

The Lovász Local Lemma

From existence to randomized construction

Rare failures, local dependence, and local repair

Many local constraints may be simultaneously satisfiable even when a union bound says nothing. The Local Lemma proves existence from local dependence. Resampling turns that existence argument into an algorithm; witness trees and reversible records explain why local repairs finish quickly.

Advanced Algorithms teaching team

Nanjing University

Lecture 6 - randomized algorithms and randomization techniques

Learning goals and reading routes

This lecture asks how a large collection of local constraints can all be satisfied, and how to find a satisfying assignment efficiently. By the end, you should be able to:

1. build a dependency graph and distinguish joint independence from pairwise independence;
2. apply the symmetric Local Lemma to satisfiability and hypergraph coloring, and prove the asymmetric lemma;
3. implement local resampling and separate resampling counts from sampling and event-testing costs;
4. prove the Moser–Tardos bound using a resampling table, witness trees, and a branching process; and
5. analyze recursive repair by encoding and then recovering the random bits used in a long execution.

For a first reading, follow the satisfiability motivation in Section 1, the Local Lemma and applications in Section 2, and its conditional-probability proof in Section 3. Section 4 introduces the resampling algorithm and its guarantees before Section 5 proves them. Section 6 gives a second, independent algorithmic proof through recursive repair and entropy compression. The two proofs certify different objects: a witness tree selected from the past, and a record of the actual recursion.

The notation and computational assumptions are introduced where needed. Throughout, n is the number of variables, m the number of constraints or bad events, and d an upper bound on the number of *other* events adjacent to one event. The symbols $\Gamma(A)$ and $\Gamma^+(A) = \Gamma(A) \cup \{A\}$ denote the open and inclusive neighborhoods. Write $[n] = \{1, \dots, n\}$; $\ln$ is natural logarithm and $\log_2$ counts bits. Empty products equal 1. A displayed probability or expectation refers to the specified random experiment, not to an arbitrary distribution on satisfying assignments.

Contents

Learning goals and reading routes	i
1 Local constraints and the probabilistic method	1
2 The Local Lemma and its applications	2
3 Why the Local Lemma is true	5
4 Finding a solution by local resampling	6
5 Why resampling terminates: tables and witness trees	8
6 Recursive repair and entropy compression	12
7 What the three stages establish	16
8 Exercises	17
A Elementary tools used in the proofs	18
Bibliographic notes	19

1 Local constraints and the probabilistic method

1.1 Satisfiability as a running example

A Boolean variable takes a value in $\{0, 1\}$, with 1 interpreted as true. A *literal* is a variable or its negation. A *clause* is a disjunction of literals; a conjunctive normal form (CNF) formula is a conjunction of clauses. For example,

$$\Phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee \neg x_4 \vee \neg x_5).$$

An assignment satisfies Φ exactly when it satisfies every clause. We use k -CNF to mean that each clause has exactly k distinct variables, with one literal per variable. In particular, a clause is not tautological. These conventions make its unique falsifying assignment on those k variables well-defined.

k -SAT and a structural search question

Given a k -CNF formula $\Phi = C_1 \wedge \dots \wedge C_m$ on n variables, k -SAT asks whether it is satisfiable. For fixed $k \geq 3$, the unrestricted problem is NP-complete. Here we seek a sufficient condition, expressed through how often clauses intersect, that guarantees a satisfying assignment and permits an efficient randomized construction.

The complexity of unrestricted satisfiability motivates the structural restriction; it does not rule out algorithms on the restricted instances studied here. The NP-completeness background originates in the Cook–Levin theory; see the bibliographic notes and [7].

If the clauses use disjoint variable sets, we can satisfy them separately. Another easy case comes from an entirely different argument. Choose $X_1, \dots, X_n$ independently and uniformly from $\{0, 1\}$. Let A_i be the bad event that C_i is violated. Then

$$\mathbb{P}(A_i) = 2^{-k}, \quad \mathbb{P}\left(\bigcup_{i=1}^m A_i\right) \leq \sum_{i=1}^m \mathbb{P}(A_i) = m2^{-k}. \quad (1)$$

The probability 2^{-k} comes from the single falsifying bit pattern, not from independence of the clauses. Only the input variables are assumed independent. Consequently, $m < 2^k$ implies a positive probability of satisfying every clause.

The probabilistic method

To prove the existence of an object with a desired property, design a probability space and show that the desired event has positive probability. An outcome in that event must then exist. Positive probability alone neither gives an efficient search algorithm nor says that a random trial is likely to succeed.

1.2 Why the total number of constraints is the wrong parameter

For disjoint clauses, the bad events are mutually independent, so

$$\mathbb{P}\left(\bigcap_{i=1}^m \overline{A_i}\right) = (1 - 2^{-k})^m > 0 \quad (2)$$

for every finite m . The union-bound restriction $m < 2^k$ is therefore unnecessary in this case. Moreover, for fixed k , the probability in (2) decays exponentially in m . Repeatedly drawing

a whole new assignment can be slow even on an instance whose clauses can be repaired independently.

Between disjoint clauses and arbitrary clauses lies *limited dependency*. Join two clauses when they share a variable, and suppose each clause meets at most d others. We would like to replace a restriction involving m by one involving this local parameter d . The Local Lemma makes this precise: one sufficient condition is

$$e(d+1)2^{-k} \leq 1. \quad (3)$$

Unlike a union bound over all clauses, this condition allows arbitrarily many constraints. It is not a union bound performed separately at every vertex; its proof must control what happens after conditioning on other constraints being satisfied.

1.3 The constraint-satisfaction viewpoint

A finite-domain constraint-satisfaction problem (CSP) has variables $x_j \in \Omega_j$ and predicates C_i defined on specified subsets of the variables. A solution makes every predicate true. Satisfiability and hypergraph coloring are two examples. Many other graph and set-system problems admit constraint formulations, but such a formulation by itself does not establish an LLL condition or an efficient event-testing routine.

Choose independent distributions for the variables and define A_i to be the event that C_i fails. An LLL application has three separate tasks: choose the probability space, bound each bad-event probability, and describe the relevant dependence. The probability space is part of the proof design; uniform choices are convenient in our examples but are not required by the lemma.

2 The Local Lemma and its applications

2.1 What a dependency graph means

Let $\mathcal{A} = \{A_1, \dots, A_m\}$ be events in a probability space. Independence of A and B means $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$. Equivalently, when $\mathbb{P}(B) > 0$, conditioning on B does not change the probability of A .

Definition 2.1 (Dependency graph). A simple undirected graph on $\mathcal{A}$ is a dependency graph if each A_i is independent of the entire family of events outside $\Gamma^+(A_i)$. Concretely, for every event B obtained by specifying occurrences or nonoccurrences of any collection of nonneighbors,

$$\mathbb{P}(A_i \cap B) = \mathbb{P}(A_i)\mathbb{P}(B). \quad (4)$$

Thus A_i is independent of the information determined jointly by its nonneighbors, not merely of each nonneighbor separately.

