Enumeration and Random Generation of Concurrent Computations

O. Bodini[1] *A. Genitrini*[2] F. Peschanski[2]

[1]Université Paris 13 – LIPN

[2]UPMC Paris – LIP6

Aléa – March, 2012

Outline

1. Motivations
   - Concurrent computations
   - Related works
2. Shuffle trees and their typical shape
   - Recursive construction
   - Quantitative analysis
3. Algorithms
   - Probability of a concurrent run prefix
   - Uniform random generation of a run

Outline

When analyzing concurrent processes, the shuffle operator is the main source of ***combinatorial explosion****.* [Mi80], [ClGrPe99]

Concurrency theory and combinatorics

In concurrency theory, one manipulates:

- syntactic objects $\Rightarrow$ Process trees
- their semantic interpretation $\Rightarrow$ Shuffle trees

Concurrency theory and combinatorics

In concurrency theory, one manipulates:

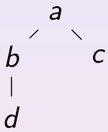
- syntactic objects $\Rightarrow$ Process trees
- their semantic interpretation $\Rightarrow$ Shuffle trees

Ideas

- to consider these objects as combinatorial structures
- to use analytic combinatorics for quantitative studies

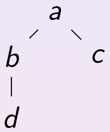
Process trees and shuffle trees

A **process tree** is a specification of events with precedence constraints:

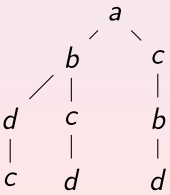


Process trees and shuffle trees

A **process tree** is a specification of events with precedence constraints:

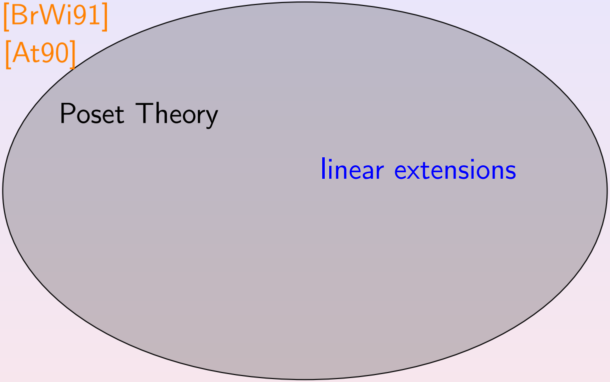


The induced **shuffle tree** lists all admissible concurrent runs by sharing prefixes, as in a trie:

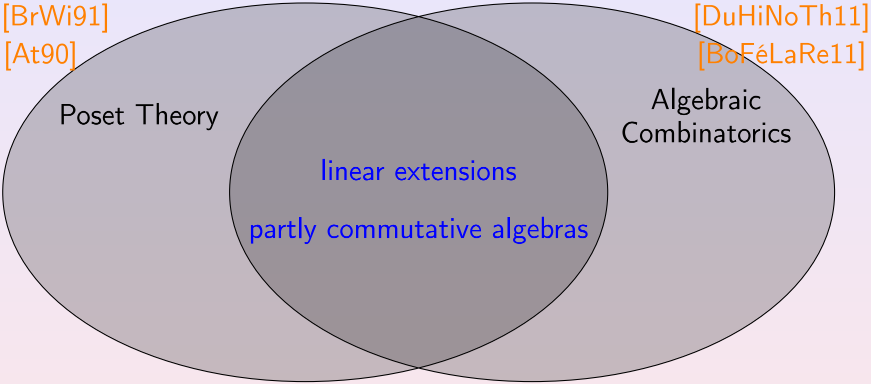


Related works

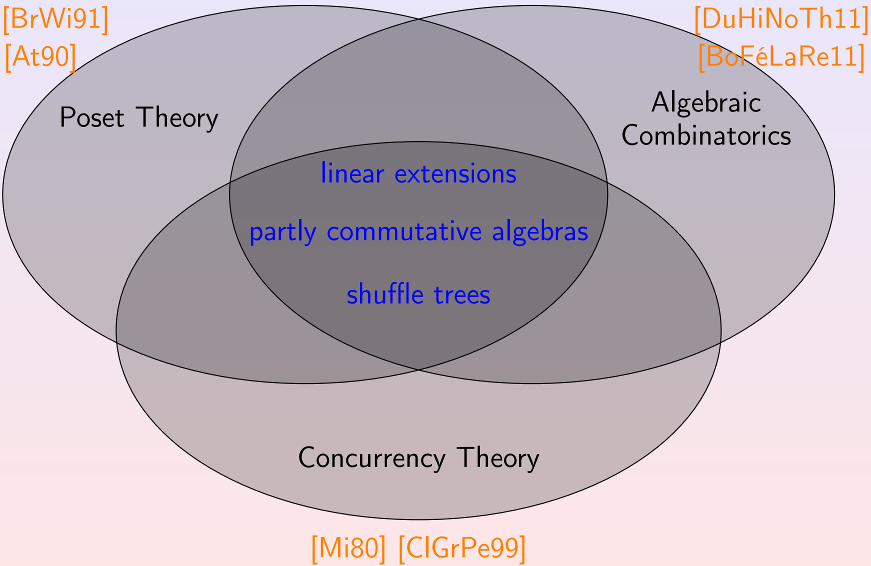
[BrWi91]
[At90]



Related works

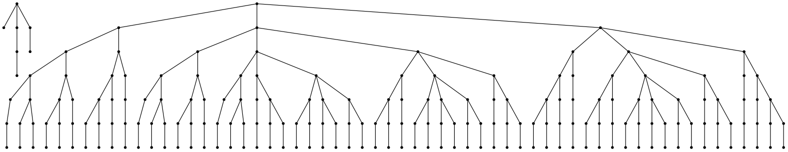


Related works



Outline

1. Motivations
2. Shuffle trees and their typical shape
3. Algorithms



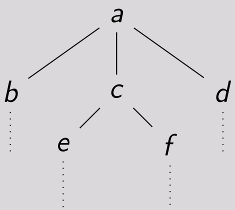
Building shuffle trees (1)

Definition: Child contraction

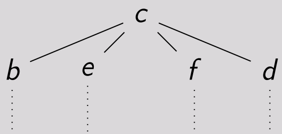
Let T be a tree with children $T_1, \ldots, T_r$ whose root-events are $\ell_1, \ldots, \ell_r$ ($r \in \mathbb{N}^*$). The i-**contraction** of T is the tree $T \lhd i$ with root ℓ_i and children $T_1, \ldots, T_{i-1}, T_{i_1}, \ldots, T_{i_m}, T_{i+1}, \ldots, T_r$ where $T_{i_1}, \ldots, T_{i_m}$ are the children of T_i.

Example

$T =$



$\Rightarrow T \lhd 2 =$



Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $\mathit{Shuf}(T)$ is defined inductively as:

- if T is a leaf, then $\mathit{Shuf}(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$ then $\mathit{Shuf}(T)$ is the tree with root-event ℓ and children $\mathit{Shuf}(T \triangleleft 1), \ldots, \mathit{Shuf}(T \triangleleft r)$

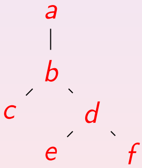
Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $Shuf(T)$ is defined inductively as:

- if T is a leaf, then $Shuf(T) := T$
- if T has root-event ℓ and children $T_1, \dots, T_r$ $(r \in \mathbb{N}^*)$ then $Shuf(T)$ is the tree with root-event ℓ and children $Shuf(T \lhd 1), \dots, Shuf(T \lhd r)$

Example (Shuffle / Contraction):



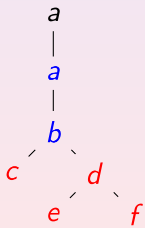
Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $\mathit{Shuf}(T)$ is defined inductively as:

- if T is a leaf, then $\mathit{Shuf}(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$
  then $\mathit{Shuf}(T)$ is the tree with root-event ℓ
  and children $\mathit{Shuf}(T \lhd 1), \ldots, \mathit{Shuf}(T \lhd r)$

Example (Shuffle / Contraction):



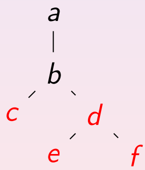
Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $Shuf(T)$ is defined inductively as:

- if T is a leaf, then $Shuf(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$ then $Shuf(T)$ is the tree with root-event ℓ and children $Shuf(T \lhd 1), \ldots, Shuf(T \lhd r)$

Example (Shuffle / Contraction):



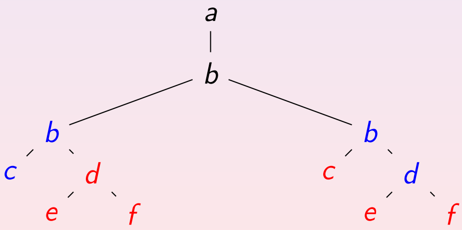
Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $Shuf(T)$ is defined inductively as:

- if T is a leaf, then $Shuf(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$ then $Shuf(T)$ is the tree with root-event ℓ and children $Shuf(T \triangleleft 1), \ldots, Shuf(T \triangleleft r)$

Example (Shuffle / Contraction):



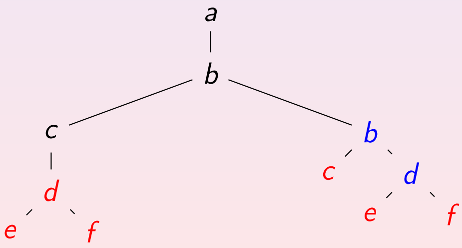
Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $Shuf(T)$ is defined inductively as:

- if T is a leaf, then $Shuf(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$
  then $Shuf(T)$ is the tree with root-event ℓ
  and children $Shuf(T \lhd 1), \ldots, Shuf(T \lhd r)$

Example (Shuffle / Contraction):



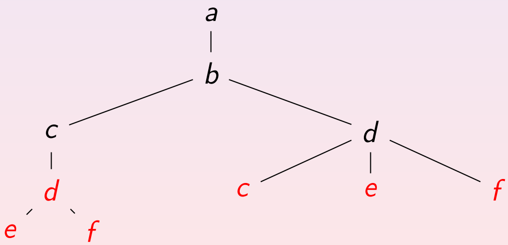
Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $Shuf(T)$ is defined inductively as:

- if T is a leaf, then $Shuf(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$ then $Shuf(T)$ is the tree with root-event ℓ and children $Shuf(T \lhd 1), \ldots, Shuf(T \lhd r)$

Example (Shuffle / Contraction):



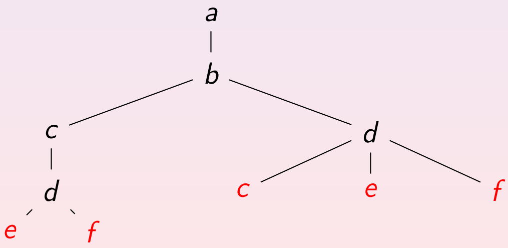
Building shuffle trees (2)

Recursive definition

Let T be a tree. Its **shuffle tree** $Shuf(T)$ is defined inductively as:

- if T is a leaf, then $Shuf(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$ then $Shuf(T)$ is the tree with root-event ℓ and children $Shuf(T \lhd 1), \ldots, Shuf(T \lhd r)$

Example (Shuffle / Contraction):



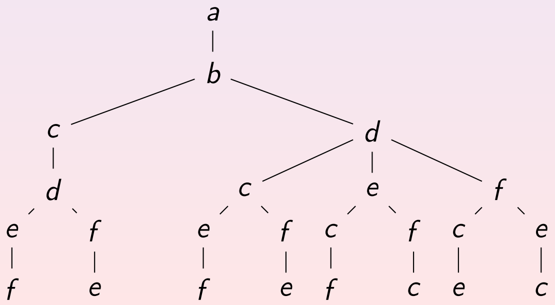
Building shuffle trees (2)

Recursive definition

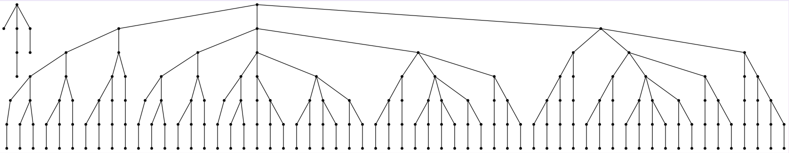
Let T be a tree. Its **shuffle tree** $\mathit{Shuf}(T)$ is defined inductively as:

- if T is a leaf, then $\mathit{Shuf}(T) := T$
- if T has root-event ℓ and children $T_1, \ldots, T_r$ $(r \in \mathbb{N}^*)$ then $\mathit{Shuf}(T)$ is the tree with root-event ℓ and children $\mathit{Shuf}(T \lhd 1), \ldots, \mathit{Shuf}(T \lhd r)$

Example (Shuffle / Contraction):



Branches of shuffle trees



Observation

Information is extremely redundant in shuffle trees:
One can recover the process tree by traversing a single branch of the shuffle tree.

