Combinatorial Mathematics

Mong-Jen Kao (高孟駿)

Monday 18:30 – 20:20

Theorem 19.2 (The Lovász Local Lemma – Asymmetric version).

Let $G = (V, E)$ be a dependency graph of events $A_1, A_2, \dots, A_n$.

Suppose that there exists real numbers $x_1, x_2, \dots, x_n$ with $0 \le x_i < 1$ such that, for all i,

$$\Pr[A_i] \;\le\; x_i \cdot \prod_{j:(i,j)\in E} \left(1 - x_j\right) .$$

Then

$$\Pr\left[\overline{A_1}\,\overline{A_2}\cdots\overline{A_n}\right] \;\ge\; \prod_{1\le i\le n} (1 - x_i) .$$

In particular, with positive probability, no A_i occurs.

Q: Can we actually construct the object ?

We will show in this lecture that,

the object can be *constructed* **in *expected*** $\boldsymbol{\sum_i \frac{x_i}{1-x_i}}$ ***number of resamples*,**

assuming the prerequisite conditions of the local lemma,

under a *common algorithmic variable setting*.

Some Notes

- The result is from the following *award-winning* paper.
  - Robin A. Moser, Gabor Tardos,
    "A constructive proof of the general Lovász local lemma."
    Journal of ACM 57(2): 11:1 – 11:15, 2010.

The result is described using only 4 pages !

- It answers a general & fundamental problem,
  with a *surprisingly simple* algorithm and analysis, and beautiful ideas.

- This paper was awarded *the Gödel prize* by the European Association for Theoretical Computer Science (EATCS) in 2020.

Outline

- Algorithmic Lovász Local Lemma

  (A ***constructive proof*** for the Lovász Local Lemma)

  - The Variable Setting Assumption
  - A Simple Randomized Algorithm
  - The Analysis
    - Notations & Definitions
    - The Galton-Watson branching process
    - Coupling the execution & evaluation

The *Variable Setting Assumption*

- We assume the following setting,
  which is common in algorithmic context.
  - The <u>*object to compute*</u> is *described by*
    a set of random variables, $Z_1, Z_2, \dots, Z_n$,
    that are <u>*mutually independent*</u> in a *fixed probability space*.
  - Each bad event A_i is determined by
    a subset of variables in $\{Z_1, \dots, Z_n\}$, denoted by $vbl(A_i)$.

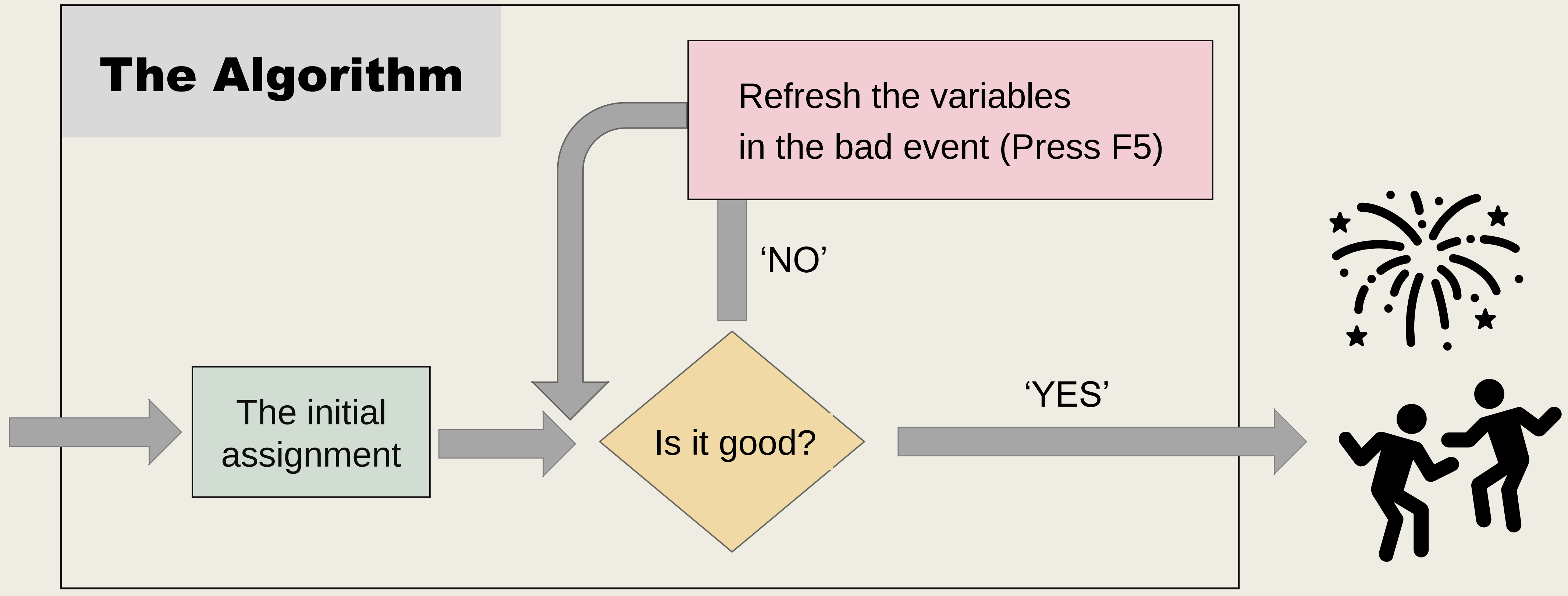
A *Simple & Elegant* Randomized Algorithm

- The following algorithm is due to [Moser & Tardos, 2010].

1. Pick an independent random assignment for Z_j, $1 \leq j \leq n$.
2. Repeat until none of the bad events A_i holds.
   - *Pick a violated event*, say A_i .
   - *Resample the value of* Z_j for all $Z_j \in vbl(A_i)$.

Roughly Speaking…

- The algorithm *keeps refreshing* the variables in the violating event until all the events are avoided.



IS THAT IT ?

- Clearly,
  when the algorithm stops, we have a feasible set of assignments.

- The question is,

Is the ‘<u>seemingly inefficient</u>’ algorithm efficient?

We can always come up with all sorts of algorithms.
The question is always, how do we be sure that it’s a good one?

The Dependency Graph

- Define the dependency graph for the events as follows.
  - For any i, j,
    there is an edge between A_i and A_j if and only if
    $$vbl(A_i) \cap vbl(A_j) \neq \emptyset\,.$$

- For any i,
  let D_i be the neighbors of A_i in the dependency graph.

The Algorithmic Lovász Local Lemma

Theorem 1 (Moser-Tardos 2010).

In the variable setting, if there exists $x_i \in (0,1)$ such that

$$\Pr[A_i] \leq x_i \cdot \prod_{j \in D_i} (1 - x_j), \qquad \forall 1 \leq i \leq n,$$

then the algorithm resamples an event A_i at most an expected number of $\frac{x_i}{1-x_i}$ times before it finds a feasible assignment.

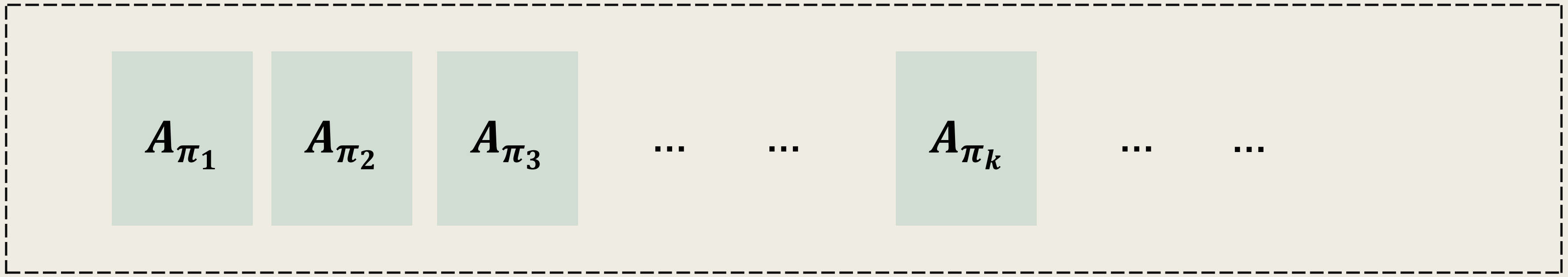
(Sketch)

Proof of Theorem 1

The Idea

- For any $1 \leq i \leq m$,
  let N_i denote the number of times the event A_i is resampled.
  - We will show that,