A dependency graph need not be unique or minimal. Adding edges preserves the definition, although it may worsen the resulting numerical bound. The event A_i itself is excluded from the nonneighbor family.

For independent variables $X_1, \dots, X_n$, let $\text{vbl}(A_i) \subseteq [n]$ be a specified set of indices determining A_i . The *variable-overlap graph* has

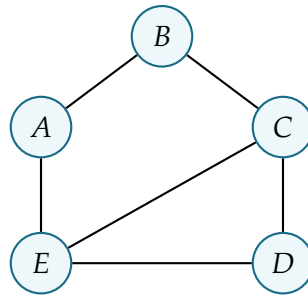
$$\Gamma(A_i) = \{A_j \neq A_i : \text{vbl}(A_i) \cap \text{vbl}(A_j) \neq \emptyset\}. \quad (5)$$

Every event outside $\Gamma^+(A_i)$ is determined by variables outside $\text{vbl}(A_i)$, as is their whole joint family. Independence of the underlying variable blocks proves that this is a dependency graph.

Example 2.2 (A small overlap graph). Consider events with scopes

$$A(X_1, X_2), \quad B(X_2, X_3), \quad C(X_3, X_4), \quad D(X_4), \quad E(X_1, X_4).$$

Their overlap graph has edges AB, AE, BC, CD, CE, DE . In particular, A is independent of the pair (C, D) jointly, even though C and D need not be independent of each other.



Pairwise independence does not suffice

Let U, V be independent fair bits and take $A_1 = \{U = 0\}$, $A_2 = \{V = 0\}$, and $A_3 = \{U \oplus V = 0\}$. The three events are pairwise independent. Nevertheless, avoiding the first two forces $U = V = 1$, so the third occurs. The empty graph is *not* a dependency graph for this family. Pairwise probability calculations cannot replace the joint condition in Definition 2.1.

2.2 The symmetric lemma, then two applications

Theorem 2.3 (Symmetric Lovász Local Lemma). Suppose $\mathbb{P}(A_i) \leq p$ for every i , and the events have a dependency graph of maximum degree at most d . If

$$ep(d+1) \leq 1, \tag{6}$$

then $\mathbb{P}(\bigcap_i \overline{A_i}) > 0$. For $d \geq 1$, the alternative condition $4pd \leq 1$ also suffices. For $d = 0$, the events are mutually independent and the conclusion holds whenever every $\mathbb{P}(A_i) < 1$.

We first apply the theorem and then derive both numerical conditions from its asymmetric version. Its two ingredients are small individual failure probabilities and limited joint dependence; a small failure probability without any dependence control is not enough.

Corollary 2.4 (Sparse k -CNF satisfiability). A k -CNF formula in which each clause shares variables with at most d other clauses is satisfiable if $e(d+1) \leq 2^k$. For $d \geq 1$, the condition $d \leq 2^{k-2}$ is another sufficient condition.

Proof. Choose independent fair bits. Each clause-failure event has probability $p = 2^{-k}$, and the clause-intersection graph is the variable-overlap graph. Apply Theorem 2.3; its alternative form is $4d2^{-k} \leq 1$. $\square$

If each variable occurs in at most L clauses, then $d \leq k(L-1)$: each of the k variables in a clause can connect it to at most $L-1$ others. Multiple shared variables only cause overcounting, which is harmless for an upper bound.

Definition 2.5 (Hypergraph coloring). A k -uniform hypergraph $H = (V, E)$ has edges that are k -element subsets of V . The degree of v is the number of edges containing it; let Δ be the maximum degree. A proper q -coloring assigns a color in $[q]$ to each vertex so that no edge is monochromatic.

Theorem 2.6 (Coloring sparse uniform hypergraphs). For $k, q \geq 2$, a k -uniform hypergraph of maximum degree Δ is q -colorable whenever

$$\Delta \leq \frac{q^{k-1}}{ek}. \quad (7)$$

Proof. Independently give each vertex a uniform color. For each edge e , let A_e be the event that all its vertices receive the same color. There are q such assignments among q^k possibilities, so $\mathbb{P}(A_e) = q^{1-k}$. Join intersecting edges. Each edge meets at most $k(\Delta - 1)$ others. If E is nonempty, $\Delta \geq 1$ and

$$ep(d+1) \leq eq^{1-k}(k(\Delta - 1) + 1) \leq eq^{1-k}k\Delta \leq 1.$$

The symmetric lemma applies. An edgeless hypergraph is trivially colorable. $\square$

For fixed q and large Δ , the condition is achieved by $k \geq \log_q \Delta + \log_q \log_q \Delta + O_q(1)$: the extra logarithm pays for the factor k . The exact finite-parameter guarantee is (7). The degree Δ of a *variable* is not the same parameter as the dependency degree d of an *event*.

2.3 Asymmetric probabilities and local budgets

The symmetric theorem uses only the largest probability and degree. When these quantities vary from event to event, a separate parameter for each event can be more effective.

Theorem 2.7 (Asymmetric Lovász Local Lemma). Let $\mathcal{A} = \{A_1, \dots, A_m\}$ have a dependency graph. Suppose there are numbers $\alpha_i \in [0, 1)$ such that

$$\mathbb{P}(A_i) \leq \alpha_i \prod_{A_j \in \Gamma(A_i)} (1 - \alpha_j) \quad \text{for every } i. \quad (8)$$

Then

$$\mathbb{P}\left(\bigcap_{i=1}^m \overline{A_i}\right) \geq \prod_{i=1}^m (1 - \alpha_i) > 0. \quad (9)$$

The parameters α_i are certificates for the inequality, not resampling probabilities or necessarily the true probabilities of A_i . Their interpretation will emerge from the proof: they bound the probability of an event even after conditioning on avoidance of other events.

Recovering the symmetric conditions. For $d \geq 1$, choose every $\alpha_i = 1/(d+1)$. Since an event has at most d neighbors,

$$\alpha_i \prod_{A_j \in \Gamma(A_i)} (1 - \alpha_j) \geq \frac{1}{d+1} \left(\frac{d}{d+1}\right)^d \geq \frac{1}{e(d+1)}.$$

The last inequality is equivalent to $(1 + 1/d)^d \leq e$. Thus (6) implies (8). Alternatively, choose $\alpha_i = 1/(2d)$ and use Bernoulli's inequality:

$$\frac{1}{2d} \left(1 - \frac{1}{2d}\right)^d \geq \frac{1}{2d} \left(1 - \frac{d}{2d}\right) = \frac{1}{4d}.$$

For $d = 0$, take $\alpha_i = \mathbb{P}(A_i) < 1$ instead; neither division by d nor the choice $1/(d+1) = 1$ is appropriate. These elementary estimates are justified in Appendix A.