Goals

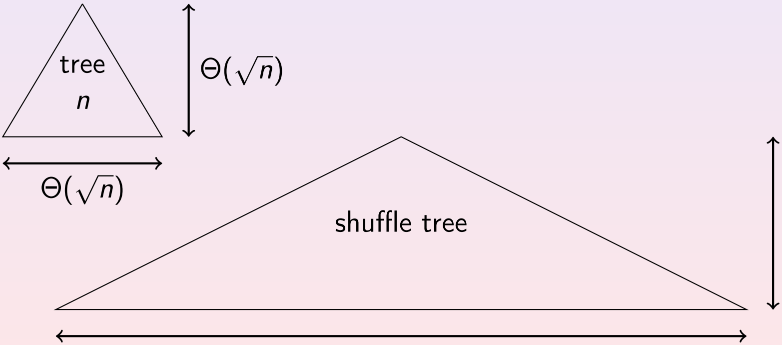
In order to analyze the combinatorial explosion of shuffle trees, we want to answer the following questions:

- What is the number of runs for a given process tree T ?
  $\Rightarrow$ the number of leaves in $Shuf(T)$
- What is the size of the shuffle tree induced by T ?
  $\Rightarrow$ no correlation known with the number of runs (sharing)

Main results

Theorem

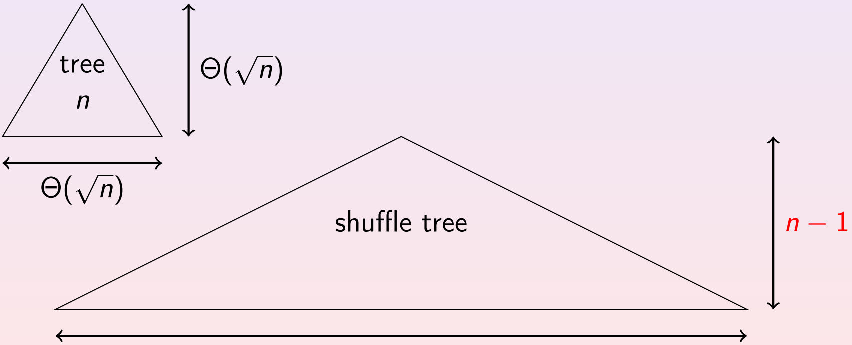
The typical shape of a shuffle tree built on a process tree of size n:



Main results

Theorem

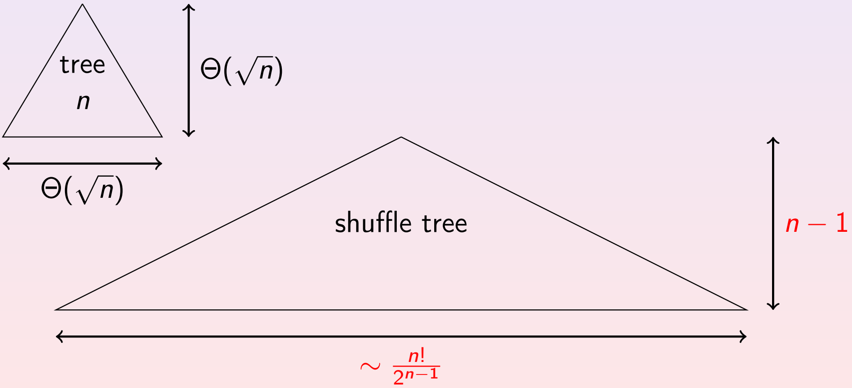
The typical shape of a shuffle tree built on a process tree of size n:



Main results

Theorem

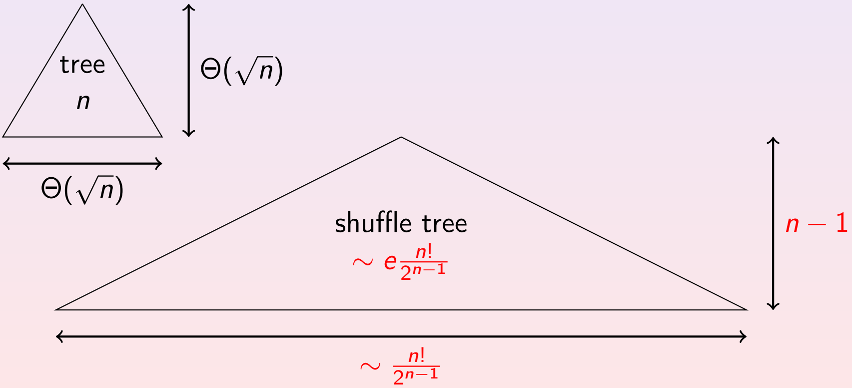
The typical shape of a shuffle tree built on a process tree of size n:



Main results

Theorem

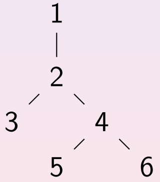
The typical shape of a shuffle tree built on a process tree of size n:

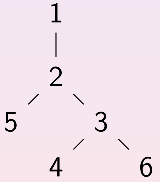


Concurrent runs and increasing trees (1)

Definition: Increasing tree

An *increasing tree* is a labelled plane tree such that the sequence of labels along any branch starting at the root is increasing.

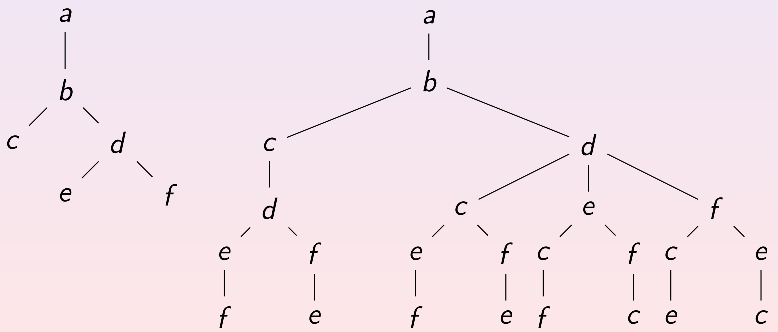




Concurrent runs and increasing trees (2)

Lemma: Bijection

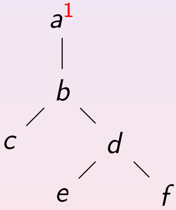
Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.

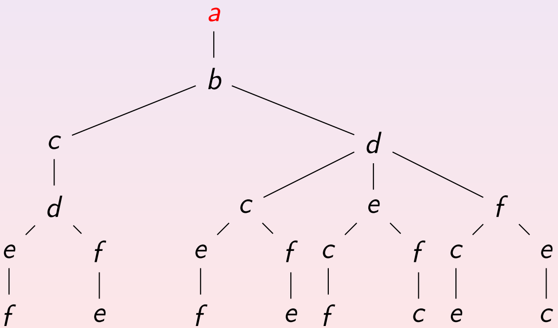


Concurrent runs and increasing trees (2)

Lemma: Bijection

Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.

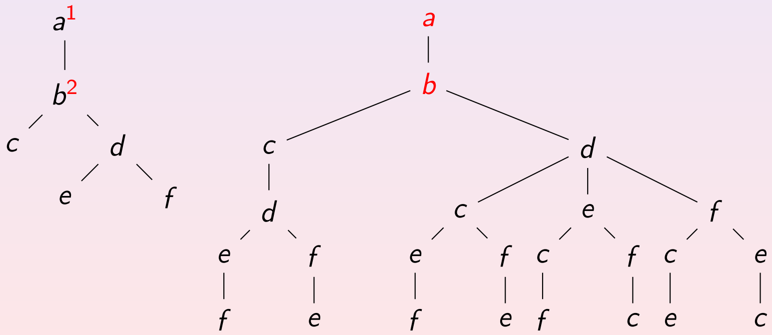




Concurrent runs and increasing trees (2)

Lemma: Bijection

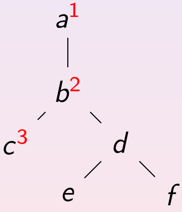
Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.

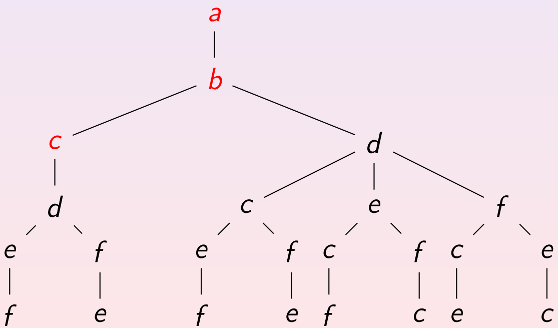


Concurrent runs and increasing trees (2)

Lemma: Bijection

Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.

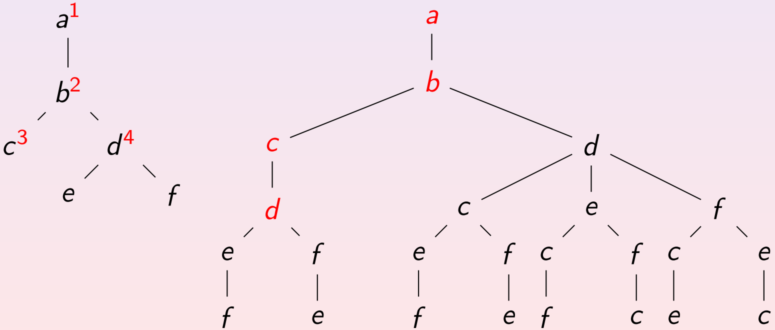




Concurrent runs and increasing trees (2)

Lemma: Bijection

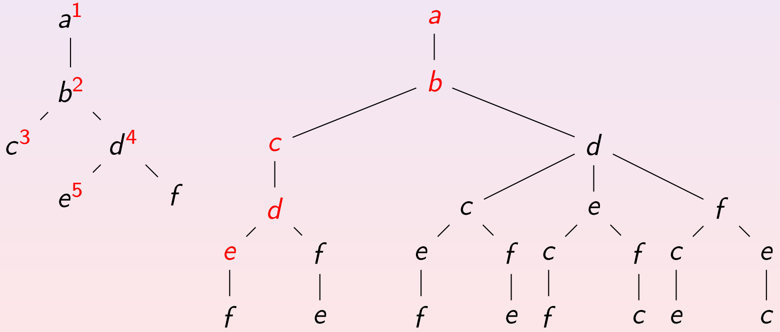
Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.



Concurrent runs and increasing trees (2)

Lemma: Bijection

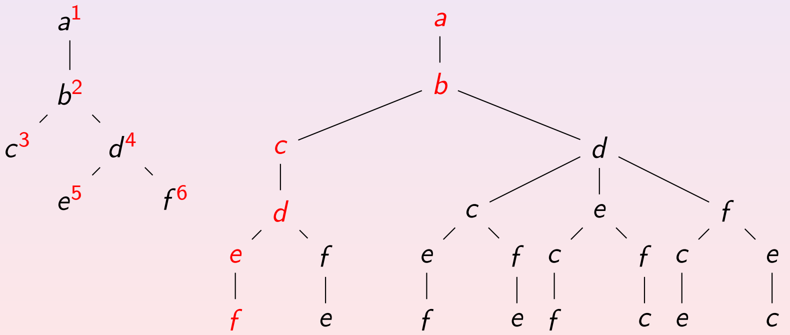
Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.



Concurrent runs and increasing trees (2)

Lemma: Bijection

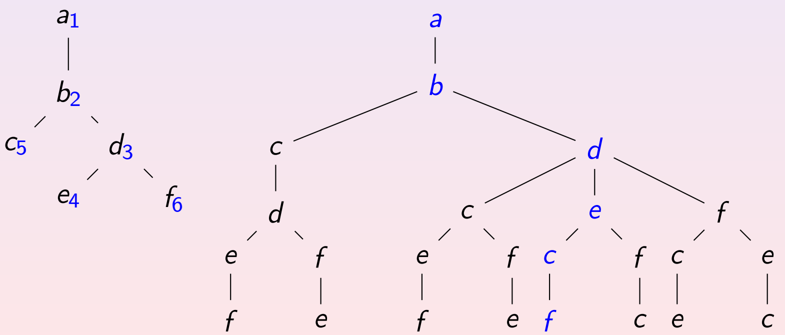
Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.



Concurrent runs and increasing trees (2)

Lemma: Bijection

Let T be a process tree. The number of runs associated to T corresponds to the number of increasing trees whose structure is the unlabelled tree T.