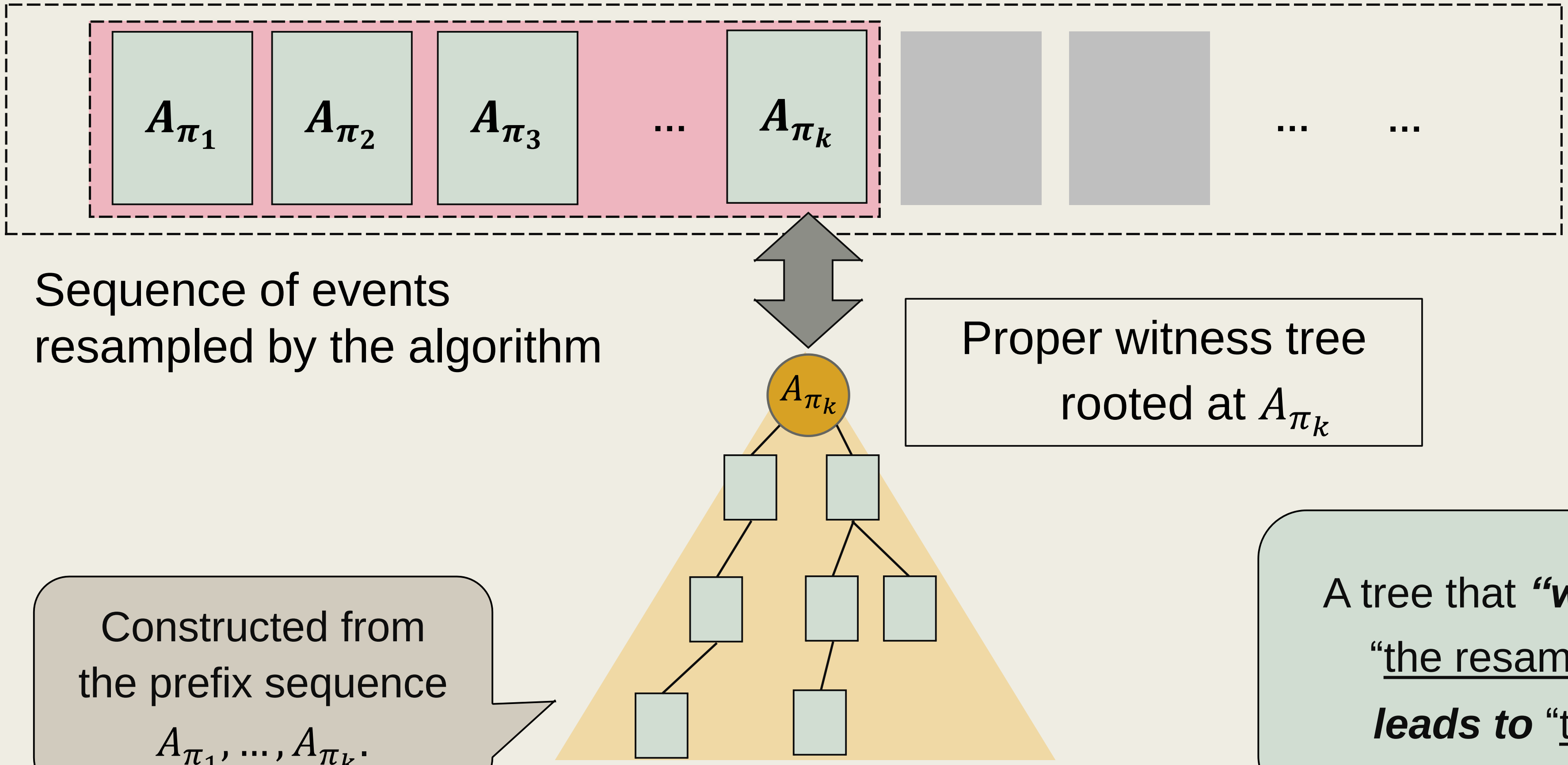
$$E[N_i] \leq \frac{x_i}{1 - x_i}.$$



Sequence of events resampled by the algorithm

- To bound $E[N_i]$,
  for any $k \geq 1$, consider the first k events resampled by the algorithm.

  We will associate the sequence $A_{\pi_1}, A_{\pi_2}, \ldots, A_{\pi_k}$ with a ***Proper Witness Tree***.



Sequence of events resampled by the algorithm

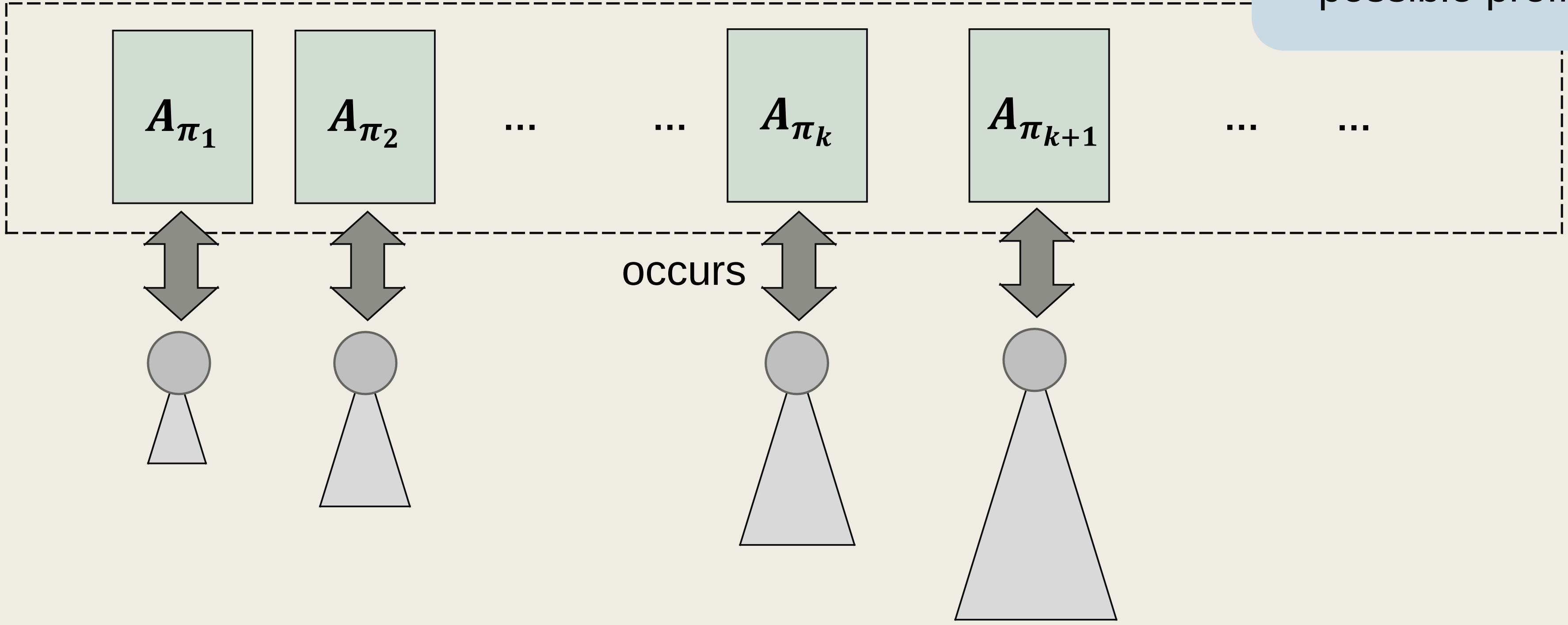
Proper witness tree rooted at A_{π_k}

Constructed from the prefix sequence $A_{\pi_1}, \ldots, A_{\pi_k}$.

A tree that ***"witnesses"*** the fact that "the resamples of $A_{\pi_1}, \ldots, A_{\pi_{k-1}}$" ***leads to*** "the resample of A_{π_k}."

Sequence of events
resampled by the algorithm

Consider the witness trees for all possible prefixes of the sequence.



- Then

$$E[N_i] = \sum_{\substack{T:\\ \text{possible proper}\\ \text{witness trees } \textbf{with root } A_i}} \Pr[\ T \text{ occurs in the sequence }]$$

<u>Lemma 2.</u> (To be proved later)

For any proper witness tree T of the events, we have

$$\Pr[\, T \text{ occurs} \,] \le \prod_{v \in T} \Pr[\, A_{[v]} \,] \; .$$

$A_{[v]}$ denotes the event to which node v corresponds.

- By Lemma 2, we have

$$E[N_i] = \sum_{T \in T_i} \Pr[\, T\ occurs \,] \le \sum_{T \in T_i} \prod_{v \in T} \left(x_{[v]} \cdot \prod_{j \in D_{[v]}} (1 - x_j) \right) \; .$$

We bound the sum using the ***<u>"Galton-Watson" random branching process</u>***.

- For any $T \in T_i$, let p_T be the probability that the random *Galton-Watson process* generates T.

Lemma 3. (To be proved)

For any $T \in T_i$, we have

$$p_T = \frac{1 - x_i}{x_i} \cdot \prod_{v \in T} \left(x_{[v]} \cdot \prod_{j \in D_{[v]}} (1 - x_j) \right).$$

We will describe the random process later.

Putting Things Together…

- By Lemma 2 and Lemma 3, we obtain

$$E[N_i] = \sum_{T \in T_i} \Pr[\, T\ occurs\,] \leq \sum_{T \in T_i} \prod_{v \in T} \left(x_{[v]} \cdot \prod_{j \in D_{[v]}} (1 - x_j) \right)$$

$$= \frac{x_i}{1 - x_i} \cdot \sum_{T \in T_i} p_T$$

$$\leq \frac{x_i}{1 - x_i}\,.$$

It remains to prove the two Lemmas.