3 Why the Local Lemma is true

3.1 The stronger conditional statement

For $S \subseteq [m]$, let $E_S = \bigcap_{j \in S} \overline{A_j}$. To bound the probability of avoiding all events, the chain rule suggests

$$\mathbb{P}(E_{[m]}) = \prod_{i=1}^m (1 - \mathbb{P}(A_i \mid E_{\{1, \dots, i-1\}})).$$

However, each conditional probability is defined only if the conditioning event has positive probability. The following induction establishes both facts together.

Lemma 3.1 (Conditional avoidance bound). *Under (8), for every $S \subseteq [m]$, the event E_S has positive probability. For every $i \notin S$,*

$$\mathbb{P}(A_i \mid E_S) \leq \alpha_i. \quad (10)$$

Proof. Induct on $s = |S|$. At $s = 0$, $E_\emptyset$ has probability 1 and the LLL hypothesis gives $\mathbb{P}(A_i) \leq \alpha_i$.

Suppose the assertions hold for smaller sets. First establish positivity for a set S of size $s \geq 1$. Choose $j \in S$. By induction,

$$\mathbb{P}(E_S) = \mathbb{P}(E_{S \setminus \{j\}})(1 - \mathbb{P}(A_j \mid E_{S \setminus \{j\}})) \geq \mathbb{P}(E_{S \setminus \{j\}})(1 - \alpha_j) > 0.$$

Now fix $i \notin S$ and split S into its neighbors of i and its nonneighbors:

$$S_1 = \{j \in S : A_j \in \Gamma(A_i)\}, \quad S_2 = S \setminus S_1.$$

If $S_1 = \emptyset$, joint independence gives $\mathbb{P}(A_i \mid E_S) = \mathbb{P}(A_i) \leq \alpha_i$. Otherwise, write $S_1 = \{j_1, \dots, j_\ell\}$, with $\ell \geq 1$. The definition of conditional probability gives

$$\mathbb{P}(A_i \mid E_S) = \frac{\mathbb{P}(A_i \cap E_{S_1} \mid E_{S_2})}{\mathbb{P}(E_{S_1} \mid E_{S_2})} \leq \frac{\mathbb{P}(A_i)}{\mathbb{P}(E_{S_1} \mid E_{S_2})}. \quad (11)$$

Here the numerator is at most $\mathbb{P}(A_i \mid E_{S_2}) = \mathbb{P}(A_i)$; this is precisely where independence from the entire nonneighbor family is used.

Expand the denominator, not the numerator, by the chain rule:

$$\begin{aligned} \mathbb{P}(E_{S_1} \mid E_{S_2}) &= \prod_{r=1}^{\ell} \left(1 - \mathbb{P}(A_{j_r} \mid E_{S_2 \cup \{j_1, \dots, j_{r-1}\}})\right) \\ &\geq \prod_{r=1}^{\ell} (1 - \alpha_{j_r}). \end{aligned}$$

Every conditioning set has at most $s - 1$ elements, so the induction hypothesis applies. Substitution in (11) and the LLL condition yield

$$\mathbb{P}(A_i \mid E_S) \leq \frac{\alpha_i \prod_{A_j \in \Gamma(A_i)} (1 - \alpha_j)}{\prod_{j \in S_1} (1 - \alpha_j)} = \alpha_i \prod_{\substack{A_j \in \Gamma(A_i) \\ j \notin S_1}} (1 - \alpha_j) \leq \alpha_i.$$

This completes both parts of the induction. □

3.2 From conditional bounds to a positive probability

Proof of Theorem 2.7. Lemma 3.1 makes every factor in the chain rule well-defined and bounds it below by $1 - \alpha_i$. Multiplying gives

$$\mathbb{P}(E_{[m]}) = \prod_{i=1}^m \mathbb{P}(\overline{A_i} \mid E_{\{1, \dots, i-1\}}) \geq \prod_{i=1}^m (1 - \alpha_i) > 0.$$

□

The local-to-global mechanism

Conditioning on nonneighbors does not change A_i 's probability. Conditioning on neighbors may increase it, but their avoidance probability in the denominator is bounded below inductively. The factors $(1 - \alpha_j)$ in the hypothesis pay for that increase. This is stronger than a collection of pairwise independence statements and different from applying a union bound in each neighborhood.

The proof uses only probabilities and a dependency graph. It says nothing about how an event is represented, how to sample an outcome, or how to discover a violated constraint. Those are the additional issues in an algorithmic Local Lemma.

4 Finding a solution by local resampling

4.1 The variable framework and algorithm

Let $X_1, \dots, X_n$ be independent random variables with prescribed distributions, and let $\mathcal{A} = \{A_1, \dots, A_m\}$ be bad events with given scopes $\text{vbl}(A_i)$. For the concrete algorithms, we work with finite domains after removing zero-probability values. A draw of a variable uses its own prescribed distribution; it need not be uniform. We assume we can sample variables independently, test whether a given event occurs, and access the scopes and their incidences. The analysis counts these operations before converting them to implementation time.

Use the variable-overlap graph (5). A statistical dependency graph with fewer edges might suffice for the existence proof, but cannot automatically replace this graph in the resampling analysis: two statistically independent events can still involve a common variable that a repair changes.

Moser–Tardos resampling algorithm

1. Draw each X_j independently from its prescribed distribution.
2. While some bad event occurs, choose one such event A_i and independently resample every variable X_j with $j \in \text{vbl}(A_i)$, leaving all other variables unchanged.
3. Return the assignment when no bad event occurs.

A fixed deterministic priority among violated events is sufficient. More generally, the choice may depend on the already revealed history, but not on future random samples. One *resampling* below means an event-resampling batch, which may draw several variables. Every chosen event must actually hold immediately before its resampling.

Theorem 4.1 (Moser–Tardos). Assume (8) for the variable-overlap graph. Let R_i be the total number of times A_i is resampled and $R = \sum_i R_i$. Then

$$\mathbb{E}R_i \leq \frac{\alpha_i}{1 - \alpha_i}, \quad \mathbb{E}R \leq \sum_{i=1}^m \frac{\alpha_i}{1 - \alpha_i}. \quad (12)$$

The algorithm terminates almost surely, and its returned assignment avoids every bad event.

Section 5 proves the theorem using witness trees [2]. It does not assume that the number of currently violated events decreases, or that the current assignment continues to have the original product distribution. Both assertions would generally be false.

4.2 Consequences before the proof

For $d \geq 1$ and $ep(d+1) \leq 1$, the choice $\alpha_i = 1/(d+1)$ yields

$$\mathbb{E}R_i \leq \frac{1}{d}, \quad \mathbb{E}R \leq \frac{m}{d}. \quad (13)$$

Under $4pd \leq 1$, the alternative choice $\alpha_i = 1/(2d)$ gives $\mathbb{E}R \leq m/(2d-1)$. In particular, the satisfiability and hypergraph-coloring applications become constructive: repair a violated clause, or recolor every vertex of a monochromatic edge.