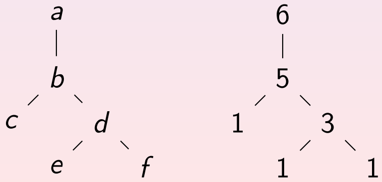
Number of concurrent runs

Theorem: Hook length in trees [Kn73]

Let T be a unlabelled tree.
The number of increasing trees built on T equals:

$$\ell_T = \frac{|T|!}{\displaystyle\prod_{R \text{ subtree of } T} |R|}.$$

This corresponds equivalently to the number of runs induced by T.



$$\ell_T = \frac{6!}{6 \cdot 5 \cdot 1 \cdot 3 \cdot 1 \cdot 1} = 8.$$

Mean number of runs and mean growth

Proposition

The *arithmetic* mean number of runs built on trees of size n is:

$$\bar{\ell}_n \sim_{n\to\infty} \frac{n!}{2^{n-1}} \sim 2\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n .$$

Mean number of runs and mean growth

Proposition

The *arithmetic* mean number of runs built on trees of size n is:

$$\bar{\ell}_n \sim_{n\to\infty} \frac{n!}{2^{n-1}} \sim 2\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n .$$

Proposition

The *geometric* mean growth between trees of size n and their number of runs is:

$$\bar{\Gamma}_n \sim_{n\to\infty} \sqrt{2\pi}n^{n-1}\exp\left(-(1+2L(1/4))n+\sqrt{\pi n}+L(1/4)\right),$$

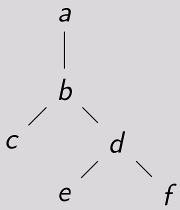
with $L\,(1/4) = \sum_{n>1} \log n \cdot Cat_n \cdot 4^{-n} \approx \mathbf{0.579043921} \pm 5\cdot 10^{-9}$.

Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

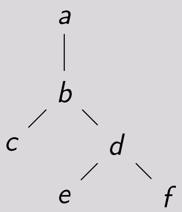


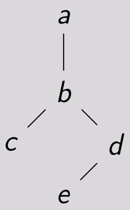
Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example



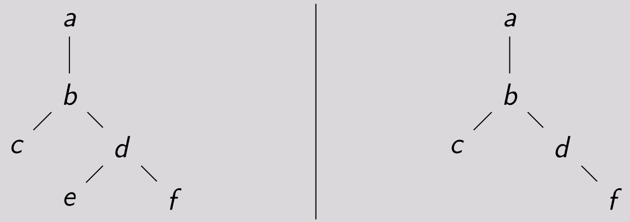


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

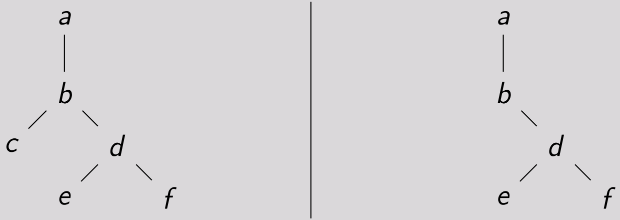


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

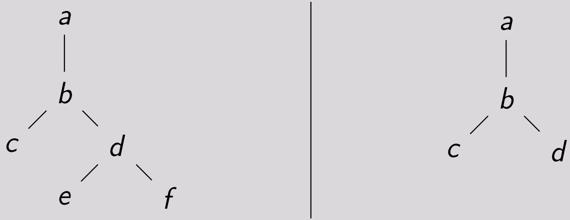


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

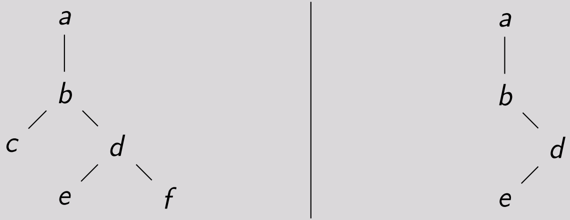


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

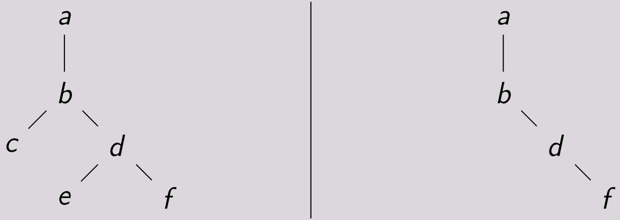


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

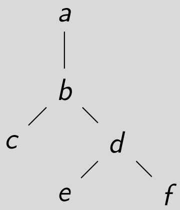


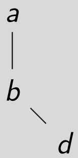
Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example



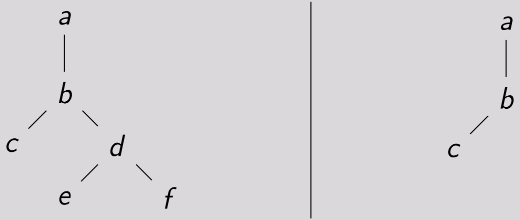


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

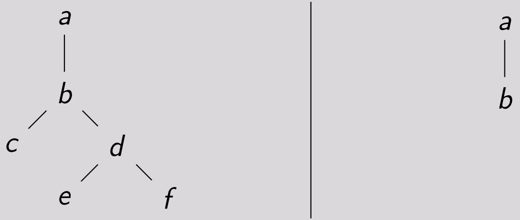


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example

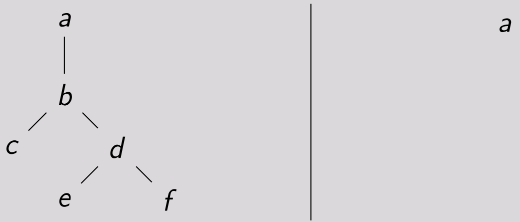


Size of shuffle trees: substructures

Definition

Let T be a process tree. We define a *substructure* of T a tree obtained by removing some subtrees of T.

Example



Size of a shuffle tree

Proposition

The size of the shuffle tree built on T satisfies:

$$n_T = \sum_{R \text{ substructure of } T} \ell_R.$$

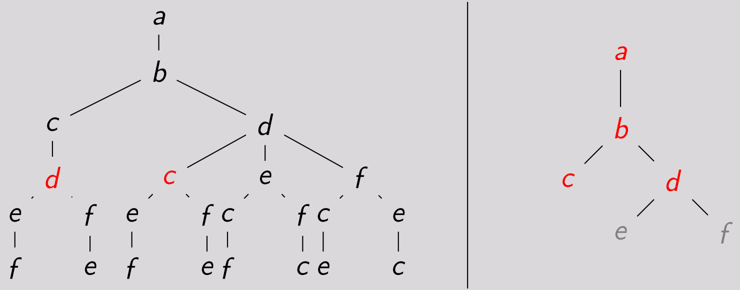
Size of a shuffle tree

Proposition

The size of the shuffle tree built on T satisfies:

$$n_T = \sum_{R \text{ substructure of } T} \ell_R.$$

Example



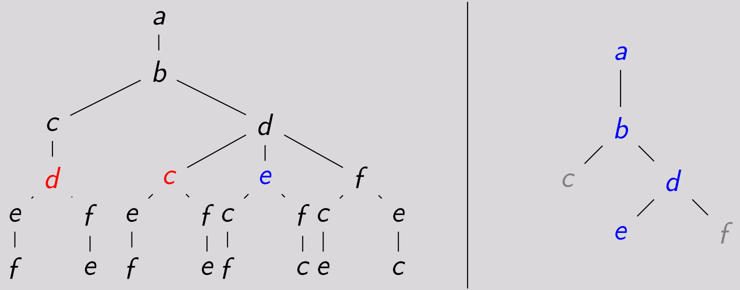
Size of a shuffle tree

Proposition

The size of the shuffle tree built on T satisfies:

$$n_T = \sum_{R \text{ substructure of } T} \ell_R.$$

Example



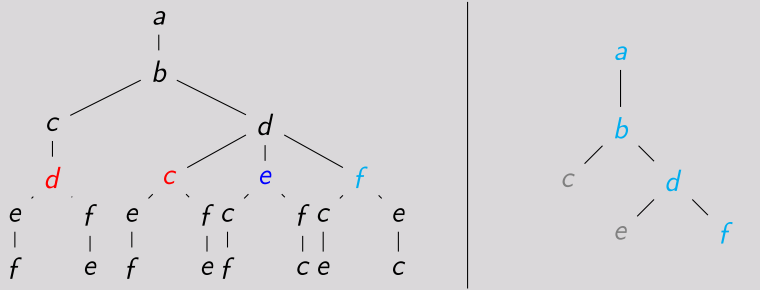
Size of a shuffle tree

Proposition

The size of the shuffle tree built on T satisfies:

$$n_T = \sum_{R \text{ substructure of } T} \ell_R.$$

Example



Mean size of shuffle trees

Theorem

The mean size $\bar{s}_n$ of a shuffle tree induced by a tree of size n follows a *P-recurrence* and satisfies:

$$\bar{s}_n \sim_{n\to\infty} e\frac{n!}{2^{n-1}} \sim 2e\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n .$$

Outline of the proof (1)

First step:

The generating function of the cumulative size of shuffle trees.

Outline of the proof (1)

First step:

The generating function of the cumulative size of shuffle trees.

- $\mathcal{C} = \mathcal{Z} \times \operatorname{Seq} \mathcal{C}$ $\qquad$ $\mathcal{M} = \mathcal{U} \times \mathcal{Z} \times \operatorname{Seq}(\mathcal{M} \cup \mathcal{C})$

Outline of the proof (1)

First step:

The generating function of the cumulative size of shuffle trees.

- $$\mathcal{S} = \mathcal{U}^{\square\mathcal{U}} \star \mathcal{Z} \times \mathsf{Seq}(\mathcal{S} \cup \mathcal{C})$$

Outline of the proof (1)

First step:

The generating function of the cumulative size of shuffle trees.

- $$\mathcal{S} = \mathcal{U}^{\square \mathcal{U}} \star \mathcal{Z} \times \mathrm{Seq}(\mathcal{S} \cup \mathcal{C})$$
- $$S(z,u) = \int_{v=0}^{\infty} \frac{z}{1 - S(z,v) - C(z)} dv = \sum_{n,k \in \mathbb{N}} S_{n,k} \cdot z^n \cdot \frac{u^k}{k!}$$

  where $S_{n,k}$ is $\displaystyle\sum_{\substack{T \\ |T|=n}} \sum_{S \text{ substructure of size } k \text{ of } T} \ell_S$.

Outline of the proof (1)

First step:

The generating function of the cumulative size of shuffle trees.

- $$\mathcal{S} = \mathcal{U}^{\Box u} \star \mathcal{Z} \times \mathrm{Seq}(\mathcal{S} \cup \mathcal{C})$$
- $$S(z,u) = \int_{v=0}^{\infty} \frac{z}{1 - S(z,v) - C(z)} dv = \sum_{n,k \in \mathbb{N}} S_{n,k} \cdot z^n \cdot \frac{u^k}{k!}$$
  where $S_{n,k}$ is $\displaystyle\sum_{\substack{T \\ |T| = n}} \sum_{S \text{ substructure of size } k \text{ of } T} \ell_S$.
- By substituting u^k by $k!$ (Gamma transformation) we obtain the generating function $S(z)$ for the size of the shuffle trees.
  $$S(z) = \int_{u=0}^{\infty} S(z,u) \exp(-u) du.$$

Outline of the proof (2)

Second step: Assisted proof using gfun.

Outline of the proof (2)

Second step: Assisted proof using gfun.

- As $S(z,u)$ is algebraic, it is holonomic.
- As $S(z,u)$ is holonomic, its Laplace transform is holonomic:

$$\hat{S}(z,u) = \int_{v=0}^{\infty} S(z,uv)\exp(-v)dv.$$

- Using the holonomic stability under partial evaluation, $S(z)$ is holonomic.
- As $S(z)$ is holonomic, its coefficients s_n follows a P-reccurence.

