A Quantitative Study of Pure Parallel Processes*

O. Bodini
Laboratoire d'Informatique de Paris-Nord,
CNRS UMR 7030, Institut Galilée and
Université Paris-Nord,
99, avenue Jean-Baptiste Clément,
93430 Villetaneuse, France.
Olivier.Bodini@lipn.univ-paris13.fr.

A. Genitrini and F. Peschanski
Sorbonne Universités,
UPMC Univ Paris 06,
CNRS, LIP6 UMR 7606,
4 place Jussieu 75005 Paris.
Antoine.Genitrini@lip6.fr
Frederic.Peschanski@lip6.fr.

Submitted: Sept, 2013; Accepted: XXX; Published: XX
Mathematics Subject Classifications: XXXXX

Abstract

In this paper, we study the interleaving – or pure merge – operator that most often characterizes parallelism in concurrency theory. This operator is a principal cause of the so-called combinatorial explosion that makes the analysis of process behaviours e.g. by model-checking, very hard – at least from the point of view of computational complexity. The originality of our approach is to study this combinatorial explosion phenomenon on average, relying on advanced analytic combinatorics techniques. We study various measures that contribute to a better understanding of the process behaviours represented as plane rooted trees: the number of runs (corresponding to the width of the trees), the expected total size of the trees as well as their overall shape. Two practical outcomes of our quantitative study are also presented: (1) a linear-time algorithm to compute the probability of a concurrent run prefix, and (2) an efficient algorithm for uniform random sampling of concurrent runs. These provide interesting responses to the combinatorial explosion problem.

Keywords: Pure Merge, Interleaving Semantics, Concurrency Theory, Analytic Combinatorics, Increasing Trees, Holonomic Functions, Random Generation.

*This research is supported by the CNRS project *ALPACA* (PEPS INS2I 2012), by the A.N.R. project *MAGNUM*, ANR 2010-BLAN-0204 and by the ANR project *MetACOnc*, ANR-15-CE40-0014.

1 Introduction

A significant part of *concurrency theory* is built upon a simple *interleaving* operator named the *pure merge* in [BW90]. The basic underlying idea is that two independent processes running in parallel, denoted $P \parallel Q$, can be faithfully simulated by the interleaving of their computations. We denote $a.P$ (resp. $b.Q$) a sequential process that first executes an *atomic action* a (resp. b) and then continues as a process P (resp. Q).

The interleaving law then states[1]:

$$a.P \parallel b.Q = a.(P \parallel b.Q) + b.(a.P \parallel Q),$$

where $+$ is interpreted as the union operator.

The pure merge operator is a principal source of *combinatorial explosion* of the possible behaviour, when analysing concurrent processes, e.g. by *model checking* [CGP99]. This issue has been thoroughly investigated and many approaches have been proposed to counter the explosion phenomenon, in general based on compression and abstraction/reduction techniques. If several decidability and worst-case complexity results are known, to our knowledge the interleaving of process structures as computation trees has not been studied extensively from the *average case* point of view.

In analytic combinatorics, the closest related line of work addresses the *shuffle* of regular languages, generally on disjoint alphabets [FGT92, MZ08, GDG+08, DPRS12]. The shuffle on (disjoint) words can be seen as a specific case of the interleaving of processes (for processes of the form $(a_1 \dots a_n) \parallel (b_1 \dots b_m)$). Interestingly, a quite related concept of interleaving of tree structures has been investigated in algebraic combinatorics [BFLR11], and specially in the context of partly commutative algebras [DHNT11]. We see our work as a continuation of this line of works, now focusing on the quantitative and analytic aspects.

Our objective in this work is to better characterize the *typical shape* of concurrent process behaviours as computation trees and for this we rely heavily on *analytic combinatorics* techniques, indeed on the *symbolic method*. One significant outcome of our study is the emergence of a deep connection between concurrent processes and increasing labelling of combinatorial structures. We expect the discovery of similar increasingly labelled structures if we go deeper into concurrency theory. This work follows the idea of investigating *concrete* problems with advanced analytic tools. In the same spirit, we emphasize practical applications resulting from such thorough mathematical studies. In the present case, we develop algorithmic techniques to analyse probabilistically the process behaviours through *counting* and *uniform random generation*.

Our study is organized as follows. In Section 2 we define the recursive construction of the interleaved process behaviours from syntactic process trees, and study the basic structural properties of this construction. In Section 3 we investigate the number of

[1]When one is interested in a finite axiomatization of the pure merge operator, a left variant must be introduced, cf.[BW90] for details.

concurrent runs that satisfy a given process specification. Based on an isomorphism with *increasing trees* – that proves particularly fruitful – we obtain very precise results. We then provide a precise characterization of what "exponential growth" means in the case of pure parallel processes. We also investigate the case of non-plane trees. In Section 4 we discuss, both theoretically and experimentally, the decomposition of computation trees by level. This culminates with a rather precise characterization of the typical shape of process behaviours. We then study, in Section 5, the expected size of process behaviours. This typical measure is precisely characterized by a linear recurrence relation that we obtain in three distinct ways. While reaching the same conclusion, each of these three proofs provides a complementary view of the combinatorial objects under study. Taken together, they illustrate the richness and variety of analytic combinatorics techniques. Section 6 is devoted to practical applications resulting from this quantitative study. First, we describe a simple algorithm to compute the probability of a run prefix in linear time. As a by-product, we obtain a very efficient way to calculate the number of *linear extensions* of a *tree-like partial order* or *tree-poset*. The second application is an efficient algorithm for the uniform random sampling of concurrent runs. These algorithms work directly on the syntax trees of the process without requiring the explicit construction of their behaviour, thus avoiding the combinatorial explosion issue.

This paper is an updated and extended version of [BGP12]. It contains new material, especially the study of the typical shape of process behaviours in Section 4. The more complex setting of non-plane trees is also discussed. Appendix A was added to discuss the weighted random sampling in dynamic multisets. The proofs in this extended version are also more detailed.

2 A tree model for process semantics

As a starting point, we recast our problem in combinatorial terms. The idea is to relate the *syntactic domain* of process specifications to the *semantic domain* (or model) of process behaviours.

2.1 Syntax trees

The grammar we adopt for pure parallel processes is very simple. A process is specified as follows:

- an *atomic action*, denoted by α, is a process,
- the *prefixing* $\alpha.P$ of an action α and a process P is a process, and, more precisely, a *prefixed process*,
- the *composition* $P_1 \parallel \ldots \parallel P_n$ of a finite number of atomic actions or prefixed processes is a process.

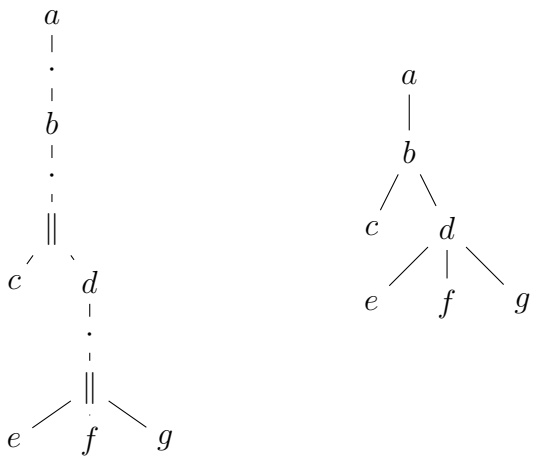


Figure 1: A process (left), its isomorphic syntax tree (right).

Furthermore each action can appear only once in a process. We consider that two processes are equivalent if and only if the first one can be transformed in the second one by a bijective relabelling of the actions.
We define the *size of a process* to be the number of actions it contains.

Let us first remark that this grammar takes the parallel operators, denoted $\|$, as associative operators and thus two of them cannot appear consecutively, i.e. in the composition $P_1 \| \ldots \| P_n$, no P_i is a composition. Moreover, in the rest of the paper we will focus on *prefixed processes*, i.e. we do not allow a whole process to be a composition. This choice does not deplete the results thanks to the bijection between prefixed processes of size n and processes that are compositions of size $n-1$.

The following is an example process:

$$a.b.(c \parallel d.(e \parallel f \parallel g)).$$

This process, that we use as an illustration throughout the paper, is depicted on the left-hand side of Figure 1. A straightforward isomorphic representation is obtained by removing the internal nodes corresponding to the $\cdot$ and the $\|$ operators (cf. on the right-hand side of Figure 1).

The process above is equivalent by relabelling to e.g. $b.a.(r \parallel d.(e \parallel f \parallel g))$. We call the representative of a class of equivalent processes a *syntax tree* with is the unique process such its actions are labelled alphabetically according to the preorder traversal.

Syntax trees are the classical plane rooted trees with some decoration: the actions in the nodes.

In the rest of the paper, we are only dealing with syntax trees, thus the names of the actions convey no combinatorial meaning. The right-hand side of Figure 1 is indeed the syntax tree of our example process.

An alternative interpretation of a syntax tree is as a set of *precedence constraints* between atomic actions. Under this light the action a at the root must be executed first and then b. There is no relation between c and d – they are said independent – and e, f, g may only happen after d. In order-theoretic terms, this forms a *tree-poset* [Atk90]

In combinatorial terms, we adopt the classical specification for *plane rooted trees* to represent the syntax trees. The *size* of a tree is its total number of nodes (i.e. the number of actions of the process).

Definition 1. *The specification* $\mathcal{C} = \mathcal{Z} \times \textsc{Seq}(\mathcal{C})$ *represents the combinatorial class of plane rooted trees.*

As a basic recall of analytic combinatorics and statement of our conventions[2], we remind the reader that for such a combinatorial class $\mathcal{C}$, we define its counting sequence C_n consisting of the number of objects of $\mathcal{C}$ of size n. This sequence is linked to the formal power series $C(z) = \sum_{n\geqslant 0} C_n z^n$, called the *(ordinary) generating function* of $\mathcal{C}$. We denote by $[z^n]C(z) = C_n$ the n-th coefficient of $C(z)$. Analogous writing conventions will be used for all combinatorial classes in this paper.

We remind the reader that in the case of class $\mathcal{C}$, the sequence C_n corresponds to the *Catalan numbers* (indeed, shifted by one). For further reference, we give the generating function of $\mathcal{C}$ and the asymptotic approximations of the Catalan numbers (obtained by the Stirling formula approximation of $n!$ as in e.g. [Com74, p. 267]).

Fact 2. $C(z) = \frac{1}{2} - \frac{\sqrt{1-4z}}{2}$ *and* $C_n = \frac{1}{n}\binom{2n-2}{n-1} = \frac{4^{n-1}}{\sqrt{\pi n^3}}\left(1 + \frac{3}{8n} + \frac{25}{128n^2} + O\left(\frac{1}{n^3}\right)\right)$.