For $d = 0$, the event scopes are disjoint. Resampling A_i until it disappears has mean number of batches $p_i/(1-p_i)$, including the possibility of zero batches when the initial sample is already good. Thus $\mathbb{E}R = \sum_i p_i/(1-p_i)$, where $p_i = \mathbb{P}(A_i) < 1$. This is also (12) with $\alpha_i = p_i$. The case does not require an expression involving m/d .

Why local repair is different from rejection sampling. In the disjoint-clause example, whole-assignment rejection waits for all m clauses to succeed simultaneously, with probability (2). Local resampling keeps the values in every successful clause and repairs only a failure. Its expected number of repairs is $m2^{-k}/(1-2^{-k})$. The general theorem shows that limited overlap still allows this local strategy, although repairs can now create new failures nearby.

4.3 From resampling counts to computational work

Let $s_i = |\text{vbl}(A_i)|$ and $d_i = |\Gamma(A_i)|$. Initially test all m events and store the set of currently violated ones. After resampling A_i , only events in $\Gamma^+(A_i)$ can change status. Recheck these events and update the stored set. The invariant is that the stored statuses agree with the actual current assignment.

For this implementation, the numbers of variable draws and event tests are at most

$$N_{\text{draw}} = n + \sum_i s_i R_i, \quad N_{\text{test}} \leq m + \sum_i (d_i + 1) R_i. \quad (14)$$

Taking expectations in Theorem 4.1 therefore gives

$$\begin{aligned} \mathbb{E}N_{\text{draw}} &\leq n + \sum_i s_i \frac{\alpha_i}{1 - \alpha_i}, \\ \mathbb{E}N_{\text{test}} &\leq m + \sum_i (d_i + 1) \frac{\alpha_i}{1 - \alpha_i}. \end{aligned} \quad (15)$$

These are oracle-operation counts. They exclude constructing the overlap graph and must be multiplied by the costs of the corresponding draws and tests, together with the bookkeeping cost of selecting and updating violated events.

For example, under the symmetric condition with $d \geq 1$, $s_i \leq k$ gives expected $O(m)$ event tests and at most $n + km/d$ variable draws. Testing a k -clause directly costs $O(k)$ bit inspections. An explicit list representation of all neighborhoods can itself contain $\Theta(md)$ entries; its preprocessing and storage are not made free by an $O(m)$ bound on subsequent tests.

An expected bound is not a deterministic limit on one execution. If $B > 0$ bounds $\mathbb{E}R$, Markov's inequality gives $\mathbb{P}(R > 2B) \leq 1/2$. Independent fresh runs, each truncated after $\lceil 2B \rceil$ batches, fail to finish in all r trials with probability at most 2^{-r} . This is a restart guarantee; it is not an exponential tail theorem for the unmodified run.

5 Why resampling terminates: tables and witness trees

5.1 Expose the random choices in a table

Instead of drawing randomness during the execution, sample an infinite independent table in advance. For every variable X_j , let

$$X_j^{(0)}, X_j^{(1)}, X_j^{(2)}, \dots$$

be independent samples from its prescribed law. All table entries, across variables as well as indices, are mutually independent. Initially use entry 0 in every column. Each resampling of an event advances the counters of its variables by one. The algorithm is otherwise unchanged and can be simulated by revealing entries only when needed.

Let $\Lambda = (\Lambda_1, \Lambda_2, \dots)$ be the *execution log*: $\Lambda_t \in \mathcal{A}$ records the event resampled at step t . Before step t , variable j has the value

$$X_j^{(h_j(t))}, \quad h_j(t) = |\{s < t : j \in \text{vbl}(\Lambda_s)\}|. \quad (16)$$

The event Λ_t is true on these values, and only then are the relevant counters advanced. The distinction between testing and resampling prevents an off-by-one error in the proof.

A fixed long log could in principle describe a rare execution, but summing over all m^t possible logs would ignore locality. A witness tree retains just the part of the past needed to certify one resampling.

5.2 Construct a witness from the past

Definition 5.1 (Proper witness tree). A proper witness tree is a finite rooted tree whose nodes u have event labels $A[u] \in \mathcal{A}$, such that every child's label is in Γ^+ of its parent's label, and siblings have distinct labels. Let $\mathcal{T}_A$ be the set of all such trees with root label A . The depth of the root is 0.

These are rooted labeled trees, not necessarily execution call trees. The order of children need not be recorded: distinct sibling labels can be arranged in a fixed canonical order.

Backward construction of $T(\Lambda, t)$

Start with one root labeled Λ_t . Scan $s = t - 1, t - 2, \dots, 1$. If Λ_s belongs to $\Gamma^+(A[u])$ for some node already in the tree, attach a new node labeled Λ_s as a child of a *deepest* such node. Resolve ties by a fixed rule. If no such node exists, omit Λ_s . The resulting tree is $T(\Lambda, t)$.

Deepest attachment matters. If two children of one node had the same label, the later-added child could instead attach below the existing child of that label, contradicting deepest attachment. Hence the resulting tree is proper.

Lemma 5.2 (Depth and repeated labels). *Trees produced by the backward construction have the following properties.*

1. If two included log positions $s < r$ have labels sharing a variable, the node for s is strictly deeper than the node for r . In particular, labels at the same depth involve disjoint variable sets.
2. If the root is the r th occurrence of an event A in the log, then the tree contains exactly r nodes labeled A . Thus distinct resampling steps produce distinct witness trees.

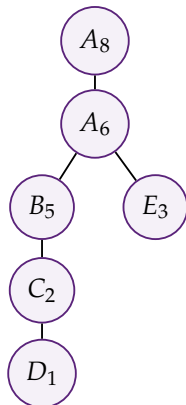
Proof. When position s is processed, the node for r is already present. Their labels overlap, so the chosen attachment point has depth at least that of the node for r . The new node is one level deeper. For the second assertion, every earlier occurrence of the root label is included because it is compatible with the root. These are all occurrences of that label up to the root time. If root labels differ, trees differ; if they agree, their counts of that label differ at different occurrence numbers. $\square$

5.3 A concrete witness and its table entries

Use the five scopes in Example 2.2, and consider the log prefix

$$\Lambda = D, C, E, D, B, A, C, A.$$

At time 8, the root is A . The scan includes positions 6, 5, 3, 2, 1 and omits 7, 4. At the tie for position 2, choose the node labeled B as parent. Subscripts in the following display identify log positions, not different event types.