Outline

- Algorithmic Lovász Local Lemma

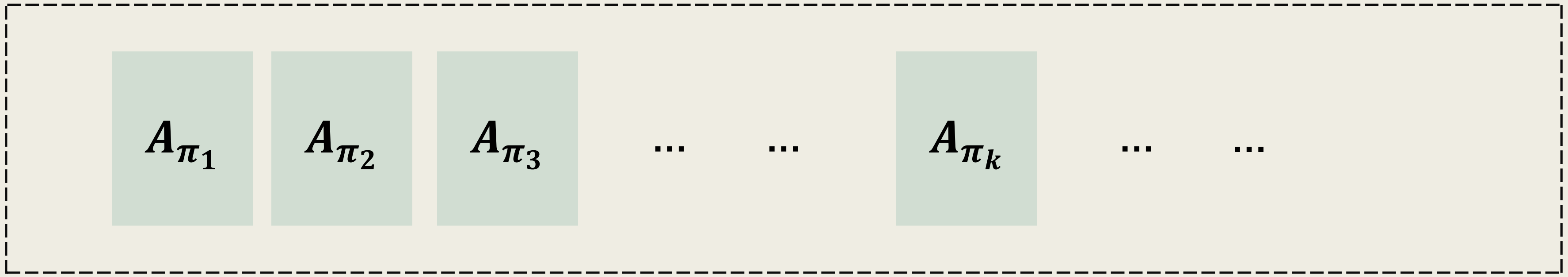
  (A ***constructive proof*** for the Lovász Local Lemma)

  - The Variable Setting Assumption
  - A Simple Randomized Algorithm
  - **The Analysis**
    - Notations & Definitions
    - The Galton-Watson branching process
    - Coupling the execution & evaluation

Notations & Definitions

The Execution Sequence

- For any $k \geq 1$,
  let π_k denote the index of the event that is resampled by the algorithm in the k^{th}-iteration.



Sequence of events resampled by the algorithm

The Closed Neighborhood D_i^+ of A_i

- For any $1 \leq i \leq m$, let

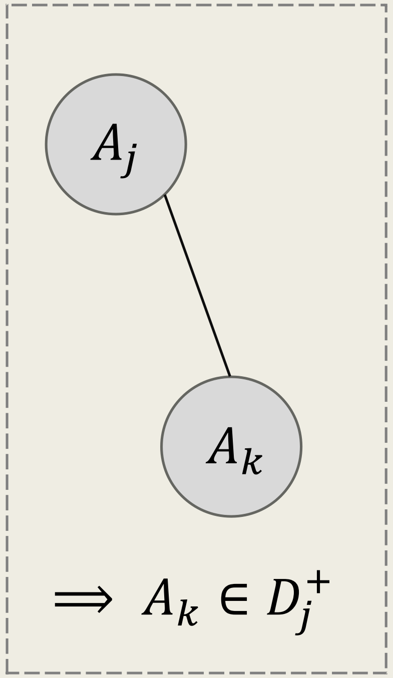
$$D_i^+ \coloneqq D_i \cup \{A_i\}$$

be the set of events that are connected to A_i in the dependency graph and the event A_i itself.

The Witness Tree

- A witness tree is a rooted tree T such that
  - Each node $v \in T$ is labeled with an event in $\{A_1, \dots, A_m\}$, denoted $A_{[v]}$.
  - If v is a child of u in T, then $A_{[v]} \in D^+_{[u]}$.
- T is called ***proper***, if for any node v,

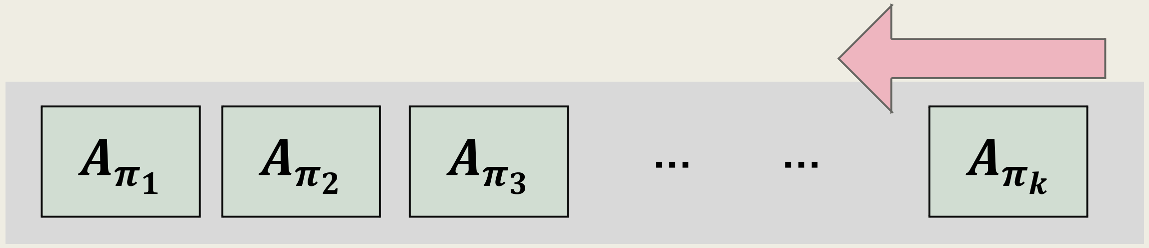
  all the events labeled on the children of v are distinct.



We use $[v]$ to denote the index of the event labeled with vertex v.

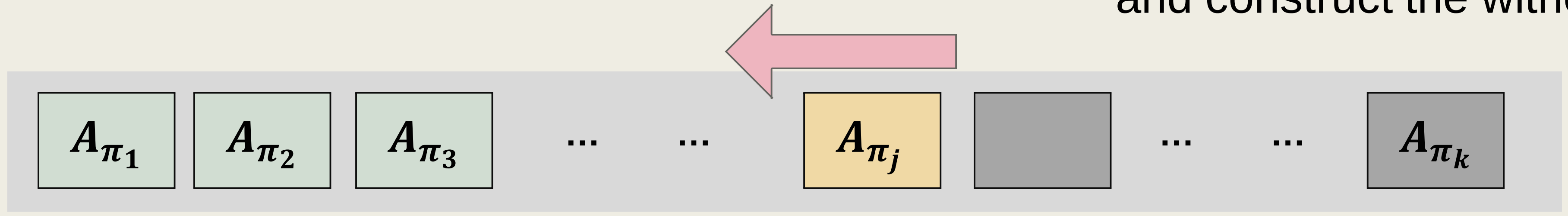
Constructing a Proper Witness Tree for any Prefix of the Execution Sequence

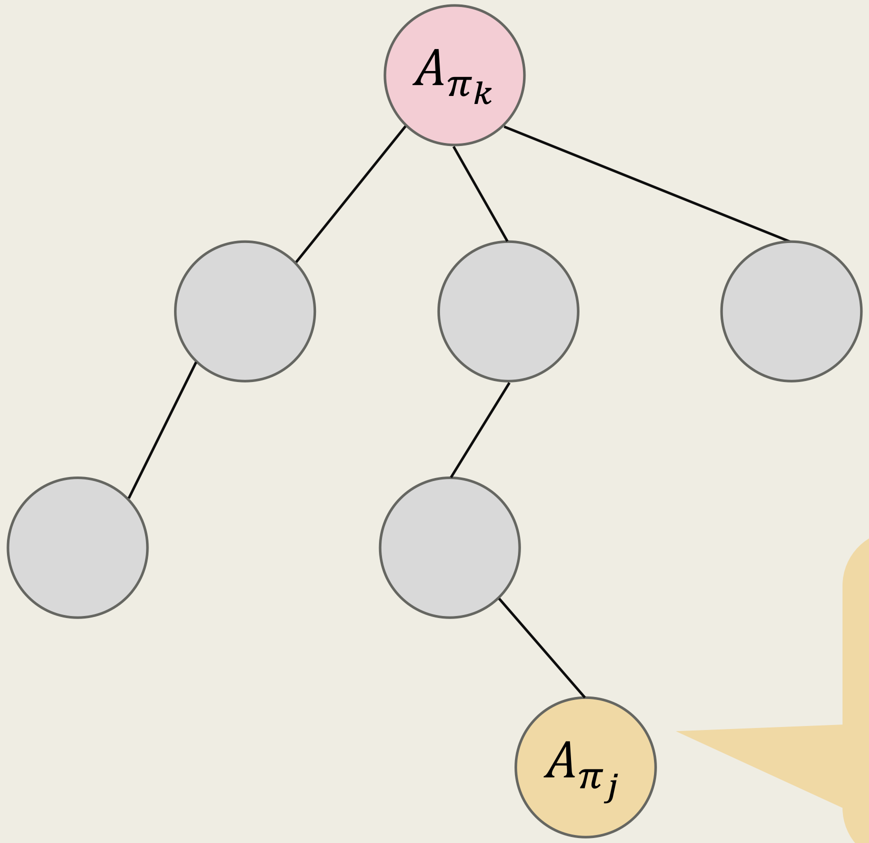
- For any $k \geq 1$, construct the tree $T(k)$ as follows.
  - Consider the execution sequence in a backward manner.
  - For each event, say, A_{π_i}, attach a node labeled with A_{π_i} as a child node to ***<u>the deepest node</u>*** in the tree that is labeled with some event in $D^+_{\pi_i}$.



Consider the events in a backward manner, and construct the witness tree.

Consider the events in a *backward manner*,
and construct the witness tree.