In the paper, we use the classical definitions in the context of plane rooted trees, (cf. e.g. [FS13]). A tree has a *root* (for example the root of the syntax tree in Figure 1 is the node a). Each node of a tree has an outdegree called the *arity* of the node. In our example, the root has arity 1. The *children* of a node ν are the nodes attached to ν: b is the single child of a. Conversely the node a is the *parent* of the node b. The *descendants* of a node are the children of the node and iteratively all their respective descendants. In fact, the descendants of a are the nodes b, c, d, e, f and g. We define a *subtree* rooted at a node to be the whole tree induced by *all* the descendants of the node. The subtree rooted in d in our example is the whole structure induced by $d.(e \parallel f \parallel g)$. Finally, let us define a *child-subtree* of a node ν to be a subtree whose root is a child node of ν.

2.2 Computation trees

As a semantic domain, we study *computation trees* [CES86], a subclass of plane rooted trees that encode the semantics of concurrent processes. An example of a computation tree is depicted in Figure 2, which is induced by our example syntax tree. Each branch of the computation tree, e.g. $\langle a, b, c, d, e, f, g\rangle$, corresponds to a *run* (or admissible computation) of the process. We note that in the computation tree all the branches share their common prefix. We now formalize the construction.

[2] The reader will find our conventions and the material about analytic combinatorics introduced in both books of Flajolet and Sedgewick [FS09, FS13].

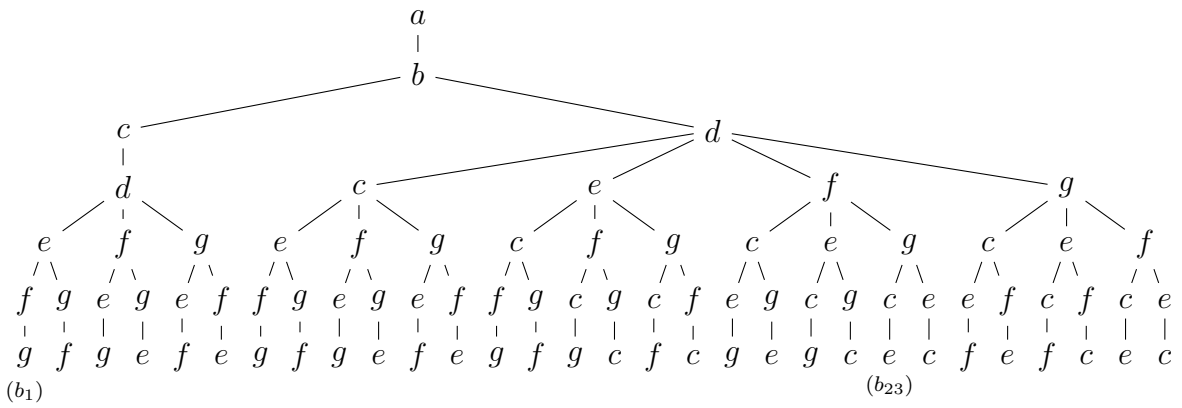
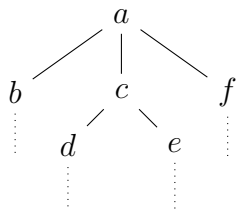
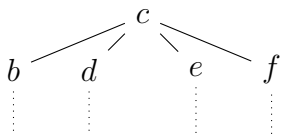


Figure 2: The computation tree induced by the syntax tree of Figure 1.

To describe the recursive construction of computation trees, we use an elementary operation of *child contraction*.

Definition 3. *Let T be a plane rooted tree and $v_1, \ldots, v_r$ be the children of the root. For $i \in \{1, \ldots, r\}$, the **i-contraction** of T is the plane tree with root v_i and the child-subtrees attached to v_i are $T(v_1), \ldots, T(v_{i-1}), T(v_{i_1}), \ldots, T(v_{i_m}), T(v_{i+1}), \ldots, T(v_r)$ (from left to right and $T(\nu)$ denotes the subtree of T whose root is ν and $v_{i_1}, \ldots, v_{i_m}$ are the children of $T(v_i)$). We denote by $T \lhd i$ the i-contraction of T.*

For example, if T is  then $T \lhd 2$ is 

Note that the root (here a) is replaced by the 2-nd child (here c). Now, the interleaving operation follows a straightforward recursive scheme.

Definition 4. *Let T be a syntax tree, then its **computation tree** $\textsc{Sem}(T)$ is defined inductively as follows:*

- *if T is a leaf, then $\textsc{Sem}(T)$ is T,*
- *if T has a root ν of arity r ($r \in \mathbb{N}\backslash\{0\}$), then $\textsc{Sem}(T)$ is the plane tree with root ν and the child-subtrees attached to ν are, from left to right, $\textsc{Sem}(T \lhd 1), \ldots, \textsc{Sem}(T \lhd r)$.*

*A **run** of the process encoded by T is the sequence of action labels encoded by a branch of $\textsc{Sem}(T)$.*

The mapping between the syntax trees on the one side, and the computation trees on the other side is trivially one-to-one. Figure 3 depicts the enumeration of the first syntax trees (from sizes 1 to 4) together with their corresponding computation trees.

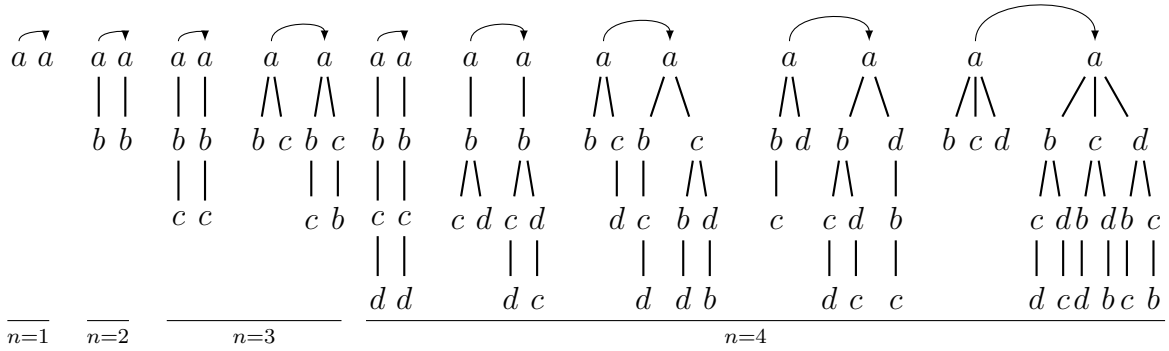


Figure 3: Enumerating the behaviours (computation trees) of processes (syntax trees).

We note that the computation trees are *balanced* (i.e. all their leaves belong to the same level), and, even more important that their height is n, where n is the size of the associated syntax tree. This is obvious since at each step of the recursive construction exactly one node is consumed. Thus there are as many computation trees of height n as there are syntax trees of size n (as counted by C_n above).

A further basic observation is that the contraction operator (cf. Definition 3) ensures that the number of nodes at a given level of a computation tree is at most equal to the number of nodes at the next level. Thus, the width of the computation tree corresponds to its number of leaves.

An important step in our study is to interpret computation trees as sets of *linear extensions* of tree-like partial orders or *tree-posets* [Atk90].

Definition 5. *Let T be the syntax tree of a process, and A its set of actions. We define the poset $(A, \prec)$ such that $a \prec b$ if a is a node that is the parent of a node b in T; moreover, the relation $\prec$ is taken to be transitive. The set of linear extensions of $(A, \prec)$ is the set of all the total orders $(A, <)$ that extend the partial order.*

For example, the syntax tree depicted in Figure 1 is interpreted as the partial order $a \prec b; b \prec c; b \prec d; d \prec e; d \prec f; d \prec g$.

Proposition 6. *Each branch of a computation tree* $\textsc{Sem}(T)$ *encodes a distinct linear extension of the tree-poset represented by T.*

Proof. At the i-th step of the construction of $\textsc{Sem}(T)$, the children of the root node α_i can either be a child in T, or a sibling, hence a relation compatible with the tree-poset encoded by T. Moreover, all the nodes are consumed by the process, it is the trivial to show that a linear extension of the order is produced. □

Proposition 7. $\textsc{Sem}(T)$ *encodes all the possible linear extensions of T as a tree-poset.*

Proof. We proceed by induction on n, the number of nodes of T. We have to show that at step k ($1 \leqslant k \leqslant n$), $\textsc{Sem}(T)$ encodes all the prefixes of linear extensions of T of length

k. For $k = 1$ there is only the root node α_1 and all the linear extensions of T must start with this node, hence satisfying the property. Now if we suppose that at step $k-1$ the construction encodes all the linear extensions ℓ_{k-1} of length $k-1$ then all the possible direct successors of α_k in a linear extension are trivially a minimum of ℓ_{k-1}. By prefixing with α_k we thus obtain ℓ_k as desired. □

Corollary 8. *ρ is a run of T if, and only if, it is a linear extension of the tree-poset encoded by T.*

Corollary 9. *Let T be a syntax tree and $b = \alpha_1, \ldots, \alpha_n$ a branch (from the root to a leaf) of the semantic tree* $\textsc{Sem}(T)$*. The branch b is a complete traversal of T.*

Interestingly, the tree T can be recovered by only considering a single branch of its computation tree, which requires some technical tools that we now define.

Definition 10. *Let T be a syntax tree and $b = \alpha_1, \ldots, \alpha_n$ a branch of the semantic tree* $\textsc{Sem}(T)$*. We define the sequences:*
$\xi_b = (\alpha_1, N_1), \ldots, (\alpha_n, N_n)$ and $\zeta_b = (\alpha_1, c_1), \ldots, (\alpha_n, c_n)$ with:

$$N_n = 0 \text{ and } \forall i,\ 1 \leqslant i < n, N_i \text{ is the number of children of node } \alpha_i \text{ in } b;$$
$$c_1 = 1 \text{ and } \forall i,\ 1 < i \leqslant n, c_i \text{ is the position of } \alpha_i \text{ below } \alpha_{i-1} \text{ in } b.$$

For example, consider the branch $b_{23} = a, b, d, f, g, c, e$ of the computation tree of Figure 2. The corresponding sequences are:

$$\left[\begin{array}{l} \xi_{b_{23}} = (a,1),(b,2),(d,4),(f,3),(g,2),(c,1),(e,0) \\ \zeta_{b_{23}} = (a,1),(b,1),(d,2),(f,3),(g,3),(c,1),(e,1) \end{array}\right.$$

The node d has 4 children (c, e, f, g) in branch b_{23}, and to reach f we select the 3rd child. From such a sequence it is easy to recover the arities of the nodes in the syntax tree T.