Node	Entries used to test the event
D_1	$X_4^{(0)}$
C_2	$X_3^{(0)}, X_4^{(1)}$
E_3	$X_1^{(0)}, X_4^{(2)}$
B_5	$X_2^{(0)}, X_3^{(1)}$
A_6	$X_1^{(1)}, X_2^{(1)}$
A_8	$X_1^{(2)}, X_2^{(2)}$

The entries assigned to different nodes are disjoint. Notice that E_3 and C_2 share X_4 but lie on different branches. Sharing a variable does *not* force two witness nodes to be ancestors of one

another; the correct ordering property is their distinct depths. The construction describes the witness associated with this log; it does not assert that an arbitrary family of predicates with these scopes realizes every such log.

5.4 The witness-tree probability bound

Lemma 5.3 (Witness-tree lemma). *For every fixed proper witness tree τ ,*

$$\mathbb{P}(\exists t : T(\Lambda, t) = \tau) \leq \prod_{u \in \tau} \mathbb{P}(A[u]). \quad (17)$$

Proof. If two nodes at the same depth have overlapping variable scopes, Lemma 5.2 shows that τ cannot occur, so the assertion is immediate. Otherwise define, for every $j \in \text{vbl}(A[u])$,

$$r_\tau(u, j) = |\{v \in \tau : \text{depth}(v) > \text{depth}(u), j \in \text{vbl}(A[v])\}|. \quad (18)$$

Test $A[u]$ using the tuple $(X_j^{(r_\tau(u, j))})_{j \in \text{vbl}(A[u])}$ from the random table. This is a test associated with a *fixed tree*, not the adaptive test made at a randomly chosen execution time.

For any fixed variable j , the nodes whose labels use j have distinct depths. Listed from deepest to shallowest, they use consecutive table indices $0, 1, \dots$. Thus no two node tests share a table cell. Independence of all cells implies that the node tests are independent, each with its original event probability. Consequently,

$$\mathbb{P}(\text{every test for } \tau \text{ succeeds}) = \prod_{u \in \tau} \mathbb{P}(A[u]). \quad (19)$$

It remains to show that an actual occurrence of τ forces all these tests to succeed *on the same table*. Suppose $\tau = T(\Lambda, t)$ and node u represents log position s . Fix a variable j used by its label. Every earlier position $r < s$ involving j is included in the tree: when the backward scan reaches r , node u is already present and provides a compatible attachment point. All those nodes are deeper than u . Conversely, an included later position involving j is shallower, by Lemma 5.2. Therefore

$$r_\tau(u, j) = |\{r < s : j \in \text{vbl}(\Lambda_r)\}| = h_j(s).$$

The fixed-tree test uses exactly the values on which Λ_s was true just before its resampling. Hence every node test succeeds. This inclusion of events, together with (19), proves the bound even after taking the union over all possible occurrence times t . $\square$

Where independence reappears

The successive resampling events need not be independent and need not have their original probabilities after conditioning on the execution history. What is independent is a collection of disjoint table-cell tests for one fixed witness tree. Local dependence limits the possible witness shapes; fresh randomness makes a fixed witness expensive in probability.

5.5 Summing all witnesses with a branching process

By distinctness of the generated trees,

$$R_i = \sum_{\tau \in \mathcal{T}_{A_i}} \mathbf{1}\{\exists t : T(\Lambda, t) = \tau\}, \quad \mathbb{E}R_i \leq \sum_{\tau \in \mathcal{T}_{A_i}} \prod_{u \in \tau} \mathbb{P}(A[u]). \quad (20)$$

These identities and inequalities also make sense before termination is proved: all terms are nonnegative, and finite truncation followed by monotone convergence justifies the sums. We must now bound a potentially infinite sum of tree weights.

First suppose all relevant α_i are positive. Define

$$w(A_i) = \alpha_i \prod_{A_j \in \Gamma(A_i)} (1 - \alpha_j). \quad (21)$$

The LLL hypothesis says $\mathbb{P}(A_i) \leq w(A_i)$.

Auxiliary branching process with root A_i

Create a root labeled A_i with certainty. For each created node u , and independently for every $B \in \Gamma^+(A[u])$, create one child labeled B with probability α_B ; otherwise create no child with that label. Repeat for new nodes. The result T_{A_i} may be finite or infinite.

The process is a multitype Galton–Watson process: the event label is the type. It creates proper witness trees, but not every tree it can create must be realizable by an execution. That extra mass only helps an upper bound.

Lemma 5.4 (Branching probability). *For a finite proper tree τ with root label A_i ,*

$$\mathbb{P}(T_{A_i} = \tau) = \frac{1 - \alpha_i}{\alpha_i} \prod_{u \in \tau} w(A[u]). \quad (22)$$

Proof. Let $C(u)$ be the set of child labels at u . Every nonroot node contributes the probability $\alpha_{A[u]}$ that it is born, while each missing potential child contributes its nonbirth probability. Thus

$$\mathbb{P}(T_{A_i} = \tau) = \prod_{u \neq \text{root}} \alpha_{A[u]} \prod_{u \in \tau} \prod_{B \in \Gamma^+(A[u]) \setminus C(u)} (1 - \alpha_B). \quad (23)$$

Start instead with all potential nonbirth factors at each node. Remove the factor corresponding to each actual child and replace it by its birth factor. This rewrites (23) as

$$\left(\prod_{u \in \tau} \prod_{B \in \Gamma^+(A[u])} (1 - \alpha_B) \right) \left(\prod_{u \neq \text{root}} \frac{\alpha_{A[u]}}{1 - \alpha_{A[u]}} \right).$$

Insert the missing root ratio and compensate by $(1 - \alpha_i)/\alpha_i$. At every node the own-label factor cancels, leaving $\alpha_{A[u]} \prod_{B \in \Gamma(A[u])} (1 - \alpha_B) = w(A[u])$. $\square$

Proof of Theorem 4.1. Combining (20), the LLL condition, and Lemma 5.4, we obtain

$$\begin{aligned} \mathbb{E}R_i &\leq \sum_{\tau \in \mathcal{T}_{A_i}} \prod_{u \in \tau} \mathbb{P}(A[u]) \\ &\leq \sum_{\tau \in \mathcal{T}_{A_i}} \prod_{u \in \tau} w(A[u]) \\ &= \frac{\alpha_i}{1 - \alpha_i} \sum_{\tau \in \mathcal{T}_{A_i}} \mathbb{P}(T_{A_i} = \tau) \leq \frac{\alpha_i}{1 - \alpha_i}. \end{aligned}$$

The last sum is at most 1, because the branching process might be infinite. No extinction theorem is needed.

If $\alpha_i = 0$, the hypothesis forces $\mathbb{P}(A_i) = 0$. In the finite-domain model with positive marginal probabilities on the domains, such an event is impossible and can be removed; it is never

resampled. Removing it from neighborhoods deletes only a factor 1. Apply the preceding proof to the remaining events. Thus the statement holds for the allowed interval $[0, 1)$ without dividing by zero.