Computer assisted ?

```
144*(diff(S(z, u), u, u, z))*u^4*z^3+12*(diff(S(z, u), u, u,
z, z))*u^6*z+108*(diff(S(z, u), u, u, z, z))*u^5*z+648*
(diff(S(z, u), u, u, z))*u^5*z^2+72*(diff(S(z, u), u, u, u,z))*
u^6*z^2+576*(diff(S(z, u), u, z))*u^3*z^3-756*(diff(S(z, u), u,
z))*z*u^4-96*(diff(S(z, u), u, u, u, z, z))*u^6*z^2+72*(diff(S(z, u),
u, z))*u^2*z^3+3456*(diff(S(z, u), u, u, z, z))*u^5*z^4+96*(diff(S(z,
u), u, u, u, z, z))*u^6*z^3+1728*(diff(S(z, u), u,z))*u^4*z^3-336*(diff(S(z, u)
z))*u^4*z^2-60*(diff(S(z, u), u,z))*u^3*z+6*u^2*z+36*u^3*z-18*u^3-378*(diff(S(z
u), u, u, u, z))*u^6*z^3-42*(diff(S(z, u), u,u, u, z))*u^6*z-12*(diff(S(z, u), u, z, z))*u^3+
384*(diff(S(z, u), u,u, u, z, z))*z^4*u^6-864*(diff(S(z, u), u, u, z,z))*u^5*z^2-15*(diff(S(z, u), u, u, z
z, z))*u^2*z^2+96*(diff(S(z, u), z, z, z))*u*z^5+24*(diff(S(z, u), z,z, z))*u^2*z^3+384*(diff(S(z, u), z,
z))*u^4*z^3+6*(diff(S(z, u), u, u, u))*u^6+27*(diff(S(z, u), u, u,z))*u^5-128*(diff(S(z, u), u, z, z, z))*
u), z, z, z))*z^4*u+2*(diff(S(z, u), u, z, z, z))*u^2*z+54*(diff(S(z,u), u, z))*u^4+192*(diff(S(z, u), z,
u, z, z))*u^2+24*(diff(S(z, u), z, z, z))*u^3*z^2+576*(diff(S(z, u),u, u, z, z))*u^4*z^4+16*(diff(S(z, u),
z, z, z))*u^4*z^2+72*(diff(S(z, u), z, z))*u^3*z-1344*(diff(S(z, u),u, z, z))*u^3*z^3+54*(diff(S(z, u), u,
u, z, z, z))*u^4*z^2+3*(diff(S(z, u), z, z))*u+36*(diff(S(z, u), u, u,z, z, z))*u^5*z^2-144*(diff(S(z, u),
u), z, z, z))*u^2*z+1152*(diff(S(z, u), z, z))*u^2*z^4+288*(diff(S(z,u), z, z))*z^4*u+30*(diff(S(z, u), u,
u), u, u, z, z))*u^5*z^3+24*(diff(S(z, u), u, z))*u^3-6*(diff(S(z, u),u, z))*u^2+6*(diff(S(z, u), u, u, z)
u), u, u))*u^5*z-360*(diff(S(z, u), z, z))*u*z^3-672*(diff(S(z, u), z,z))*u^2*z^3+60*(diff(S(z, u), z, z))
z))*u^2*z^2+432*(diff(S(z, u), z))*u^3*z^2+72*(diff(S(z, u),z))*u*z^3-84*(diff(S(z, u), z))*u*z^2-30*(dif
z))*u^2*z-252*(diff(S(z, u), z))*u^3*z+36*(diff(S(z, u),z))*u*z-72*S(z, u)*u^3*z-12*S(z, u)*u^2*z-216*(di
u))*z*u^4-24*(diff(S(z, u), u))*u^3*z+48*(diff(S(z, u), z,z))*z^4+2304*(diff(S(z, u), u, z, z))*u^3*z^4+18
```

Outline of the proof (3)

Third step: Asymptotic behaviour of the coefficients of $\bar{s}_n$.

Outline of the proof (3)

Third step: Asymptotic behaviour of the coefficients of $\bar{s}_n$.

- Classical method gives:

$$\bar{s}_n \cdot \frac{2^{n-1}}{n!} = \theta(1).$$

- Some more work is necessary to obtain the constant.
- Finally,

$$\bar{s}_n \sim_{n\to\infty} e\frac{n!}{2^{n-1}}.$$

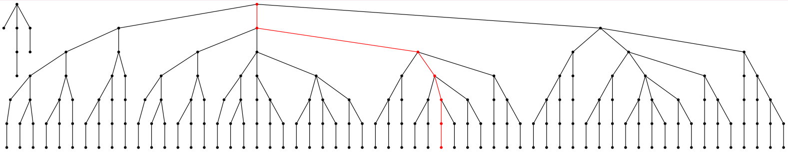
□

Outline

1 Motivations

2 Shuffle trees and their typical shape

3 Algorithms



Probability of a run prefix

Data: T: a weighted process tree of size n
Data: $\sigma := \langle \alpha_1, \ldots, \alpha_p \rangle$: a run prefix of length $p \leq n$
Result: ρ_σ: the probability of σ in the shuffle of T

$\rho_\sigma := 1$
$i := 1$
for *i from 1 to $p-1$* **do**
  $\rho_\sigma := \rho_\sigma \times \frac{|T(\alpha_{i+1})|}{n-i}$
  $i := i + 1$
return ρ_σ

Directly deduced from the hook length formula.

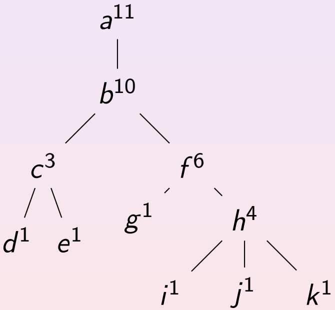
Proposition

The number of runs of a process tree T of size n can be computed in $O(n)$ operations.

[At90] gave a quadratic complexity algorithm.

Uniform random generation example

$\{1..11\}$



empty

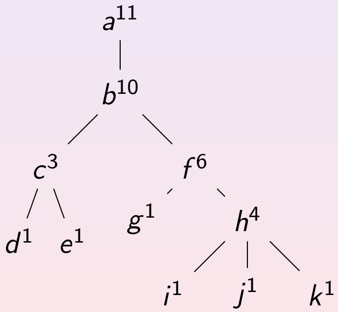
run = []

Uniform random generation example

construct

$\{1..11\}$

$\boxed{a \mid 0, 11, 0 \mid L}$



empty

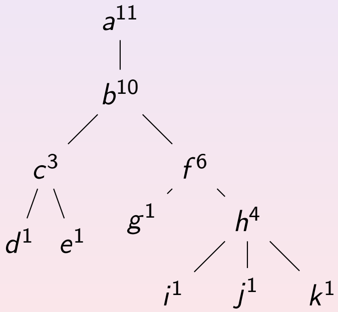
run = []

Uniform random generation example

random choice; search

$5 \in \{1..11\}$

$\boxed{a \mid 0, 11, 0 \mid L}$



empty

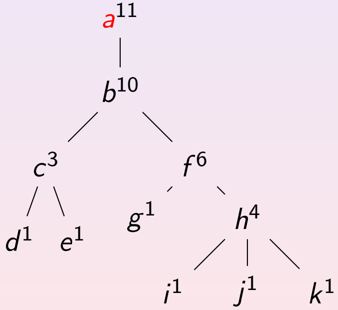
run = []