Proposition 11. *Let T be a syntax tree and $\xi_b = (\alpha_1, N_1), \ldots, (\alpha_n, N_n)$ a branch sequence of* $\textsc{Sem}(T)$*. Then the number of children of α_i in T is $\mathsf{arity}(\alpha_i)$ with*

$$\mathsf{arity}(\alpha_1) = N_1 \text{ and } \forall i,\ 1 < i \leqslant n,\ \mathsf{arity}(\alpha_i) = N_i - N_{i-1} + 1.$$

Proof. In a branch sequence ξ_b the passage from step (α_{i-1}, N_{i-1}) to (α_i, N_i) corresponds to the c_i-contraction in the construction of $\textsc{Sem}(T)$ at node α_{i-1}. An obvious fact is that N_i is the number of children of α_i after the contraction (hence in the semantic tree). This also contains the $N_{i-1} - 1$ children of α_{i-1} inherited by the contraction process (i.e. all its children except α_i itself). □

Now, we can distinguish the first branch of a computation tree (e.g. b_1 in Figure 2) as the preorder traversal of the syntax tree.

Proposition 12. *The leftmost branch b_1 of a computation tree* $\textsc{Sem}(T)$ *corresponds to the preorder traversal of T.*

Proof. In the sequence $\zeta_{b_1} = (\alpha_1, c_1), \ldots (\alpha_n, c_n)$ we have $c_i = 1$ (for all i), hence we only consider 1-contractions, i.e. contracting the leftmost child in the construction $\textsc{Sem}(T)$. If instead of attaching all the child subtrees but only the first (i.e. only $\textsc{Sem}(T \lhd 1)$ instead of $\textsc{Sem}(T \lhd 1), \ldots, \textsc{Sem}(T \lhd r)$) then Definition 4 trivially becomes a definition of the preorder traversal of T. □

Corollary 13. *A syntax tree T can be recovered from the leftmost branch b_1 of $\textsc{Sem}(T)$.*

Proof. Based on Proposition 11 we can obtain the successive arities of the leftmost b_1, which according to Proposition 12 corresponds to the preorder traversal of T. This gives us the *preorder degree sequence* of T, which is isomorphic to T itself, cf. [FS09, p. 74]. □

Interestingly, the original tree can be in fact recovered by any branch of the semantic tree, provided a good witness to its extreme level of redundancy.

Proposition 14. *A syntax tree T can be recovered from any branch b_k of $\textsc{Sem}(T)$.*

Proof. Let $\xi_{b_k} = (\alpha_1, N_1), \ldots, (\alpha_n, N_n)$ and $\xi_{b_k} = (\alpha_1, c_1), \ldots, (\alpha_n, c_n)$ be the sequences associated to b_k. By Proposition 11 we obtain the arities $\mathsf{arity}(\alpha_1), \ldots, \mathsf{arity}(\alpha_n)$. The next step is to recover the permutation of the α_i's corresponding to the preorder traversal. First let us introduce two notations. An empty sequence is denoted by $\langle\rangle$ and a sequence ζ with its first element e singled out is denoted by $e :: \zeta$. Then, we proceed recursively according to the following definitions:

$$\mathsf{next}((\alpha, c) :: \zeta) = \begin{cases} ((\alpha, c), \zeta) \text{ if } c = 1 \\ ((\beta, d), (\alpha, c - 1 + \mathsf{arity}(\beta)) :: \zeta') \\ \quad \text{with } ((\beta, d), \zeta') = \mathsf{next}(\zeta) \text{ otherwise} \end{cases}$$

$$\mathsf{prefix}(\zeta) = \begin{cases} \langle\rangle \text{ if } \zeta = \langle\rangle \\ \nu :: \mathsf{prefix}(\zeta') \text{ with } (\nu, \zeta') = \mathsf{next}(\zeta) \text{ otherwise} \end{cases}$$

The role of the function next is to fetch in a sequence corresponding to a branch suffix in the computation tree, the element that would be the next element in the preorder traversal. The function also performs the permutation that allows this next element to bubble-up until the head of the sequence. The function prefix now repeats this process, hence generating the preorder traversal as expected. The central element of this constructive demonstration is as follows. We consider $(\alpha, c) :: \zeta$ is a non-empty sequence of a given suffix of a given branch b in $\textsc{Sem}(T)$. This yields a pair $((\beta, d), \zeta'')$ and we want to prove that, first $d = 1$ which means it corresponds to a 1-contraction in the elaboration of the semantic tree. Moreover, the sequence ζ'' must correspond to a valid branch suffix b' in $\textsc{Sem}(T)$, and moreover a branch located to the left of b in the semantic tree. First, if $c = 1$ then by definition of next we have: $\beta = \alpha$ and $d = 1$, and the suffix ζ is left unchanged thus $b' = b$ is a correct branch suffix. In the case $c > 1$ we proceed by induction. The next element is (β, d) with $\beta \neq \alpha$ and the updated suffix is $(\alpha, c - 1 + \mathsf{arity}(\beta) :: \zeta')$. An important information is that α cannot be a descendant of β because α preceeded β in the original branch suffix b. In the original sequence, we were in fact performing a c-contraction for node α. But now, operating the permutation $\alpha \leftrightarrow \beta$ corresponds to enforcing a 1-contraction for β. Consequently, the contraction for α must take place after

the $\mathsf{arity}(\beta)$ chidren of β to update the resulting sequence so that the b' remains a correct branch suffix. By hypothesis of induction we also know that $d = 1$ (because the element β is simply propagated) and the remaining sequence ζ' corresponds to a correct suffix.

Now, the prefix function applies recursively the next function, i.e. prefixing the sequence by the next element in the preorder traversal, hence $\mathsf{prefix}(\zeta_{b_k})$ is exactly the sequence corresponding to the leftmost branch b_1 (i.e. ζ_{b_1}).

□

3 Enumeration of concurrent runs

Our quantitative study begins by measuring the number of concurrent runs of a process encoded as a syntax tree T. This measure in fact corresponds to the number of leaves – and thus the width – of the computation tree $\textsc{Sem}(T)$. Given the exponential nature of the merge operator, measuring efficiently the *dimensions* of the concurrent systems under study is of a great practical interest. In a second step, we quantify precisely the exponential growth of the computation trees, which provides a refined interpretation of the so-called combinatorial explosion phenomenon. Finally, we study the impact of characterizing commutativity for the merge operator. As a particularly notable fact, this section reveals a deep connection between increasingly labelled structures and concurrency theory.

3.1 An isomorphism with increasing trees

The connection between (syntactic) processes and tree-posets can be extended to *increasing trees*. Indeed, as observed e.g. in [KMPW14], the set of linear extensions of size n tree-posets are in one-to-one correspondence with the set of *increasing trees* of size n [BFS92, Drm09].

Definition 15. *An* ***increasing tree*** *is a labelled plane rooted tree such that the set of labels is* $\{1, 2, \ldots, \mathrm{size}(T)\}$ *and the sequence of labels along any branch starting at the root is increasing.*

For example, to label the syntax tree of Figure 1, a must take the label 1, b thus takes the label 2. Then the label of c must belong to $\{3, 4, 5, 6\}$, which then induces constraints on the other node labellings. Finally, for our example, only 30 labelled trees are increasing trees among the $7! = 5040$ possible unconstrained labellings.

The class of increasing trees satisfies the following specification (using the classical boxed product $^{\square}\star$ see [FS09, p. 139] for details):

$$\mathcal{G} = \mathcal{Z}^{\square} \star \textsc{Seq}(\mathcal{G}).$$

It is easy to obtain the coefficients of the associated *exponential* generating function $G(z)$ (e.g. from [BFS92]). In fact, it satisfies a classical first order differential equation. Once the differential equation is solved, the extraction of the coefficients is direct.

Fact 16. *The number of increasing trees of size n is*

$$n! \cdot [z^n]G(z) = 1 \cdot 3 \cdots (2n-3) = \frac{(2n-2)!}{2^{n-1}(n-1)!}.$$

From this fact, we obtain our first significant measure.

Theorem 17. *The mean number of concurrent runs induced by syntax trees of size n is*

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

Proof. This result is obtained from Facts 16 and 2. By dividing the number of increasing trees of size n by the number of syntax trees of size n (i.e. the Catalan number C_n) we get the stated value for $\bar{W}_n$. The asymptotic relation is derived from Stirling's formula [FS09, p. 37]. □

A further information that will prove particularly useful is the number of increasing labellings for a given tree. This can be obtained by the famous *hook-length* formula [Knu98, p. 60]. The relation between the hook-length formula and increasing trees belongs to the folklore and is in fact an exercise of [Knu98, p. 67].

Fact 18 (Hook-length formula)**.** *The number ℓ_T of increasing trees induced by a plane rooted tree structure T is*

$$\ell_T = \frac{|T|!}{\prod_{S \text{ subtree of } T} |S|},$$

where $|\cdot|$ corresponds to the tree size measure.

Corollary 19. *The number of concurrent runs induced by a syntax tree T is the number ℓ_T.*

We remark that the hook-length gives us "for free" a direct algorithm to compute the number of linear extensions of a tree-poset in linear time (i.e. the number of arithmetic operations is linear in the size of the tree). This is clearly an improvement if compared to related algorithms, e.g. [Atk90]. In Section 6 we discuss a slightly more general and more efficient algorithm that proves quite useful.

3.2 Analysis of growth

To analyse quantitatively the growth between the processes and their behaviours, we measure the average number of concurrent runs induced by large syntax trees of size n. The arithmetic mean given in Theorem 17 is the usual way to measure in average. Nevertheless, a small number of compact syntax trees (in the sense that internal nodes have a large arity: e.g. a root followed by $(n-1)$ children) produces a huge number of runs and unbalances the mean. So, a natural way to avoid such a bias is to compute the geometric mean which is less sensitive to extremal data. This subsection is devoted to prove the following theorem about the geometric mean number of concurrent runs.

Theorem 20. *The geometric mean number, denoted by $\bar{\Gamma}_n$, of concurrent runs induced by process trees of size n satisfies:*

$$\bar{\Gamma}_n = \prod_{k=2}^{n-1} k^{1-\frac{n-k+1}{2}\frac{C_k C_{n-k+1}}{C_n}} \sim_{n\to\infty} \sqrt{2\pi}\frac{e^{\sqrt{\pi n}+L(1/4)}}{n}\left(\frac{n}{e^{1+2L(1/4)}}\right)^n,$$

where $L(1/4) = \sum_{n>1} \ln n \cdot C_n \cdot 4^{-n} \approx 0.57889$.

Although this geometric mean is exponentially smaller than the arithmetic mean and both means are far smaller than the upper bound (worst case) $(n-1)!$, we can point here that the number of runs of a typical large syntax tree explodes almost like the factorial function.