Summing over i gives a finite upper bound on $\mathbb{E}R$. A nonnegative extended random variable with finite expectation is finite almost surely, so the algorithm terminates almost surely. When it stops, its stopping condition certifies that all constraints are satisfied. $\square$

What the proof has accomplished. A potentially long adaptive execution was replaced by countably many finite certificates. Each certificate has probability at most the product of its local event probabilities. A separate random tree accounts for the total weight of all certificates. The auxiliary tree is a tool for summation, not the algorithm's actual trajectory or a distribution of satisfying assignments.

6 Recursive repair and entropy compression

6.1 Moser's recursive Fix-It algorithm

Return to a k -CNF formula, and fix an ordering of the clauses and of every inclusive neighborhood. The resampling algorithm can be scheduled recursively so that a repaired clause is checked together with everything it may directly disturb.

Moser's recursive algorithm

Main procedure. Draw n independent fair bits. While a violated clause exists, choose the first such clause C and call $\text{Fix}(C)$. Return the assignment when all clauses are satisfied.

Procedure $\text{Fix}(C)$.

1. Resample the k variables of C with fresh independent fair bits.
2. While a clause $D \in \Gamma^+(C)$ is violated, choose the first such D and call $\text{Fix}(D)$.
3. Return when every clause in $\Gamma^+(C)$ is satisfied.

The inclusive neighborhood contains C itself: a resampling may leave the same clause violated. Every call begins with a violated clause. One call performs one batch of k fresh draws, followed by zero or more recursive calls. The chronological batches are a legal Moser–Tardos execution, so Theorem 4.1 already applies when its condition holds. We now give an independent proof under a slightly stronger simple numerical condition. Its purpose is to expose a different mechanism, not to improve the Moser–Tardos threshold.

Lemma 6.1 (Preservation at return times). *If a call $\text{Fix}(C)$ returns, every clause satisfied at entry is satisfied at return, and every clause in $\Gamma^+(C)$ is satisfied at return. Consequently, at most m top-level calls can be started, including a possible final unfinished call.*

Proof. The first assertion concerns a finite completed recursion and can be proved by induction on its number of calls; it does not presume that every call terminates. The first resampling changes only variables of C , so it cannot affect a clause outside $\Gamma^+(C)$. Each completed recursive child preserves all clauses satisfied at its own entry, by induction. Thus the clauses outside $\Gamma^+(C)$ that were satisfied at entry to the parent remain satisfied at the parent's return. Its stopping

condition makes every clause inside $\Gamma^+(C)$ satisfied as well. This proves the preservation statement.

At each completed top-level call, all previously satisfied clauses are restored and its initially violated root clause becomes satisfied. At these top-level boundaries the set of satisfied clauses strictly increases. A later top-level root cannot be any previously repaired root, so there are at most m started top-level calls. $\square$

The invariant is about return times

A clause repaired by an earlier top-level call may temporarily become false during a later recursive repair. Preservation is guaranteed after that later call returns, not at every intermediate resampling. The argument counts distinct top-level roots; it does not say that each clause is resampled only once.

6.2 Why a long execution might encode its own randomness

Just before resampling a violated k -clause, the values of all its k variables are known from the clause label: every literal is false. Thus resampling overwrites a *known* local assignment with k random bits. If the labels of the resampled clauses are recorded, a final assignment lets us undo all those overwrites and recover the random bits.

Recording a clause label naively takes $\lceil \log_2 m \rceil$ bits per call. Recursion changes that cost. A nonroot call is to a clause in the inclusive neighborhood of its parent, so its label needs only a rank among at most $d + 1$ choices. Global names are needed only at the at most m top-level roots. If a call consumes more random bits than the record needs to describe it, a long run would turn many uniformly random bits into a shorter, recoverable description.

Lemma 6.2 (Finite incompressibility). *Let $\mathcal{B} \subseteq \{0,1\}^N$. If an encoding is injective on $\mathcal{B}$ and uses at most L bits per encoded string, then*

$$\frac{|\mathcal{B}|}{2^N} < 2^{L+1-N}. \quad (24)$$

If all codes have exactly L bits, the sharper bound is $|\mathcal{B}|/2^N \leq 2^{L-N}$. In particular, a uniform N -bit string has an injective description of length at most $N - \ell$ only with probability less than $2^{1-\ell}$.

Proof. There are $\sum_{r=0}^L 2^r = 2^{L+1} - 1$ binary strings of length at most L , and exactly 2^L of length L . Injectivity bounds $|\mathcal{B}|$ by the appropriate count. Divide by the 2^N equally likely inputs. $\square$

The lemma does not prohibit compression of a particular string. It says that a large fraction of uniformly random strings cannot all receive much shorter distinct descriptions. We need no machinery of Kolmogorov complexity beyond this elementary counting fact.

6.3 Freeze after a fixed number of resamplings

Fix integers n, m, k, t and the formula before choosing randomness, with $m \geq 1$ and $t \geq 1$. Supply a uniform tape of length

$$N = n + tk. \quad (25)$$

The first n bits initialize the assignment; successive blocks of k bits supply successive calls, in a fixed order of the variables of each clause. If the run reaches its t th call, stop it *immediately after*

that call's first resampling. The entire tape has then been used. If it terminates earlier, the tape does not belong to the long-run set $\mathcal{B}_t$ that we will encode.

A call opens a node of a recursion tree; its recursive children are ordered by their call times. Completed top-level calls form a forest, and the last tree can have an unfinished recursion stack. Record the actual openings and returns up to the freeze, then append closing parentheses for the remaining open stack. This produces a complete ordered forest with exactly t nodes. The added closing symbols describe the *record* only; they do not assert that those calls actually returned.

Set

$$a = \lceil \log_2 m \rceil, \quad b = \lceil \log_2 (d + 1) \rceil. \quad (26)$$

The following fixed-length encoding avoids any ambiguity about the end of a partial recursion record.

Encode the first t calls and the current assignment

Record four fields, whose lengths are known from n, m, d, t :

1. **Forest shape:** $2t$ bits, using 1 to open a node and 0 to close it in depth-first order.
2. **Root labels:** m slots of a bits. Store the root labels in order and fill unused slots with zeroes.
3. **Nonroot labels:** t slots of b bits. Store each nonroot's rank in its parent's ordered inclusive neighborhood, in preorder; fill unused slots with zeroes.
4. **Current assignment:** the n bits after the t th resampling.

Ranks and labels are stored as zero-based integers in fixed-width binary. When a width is zero, the unique available choice needs no bits.

The number of roots and the number of nonroot nodes are determined by the forest shape, so the decoder knows which slots are padding. The shape uses at most m roots by Lemma 6.1. The total code length is

$$L = n + ma + t(b + 2). \quad (27)$$

Neither the formula nor the parameters are part of the transmitted record: they are fixed side information common to all tapes in this counting argument. The length includes the complete binary encoding of the forest, not just its clause labels.