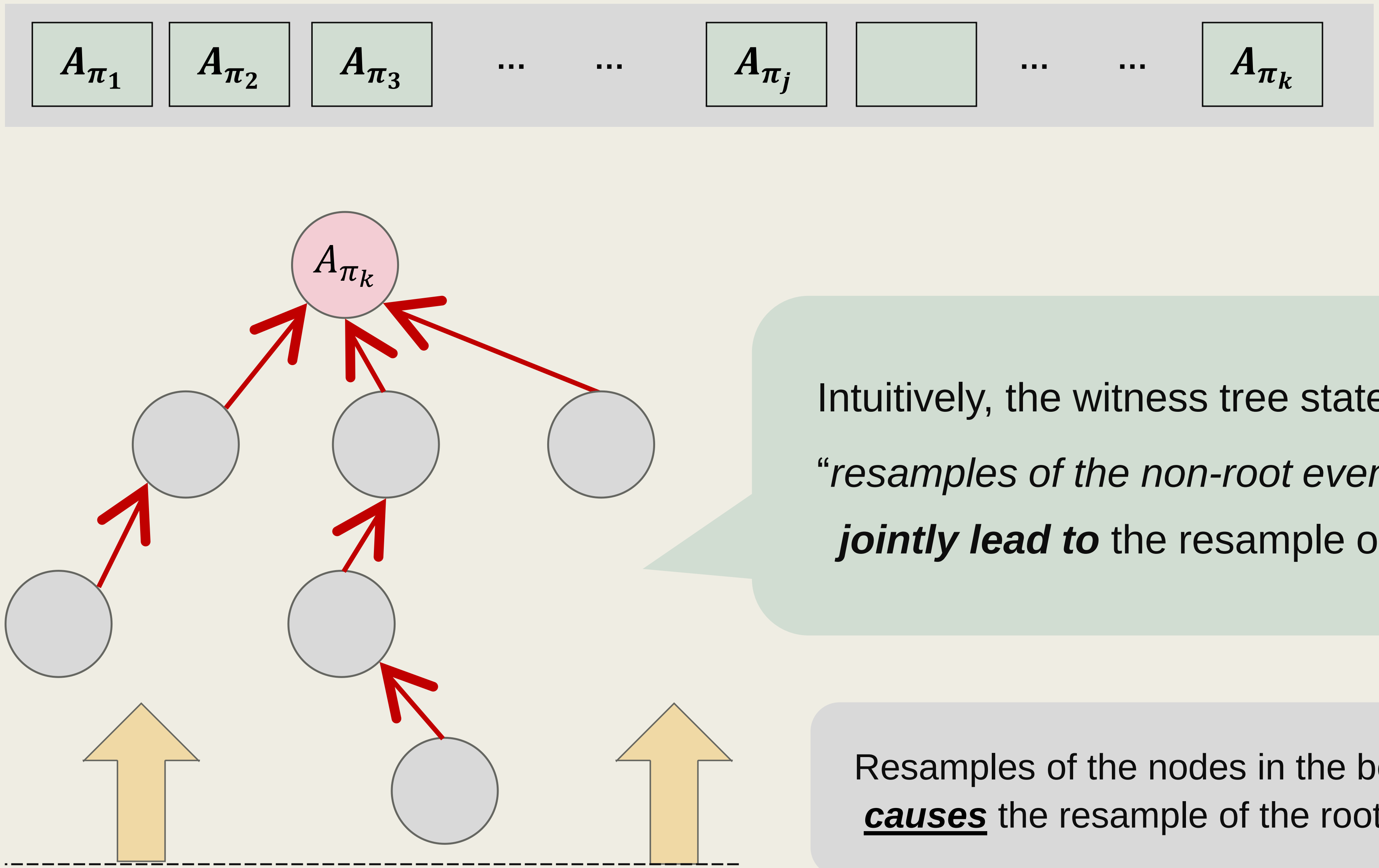




Hence,
the tree is a ***witness tree***.

Attach this node as a child to
the deepest node in the tree
that is labeled with some event in $D^+_{\pi_j}$

Consider the events in a *backward manner*,
and construct the witness tree.



Intuitively, the witness tree states that

"*resamples of the non-root events in* $T(k)$ ***jointly lead to*** the resample of A_{π_k}."

Resamples of the nodes in the bottom-up order ***causes*** the resample of the root event.

Properties of

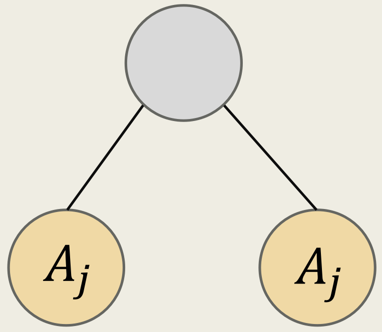
the Constructed Witness Trees

Proposition 1.

For any $k \geq 1$,

$T(k)$ is a proper witness tree.

- $T(k)$ is a witness tree by the way it is constructed.



- If it is not proper, then

  some A_j is labeled at least twice as children of some node.

By the construction rule, one of them should be attached deeper.
A contradiction.

- For any proper witness tree T,

  we say that it occurs (in the execution sequence),

  if $T = T(k)$ for some $k \geq 1$.

Lemma 2.

For any proper witness tree T of the events, we have

$$\Pr[\, T \text{ occurs } \,] \leq \prod_{v \in T} \Pr[\, A_{[v]} \,] \,.$$

We will leave the proof of this lemma to the end of the slides.

Lemma 2.

For any proper witness tree T of the events, we have

$$\Pr[\, T \text{ occurs } \,] \;\le\; \prod_{v \in T} \Pr[\, A_{[v]} \,]\;.$$

- Let T_i be the set of proper witness trees with root labeled with A_i.
- By Lemma 2, we have

$$E[N_i] \;=\; \sum_{T \in T_i} \Pr[T\, occurs] \;\le\; \sum_{T \in T_i} \prod_{v \in T} \left(x_{[v]} \cdot \prod_{j \in D_{[v]}} (1 - x_j) \right) \;.$$

We bound the sum by *relating it to a simple random process*.

Outline

- Algorithmic Lovász Local Lemma

  (A ***constructive proof*** for the Lovász Local Lemma)

  - The Variable Setting Assumption
  - A Simple Randomized Algorithm
  - **The Analysis**
    - Notations & Definitions
    - The Galton-Watson branching process
    - Coupling the execution & evaluation

The Multi-type

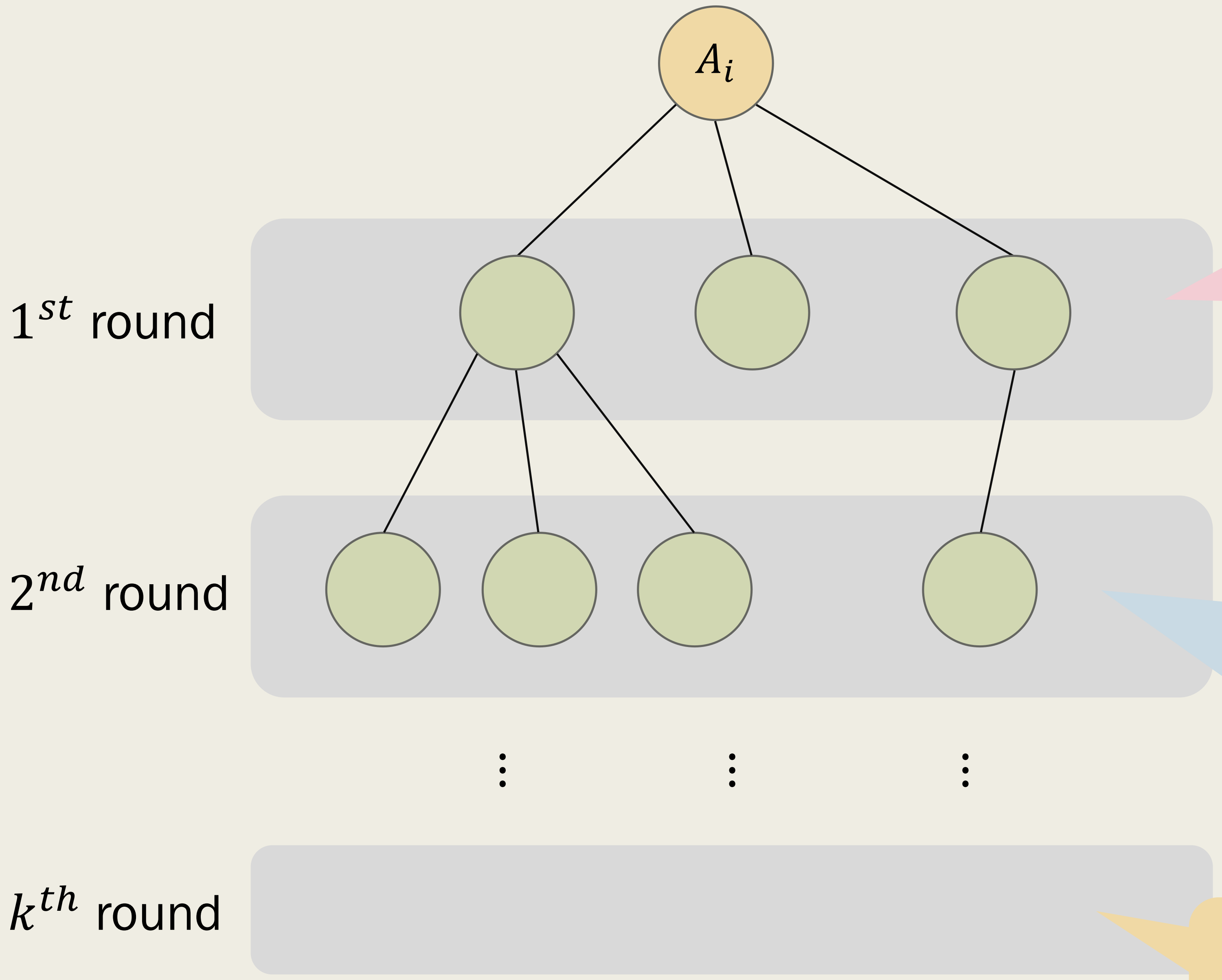
Galton-Watson Branching Process

The Galton-Watson Branching Process

- Consider the following simple random process for generating $T \in T_i$.

```
1. Generate the root node with label $A_i$.
2. While at least one node was generated in the previous iteration,
   do
      ▪ For each of these newly-generated nodes, say, $v$, do
           ▪ For each event $B \in D_{[v]}^{+}$,
             with probability $x_{[B]}$, generate a new child node for $v$ with label $B$.
3. Return the tree generated.
```

Let $[B]$ denote the index of the event B in $\{A_1, A_2, \ldots, A_m\}$.



For each $A_b \in D_i^+$,
generate a new branch
node A_b with probability x_b.

For each newly generated
branch node, say, v, and
each $A_b \in D_{[v]}^+$,
generate a new branch
node A_b with probability x_b.

Repeat until
no vertices are newly generated.

The Process *Generates* a *Proper Witness Tree*

- We only branch for events in D^+.
  - So it is a witness tree.
- Each event in D^+ is branched at most once.
  - The witness tree is proper.