Uniform random generation example

take

$5 \in \{1..11\}$

$\boxed{a \mid 0, 11, 0 \mid L}$



empty

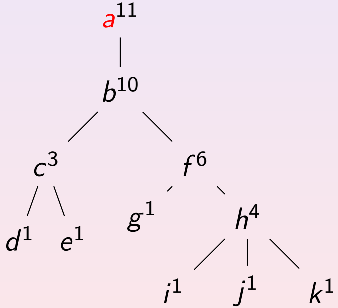
run = $[a]$

Uniform random generation example

swap

$\{1..10\}$

$\boxed{b \mid 0, 10, 0 \mid L}$



empty

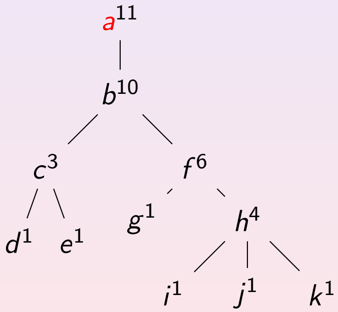
run = $[a]$

Uniform random generation example

random choice; search

$7 \in \{1..10\}$

$\boxed{b \mid 0, 10, 0 \mid L}$



$\boxed{\text{empty}}$

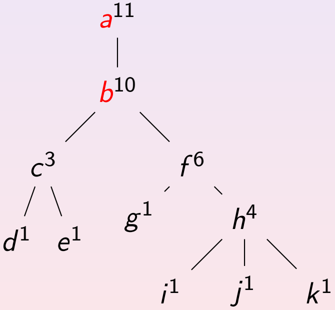
run $= [a]$

Uniform random generation example

take

$7 \in \{1..10\}$

$\boxed{b \mid 0, 10, 0 \mid L}$



empty

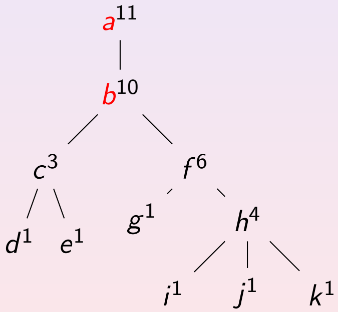
run = $[a, b]$

Uniform random generation example

swap

$\{1..9\}$

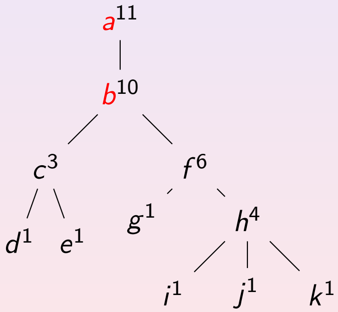
$\boxed{c \mid 0, 3, 0 \mid L}$



empty

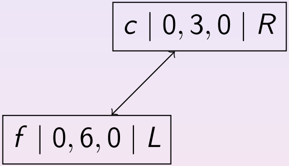
run $= [a, b]$

Uniform random generation example



construct; invert bit

$\{1..9\}$



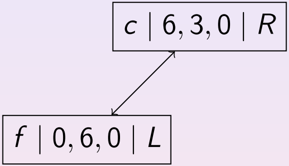
empty

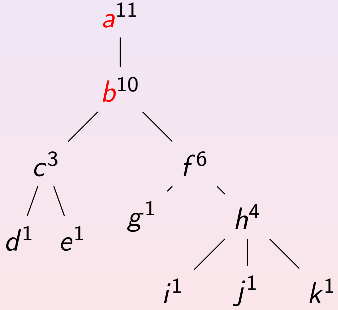
run $= [a, b]$

Uniform random generation example

update

$\{1..9\}$





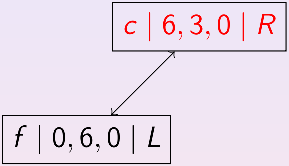
empty

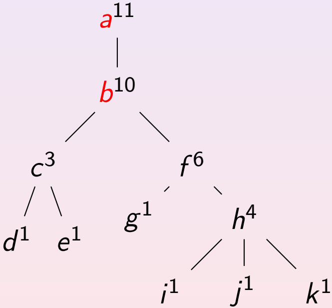
run $= [a, b]$

Uniform random generation example

random choice; search

$8 \in \{1..9\}$





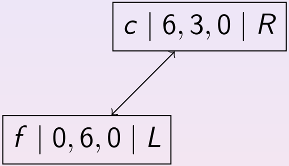
empty

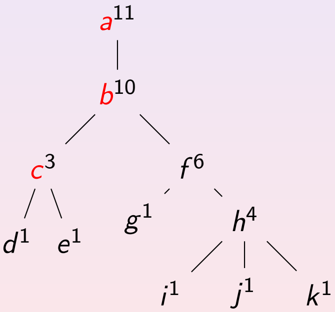
run = $[a, b]$

Uniform random generation example

take

$8 \in \{1..9\}$





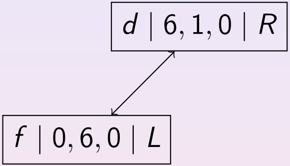
empty

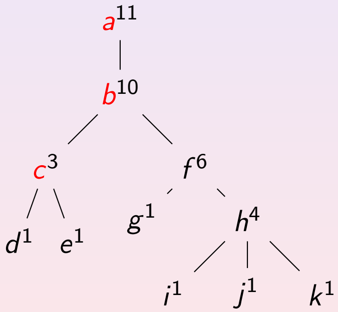
run $= [a, b, c]$

Uniform random generation example

swap

$\{1..8\}$





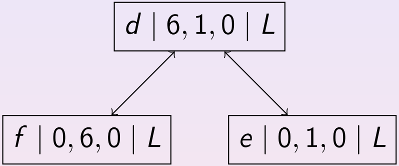
empty

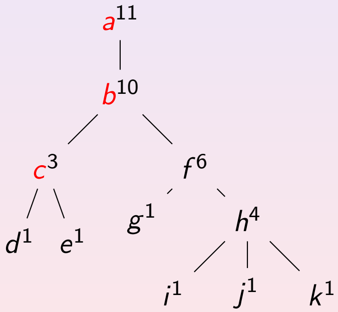
run $= [a, b, c]$

Uniform random generation example

construct; invert bit

$\{1..8\}$

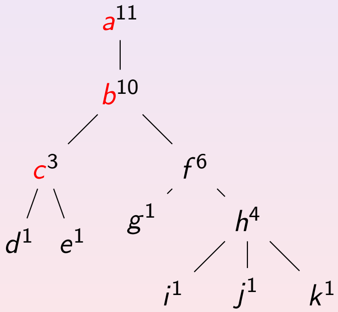




empty

run $= [a, b, c]$

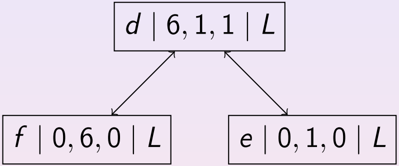
Uniform random generation example



run $= [a, b, c]$

update

$\{1..8\}$

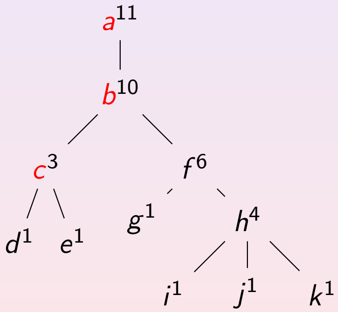


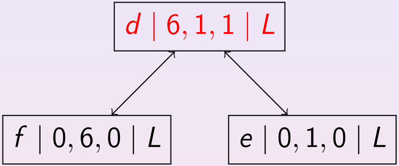
empty

Uniform random generation example

random choice; search

$8 \in \{1..8\}$

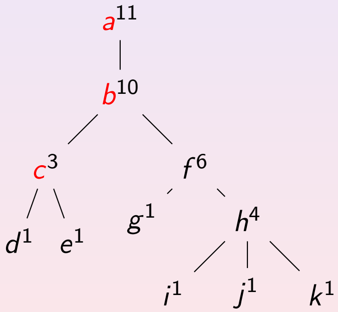




empty

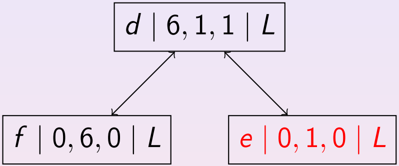
run = $[a, b, c]$

Uniform random generation example



search

$$8 - (6+1) = 1 \in \{1..1\}$$



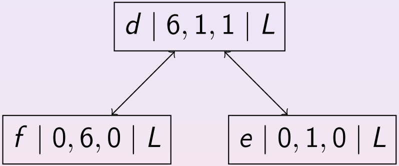
empty

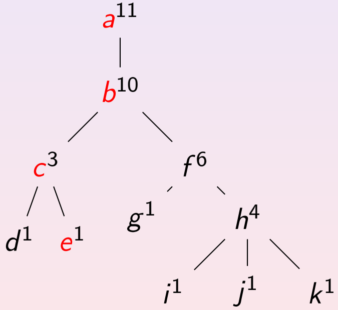