6.4 Decode by running the overwrites backwards

The forest and its labels determine the chronological sequence $C_{i_1}, \dots, C_{i_t}$: every call resamples immediately upon entry, so call preorder is resampling order. The final assignment supplies the state from which to reverse that sequence.

Recover the random tape

Start with the recorded assignment. For $s = t, t-1, \dots, 1$:

1. Read the current bits on the variables of C_{i_s} , in their prescribed order. Save them as the k random bits used at call s .
2. Replace those variables by the unique assignment that falsifies C_{i_s} ; leave all other variables unchanged.

After all t reversals, the current assignment is the initial n bits. Output them followed by the recovered blocks in order $1, \dots, t$.

Lemma 6.3 (Reversibility). *The decoding procedure reconstructs the entire $n + tk$ -bit tape for every tape in $\mathcal{B}_t$. Thus the encoding is injective on $\mathcal{B}_t$.*

Proof. Induct backwards on s . Before undoing call s , all later resamplings have already been undone, so the maintained assignment is exactly the state immediately after its resampling. On its scope, those values are exactly the newly drawn random block. Immediately before the call, its clause was violated, which uniquely determines the old values on the scope. Replacing the scope by that falsifying pattern therefore reconstructs the preceding state. Repeating reaches the initial assignment and recovers every block. $\square$

For instance, if $C = x_1 \vee \neg x_2 \vee x_3$, the predecessor values on its scope must have been $(0, 1, 0)$. After later changes are undone, the current scope values reveal the three bits drawn to replace that pattern. Overlap with other clauses does not prevent reversal, because those later overwrites are undone first.

6.5 Counting gives a tail bound

Let R be the number of calls in the recursive algorithm. This is also its number of resampling batches. Define the bit surplus

$$\gamma = k - b - 2. \quad (28)$$

The two subtracted terms are the local label cost and the per-node forest-shape cost. From (25), (27), and Lemma 6.3, fixed-length incompressibility gives

$$\mathbb{P}(R \geq t) = \frac{|\mathcal{B}_t|}{2^{n+tk}} \leq \min\{1, 2^{ma-\gamma t}\}. \quad (29)$$

The equality uses an *unconditioned* uniform tape. We have not assumed that the tape remains uniform after conditioning on a long execution. Only the long-run subset is encoded, and the counting lemma was stated for such subsets.

Theorem 6.4 (Entropy-compression bound for recursive repair). *For $k \geq 3$, suppose each clause intersects at most d others and $d < 2^{k-3}$. For every $0 < \delta < 1$, the recursive algorithm returns a satisfying assignment, with probability at least $1 - \delta$, before*

$$t_\delta = \lceil m \lceil \log_2 m \rceil + \log_2(1/\delta) \rceil \quad (30)$$

resampling calls have been made. In particular, the number of calls is $O(m \log(2m) + \log n)$ with probability at least $1 - 1/n$ for $n \geq 2$.

Proof. Since d is an integer and $k \geq 3$, $d < 2^{k-3}$ implies $d + 1 \leq 2^{k-3}$. Hence $b \leq k - 3$ and $\gamma \geq 1$. Formula (29) gives $\mathbb{P}(R \geq t_\delta) \leq \delta$. Its tail tends to zero, proving almost-sure termination independently of the Moser–Tardos theorem. At termination the main loop certifies satisfaction. Taking $\delta = 1/n$ gives the final statement; $\log(2m)$ keeps it meaningful for $m = 1$. $\square$

More generally, whenever $\gamma > 0$, the threshold $\lceil (ma + \log_2(1/\delta))/\gamma \rceil$ suffices. The stated condition is a convenient way to guarantee at least one saved bit per call. When $d = 0$, disjoint clauses can also be handled directly, for every $k \geq 1$, without requiring this particular code to save bits.

The bound counts calls, not only top-level invocations and not individual variable assignments. Each call draws k bits. A direct implementation can maintain violated-event statuses using the same local updates as in Section 4.3; a recursion stack and searches for violated neighbors add bookkeeping. With finite explicit scopes and neighborhoods, these costs are polynomial in the input size and the number of calls. A fixed execution budget bounds the stack by that same budget.

Two kinds of trees, two kinds of accounting

A Moser–Tardos witness tree is extracted *backwards* from the log to certify one resampling. Its weight is bounded using independent table cells. A recursive call tree records the algorithm's *actual* nested calls. Together with the final assignment, it recovers the random tape. The former supports a sum of witness probabilities; the latter supports an injective encoding of long runs.

7 What the three stages establish

The Local Lemma separates local failure from global feasibility. The Moser–Tardos theorem adds an algorithmic representation and a local repair rule. Entropy compression provides a second explanation of efficient repair by showing that too much failure would reveal too much about the random choices.

Method	Additional structure	Conclusion proved here
Classical LLL	A dependency graph and event probabilities	Positive avoidance probability; no search-time claim.
Moser–Tardos	Independent variable scopes, fresh sampling, event tests	Expected resampling count $\sum_i \alpha_i / (1 - \alpha_i)$.
Recursive entropy proof	A k -clause has one falsifying local assignment; recursive neighbor ranks	Explicit high-probability call bound when $d < 2^{k-3}$.

Further directions

The criterion $ep(d + 1) \leq 1$ is a useful sufficient condition, not the exact characterization of every dependency graph. Shearer's work gives a sharp description in terms of the graph and event probabilities [4]. More information about the events can support different or stronger arguments than a graph-only criterion. We do not need those refinements for any theorem in this lecture.

Outside a product space, there may be no independent variable block to redraw. Resampling-oracle frameworks formalize alternative local repair operations [5]. Their hypotheses must be verified; an abstract dependency graph alone does not supply an efficient repair operation.

Finally, *finding* a satisfying assignment and *sampling* from a specified law on satisfying assignments are different tasks. For a product measure μ and $\mathcal{S} = \bigcap_i \overline{A_i}$, the natural conditional sampling target is

$$\mu(\sigma \mid \mathcal{S}) = \frac{\mu(\sigma) \mathbf{1}\{\sigma \in \mathcal{S}\}}{\mu(\mathcal{S})}.$$

The termination proofs above establish support on $\mathcal{S}$, not this output formula. Distributional properties of the Moser–Tardos output require additional analysis [6]. The expected-time theorem should not be read as an exact-sampling theorem.