The Galton-Watson Branching Process

- The speed for which the process terminates depends on the values of x_j, for all A_j that is reachable from A_i in the dependency graph.
  - The process dies out quickly when the x_j are small.
  - On the contrary,
    when x_j are large, the branching process may not stop at all.

- For any $T \in T_i$, let p_T denote the probability that the Galton-Watson process generates T.

Lemma 3.

For any $T \in T_i$, we have

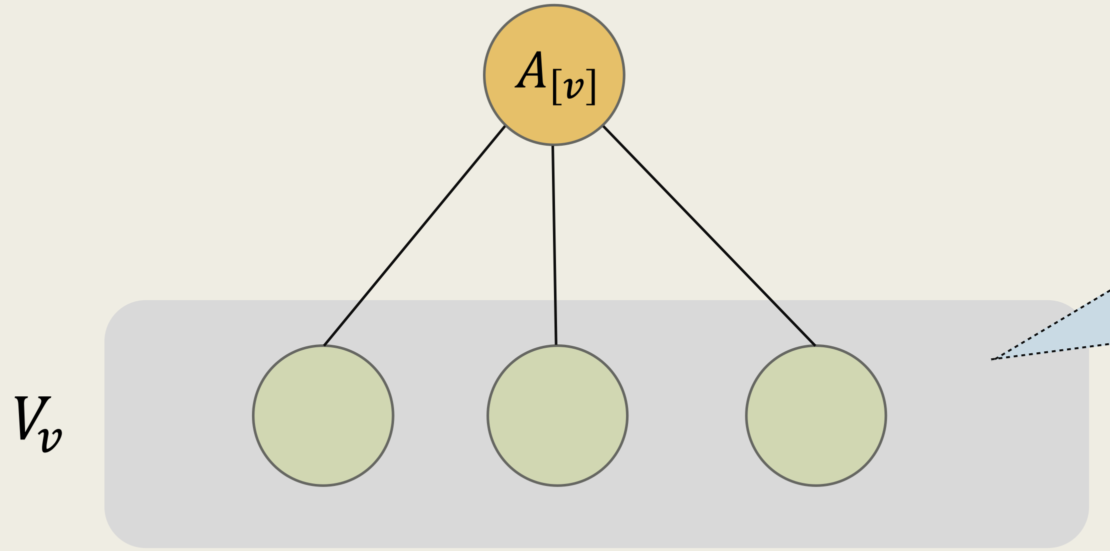
$$p_T = \frac{1 - x_i}{x_i} \cdot \prod_{v \in T} \left(x_{[v]} \cdot \prod_{j \in D_{[v]}} (1 - x_j) \right).$$

This lemma can be verified directly from the process.

Proof of Lemma 3

- Consider any vertex $v \in T$.

  Suppose that it **has children set V_v**.

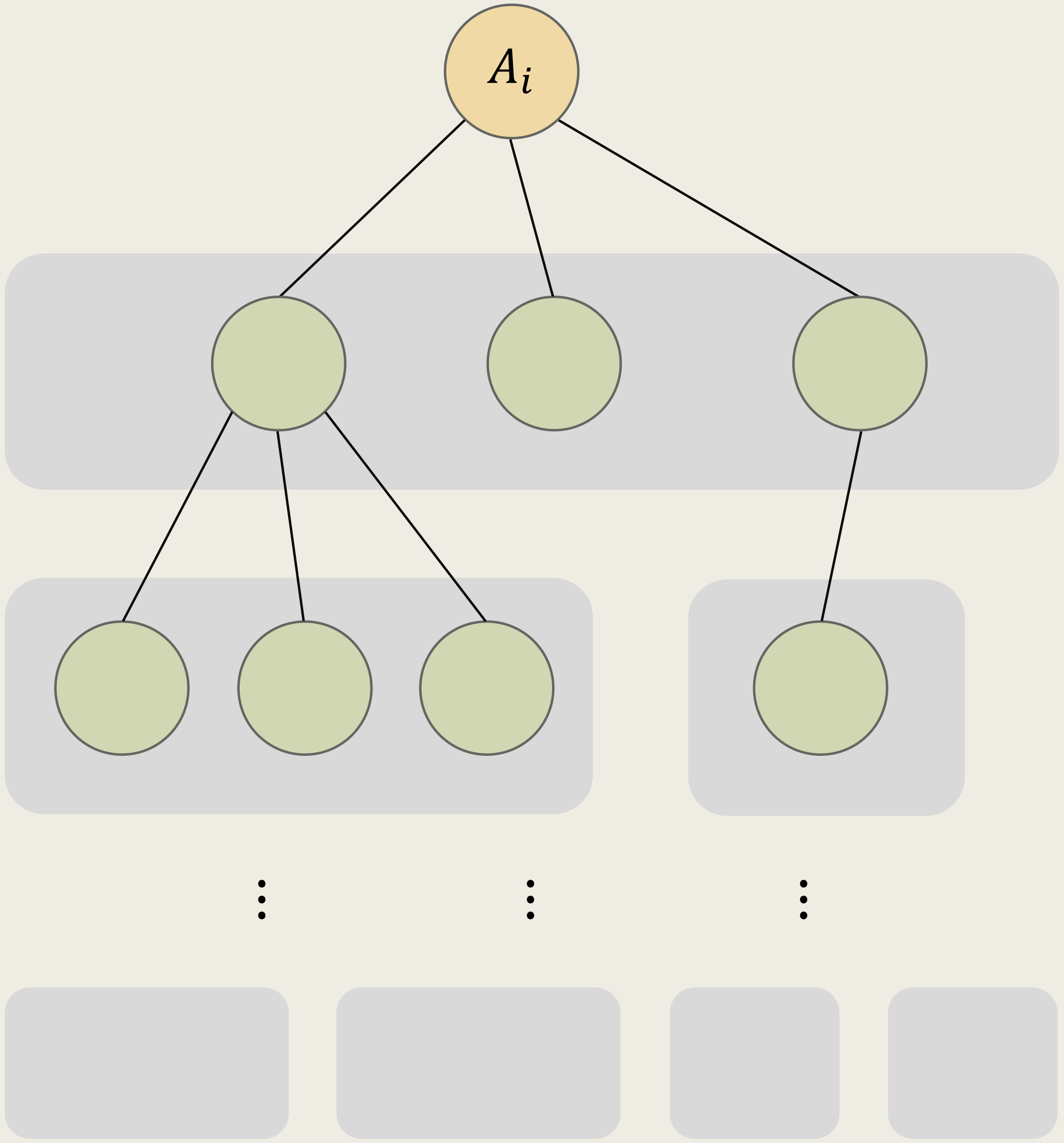


This happens with probability

$$\prod_{u \in V_v} x_{[u]} \cdot \prod_{j \in D^+_{[v]} \setminus [V_v]} (1 - x_j)$$

Which is equal to

$$\prod_{u \in V_v} \frac{x_{[u]}}{1 - x_{[u]}} \cdot \prod_{j \in D^+_{[v]}} (1 - x_j)$$



- We have

$$p_T = \prod_{v \in T} \left(\prod_{u \in V_v} \frac{x_{[u]}}{1 - x_{[u]}} \cdot \prod_{j \in D_{[v]}^+} (1 - x_j) \right)$$

$$= \frac{1 - x_i}{x_i} \cdot \prod_{v \in T} \left(\frac{x_{[v]}}{1 - x_{[v]}} \cdot \prod_{j \in D_{[v]}^+} (1 - x_j) \right)$$

$$= \frac{1 - x_i}{x_i} \cdot \prod_{v \in T} \left(x_{[v]} \cdot \prod_{j \in D_{[v]}} (1 - x_j) \right) .$$

- This proves the lemma.

Outline

- Algorithmic Lovász Local Lemma

  (A ***constructive proof*** for the Lovász Local Lemma)

  - The Variable Setting Assumption
  - A Simple Randomized Algorithm
  - **The Analysis**
    - Notations & Definitions
    - The Galton-Watson branching process
    - Coupling the execution & evaluation

Strictly Proper Witness Trees

- Let T be a witness tree.
  - For any $v \in T$, let $\mathrm{depth}(v)$ be its distance to the root.
  - We say that T is *strictly proper*,

    if for any $u, v \in T$ with $depth(u) = depth(v)$,

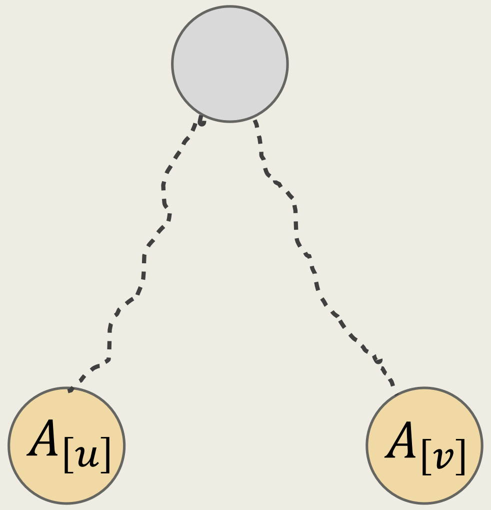
    we always have

$$vbl\left(A_{[u]}\right) \cap vbl\left(A_{[v]}\right) = \emptyset\ .$$

Proposition 4.

If T occurs in the execution sequence, then T is strictly proper.

- The proof is straightforward,
  by the way how witness trees are constructed
  from the execution sequence.
  - If there exist $u, v \in T$ with the same depth
    and $vbl(A_{[u]}) \cap vbl(A_{[v]}) \neq \emptyset$,
    then one of them should be attached at a deeper level.