run $= [a, b, c]$

Uniform random generation example

take

$\{1..7\}$

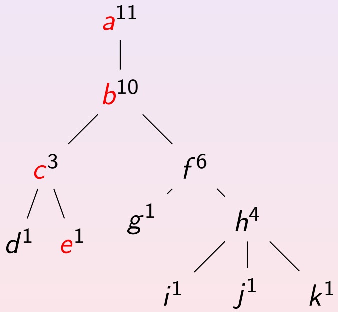


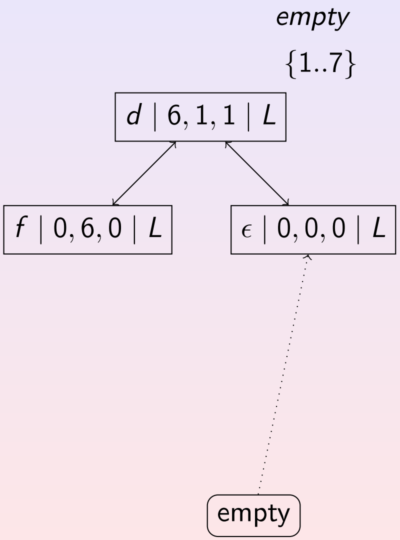


empty

run $= [a, b, c, e]$

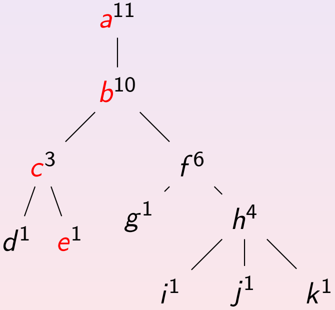
Uniform random generation example

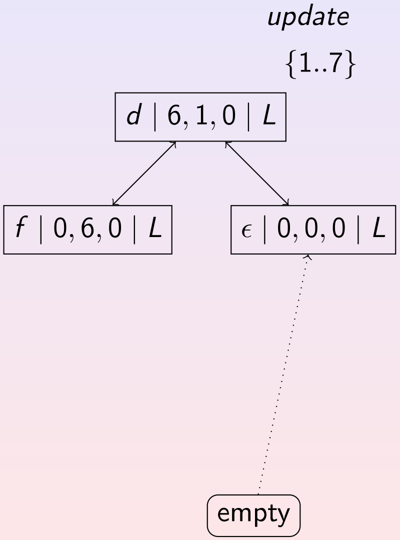




run $= [a, b, c, e]$

Uniform random generation example



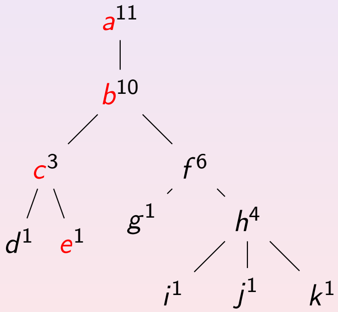


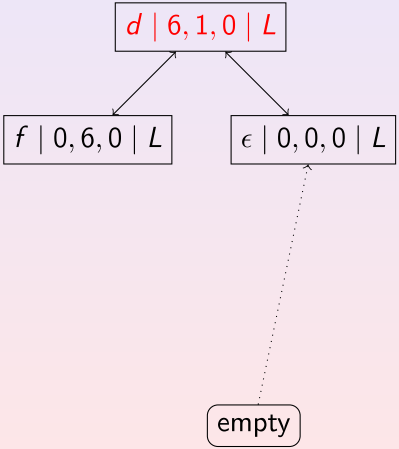
run $= [a, b, c, e]$

Uniform random generation example

random choice; search

$2 \in \{1..7\}$



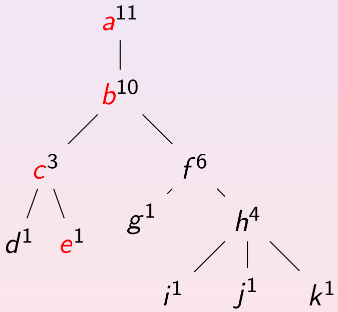


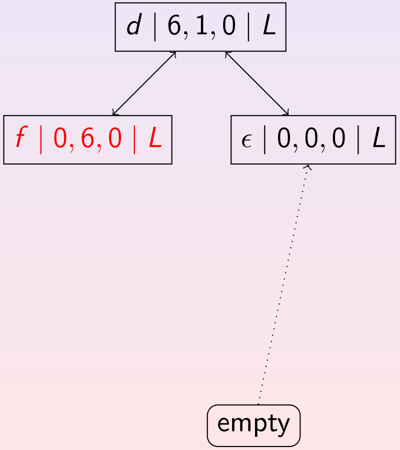
run $= [a, b, c, e]$

Uniform random generation example

search

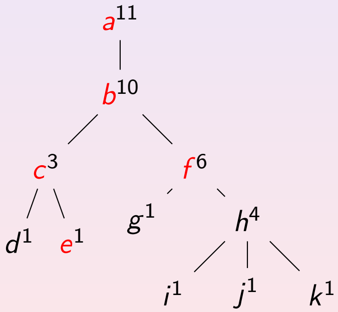
$2 \in \{1..6\}$

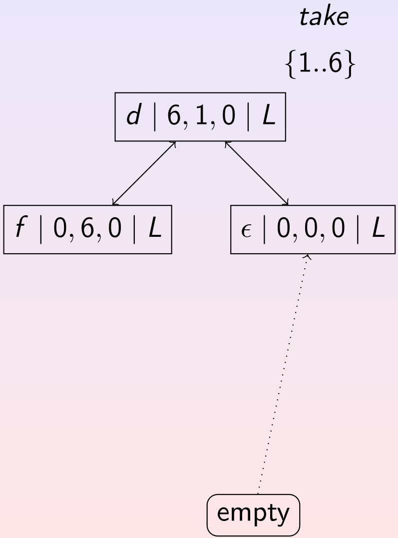




run $= [a, b, c, e]$

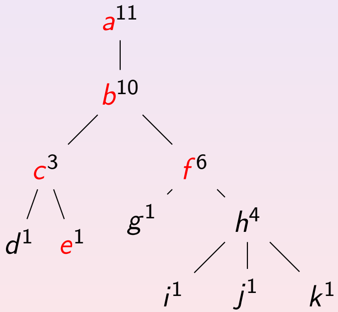
Uniform random generation example

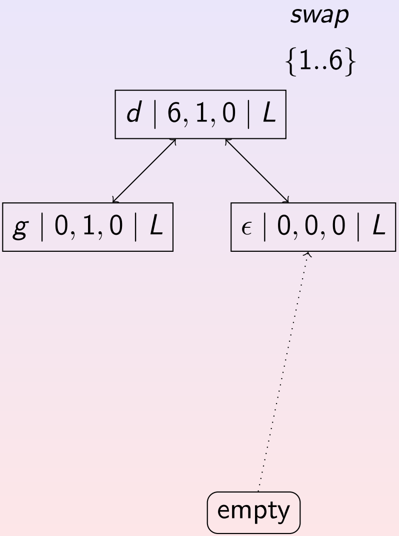




run $= [a, b, c, e, f]$

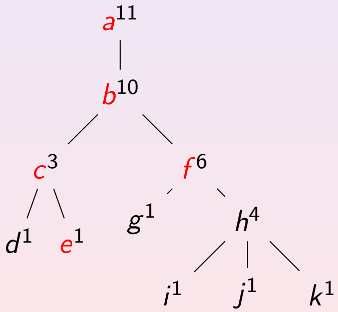
Uniform random generation example

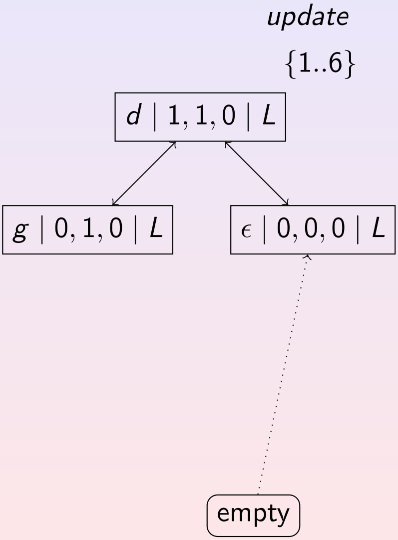




run $= [a, b, c, e, f]$

Uniform random generation example



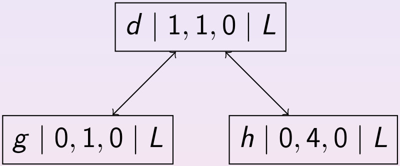


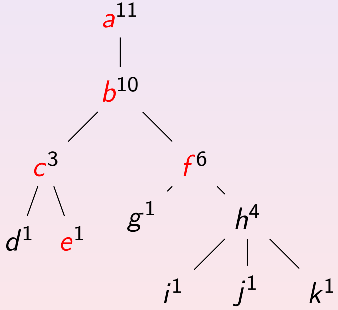
run $= [a, b, c, e, f]$

Uniform random generation example

fill

$\{1..6\}$





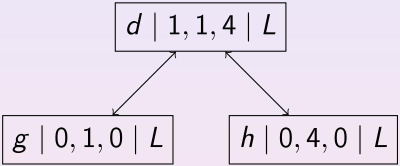
empty

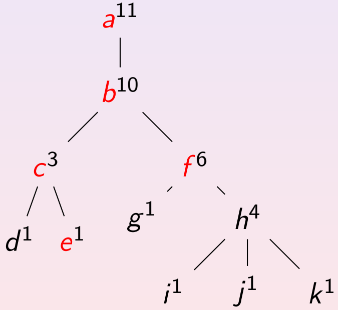
run $= [a, b, c, e, f]$

Uniform random generation example

update

$\{1..6\}$





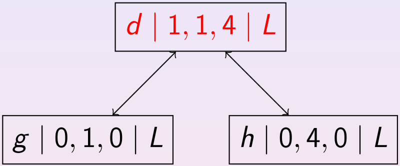
empty

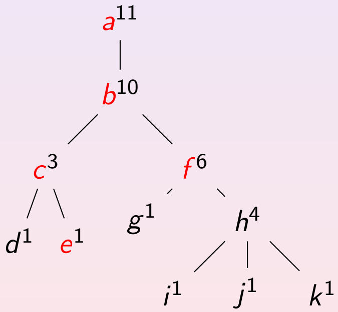
run $= [a, b, c, e, f]$

Uniform random generation example

random choice; search

$1 \in \{1..6\}$





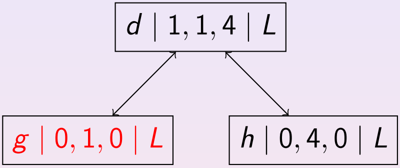
empty

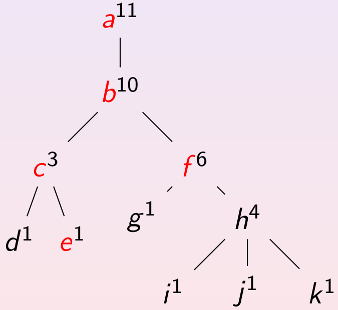
run $= [a, b, c, e, f]$

Uniform random generation example

search

$1 \in \{1..1\}$





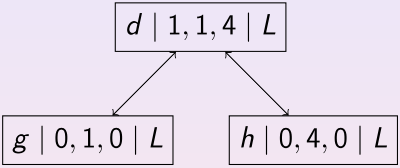
empty

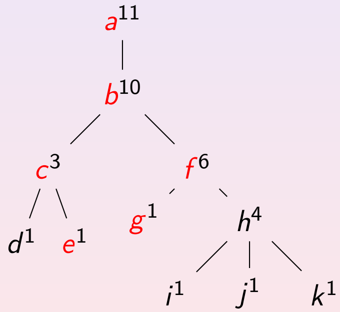
run $= [a, b, c, e, f]$

Uniform random generation example

take

$\{1..5\}$

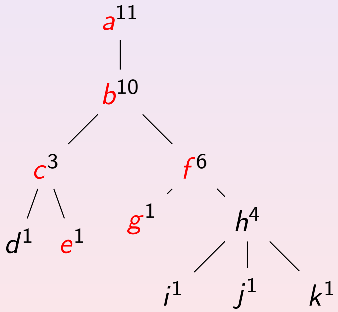


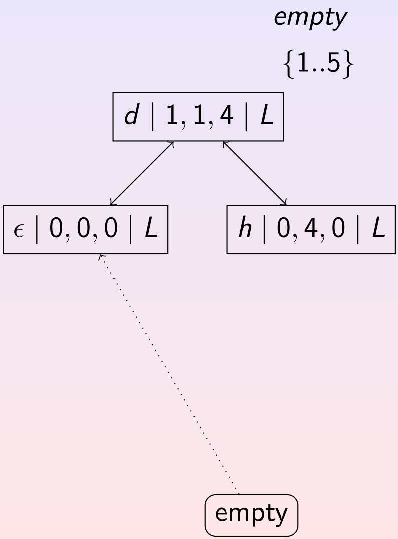


empty

run $= [a, b, c, e, f, g]$

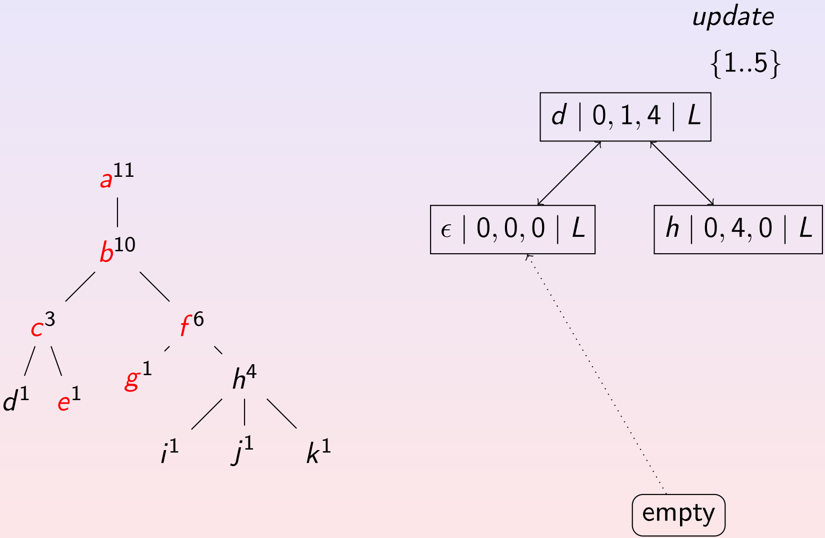
Uniform random generation example





run $= [a, b, c, e, f, g]$

Uniform random generation example

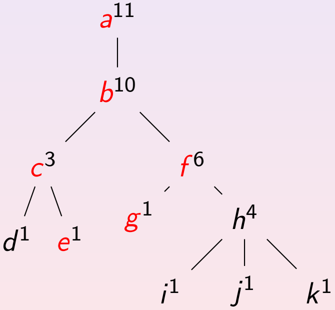


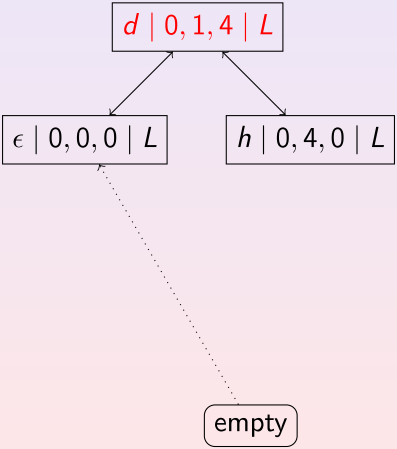
run $= [a, b, c, e, f, g]$