Proof. Let us recall a notation: for a node ν in a tree T, we denote by $T(\nu)$ the subtree which root is ν. First we need to obtain a recurrence formula based on the hook-length formula. Let us give the following observation:

$$\prod_{S \text{ subtree of } T} |S| = |T| \cdot \prod_{\substack{\nu \text{ child of} \\ \text{the root of } T}} \left(\prod_{S \text{ subtree of } T(\nu)} |S| \right).$$

Now, from the Fact 18 and a proof by induction, we deduce another equation for ℓ_T:

$$\ell_T = (|T|-1)! \left(\prod_{\substack{\nu \text{ child of} \\ \text{the root of } T}} \frac{\ell_{T(\nu)}}{|T(\nu)|!} \right).$$

Since the geometric mean of the numbers ℓ_T is related to the arithmetic mean of the numbers $\ln(\ell_T)$, we introduce the sequence $h_T = \ln(\ell_T/|T|!)$ and its generating function $H(z) = \sum_T h_T z^{|T|}$. Using the latter recursive formula for ℓ_T, we deduce

$$H(z) = -L(z) + \sum_T \sum_{\substack{\nu \text{ child of} \\ \text{the root of } T}} h_{T(\nu)} z^{|T|},$$

where $L(z) = \sum_T \ln|T| z^{|T|} = \sum_{n\geqslant 1} C_n \ln(n) z^n$ and $C(z) = \sum_n C_n z^n$ enumerates syntax trees.
By partitioning the trees according to their root arities and the child-subtrees attached at their roots, we get

$$H(z) = -L(z) + \sum_{r\geqslant 1} \sum_{R_1,\ldots,R_r} \left(\sum_{i=1}^{r} h_{R_i} \right) z^{1+\sum_{j=1}^r |R_j|},$$

where the trees R_i's are the child-subtrees attached at the root of each syntax tree. Let us now focus on the second term of $H(z)$. Set

$$S_r = \sum_{R_1,\ldots,R_r} \left(\sum_{i=1}^{r} h_{R_i} \right) z^{1+\sum_{j=1}^r |R_j|}.$$

We first partition the sums over the trees R_i's according to their sizes (suppose $r > 1$, otherwise the computation is obvious):

$$\begin{aligned}
S_r &= \sum_{s_1,\ldots,s_r} \sum_{\substack{R_1,\ldots,R_r \\ |R_1|=s_1,\ldots,|R_r|=s_r}} \left(\sum_{i=1}^{r} h_{R_i}\right) z^{1+\sum_{j=1}^{r} s_j} \\
&= z \cdot \sum_{s_1,\ldots,s_r} \left(\sum_{R_1:|R_1|=s_1} h_{R_1} z^{s_1} \cdot \sum_{\substack{R_2,\ldots,R_r \\ |R_2|=s_2,\ldots,|R_r|=s_r}} z^{\sum_{j=2}^{r} s_j} + \cdots + \right. \\
&\qquad \left. \sum_{R_r:|R_r|=s_r} h_{R_r} z^{s_r} \cdot \sum_{\substack{R_1,\ldots,R_{r-1} \\ |R_1|=s_1,\ldots,|R_{r-1}|=s_{r-1}}} z^{\sum_{j=1}^{r-1} s_j} \right) \\
&= z \cdot \sum_{s_1} \sum_{R_1:|R_1|=s_1} h_{R_1} z^{s_1} \cdot C^{r-1}(z) + \cdots + z \cdot \sum_{s_r} \sum_{R_r:|R_r|=s_r} h_{R_r} z^{s_r} \cdot C^{r-1}(z) \\
&= zH(z) \cdot rC^{r-1}(z).
\end{aligned}$$

Thus, we get:

$$H(z) = -L(z) + z \cdot H(z) \sum_{r \geqslant 1} rC^{r-1}(z).$$

By differentiating the formal equation $\sum_{r \geqslant 0} C^r(z) = 1/(1 - C(z))$, we get the next equation: $\sum_{r \geqslant 1} rC^{r-1}(z) = (1 - C(z))^{-2}$, and thus,

$$H(z) = \frac{-L(z)}{2} \left(1 + \frac{1}{\sqrt{1-4z}}\right).$$

We can compute the geometric mean

$$\bar{\Gamma}_n = \left(\prod_{\substack{T \\ |T|=n}} \ell_T \right)^{1/C_n} = \left(n!^{C_n} \prod_{\substack{T \\ |T|=n}} \frac{\ell_T}{|T|!} \right)^{1/C_n} = n! \exp\left(\frac{1}{C_n} \cdot [z^n] H(z) \right).$$

It remains to extract the n-th coefficient of $H(z)$.

$$\begin{aligned}
[z^n]H(z) &= -\frac{1}{2} \sum_{k=1}^{n} [z^k] L(z) \cdot [z^{n-k}] \left(1 + \frac{1}{\sqrt{1-4z}}\right) \\
&= -\frac{1}{2} \sum_{k=1}^{n} C_k \ln(k) \cdot \left(1_{\{k=n\}} + (n-k+1) C_{n-k+1}\right),
\end{aligned}$$

where $1_{\{k=n\}}$ is 1 when $k=n$ and 0 otherwise. Thus,

$$\begin{aligned}\bar{\Gamma}_n &= n!\exp\left(-\ln(n)-\frac{1}{2C_n}\cdot\sum_{k=2}^{n-1}C_k\ln(k)\cdot(n-k+1)C_{n-k+1}\right)\\ &= (n-1)!\cdot\prod_{k=2}^{n-1}k^{-\frac{n-k+1}{2}\frac{C_kC_{n-k+1}}{C_n}}=\prod_{k=2}^{n-1}k^{1-\frac{n-k+1}{2}\frac{C_kC_{n-k+1}}{C_n}}.\end{aligned}$$

To complete the proof, it remains to obtain the asymptotic approximation for $\bar{\Gamma}_n$.

First, let us approximate the coefficients H_n. The generating function $L(z)$ has its dominant singularity $\rho=1/4$. In fact it cannot be larger than the one of the Catalan generating function, and the $L(1/4)$ is finite. Furthermore, $L(z)$ is Δ-analytic. Let us introduce the generating function $A(z)$ that will be used to approximate $L(z)$ near the dominant singularity.

$$A(z)=L(1/4)-\frac{\sqrt{1-4z}}{2}\ln\left(\frac{1}{1-4z}\right)+\frac{(\gamma+2\ln 2-2)\sqrt{1-4z}}{2},$$

where γ is Euler's constant. By using the two first terms in the development of Catalan numbers (see Fact 2) and asymptotic formula for the coefficients of $\sqrt{1-z}\ln(1-z)^{-1}$ (see e.g. [FS09, p. 388]), we obtain

$$\begin{aligned}A_n &=_{n\to\infty} -\frac{4^n}{2}\left(-\frac{1}{\sqrt{\pi n^3}}\left(\frac{1}{2}\ln n+\frac{\gamma+2\ln 2-2}{2}\right)+O\left(n^{-\frac{5}{2}}\ln n\right)\right)\\ &\quad -(\gamma+2\ln 2-2)\frac{4^{n-1}}{\sqrt{\pi n^3}}\left(1+O\left(\frac{1}{n}\right)\right)\\ &=_{n\to\infty} \frac{4^{n-1}\ln n}{\sqrt{\pi n^3}}+O\left(\frac{4^n\ln n}{\sqrt{n^5}}\right).\end{aligned}$$

Since, $L_n=\ln(n)C_n$, thus there exists $\bar{\alpha}$, $\bar{\beta}$ and n_0, such that, for all $n\geqslant n_0$,

$$\frac{4^{n-1}\ln n}{\sqrt{\pi n^3}}\cdot\left(1+\frac{\bar{\alpha}}{n}\right)\leqslant L_n\leqslant\frac{4^{n-1}\ln n}{\sqrt{\pi n^3}}\cdot\left(1+\frac{\bar{\beta}}{n}\right).$$

For all $i\in\{2,\ldots,n_0-1\}$, let us define $\alpha_i=i\left(\frac{\sqrt{\pi i^3}L_i}{2\ln(i)4^{i-1}}-1\right)$ and $\beta_i=i\left(\frac{2\sqrt{\pi i^3}L_i}{\ln(i)4^{i-1}}-1\right)$. Thus we denote by α the minimal value among $\bar{\alpha}$ and the set of α_i's and β, the maximal value among $\bar{\beta}$ and the set of β_i's. Finally we conclude, for all $n\geqslant 1$

$$\frac{4^{n-1}\ln n}{\sqrt{\pi n^3}}\cdot\left(1+\frac{\alpha}{n}\right)\leqslant L_n\leqslant\frac{4^{n-1}\ln n}{\sqrt{\pi n^3}}\cdot\left(1+\frac{\beta}{n}\right).$$

Thus, by summing over n and by using Equation (25) in [FS09, p. 387], we conclude

$$L(z)=_{z\to 1/4}A(z)+O\left((1-4z)^{3/2}\ln\left(\frac{1}{1-4z}\right)\right).$$

Consequently, by using Theorem VI [FS09, p. 394], we directly derive

$$H_n = -\frac{1}{2}\left(L(1/4)\frac{4^n}{\sqrt{\pi n}} - \frac{4^n}{2n} + \frac{4^{n-1}\ln n}{\sqrt{\pi n^3}} - L(1/4)\frac{4^{n-1}}{2\sqrt{\pi n^3}} + O\left(\frac{4^n \ln n}{n^{5/2}}\right)\right).$$

Thus,

$$\frac{H_n}{C_n} = H_n \frac{\sqrt{\pi n^3}}{4^{n-1}}\left(1 - \frac{3}{8n} + O\left(\frac{1}{n^2}\right)\right) = -L(1/4)\cdot 2n + \sqrt{\pi n} - \frac{\ln n}{2} + L(1/4) + O\left(\frac{1}{n^2}\right).$$

Finally with Stirling formula, we conclude

$$\bar{\Gamma}_n = (n-1)!\cdot \exp\left(\frac{H_n}{C_n}\right) \sim_{n\to\infty} \sqrt{2\pi}\frac{e^{\sqrt{\pi n}+L(1/4)}}{n}\left(\frac{n}{e^{1+2L(1/4)}}\right)^n.$$

Concerning the constant $L(1/4)$, it has been obtained by using the Shanks transformation [Sha55] that consists in a nonlinear acceleration method for computing the limit of a series. More precisely, we have used the first $500,000$ terms of the series $L(1/4)$ and then we applied 10 iterations of the Shanks transformation in order to obtain the stated approximation for $L(1/4)$. □

3.3 The case of non-plane trees

In classical concurrency theory, the pure merge operator often comes with commutativity laws, e.g. $P \parallel Q \equiv Q \parallel P$. From a combinatorial point of view, the idea is to consider the syntax and computation trees as non-plane (or unordered) rooted trees.

The numbers of non-plane rooted trees, i.e. the analogue of the Catalan numbers, are well known thanks to the work of Pólya [Pó37]. They are directly derived from the next specification studied in [FS09, p. 475–477].