Lemma 2.

For any proper witness tree T of the events, we have

$$\Pr[\ T \text{ occurs in execution }\ ] \ \le \ \prod_{v \in T} \Pr\left[\ A_{[v]}\ \right] .$$

- By Proposition 4, for witness trees that are not strictly proper,

$$\Pr[\ T \text{ not strictly proper occurs}\ ] \ = \ 0 \ \le \ \prod_{v \in T} \Pr\left[\ A_{[v]}\ \right] .$$

Hence, it suffices to prove the statement for *strictly proper witness trees*.

Proof of Lemma 2

It remains to prove the statement of Lemma 2.

This is the part for which the *algorithmic* ***variable-setting*** is truly involved.

To Prove :

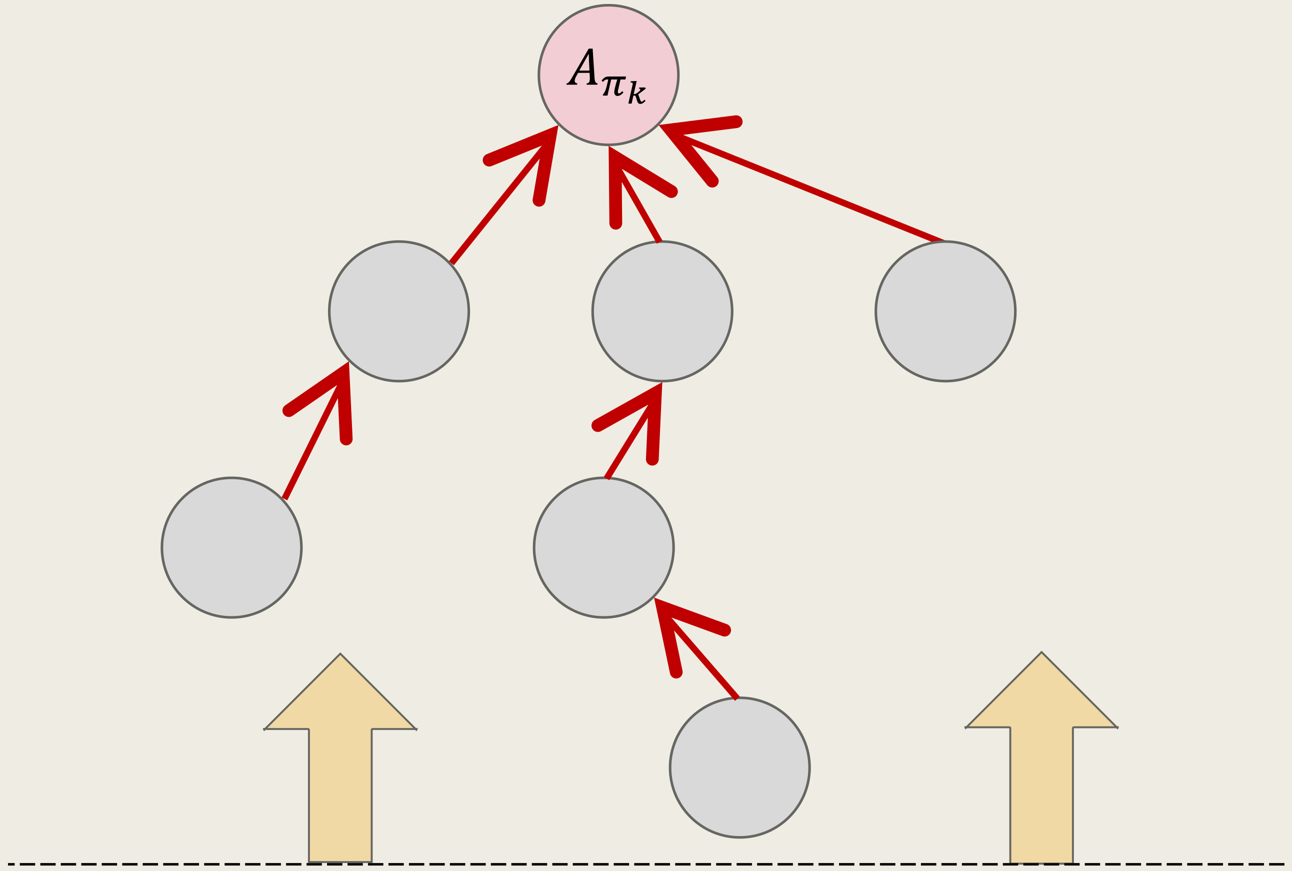
For any strictly proper witness tree T of the events, we have

$$\Pr[\ T \text{ occurs in execution }\ ] \ \leq \ \prod_{v \in T} \Pr\left[\, A_{[v]} \,\right] .$$

- Consider the following ***evaluation process*** for T.
  - For each $v \in T$ in a *reversed-BFS order*,

    sample the values of the variables in $vbl\left(A_{[v]}\right)$.

- For each $v \in T$ in a reversed-BFS order, sample the values of the variables in $vbl(A_{[v]})$.



<u>To Prove :</u>

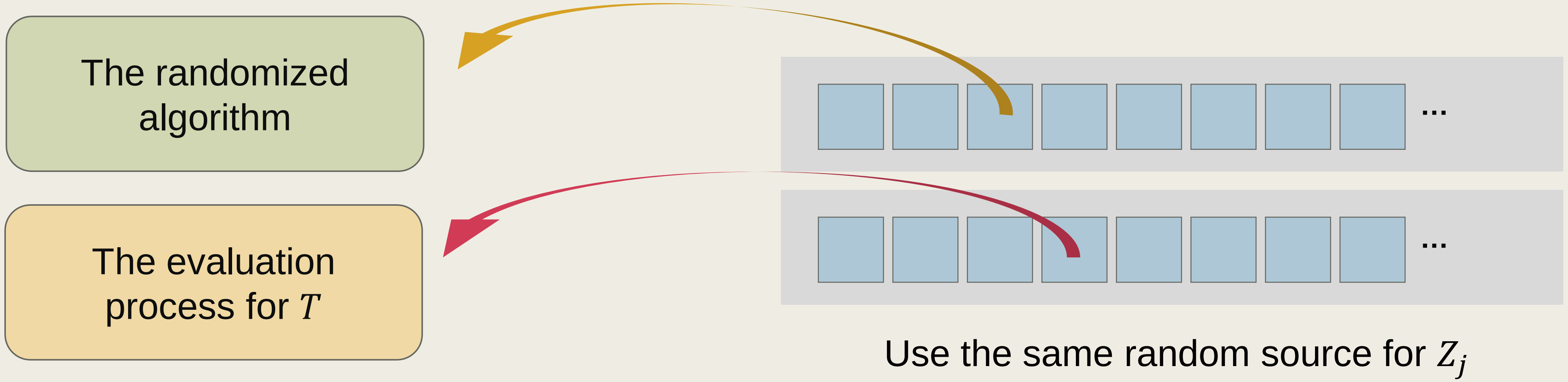
For any strictly proper witness tree T of the events, we have

$$\Pr[\ T \text{ occurs in execution }\ ] \le \prod_{v \in T} \Pr\left[A_{[v]} \right].$$

- Consider the following ***evaluation process*** for T.
  - For each $v \in T$ in a *reversed-BFS order*, sample the values of the variables in $vbl\left(A_{[v]}\right)$.
  - Furthermore, suppose that, in the evaluation process, we use ***the same random source*** with ***the algorithm execution***.

The Execution Coupling

- Imagine that, for each $1 \le j \le n$, in the evaluation process, we use ***an identical random source*** that is used in the algorithm execution for variable Z_j.
  - Therefore, the evaluation process gets ***the same random sequence*** with the algorithm execution when it samples Z_j.


The randomized algorithm
The evaluation process for T
…
…
Use the same random source for Z_j

- Consider the following evaluation process.
  - For each $v \in T$ in a reversed-BFS order, sample the values of the variables in $vbl(A\_[v])$.

- We say that the sample in v is ***successful***, if it makes $A_{[v]}$ true.

  Clearly,
  $$\Pr[\text{ sample in } v \text{ successful }] = \Pr\left[A_{[v]}\right].$$

- We say that the ***evaluation process succeeds***, if the samples in all vertices are successful.