Uniform random generation example

random choice; search

$3 \in \{1..5\}$



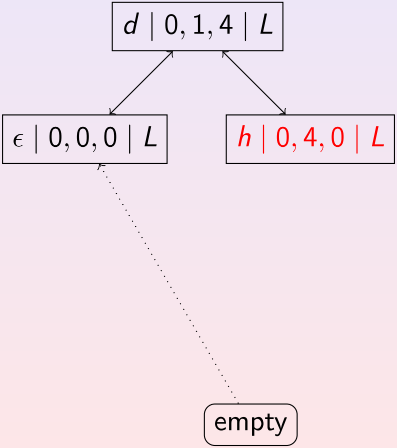


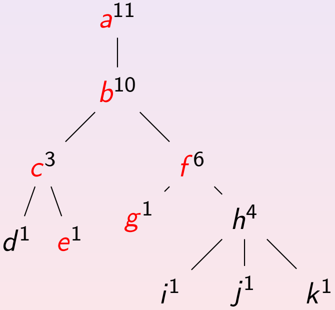
run $= [a, b, c, e, f, g]$

Uniform random generation example

search

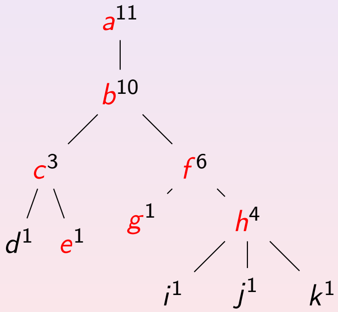
$2 \in \{1..4\}$

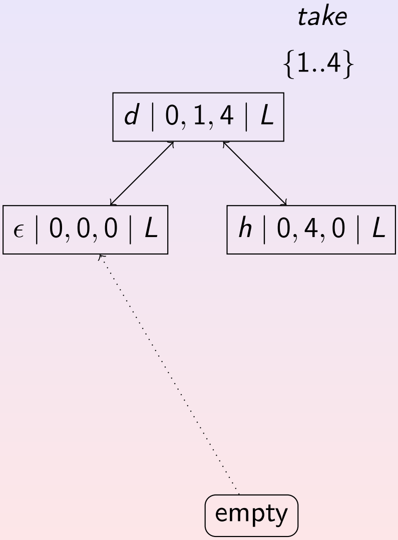




run $= [a, b, c, e, f, g]$

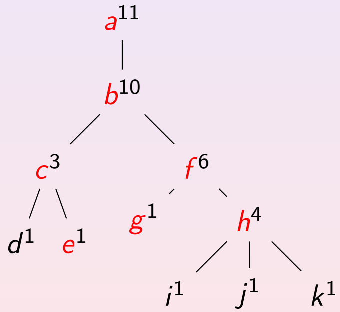
Uniform random generation example

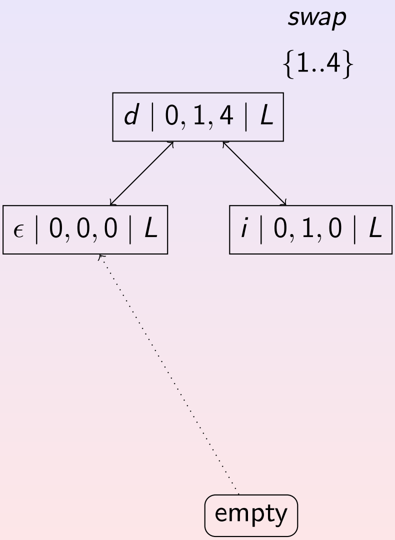




run $= [a, b, c, e, f, g, h]$

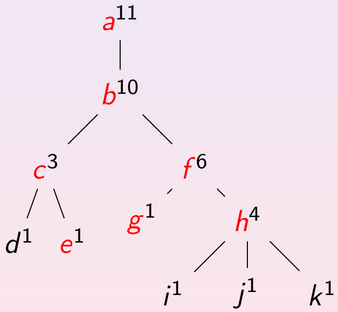
Uniform random generation example

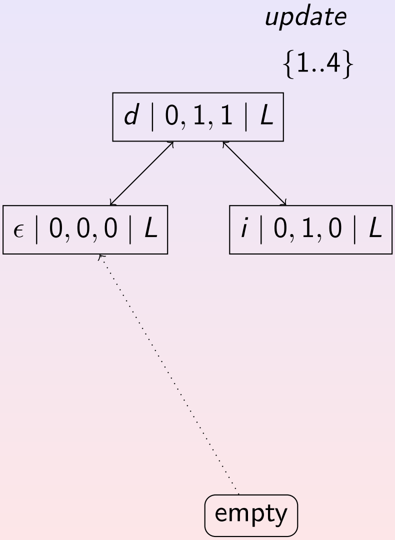




run $= [a, b, c, e, f, g, h]$

Uniform random generation example



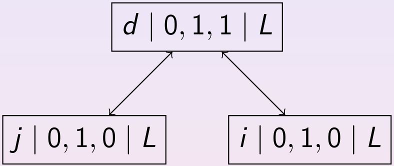


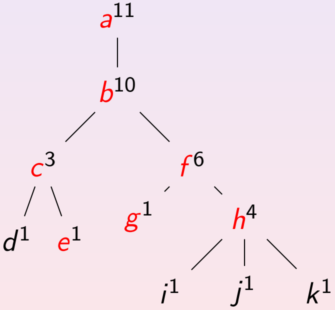
run $= [a, b, c, e, f, g, h]$

Uniform random generation example

fill

$\{1..4\}$

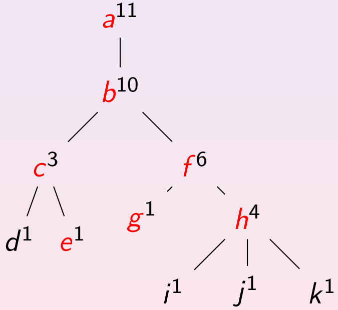


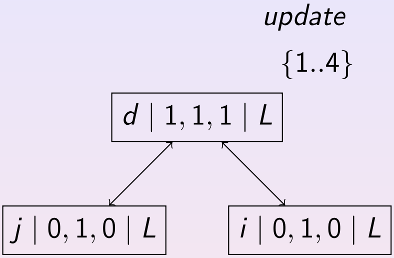


empty

run $= [a, b, c, e, f, g, h]$

Uniform random generation example





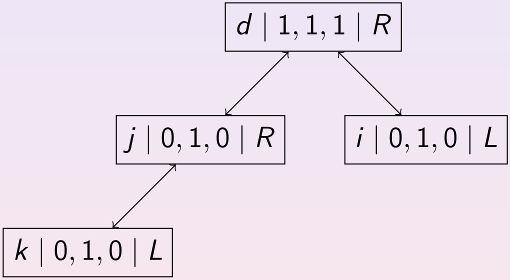
empty

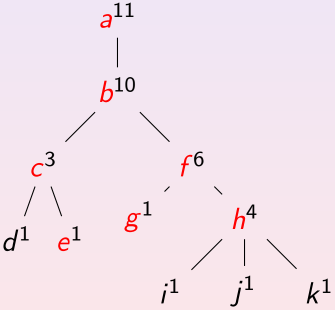
run $= [a, b, c, e, f, g, h]$

Uniform random generation example

construct ; invert bits

$\{1..4\}$

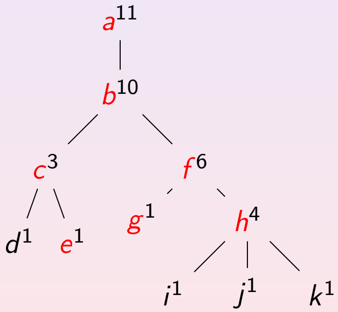


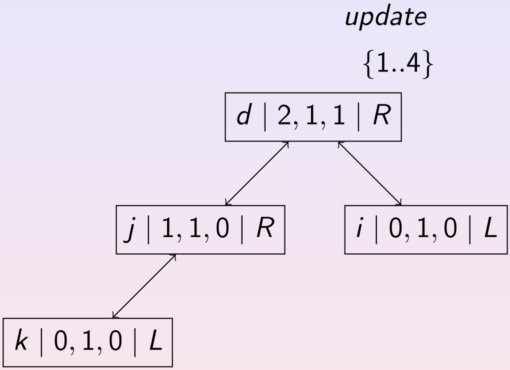


empty

run $= [a, b, c, e, f, g, h]$

Uniform random generation example





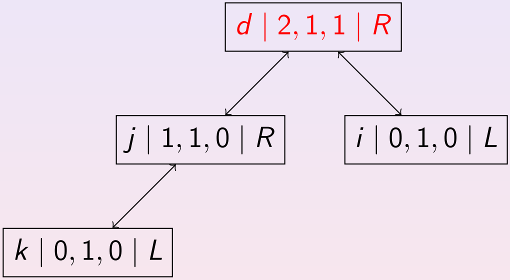
empty

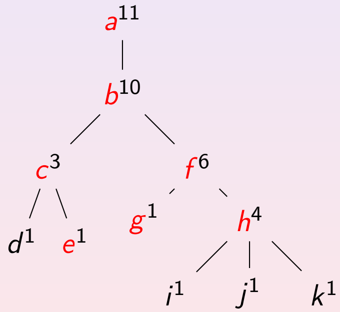
run $= [a, b, c, e, f, g, h]$

Uniform random generation example

random choice; search

$2 \in \{1..4\}$





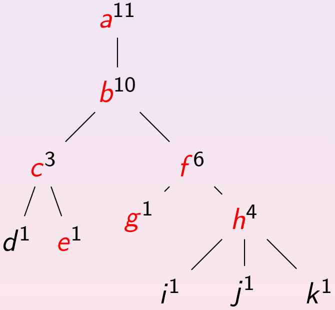
empty

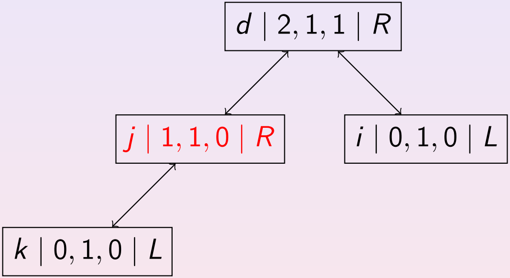
run $= [a, b, c, e, f, g, h]$

Uniform random generation example

search

$2 \in \{1..2\}$

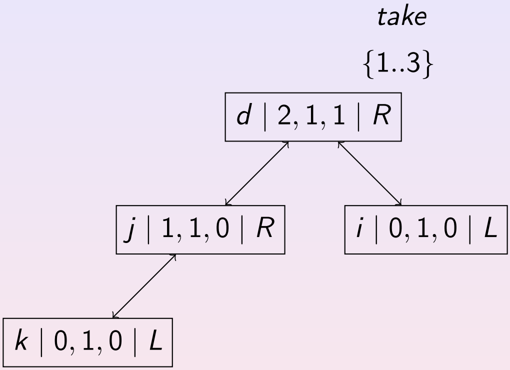


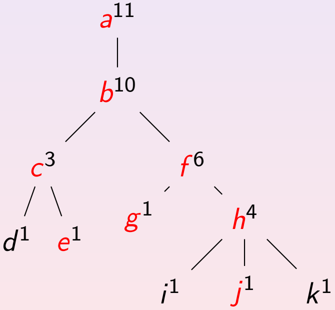


empty

run $= [a, b, c, e, f, g, h]$

Uniform random generation example

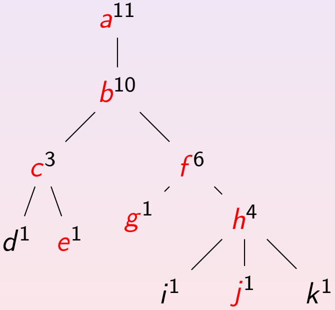


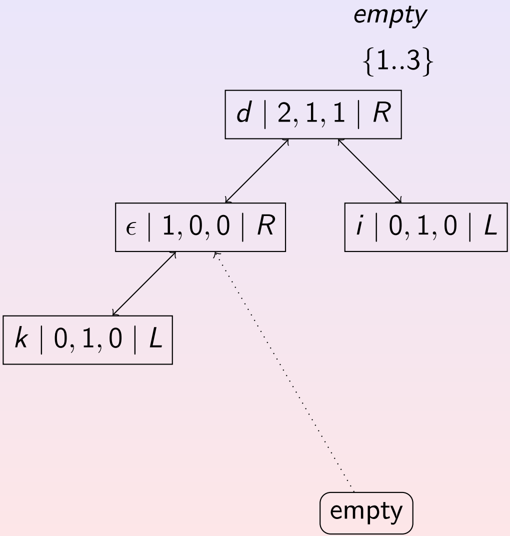


empty

run = $[a, b, c, e, f, g, h, j]$

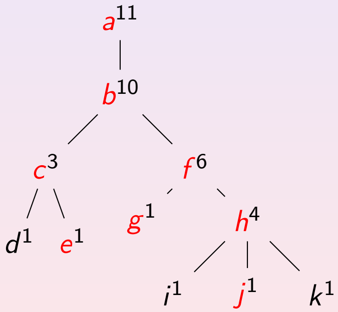
Uniform random generation example

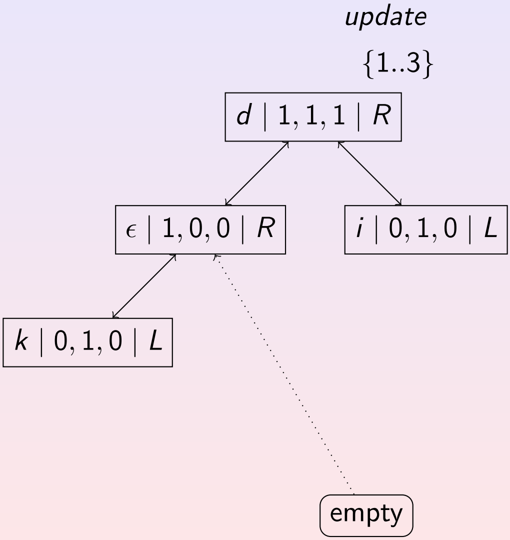




run $= [a, b, c, e, f, g, h, j]$

Uniform random generation example



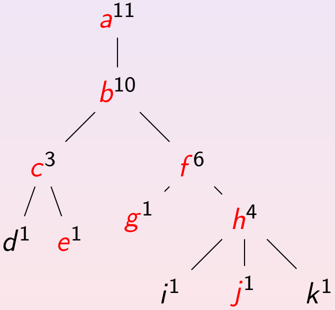


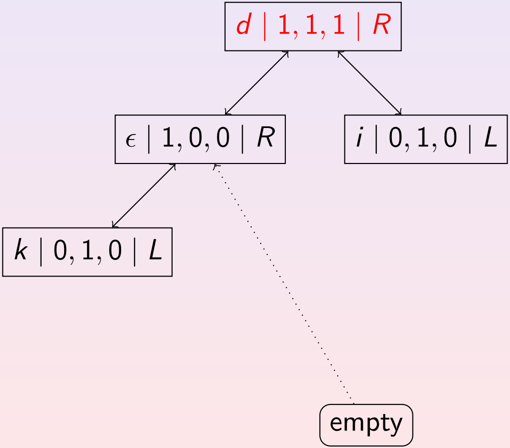
run $= [a, b, c, e, f, g, h, j]$

Uniform random generation example

random choice; search