Fact 21. *The specification of unlabelled non-plane rooted trees is* $\mathcal{T} = \mathcal{Z} \times \textsc{MSet}\,\mathcal{T}$. *The number* T_n *of such trees of size* n *is:*

$$T_n \sim_{n\to\infty} \frac{\mu}{2\sqrt{\pi n^3}}\eta^{-n},$$

where $\eta \in [\frac{1}{4}, \frac{1}{e}]$ *and, approximately,* $\eta \approx 0.3383218$ *and* $\mu \approx 1.559490$.

Compared to rooted plane trees, no known closed form exists to characterize the symmetries involved in the non-plane case. One must indeed work with rather complex approximations. The increasing variant of non-plane rooted trees has been studied, too, in the model of *increasing Cayley trees*, or *recursive trees*, introduced by Meir and Moon [MM78], and studied in details in [FS09, p. 526–527]:

Fact 22. *The specification of increasing non-plane rooted trees is* $\mathcal{I} = \mathcal{Z}^{\square} \star \textsc{Set}(\mathcal{I})$. *The number* I_n *of such trees of size* n *is* $I_n = (n-1)!$.

Theorem 23. *The mean number of concurrent runs built on non-plane syntax trees of size n is*

$$\bar{V}_n \sim_{n\to\infty} \frac{2\sqrt{2}\pi n}{\mu}\left(\frac{n\eta}{e}\right)^n,$$

where η and μ are the constants introduced in Fact 21.

Of course, we obtain different approximations for the plane vs. non-plane case. The ratio $\bar{W}_n/\bar{V}_n$ is equivalent to $\gamma(2\eta)^{-n}/\sqrt{\pi n}$, which means that although the exponential growths are not of the same order. However, the two asymptotic formulas follow a same universal *shape*. This comparison between plane and non-plane combinatorial structures is a recurring theme in combinatorics. It has often been pointed out that in most cases the asymptotic formulas look very similar. Citing Flajolet and Sedgewick (cf. [FS09, p. 71–72]):

> "[...] (some) universal law governs the singularities of simple tree generating functions, either plane or non-plane [...]".

Our study echoes quite faithfully such an intuition.

4 Typical shape of process behaviours

Our goal in this section is to provide a more refined view of the process behaviours by studying the typical shape of the computation trees. This study puts into light a new – and, we think, interesting – combinatorial class: the model of *increasing admissible cuts* (of plane rooted trees). In the first part we recall the notion of admissible cuts and define their increasing variant. This naturally leads to a generalization of the hook-length formula of Fact 18 that enables the decomposition of a computation tree by levels. Based on this construction, we study experimentally the level decomposition of computation trees corresponding to syntax trees of the given size 40 (which yields computation trees with more than 10^{28} nodes). Finally, we discuss the mean number of nodes by level, which is obtained by enumerating increasing admissible cuts. This provides a fairly precise characterization for the typical shape of process behaviours.

4.1 Increasing admissible cuts

The notion of an *admissible cut* has been already studied in algebraic combinatorics, see for example [CK98]. The novelty here is the consideration of the increasingly labelled variant.

Definition 24. *Let T be a syntax tree of size n. An **admissible cut** of T of size $k = n-i$ $(0 \leqslant i < n)$ is a tree obtained by starting with T and removing recursively i leaves from it. An **increasing admissible cut** of T of size k is an admissible cut of size k of T that is increasingly labelled.*

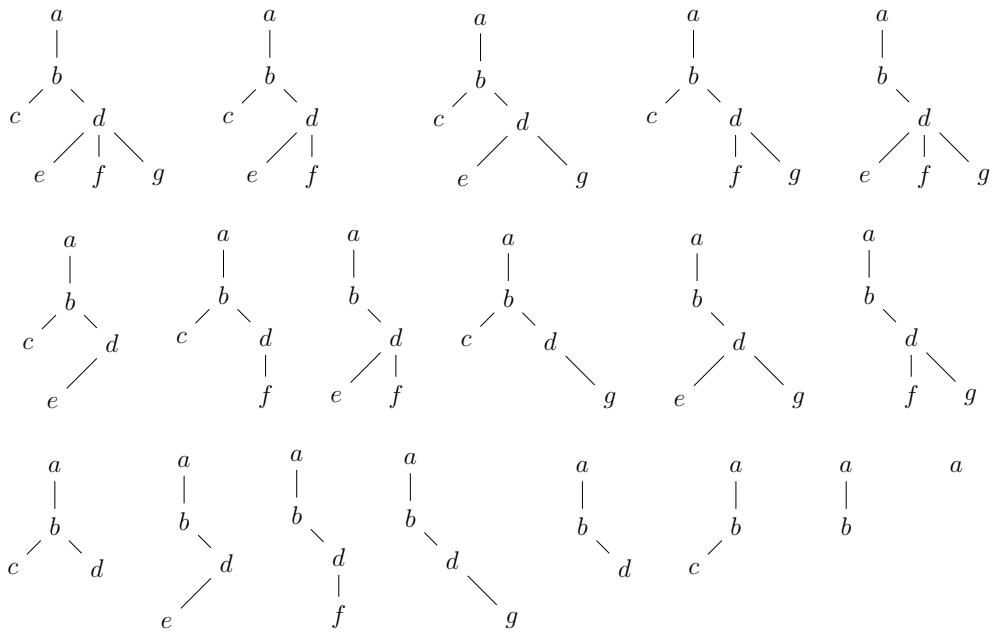


Figure 4: All admissible cuts of the syntax tree of Figure 1.

Figure 4 depicts all admissible cuts for the syntax tree T of Figure 1. We remark that the tree T is itself an admissible cut of T.

To establish a link with increasing admissible cuts, we first make a simple albeit important observation.

Proposition 25. *Let T be a syntax tree of size n. Any run prefix of length k ($1 \leqslant k \leqslant n$) in $\textsc{Sem}(T)$ is uniquely encoded by an admissible cut of T whose size is k.*

Proof. We proceed by induction on k. For $k = 1$, there is a single run prefix of length $k = 1$ and the corresponding admissible cut is the root. Now suppose that the property holds for run prefixes of length k, $1 < k < n$, let us show that it also holds for run prefixes of length $k + 1$. By induction hypothesis, any run prefix σ_k of length k is encoded by exactly one admissible cut of size k. Let us denote by $C(\sigma_k)$ this admissible cut. Now, any prefix σ_{k+1} of length $k + 1$ is obtained by appending an action α to a prefix σ_k of length k. For σ_{k+1} to be a valid prefix, α must correspond to a node in T that is a child of one of the nodes of $C(\sigma_k)$. Thus we obtain a unique $C(\sigma_{k+1})$ as $C(\sigma_k)$ completed by a single leaf α. □

For example the run prefixes $\langle a, b, c, d\rangle$ and $\langle a, b, d, e\rangle$ are encoded by the first two admissible cuts of size 4 corresponding to the first two cuts of the third line of Figure 4.

This result leads to a fundamental connection with increasing admissible cuts.

Proposition 26. *Let T be a syntax tree of size n. The number of run prefixes of length k ($1 \leqslant k \leqslant n$) in $\textsc{Sem}(T)$ is the number of increasing labellings of the admissible cuts of T of size k.*

Proof. This is obtained by a direct order-theoretic argument. Each admissible cut is a tree-poset and thus the number of runs it encodes is the number of its linear extensions. □

For example, there are four admissible cuts of size 4 in Figure 4 (the leftmost cuts of the third line). The first one admits two increasing labellings and the other ones have a single labelling. This gives $2+1+1+1=5$ run prefixes of length 4 for the syntax tree T of Figure 1. Now, we observe that this is also the number of nodes at level 3 in the corresponding computation tree. And this of course can be generalized: the number of run prefixes of length k corresponds to the number of nodes at level $k-1$ in the computation tree.

From this we can characterize precisely the number of nodes by level based on a generalization of the hook-length formula.

Corollary 27. *Let T be a process tree of size n. The number of nodes at level $n-1-i$ $(0 \leqslant i < n)$ of* $\textsc{Sem}(T)$ *is*

$$n_T^i = \sum_{\substack{S \text{ admissible cut of } T \\ |S|=n-i}} \ell_S,$$

where ℓ_S is the hook-length formula applied to the admissible cut S (cf. Fact 18). Moreover, the total number of nodes of $\textsc{Sem}(T)$ *is*

$$n_T = \sum_{i=0}^{n-1} n_T^i = \sum_{S \text{ admissible cut of } T} \ell_S.$$

4.2 Level decomposition

Before deriving an exact formula for the mean number of nodes by level, we can take advantage of Corollary 27 to compute the shape of some typical computation trees.

4.2.1 Experimental study

Our experiments consist in generating uniformly at random some syntax trees (using our *arbogen tool*[3]) of size n for n not too small. Then we can compute n_T as defined above by first listing all the admissible cuts of T.

However we cannot take a syntax tree with a very large size n due to the following result.

Proposition 28. *The mean number $\bar{m}_n$ of admissible cuts of trees of size n satisfies*

$$\bar{M}_n \sim_{n\to\infty} \frac{4}{\sqrt{15}} \left(\frac{5}{4}\right)^{2n}.$$

[3] https://github.com/fredokun/arbogen

Proof. Let us denote by $M(z)$ the ordinary generating function enumerating the multiset of admissible cuts of all trees. More precisely, we get $M(z) = \sum_{n\in\mathbb{N}} M_n z^n$ where $M_n = \sum_{T;|T|=n} |\{S \text{ adm. cut of } T\}|$.
The specification of $\mathcal{M}$ is $\mathcal{Z} \times \textsc{Seq}(\mathcal{M} \cup \mathcal{C})$, where the tag $\mathcal{Z}$ marks the nodes of the tree carrying the admissible cut, and $\mathcal{C}$ is the class of syntax trees. In fact, an admissible cut is a root and a sequence of trees that are either admissible cuts or trees. In that last case, trees correspond to a branch of the original tree that has been entirely cut. Consequently, $M(z)$ satisfies the following equation that can be easily solved.

$$M(z) = \frac{z}{1 - M(z) - C(z)}, \quad \text{thus} \quad M(z) = \frac{1 + \sqrt{1-4z} - \sqrt{2 - 20z + 2\sqrt{1-4z}}}{4}.$$

The singularities of $M(z)$ are $1/4$ and $4/25$: the latter one is the dominant. The generating function is analytic in a Δ-domain around $4/25$ because of the square-root type of the dominant singularity. Since $M(z)$ is algebraic, it admits a Puiseux development (e.g. cf. [FS09, p. 498]) near its singularity $\frac{4}{25}$ of the following form:

$$M(z) =_{z\to\frac{4}{25}} -\frac{2}{5} - \frac{2}{\sqrt{15}}\sqrt{1 - \frac{25}{4}z} + \frac{2}{15}\left(1 - \frac{25}{4}z\right) + O\left(\left(1 - \frac{25}{4}z\right)^{\frac{3}{2}}\right).$$

Thus, by using transfer lemmas [FS09, p. 501], we get the asymptotic behaviour:

$$M_n \sim_{n\to\infty} \frac{1}{\sqrt{15\pi n^3}}\left(\frac{25}{4}\right)^n.$$

By dividing by the number $C_n$of syntax trees of size n (cf. Fact 2), we obtain the stated average value $\bar{M}_n$. □

Proposition 29. *The cumulative number of admissible cuts M_n satisfy the following P-recurrence.*

$$\begin{aligned}(-500n + 2000n^3)M_n + (120 - 220n - 1380n^2 - 920n^3)M_{n+1} \\ -(1488 + 1626n + 387n^2 - 21n^3)M_{n+2} + (1104 + 1088n + 351n^2 + 37n^3)M_{n+3} \\ -(168 + 146n + 42n^2 + 4n^3)M_{n+4} \;=\; 0,\end{aligned}$$

with $M_0 = 0, M_1 = 1, M_2 = 2$ and $M_3 = 7$.