  It follows that
  $$\Pr[\text{ evaluation succeeds }] = \prod_{v \in T} \Pr\left[A_{[v]}\right].$$

It suffices to prove that, for *strictly proper witness tree* T,

$$\Pr[\ T \text{ occurs in execution }\ ] \leq \Pr[\ \text{evaluation succeeds }\ ].$$

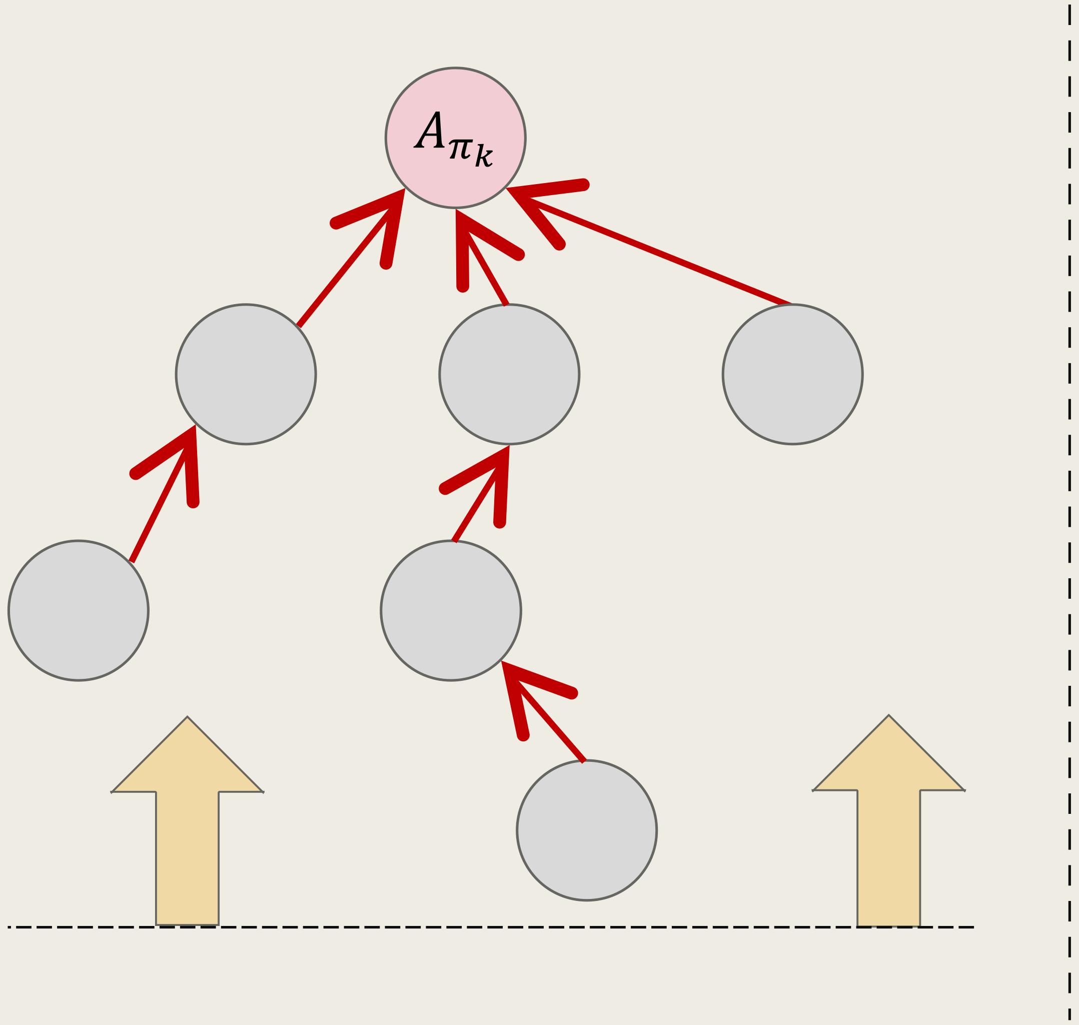
- We show that, if we couple up (by using the same random sources)
  - *the execution of the algorithm and*
  - *the evaluation process of the witness tree,*

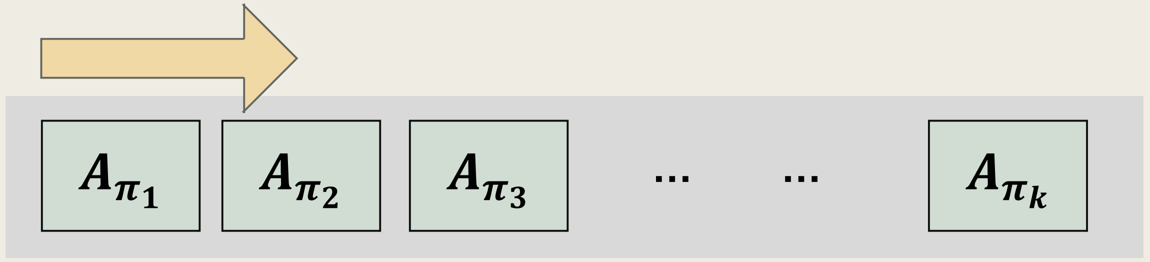
  then, *whenever T occurs in the execution sequence*,
  the *evaluation process for T must succeed*.

- Note that, this implies the conclusion we want.

$A \Rightarrow B$, then $\Pr[A] \leq \Pr[B]$.

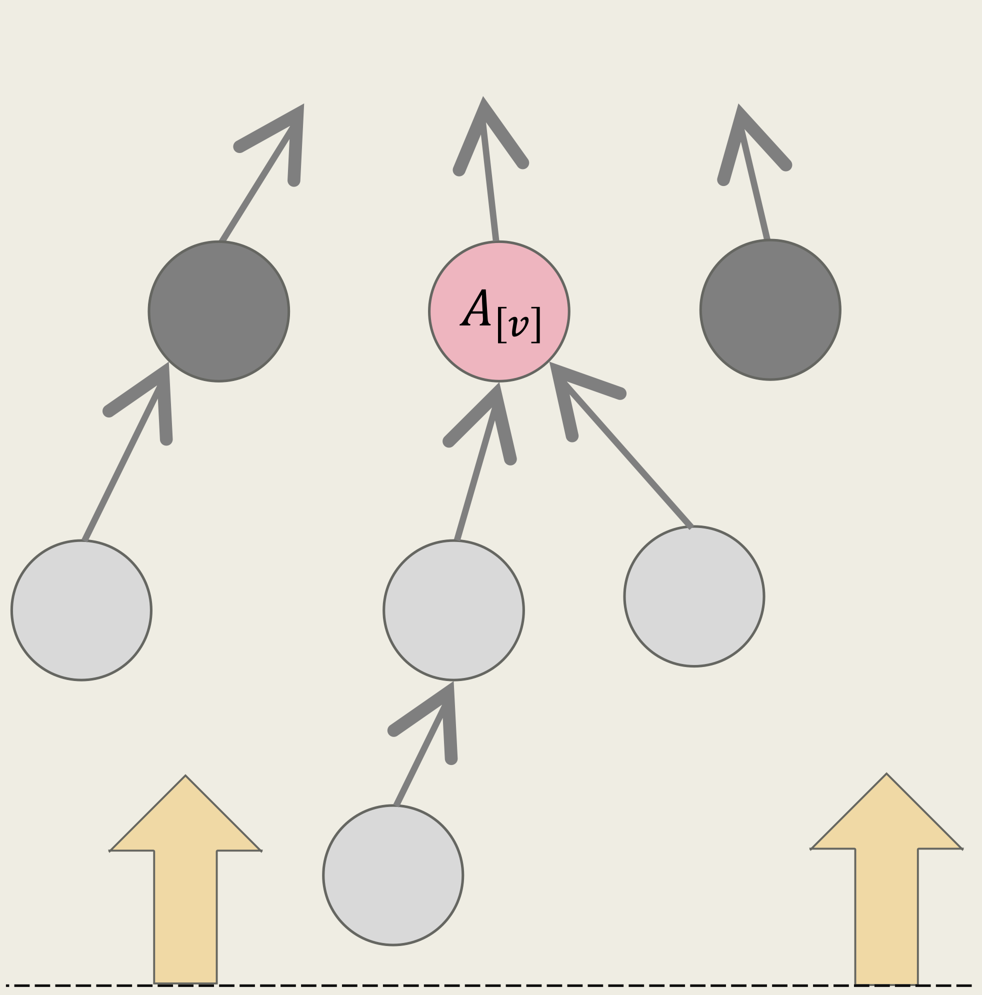
- We couple up ***the execution sequence of the algorithm*** and ***the evaluation process of the witness tree*** $T \in T_k$.