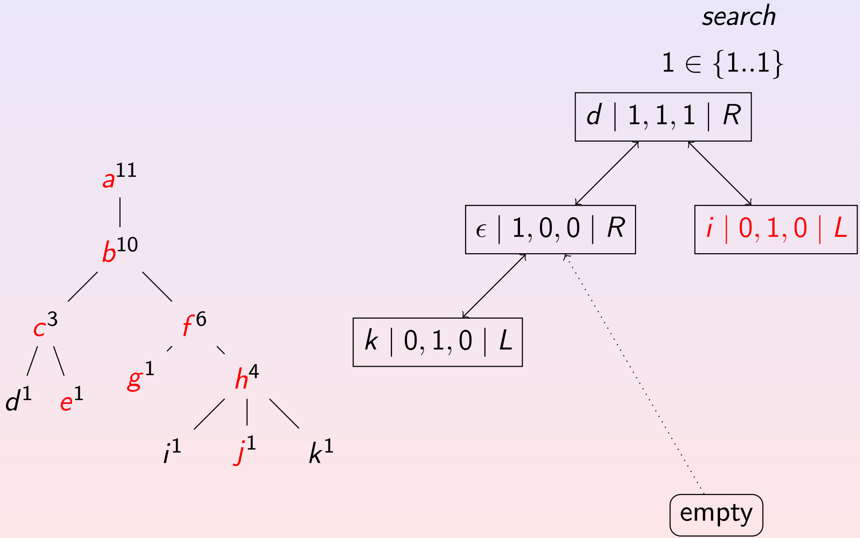
$3 \in \{1..3\}$





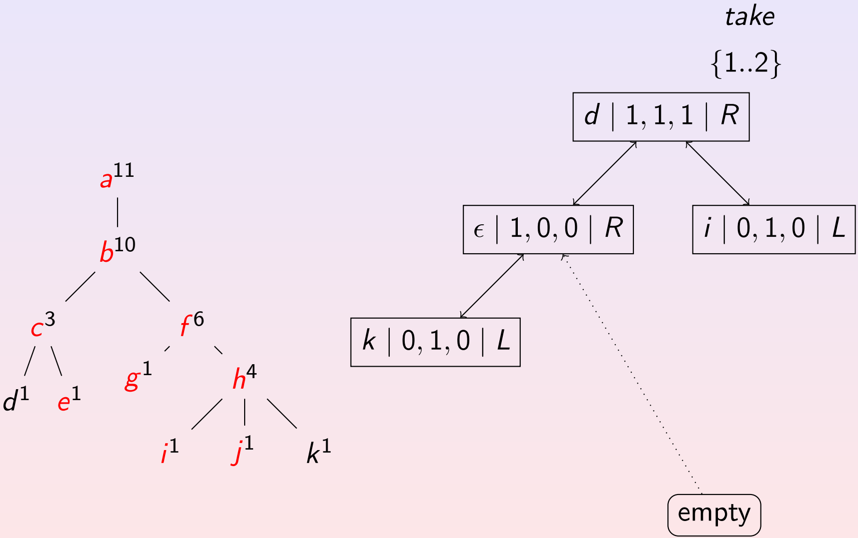
run $= [a, b, c, e, f, g, h, j]$

Uniform random generation example



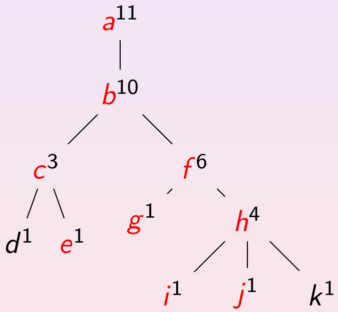
run = $[a, b, c, e, f, g, h, j]$

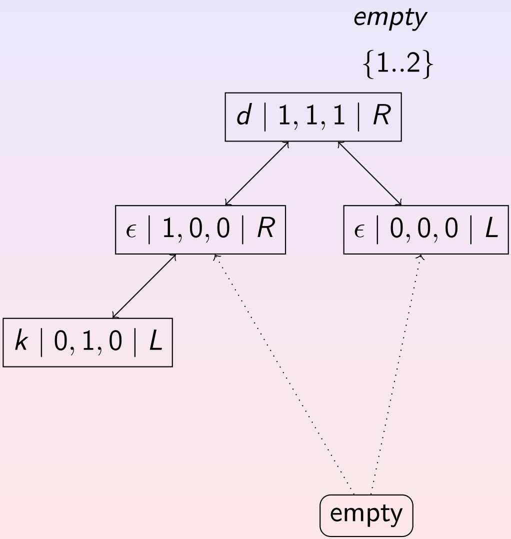
Uniform random generation example



run $= [a, b, c, e, f, g, h, j, i]$

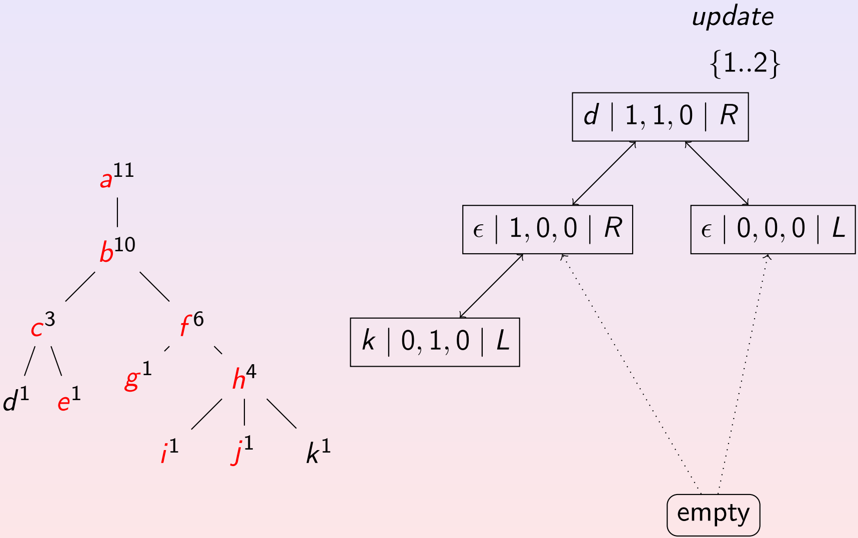
Uniform random generation example





run $= [a, b, c, e, f, g, h, j, i]$

Uniform random generation example

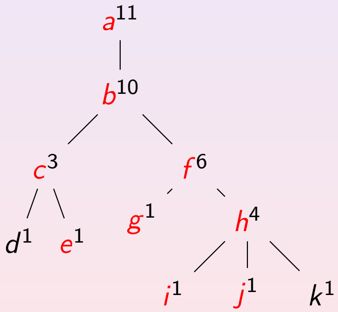


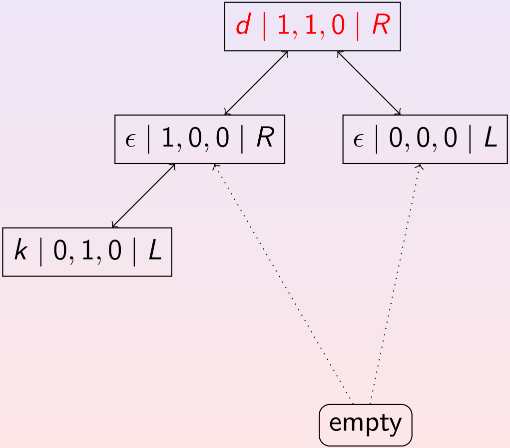
run = $[a, b, c, e, f, g, h, j, i]$

Uniform random generation example

random choice; search

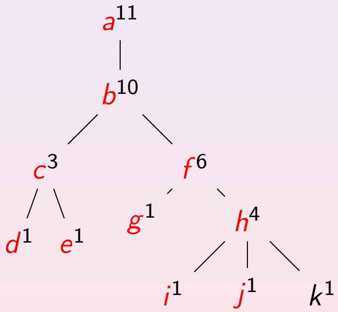
$2 \in \{1..2\}$

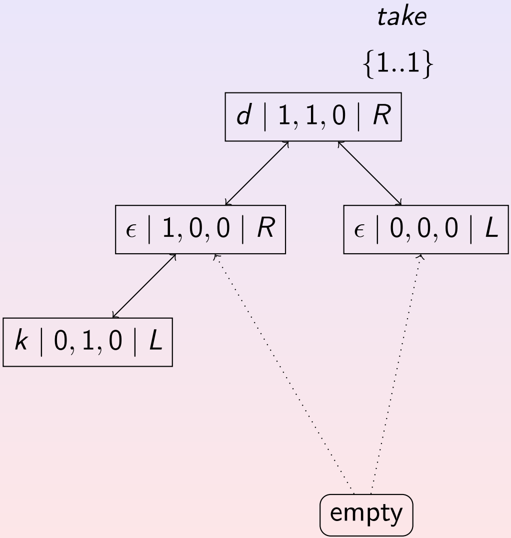




run $= [a, b, c, e, f, g, h, j, i]$

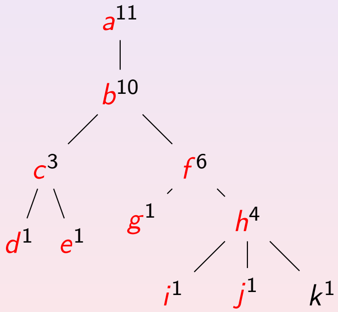
Uniform random generation example

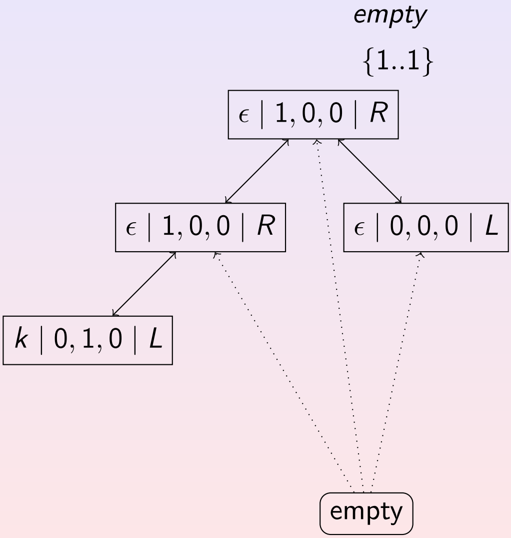




run $= [a, b, c, e, f, g, h, j, i, d]$

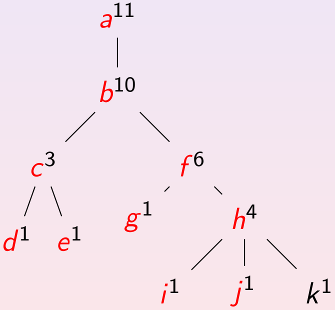
Uniform random generation example





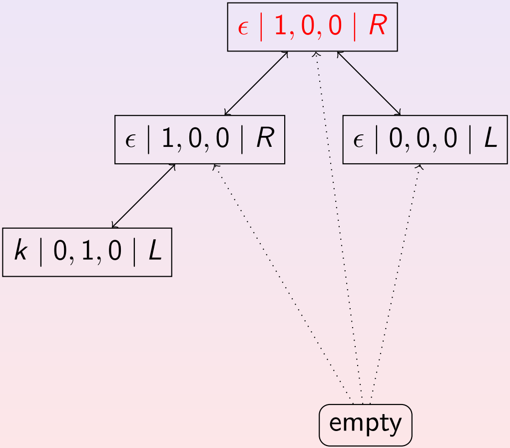
run $= [a, b, c, e, f, g, h, j, i, d]$

Uniform random generation example



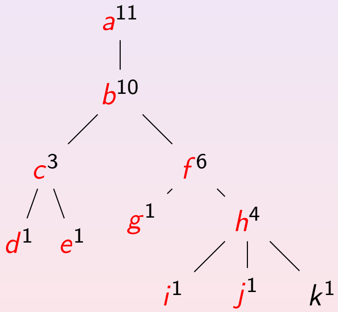
random choice; search

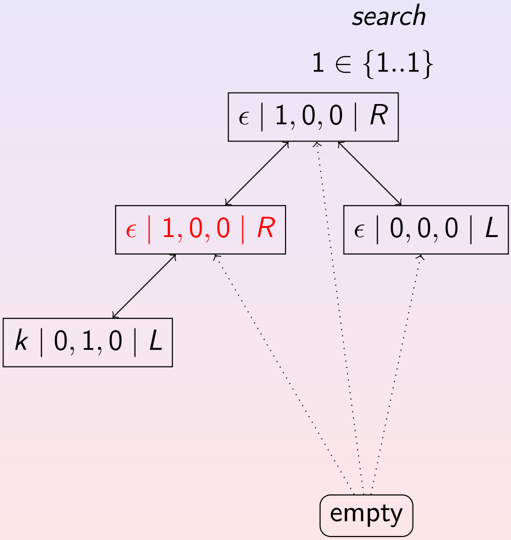
$1 \in \{1..1\}$



run $= [a, b, c, e, f, g, h, j, i, d]$

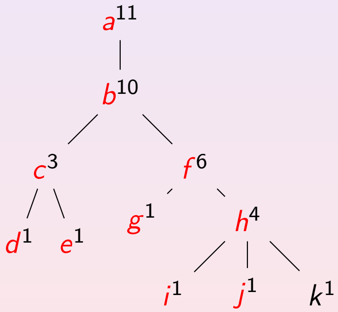
Uniform random generation example

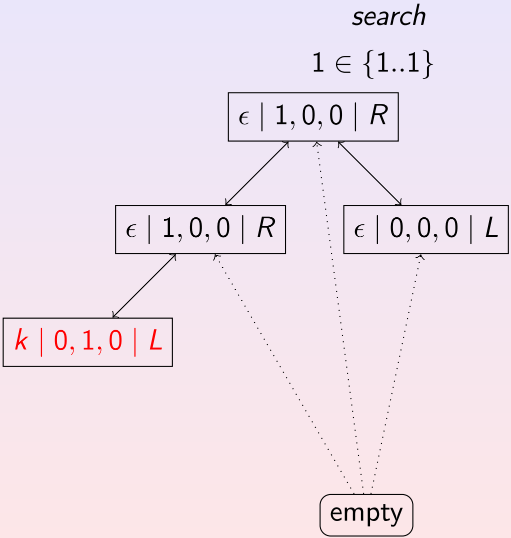




run $= [a, b, c, e, f, g, h, j, i, d]$

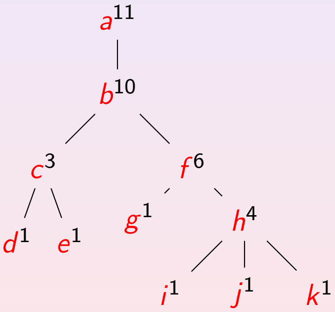
Uniform random generation example

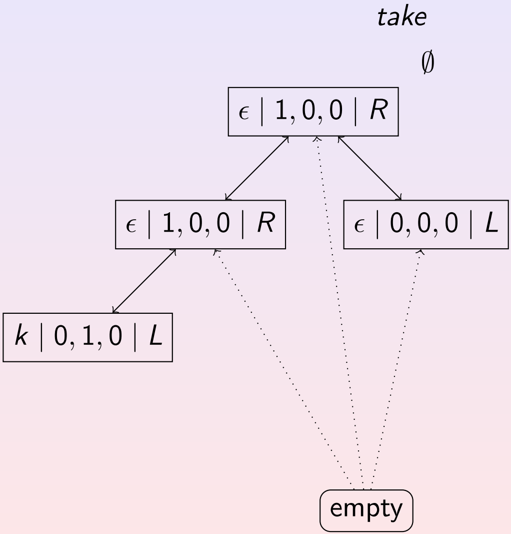




run $= [a, b, c, e, f, g, h, j, i, d]$

Uniform random generation example





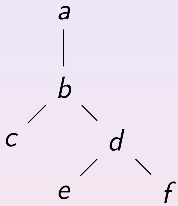
run $= [a, b, c, e, f, g, h, j, i, d, k]$

Conclusion and perspectives

First step for the quantitative analysis of concurrent theory objects,. . .

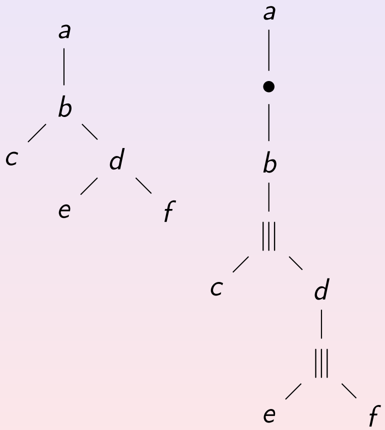
Conclusion and perspectives

First step for the quantitative analysis of concurrent theory objects,. . .



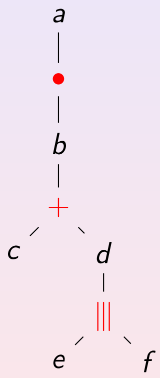
Conclusion and perspectives

First step for the quantitative analysis of concurrent theory objects,. . .



Conclusion and perspectives

First step for the quantitative analysis of concurrent theory objects,. . .



Conclusion and perspectives

First step for the quantitative analysis of concurrent theory objects,. . .