This sequence is registered in OEIS as A007852 and enumerates the number of antichains in rooted trees [Kla97]. In fact the bijection between the admissible cuts and the antichains is direct, since each antichain corresponds to the set of leaves of an admissible cut. The previous Proposition 29 is already proved in [Kla97].

Proof. The algebraic generating function $M(z)$ is solution of the equation

$$z^2 - z \cdot M(z) + 3z \cdot M(z)^2 - M(z)^3 + M(z)^4 = 0.$$

Thus, by using the results of Comtet [Com64], we can derive a differential equation satisfied by the generating function $M(z)$

$$\begin{aligned} 30z - 12 + (-120z + 48) \cdot M(z) \\ +(1500z^3 + 240z^2 - 318z + 36) \cdot M'(z) \\ +(6000z^4 - 1380z^3 - 450z^2 + 129z - 6) \cdot M''(z) \\ +(2000z^5 - 920z^4 + 21z^3 + 37z^2 - 4z) \cdot M'''(z) \quad = \quad 0, \end{aligned}$$

with $M(0) = 0, = 1, M'(0) = 4$ and $M''(0) = 4$. And finally Stanley's result about holonomic functions (cf. [Sta80]) let derive the P-recurrence satisfied by the coefficients of $M(z)$. Note that the complete proof can be directly computed by using the package Gfun [SZ94] in Maple.

Another proof would be obtained by using the Guess and Prove method (extensively detailed in a more complex context in Section 5). □

The result, given in Proposition 28, states that the average number of admissible cuts of size n syntax trees is exponential in n. Thus, we must be particularly careful when computing the shape of a computation tree in practice using the generalization of the hook-length formula for increasing admissible cuts. For syntax trees of a size $n \leqslant 40$ we are able to compute the level decomposition within a couple of days using a fast computer[4]. This must be compared to the mean width of these trees: $\bar{W}^0_{40} > 1.48 \cdot 10^{36}$!

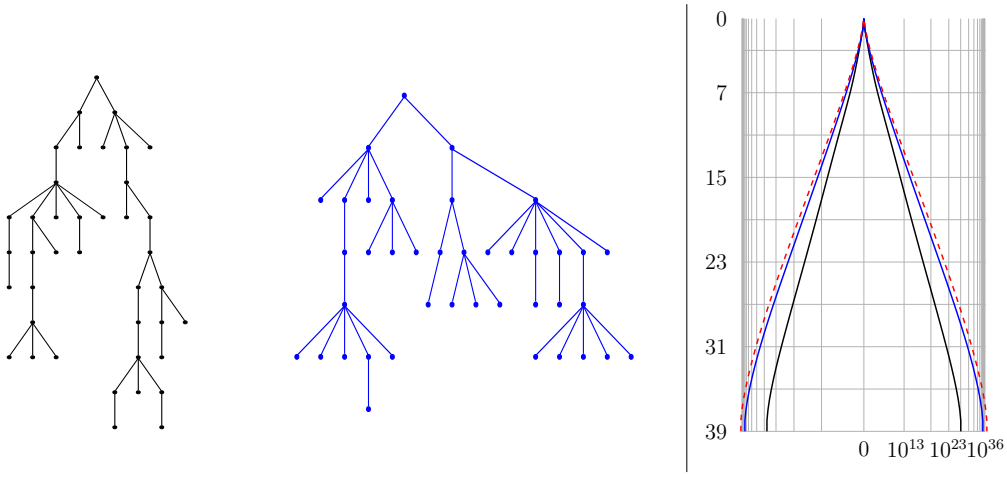


Figure 5: Two trees of size 40 and their computation tree profile behind the average profile.

Consider the two syntax trees, uniformly sampled among all trees of size 40, depicted on the left of Figure 5. The shapes of the corresponding computation trees are depicted on the right of the Figure. We use a logarithmic scale of the horizontal axis so that the

[4]The computer used for the experiment is a bi-Xeon 5420 machine with 8 cores running at 2.5Ghz each, equipped with 20GB of RAM and running linux.

exponential fringes become lines. The computation trees are of size larger than $8.74 \cdot 10^{28}$ for the one corresponding to the left process tree and $2.66 \cdot 10^{35}$ for the second one. These correspond to the two plain lines in the Figure. The dashed lines correspond to the exact computation of the mean as explained in the next section. We can see that it is almost reached by the shape of the second tree. To analyse this peculiarity, we have sampled more than fifty typical process trees of size 40 (with uniform probability among all trees of size 40). The results are fairly interesting. All the shapes that we computed follow the same kind of curve as the average. However, almost all process trees (49 among 50) have a computation tree size that is much smaller than the average ($\approx 4.06 \cdot 10^{36}$). Indeed, most of them have a size that belong to $[10^{28}, 10^{35}]$, and a single one has a size larger than the average (it is approximately twice as large).

This observation let us conjecture that only a very few special syntax trees accounts for the largest part of the average semantic tree size. These are undoubtedly process trees whose internal nodes have a large arity. The simplest one is the process tree with a single internal node. In the case of size 40, its semantic correspondence has size larger than $2.03 \cdot 10^{46}$. In fact, the combinatorial explosion in the worst case increases like a factorial function. Since the Catalan numbers (that count syntax trees) do not increase that quickly, the "worst" syntax trees (those whose computation tree sizes are the largest) do really influence the average measures.

4.2.2 Mean number of nodes by level

We may now describe one of the fundamental results of this paper: an exact formula for the mean number of nodes at each level of a computation tree.

Theorem 30. *The mean number of nodes at level $n-i-1$, for $i \in \{0, \ldots, n-1\}$ in a computation tree corresponding to a syntax tree of size n is:*

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

Proof. Let n, i be two integers such that $0 \leqslant i < n$. A direct consequence of Corollary 27

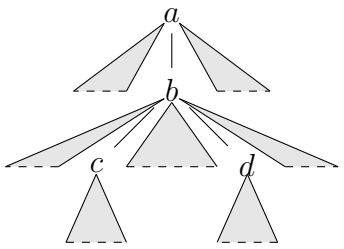


Figure 6: An admissible cut and the places where it can be enriched.

gives the cumulated number W_n^i of nodes at level $n-i-1$ in computation trees issued of syntax trees of size n to be equal to the sum on all increasingly labelled admissible cuts

of size $n-i$ induced by size n syntax trees. As in the previous section, let us denote by $\mathcal{G}$ the combinatorial class of increasing trees and thus G_{n-i} the number of increasing trees of size $n-i$. An admissible cut is obtained from a tree by pruning some of its subtrees. By the reverse process, i.e. by plugging sequences of trees to a fixed tree (that corresponds to an admissible cut), we obtain the set of trees which admit that admissible cut. In Figure 6, the fixed admissible cut is the tree with nodes a, b, c, d and the places where sequences of trees can be plugged are depicted by the grey triangles. For every node of arity η, exactly $\eta+1$ sequences of trees can be plugged near its children. So for a fixed admissible cut of size $n-i$, the number of places is obviously $2n-2i-1$. Thus, by using Definition 1 and Fact 16, we conclude:

$$W_n^i = [z^n] G_{n-i} z^{n-i} \left(\frac{1}{1-C(z)}\right)^{2n-2i-1} = \frac{(2n-2i-2)!}{2^{n-i-1}(n-i-1)!}[z^i]\left(\frac{C(z)}{z}\right)^{2n-2i-1}.$$

Results of [PS70, Part 3, Chapter 5] on powers of the Catalan generating function give:

$$[z^n]\left(\frac{C(z)}{z}\right)^k = \frac{k}{n}\binom{k+2n-1}{n-1}.$$

The former result is obtained by using Bürmann's form of Lagrange inversion. An analogous expression is given in [FS09, p. 66–68]. Thus,

$$W_n^i = \frac{(2n-2i-1)!}{i\cdot 2^{n-i-1}\cdot(n-i-1)!}\binom{2n-2}{i-1}.$$

By taking the average, i.e. by dividing by C_n, we obtain the stated value for $\bar{W}_n^i$. □

Given this result, we can complete the analysis of the shapes depicted in Figure 5. Let us first determine the limiting curve for the average shape of a computation tree. We renormalise the values $\bar{W}_n^i$ in the following way: $f(c,n) = \ln(\bar{W}_n^{\lfloor cn \rfloor})$. We thus obtain an asymptotic evaluation of $f(c,n)$ when n tends to infinity. When $\frac{1}{n} = o(c)$ and $1-c = o(\frac{1}{n})$, we have $f(c,n)$ equal to

$$(1-c)n\ln(n) + \left(c - 1 + \ln\left(\frac{(2-2\,c)^{1-c}}{c^c\,(2-c)^{2-c}}\right)\right)n + \ln\left(\frac{\sqrt{4-2c}}{\sqrt{c}}\right) + O\left(\frac{1}{c(1-c)n}\right).$$

In particular, on every compact $[a,b]$ such $0 < a < b < 1$, the function in $c \mapsto f(c,n)/(n\ln(n))$ tends uniformly to $(1-c)$ when n tends to infinity. Moreover, if we keep the second order terms, we then obtain a curve which is totally relevant with the Figure 5 near levels 0 and 40.

We can study the asymptotic behaviour of $\bar{W}_n^i$ and $\bar{W}_n^{n-i}$ for fixed constant i. A straightforward calculation shows that

$$\bar{W}_n^i \sim_{n\to\infty} \frac{n!}{2^{n-1}i!} \qquad \text{and} \qquad \bar{W}_n^{n-i} \sim_{n\to\infty} \frac{(2i-1)!}{2^{k-1}}.$$

Both give an interpretation to the inflexion of the curve near the boundary points.