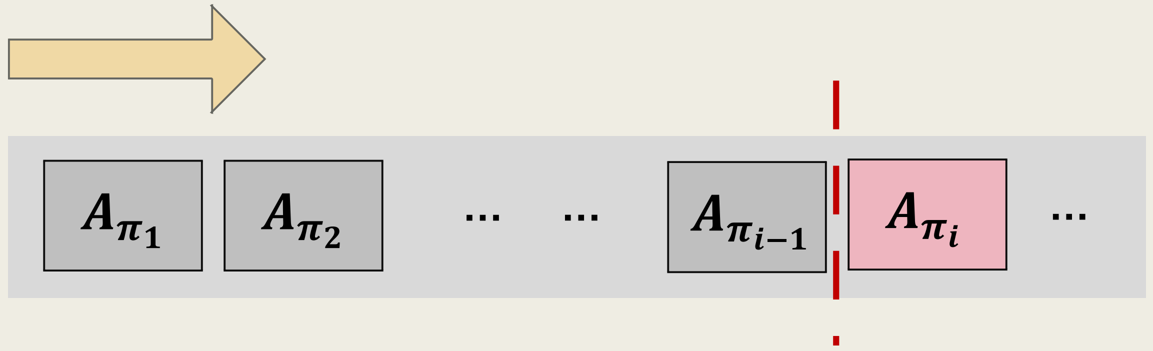




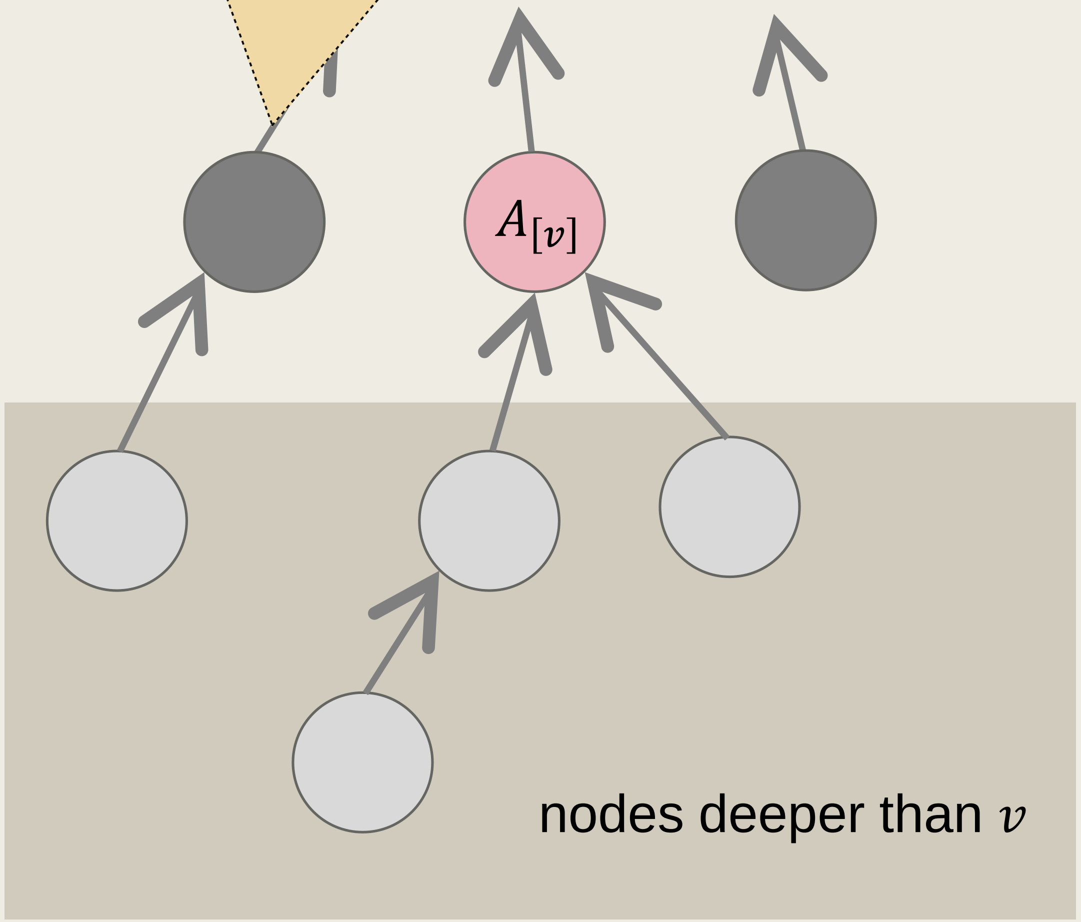
- Consider a node $v \in T \in T_k$ and any $Z_j \in vbl(A_{[v]})$.

  Suppose that it is the i^{th}-item in the execution sequence, i.e., $[v] = \pi_i$.





None of the nodes at the same level, other than v, contains Z_j.



The number of times Z_j is sampled at

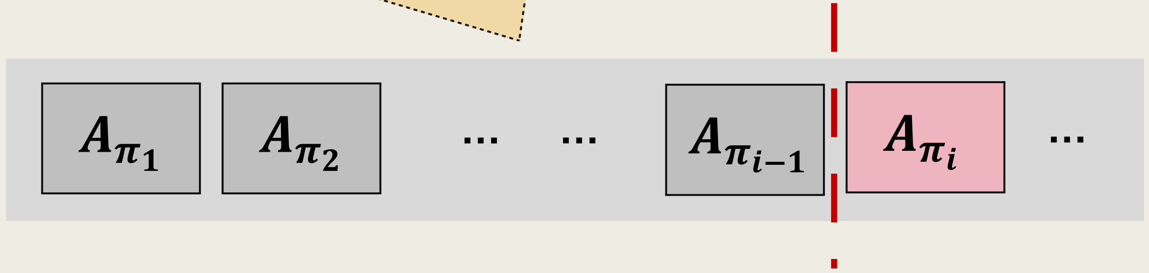
$$\{\, u \in T \;:\; depth(u) > depth(v) \,\}$$

and

$$\{\, A_{\pi_1}, A_{\pi_2}, \dots, A_{\pi_{i-1}} \,\}$$

are ***the same***, since T is strictly proper.

All of these events that contain Z_j appear at depth deeper than $depth(v)$.



- Consider a node $v \in T \in T_k$ and any $Z_j \in vbl(A_{[v]})$.
  Suppose that it is the i^{th}-item in the execution sequence, i.e., $[v] = \pi_i$.

- The number of times Z_j is sampled at

  $$\{ u \in T \; : \; depth(u) > depth(v) \} \quad \text{and} \quad \{ A_{\pi_1}, A_{\pi_2}, \ldots, A_{\pi_{i-1}} \}$$

  are ***the same***, since T is strictly proper.

- Since the algorithm *makes one more sampling on* Z_j ***initially***,

  the result the evaluation process gets at node v is

  the current value of $\boldsymbol{Z_j}$ at the i^{th}-iteration of the algorithm.

- This argument holds for all variables in $vbl(A_{[v]})$.

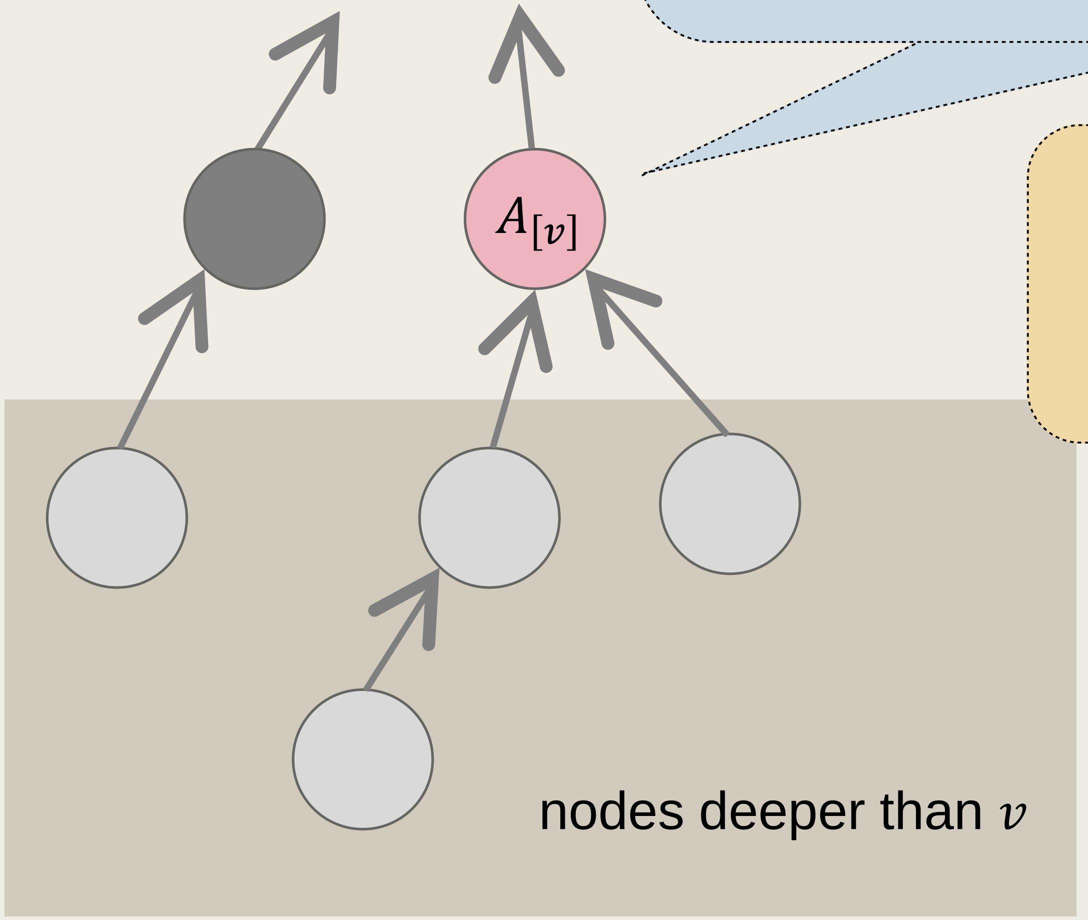
When the process samples $vbl(A_{[v]})$ at v,

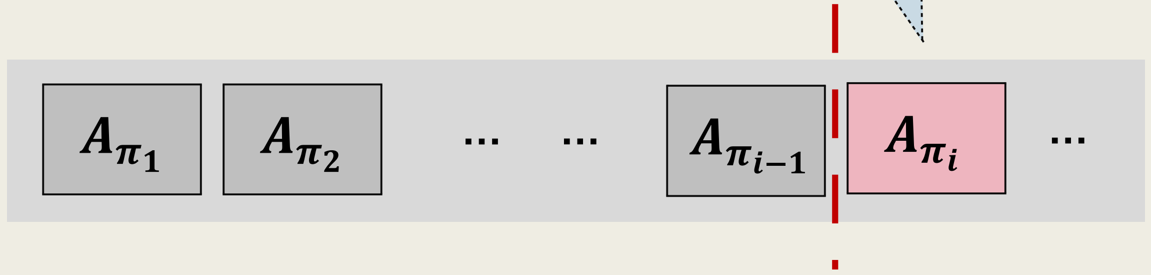
what it gets is the assignment the algorithm has for $vbl(A_{[v]})$

at the beginning of the i^{th}-iteration !

Since A_{π_i} is true (the algorithm resamples it),

the evaluation at v must be successful.


$A_{[v]}$
nodes deeper than v


A_{π_1}
A_{π_2}
… …
$A_{\pi_{i-1}}$
A_{π_i}
…