8 Exercises

Modeling dependence and applying the lemma

1. **The union-bound threshold.** Construct an unsatisfiable k -CNF with 2^k clauses on k variables. Why does this not contradict Corollary 2.4? Compare whole-assignment rejection with local repair when the clauses have disjoint scopes.
2. **Joint versus pairwise independence.** Verify all pairwise independence claims in the three-event fair-bit example. Identify a joint event formed from two of the indicators that is not independent of the third.
3. **Variable occurrences.** If every variable occurs in at most L clauses, derive a sufficient condition on k, L for satisfiability. For hypergraph coloring, compare the exact criterion $eq^{1-k}(k(\Delta - 1) + 1) \leq 1$ with (7).
4. **Unequal budgets.** Consider a star dependency graph with center probability p_0 and r leaves of probabilities $p_1, \dots, p_r$. For a proposed center parameter $a \in (0, 1)$, set $\alpha_j = p_j / (1 - a)$ at each leaf. State the resulting sufficient conditions, including $\alpha_j < 1$. Explain why the largest-probability symmetric bound can discard useful information.

Resampling and witness trees

5. **Positive conditioning events.** In Lemma 3.1, mark each use of the induction hypothesis and the size of its conditioning set. Why is a positivity argument necessary before writing (11)?
6. **Build the next witness.** Extend the log in Section 5.3 by a ninth label D . Construct $T(\Lambda, 9)$ using a fixed deepest-parent tie rule. Which positions are omitted? Verify the table index at every included node.
7. **Proper trees versus generated trees.** Give a proper witness tree whose two labels at one depth share a variable, and explain why it cannot be generated by the backward scan. Why is it nevertheless harmless for the branching-process sum to include this tree?
8. **Cost and confidence.** Derive (15) for event-specific draw and test costs. Then obtain a high-confidence restart guarantee from a known expected batch bound. Distinguish that guarantee from a tail bound for a single uninterrupted execution.

Recursive repair and reversible information

9. **Return-time preservation.** Prove Lemma 6.1 by induction on the completed recursion tree. Explain why no assumption of almost-sure termination is used in this induction, and why temporary violations do not invalidate the bound of m top-level roots.
10. **Check every field of the code.** Write a decoder for the four fields in Section 6.3. Explain how it recognizes roots, padding, and rank widths. Why may unfinished calls be closed in the encoded shape without being executed further?
11. **Several forbidden local assignments.** Suppose each bad constraint involves k fair bits and has at most $s \geq 1$ forbidden assignments. Keep the same recursive repair scheme, and add a rank specifying the pre-resampling forbidden assignment for every call. Show that a sufficient bit-surplus condition for the analogous entropy argument is $k > b + 2 + \lceil \log_2 s \rceil$. State what changes in the decoder and the tail bound.
12. **Compare the two constructive analyses.** For $k = 10$ and $d = 100$, determine which of $ep(d+1) \leq 1$, $4pd \leq 1$, and $d < 2^{k-3}$ hold when $p = 2^{-k}$. Compare the information supplied by the two proofs about resampling count, confidence, and output distribution.

A Elementary tools used in the proofs

A.1 Products, logarithms, and repeated trials

For $u > -1$, $\ln(1+u) \leq u$: the difference $u - \ln(1+u)$ is minimized at zero. Hence $(1+1/d)^d \leq e$ for $d \geq 1$. For $0 \leq x \leq 1$ and an integer $d \geq 1$, Bernoulli's inequality $(1-x)^d \geq 1-dx$ follows by induction: multiplying the bound at d by $1-x$ produces $1-(d+1)x+dx^2 \geq 1-(d+1)x$.

A trial that fails with probability $p < 1$ and is independently repeated until the first success has expected number of failures

$$\sum_{r \geq 1} p^r = \frac{p}{1-p}.$$

The initial trial matters: this counts resamplings when the initial evaluation is allowed to succeed, rather than the total number of independent evaluations, whose mean is $1/(1-p)$.

A.2 Nonnegative counts and almost-sure termination

For a nonnegative integer-valued random variable R , possibly infinite,

$$R = \sum_{t \geq 1} \mathbf{1}\{R \geq t\}, \quad \mathbb{E}R = \sum_{t \geq 1} \mathbb{P}(R \geq t).$$

Both identities follow by first summing finitely many terms and then increasing the truncation. The same argument permits swapping expectation with a countable sum of nonnegative witness indicators. If $\mathbb{E}R < \infty$, then $\mathbb{P}(R = \infty) = 0$, since $\mathbb{E}R \geq T\mathbb{P}(R = \infty)$ for every integer T .

For $a > 0$, the inequality $R \geq a\mathbf{1}\{R \geq a\}$ implies Markov's inequality $\mathbb{P}(R \geq a) \leq \mathbb{E}R/a$. It yields the restart analysis in Section 4.3. For the stronger geometric tail (29) with $\gamma \geq 1$, summing the tail also gives $\mathbb{E}R \leq ma + 2$. This is a consequence of the explicit code, separate from the sharper general Moser–Tardos expectation bound.

A.3 Why the algorithmic proof extends beyond finite domains

The table proof uses only mutually independent fresh samples, measurable event tests, and finite scopes. It therefore also applies to other variable distributions whenever these operations are well-defined. For a zero-probability bad event, each fixed tuple of table indices realizes that event with probability zero. There are only countably many such finite tuples and finitely many events; a countable union shows that almost surely no table evaluation ever realizes a zero-probability event. This replaces the finite-support deletion argument without changing the proof.

Bibliographic notes

The Local Lemma originates with Erdős and Lovász [1]. Its conditional-probability proof and its applications to local constraints form the first stage of this lecture. Moser's constructive work [3] and the general Moser–Tardos theorem [2] supply the algorithmic stage. Section 6 gives an elementary entropy-compression analysis of recursive repair, with an explicit fixed-length forest code. The explicit forest code yields the high-probability bound in Theorem 6.4. The research directions in Section 7 point to further reading; their stronger results are not prerequisites for the lecture.

References

- [1] Paul Erdős and László Lovász. Problems and Results on 3-Chromatic Hypergraphs and Some Related Questions. In *Infinite and Finite Sets*, Vol. II, Colloquia Mathematica Societatis János Bolyai 10, pages 609–627. North-Holland, 1975. [Original paper](#).
- [2] Robin A. Moser and Gábor Tardos. A Constructive Proof of the General Lovász Local Lemma. *Journal of the ACM*, 57(2), Article 11, 1–15, 2010. [Author manuscript](#); doi:10.1145/1667053.1667060.
- [3] Robin A. Moser. A Constructive Proof of the Lovász Local Lemma. In *Proceedings of STOC*, pages 343–350, 2009. [Earlier author manuscript](#).
- [4] James B. Shearer. On a Problem of Spencer. *Combinatorica*, 5:241–245, 1985. doi:10.1007/BF02579368.
- [5] Nicholas J. A. Harvey and Jan Vondrák. An Algorithmic Proof of the Lovász Local Lemma via Resampling Oracles. *SIAM Journal on Computing*, 49(2):394–428, 2020. doi:10.1137/18M1167176.
- [6] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New Constructive Aspects of the Lovász Local Lemma. [Author manuscript](#), arXiv:1001.1231, 2010.
- [7] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of STOC*, pages 151–158, 1971. doi:10.1145/800157.805047.