5 Expected size of process behaviours

In this section we study the average size of computation trees (i.e. the mean total number of nodes). In a first part we provide a first approximation based on the Theorem 30 of the previous section. Then, we make a conjecture regarding the non-plane case, which proof requires a deeper study that goes beyond the scope of this paper. Finally, we characterize the average size in a more precise way, through a P-recurrence that is obtained by various ways. We describe three different techniques of analytic combinatorics to obtain this recurrence relation: each technique has its pros and cons, as will be discussed below. Finally, we reach our goal of providing the precise asymptotic behaviour of the size of computation trees.

5.1 First approximation of mean size

Our initial approximation of the mean size of the computation trees is based on the level decomposition of Section 4, where we have given an exact formula for the mean number of nodes $\bar{W}_n^i$ at level $(n-i-1)$ for computation trees corresponding to syntax trees of size n. We first give, as a technical lemma, inequalities about $\bar{W}_n^i$.

Lemma 31.

$$1 \leqslant \frac{2^{n-1}\ i!}{n!} \cdot \bar{W}_n^i \leqslant \frac{1}{1-\frac{i^2}{2n}}.$$

Proof. We first deal with some normalized expression of Theorem 30:

$$\frac{2^{n-1}\ i!}{n!} \cdot \bar{W}_n^i = 2^i \cdot \frac{(2n-2i-1)!}{(2n-i-1)!} \cdot \frac{(n-1)!}{(n-i-1)!} = \frac{\prod_{j=0}^{i-1}(n-i+j)}{\prod_{j=0}^{i-1}(n-i+\frac{j}{2})}.$$

Obviously, we get the stated lower bound 1. Let us go on with some simplifications. The $\lceil i/2 \rceil$ first factors of the numerator can be simplified with those of the denominator when j is even:

$$\frac{2^{n-1}\ i!}{n!} \cdot \bar{W}_n^i = \frac{\prod_{j=0}^{\lfloor i/2 \rfloor -1}(n-\lfloor \frac{i}{2} \rfloor + j)}{\prod_{j=0}^{\lfloor i/2 \rfloor -1}(n-i+\frac{1}{2}+j)} = \frac{\prod_{j=0}^{\lfloor i/2 \rfloor -1}(1-\frac{\lfloor i/2 \rfloor -j}{n})}{\prod_{j=0}^{\lfloor i/2 \rfloor -1}(1-\frac{i-1/2-j}{n})}.$$

But, the numerator is smaller than 1 and the denominator satisfies:

$$\prod_{j=0}^{\lfloor i/2 \rfloor -1}\left(1-\frac{i-1/2-j}{n}\right) \geqslant 1 - \sum_{j=0}^{\lfloor i/2 \rfloor -1}\frac{i-1/2-j}{n} \geqslant 1-\frac{i^2}{2n}.$$

So we obtain the stated upper bound. □

Theorem 32. *The mean size $\bar{S}_n$ of a computation tree induced by a process tree of size n admits the following asymptotic formula*

$$\bar{S}_n = \sum_{i=0}^{n-1} \bar{W}_n^i \sim_{n\to\infty} e \cdot \frac{n!}{2^{n-1}}.$$

Proof. Using Lemma 31 and taking a large enough n, we get:

$$\sum_{i=0}^{n-1} \frac{n!}{2^{n-1}\ i!} \leqslant \bar{S}_n \leqslant \sum_{i=0}^{n-1} \frac{n!}{2^{n-1}\ i!} \cdot \frac{1}{1-\frac{i^2}{2n}} \leqslant \sum_{i=0}^{n-1} \frac{n!}{2^{n-1}\ i!} \cdot \left(1+\frac{i^2}{n}\right).$$

First let us take the lower bound into account. Using an upper bound of the tail of the series associated to the exponential function, we obtain the following inequalities

$$\frac{n!}{2^{n-1}} \left(e - \frac{e}{n!}\right) \leqslant \sum_{i=0}^{n-1} \frac{n!}{2^{n-1}\ i!} \leqslant e \cdot \frac{n!}{2^{n-1}}.$$

Both bounds tend to $e \cdot n!/2^{n-1}$. It remains to prove that $n^{-1} \cdot \sum_{i=0}^{n-1} i^2/i! = o(1)$ to complete the proof.

$$\sum_{i=0}^{n-1} \frac{i^2}{i!} = \sum_{i=0}^{n-2} \frac{i+1}{i!} = \sum_{i=0}^{n-3} \frac{1}{i!} + \sum_{i=0}^{n-2} \frac{1}{i!} \to_{n\to\infty} 2e.$$

Finally, we get the asymptotic behaviour of the mean size of the computation trees induced by size n syntax trees

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

□

Corollary 33. *Let f be a function in n that tends to infinity with n. Let a_n be the average number of nodes of the computation trees induced by all the syntax tree of size n and ℓ_n the average number of nodes belonging to the $f(n)$ last levels of the computation trees. Then, ℓ_n/a_n tends to 1 when n tends to infinity.*

The proof is analogous as the previous one using bounds on the tail of the series from the exponential function. The unique constraint for f is that it tends to infinity, but it can grow as slow as we want. Thus, for example, asymptotically almost all nodes of the average computation tree belong to the $\log(\ldots(\log n)\ldots)$ last levels.

5.2 The case of non-plane trees

In order to compute the average size in the context of non-plane trees, we need one more result that is the analogue of powers of the Catalan generating function (see proof of Theorem 30). Here, in the case of non-plane trees, this corresponds to the powers of unlabelled non-plane rooted trees. Although many results about forests of unlabelled non-plane trees have been studied in [PS79], it seems that the case of finite sequences of unlabelled non-plane trees has not been thoroughly investigated.

Conjecture 34. *The mean size $\bar{U}_n$ of a computation tree induced by a process (unlabelled non-plane rooted) tree of size n admits the following asymptotics:*

$$\bar{U}_n \sim_{n\to\infty} e \cdot \bar{V}_n \sim \frac{2\sqrt{2}e\pi n}{\gamma} \left(\frac{n\eta}{e}\right)^n,$$

where η and γ are introduced in Fact 21.

5.3 The mean size defined by a P-recurrence

In this section, we focus on the asymptotic behaviour of the average size $\bar{S}_n$ of the computation trees induced by syntax trees of size n. Our goal is to obtain more precise approximations than Theorem 32 using different analytic combinatorics techniques. Indeed, we present three distinct ways to establish our main result: a P-recurrence that precisely captures the desired quantity. These results are deeply related to the holonomy property of the generating functions into consideration.

Theorem 35. *The mean size $\bar{S}_n$ of a computation tree induced by a tree of size n satisfies the P-recurrence:*

$$(2n^4+12n^3+22n^2+12n)\bar{S}_n-(4n^4+32n^3+87n^2+87n+18)\bar{S}_{n+1}\\ +(2n^4+24n^3+85n^2+106n+39)\bar{S}_{n+2}-(4n^3+20n^2+31n+15)\bar{S}_{n+3}=0,$$

with the initial conditions: $\bar{S}_0=0, \bar{S}_1=1$ and $\bar{S}_2=2$.

We have stored the non-normalized version of this sequence in OEIS, as A216234. It deals with the sequence $(S_n)_n$, associated to the cumulated sizes of computation trees issued of process trees of size n.

Proof. A first approach to prove this theorem is based on creative telescoping. See references [PWZ96, Zei90] for a detailed explanation of the method. This proof is a direct consequence of the level decomposition detailed in Section 4. It is technically simple from two points of view: in terms of the technical mathematics necessitated and in terms of the computer assistance required. In particular, all the steps are totally automatized in classical computer algebra systems (such as Maple or Mathematica). The level decomposition is really a peculiarity of the combinatorial structure we investigate, and it is hardly a common situation.

From the exact formula for the mean number of nodes each level, $\bar{W}_n^i$, by summing on all levels, we get the mean number $\bar{S}_n$ of nodes

$$\bar{S}_n=\sum_{i=0}^{n-1}\frac{2^i\ (2n-2i-1)!\ (n-1)!}{(2n-i-1)!\ (n-i-1)!}\cdot\frac{n!}{2^{n-1}\ i!}.$$

This sum can be expressed in terms of hypergeometric functions (see e.g. [PWZ96]):

$$\bar{S}_n=\frac{n!}{2^{n-1}}\,{}_1F_1(-2\,n+1;\,-n+1/2;\,1/2)-{}_2F_2(1,-n+1;\,1/2,n+1;\,1/2). \qquad (1)$$

Now, by using the package Mgfun in Maple [Chy98], we extract, by creative telescoping, the stated P-recurrence for $\bar{S}_n$. □

Furthermore, we can prove Theorem 35 with two other distinct approaches.

- The first one is based on the multivariate holonomy theory. It is our original proof, that can be found in [BGP12], and it is clearly the proof that conveys the most combinatorial informations about the structures we study.

- The last approach is based on the concept *Guess-and-Proof*. Once again this approach lies on the fundamental property of holonomy of the generating function (c.f. [Was02]). We calculate the first values for S_n, guess a differential equation verified by $S(z)$ and prove that it is corrected. This proof style is both powerful and clever since it is almost entirely automated. However, it does not convey much information about the combinatorial structures under study.

5.4 Precise asymptotics of the size

Now that we have a P-recurrence for the mean size, we can obtain precise asymptotic behaviour in a relatively effortless way.

Theorem 36. *The mean size $\bar{S}_n$ of a computation tree induced by a tree of size n admits the following precise asymptotic relation:*

$$\bar{S}_n = e\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n\left(2+\frac{2}{3n}+\frac{49}{36n^2}+\frac{27449}{6480n^3}+O\left(\frac{1}{n^4}\right)\right).$$

Proof. We can derive this result from the hypergeometric expression (1). First, let us observe that ${}_2F_2(1,-n+1;\,1/2,n+1;\,1/2)$ tends to a constant when n tends to infinity. So, we essentially need to analyse the part $\frac{n!}{2^{n-1}}\cdot {}_1F_1(-2\,n+1;\,-n+1/2;\,1/2)$. Let us observe that the next hypergeometric function ${}_1F_1(-2\,n+1;\,-n+1/2;\,x)$ admits the following expansion:

$$\begin{aligned}{}_1F_1(-2\,n+1;\,-n+1/2;\,x) = e^{2x}+x^2e^{2x}\frac{1}{n}+\\ x^2e^{2x}\left(\frac{3}{2}+2x+\frac{1}{2}x^2\right)\frac{1}{n^2}+\frac{1}{12}x^2\mathrm{e}^{2\,x}\left(27+96\,x+84\,x^2+24\,x^3+2\,x^4\right)\frac{1}{n^3}+\\ x^2e^{2x}\left(\frac{27}{8}+\frac{49}{2}x+\frac{349}{8}x^2+29x^3+\frac{33}{4}x^4+x^5+\frac{1}{24}x^6\right)\frac{1}{n^4}+O\left(\frac{1}{n^5}\right).\end{aligned}$$

Thus the asymptotic behaviour of $\bar{S}_n$ follows directly.

Let us remark that it is also possible to reach this asymptotic behaviour by using the P-recurrence. We introduce a new auxiliary generating function which is more tractable than $S(z)$. For that purpose, recall that the total number of leaves in the computation trees induced by process trees of size n (which is also the number of increasing trees of size n) is equal to $n!/2^{n-1}$. So, it is natural to study the series $R(z)$ with general terms $\bar{S}_n\cdot 2^{n-1}/n!$ which is also holonomic and satisfies

$$\begin{aligned}-2\left(10z^2+7z+3\right)R(z)+\left(-16z^4+32z^3+18z^2+7z\right)R'(z)+\\ 4\left(4z^4-6z^3-z^2\right)R''(z)+4\left(-z^4+z^3\right)R'''(z)=4z^2+z,\end{aligned}$$

with the initial conditions $R(0)=0, R'(0)=1,\ R''(0)=4$. The coefficients R_n satisfy the P-recurrence:

$$\begin{aligned}-16nR_n+4(4n^2+12n+3)R_{n+1}-2(2n^3+18n^2+31n+13)R_{n+2}+\\ (4n^3+20n^2+31n+15)R_{n+3}=0,\end{aligned}$$

with $R_0 = 0, R_1 = 1$ and $R_2 = 2$. Now, we can easily prove that this solution of this recurrence is convergent. Indeed, the it is non-negative and asymptotically decreasing, just by observing that $R_{n+3} - R_{n+2} = \left(\frac{4}{n} + O(\frac{1}{n^2})\right)(R_{n+2} - R_{n+1}) + O\left(\frac{1}{n^2}\right) R_n$ implies that for n sufficiently large the difference is always negative.

Theorem 32 shows that the series converges to $\exp(1)$. Now, a deeper analysis of this recurrence can be done using the tools described in [FS09, p. 519–522]. Indeed, the singularities are regular. Let us remark that another way to conclude consists of guessing that the asymptotic expansion of $R(z)$, as z tends to infinity, can be expressed as $\exp(2z + a\ln(z) + bz^{-1} + cz^{-2} + dz^{-3} + O(z^{-4}))$ and to use saddle point analysis (its hypotheses being validated by Wasow's theory [Was02]) to prove the result.

□

6 Applications

We describe in this section two practical outcomes of our quantitative study of the pure merge operator. First, we present an algorithm to efficiently compute the probability of a concurrent run prefix induced by the uniform probability distribution of runs. The second application is a uniform random sampler of concurrent runs. These algorithms work directly on the syntax trees without requiring the explicit construction of the computation trees. An important remark is that these algorithms apply no matter whether the syntax trees are plane or not. In these distinct models, only the average quantities are impacted because the syntax trees do not follow the same distribution.

6.1 Probability of a run prefix

We first describe an algorithm to determinate the probability of a concurrent run prefix (i.e. the prefix of a branch in a computation tree). In practice, this algorithm can be used to guide a search in the state space of process behaviours, e.g. for statistical model checking or (uniform) random testing.

Definition 37. *Let T be a process tree and $\sigma = \langle \alpha_1, \ldots, \alpha_p \rangle$ a prefix of a run in $\textsc{Sem}(T)$. The **suspended tree** T_σ has root α_p and its children are all children of the nodes $\alpha_1, \ldots, \alpha_p$ not already in σ and ordered according to the preorder traversal of T.*

For example, the suspended tree $T_{\langle a,b,d\rangle}$ of the syntax tree T in Figure 1 has root d and children (from left to right) are reduced to the leaves c, e, f and g.

Let T be a process tree and $\sigma_p = \langle \alpha_1, \ldots, \alpha_p \rangle$ a run prefix of length p. We are interested in the probability of choosing α_{p+1} to form the prefix run $\sigma_{p+1} = \langle \alpha_1, \ldots, \alpha_p, \alpha_{p+1} \rangle$. To obtain this probability, we need to count how many runs in T_{σ_p} are first running α_{p+1}.

Proposition 38.

$$\mathbb{P}(\sigma_{p+1} \mid \sigma_p) = \frac{\ell_{T_{\sigma_{p+1}}}}{\ell_{T_{\sigma_p}}} = \frac{(|T| - p)!}{\prod_{S \text{ subtree of } T_{\sigma_{p+1}}} |S|} \cdot \frac{\prod_{S \text{ subtree of } T_{\sigma_{p+1}}} |S|}{(|T| - p + 1)!} = \frac{|T(\alpha_{p+1})|}{|T| - p + 1}.$$

The proof directly derives from the hook-length formula (cf. Fact 18).

Corollary 39. *Let T be a process tree and $\sigma = \langle \alpha_1, \ldots, \alpha_p \rangle$ be a prefix of a run in* $\textsc{Sem}(T)$. *For the uniform probability distribution on the set of all concurrent runs, the induced probability on prefixes satisfies $\mathbb{P}(\sigma) = \ell_{T_\sigma}/\ell_T$.*

Corollary 40. *Let T be a process tree. The probability of a prefix run $\sigma = \langle \alpha_1, \ldots, \alpha_p \rangle$ in the computation tree of T is equal to $\prod_{k=1}^{p} |T(\alpha_k)|/(|T| - k + 1)$.*

Algorithm 1: 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 $p \leqslant n$
Result: ρ_σ: the probability of σ in the comput. tree 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

From Corollaries 39 and 40 we derive, as Algorithm 1, the computation of the probability ρ_σ of a concurrent run prefix σ. While measuring the probability in terms of a computation tree, the latter need not be constructed explicitly. The algorithm indeed requires only the syntax tree T with added weights, and a few extra memory cells. It trivially performs in linear-time.

Proposition 41. *Algorithm 1 computes the probability ρ_σ of a length p run prefix, in $(p-1)$ steps, and $\Theta(p)$ arithmetic operations.*

If the run prefix σ we consider is a full run (i.e. a complete traversal of a syntax tree T), then we obtain the uniform probability of a run in general (since all runs have equal probability in the computation tree). Then we have the following result.

Proposition 42. *The number of concurrent runs of a process tree T is $1/\rho_\sigma$ when σ is any complete traversal of T.*

From an order-theoretic point of view, we thus obtain as a by-product a linear-time algorithm to compute the number of linear extensions of a tree-like partial order.

Corollary 43. *Let T a tree-like partial order of size n. The number of its linear extensions can be computed in $O(n)$.*

Since any full run has length the size n of the syntax tree, the upper-bound $O(n)$ is trivially obtained. Moreover, we conjecture that the problem has $\Omega(n)$ lower-bound also. Note that the hook length formula also yields a linear-time algorithm but with more

arithmetic operations. To put into a broader perspective this result, we remind the reader that the problem of counting linear extensions of partial orders is $\#P$-complete [BW91] in the general case. Moreover, the proposed solution (obtained thanks to the very fruitful isomorphism with increasing trees) is clearly an improvement if compared to the quadratic algorithm proposed in [Atk90].

6.2 Random generation of concurrent runs

The uniform random generation of concurrent runs is of great practical interest in the realm of automated testing and statistical model-checking [GS05]. The problem has a trivial solution if we work on a computation tree. Since all runs have equal probability, we may simply select a leaf at random, and reconstruct the full run by climbing the unique branch from the selected leaf to the root of the tree. Of course, this naive algorithm is highly impractical given the exponential size of the computation tree. The challenge, thus, is to find a solution which does not require the explicit construction of the computation tree. A possible way would be to rely on a Markov Chain Monte Carlo (MCMC) approach, e.g. based on [Hub06]. We describe here a simpler, more direct approach that yields a more efficient almost linear algorithm.

The main idea is to sample in a multiset containing the nodes of the syntax trees as elements, each one associated to a weight corresponding to the size of the subtree rooted at this node. A particularly efficient way to implement the required multiset structure is to use a *partial sum tree*, i.e. a balanced binary search tree in which all operations (adding or removing a node) are done in logarithmic time. The details of this implementation can be found in Appendix A.

Algorithm 2: uniform random generation of concurrent runs

Data: T: a weighted process tree of size n
Result: σ: a run (a list of nodes)
$\sigma := \langle\rangle$
$M := \{\!\{a^{|T|}\}\!\}$ # initialize a multiset with the root a with its weight
for p **from** 1 **to** $|T| - 1$ **do**
  $\alpha := \mathsf{sample}(M)$ # sample an action α according to its weight in the multiset
  $\sigma := \sigma.\alpha$ # append the sampled action to the sequence
  $M := \mathsf{update}(M, \alpha, 0)$ # α cannot be sampled anymore
  for $\beta \in \mathsf{child}(T_\sigma)$ **do**
    $M := \mathsf{update}(M, \beta, |T(\beta)|)$ # insert the children of α in the multiset
return σ

Let T be a process tree. First by one traversal, we add a label to all nodes of T that corresponds to the size of the subtree rooted in that node. We say that this size corresponds to the weight of each node. We build a list σ, at the end of size n, such that at each step i, we add one action to σ that corresponds to the i-th action in our random run. To choose this i-th action, we sample in the multiset of all possible actions available

in the considered step. Initially only the root is available. Then it is added to σ and removed from the multiset. Finally its children are enabled with their weights. And we proceed until all actions have been sampled.

Let T a syntax tree, we denote by $\mathsf{child}(T)$ the nodes at level one of T.

The following loop invariant derives easily from Algorithm 2.

Invariant 44. *At the p-th step of the algorithm, we have:*

$$|M_p| = |T| - p + 1 \text{ with } M_p = \{\alpha_{p+1} \mid \alpha_{p+1} \in \mathsf{child}(T_{\sigma_p})\}.$$

Proposition 45. *Let σ_p the prefix obtained at the p-th step in Algorithm 2. The next action α_{p+1} is chosen with probability $|T(\alpha_{p+1})|/(|T| - p + 1)$. Consequently the complete run σ is generated with uniform probability.*

Proof. Let M_p the multiset obtained at step p in Algorithm 2. We select the next action α_{p+1} with probability $\frac{M_p(\alpha_{p+1})}{|M_p|}$ (cf. Appendix A for a more detailed explanation). By Invariant 44 we have $|M_p| = |T| - p + 1$. Moreover, in the Algorithm we insert α_{p+1} with weight $M_p(\alpha_{p+1}) = |T(\alpha_{p+1})|$. Thus by Proposition 38 the prefix σ_{p+1} is obtained with the correct probability so that when completed the full run σ is generated with uniform probability. □

In the case of the partial sum tree implementation (cf. Appendix A), we have directly the following complexity results.

Proposition 46. *Let n be the size of the weighted process tree T. To obtain a random execution, we need n random choices of integers and the operations on the multiset are of order $\Theta(n \log n)$ (for the worst case).*

7 Conclusion and perspectives

The quantitative study of the pure merge operator represents a preliminary step towards our goal of investigating concurrency theory from the analysis of algorithms point of view. The special case of binary merge operator (which is classical in concurrency theory) has been investigated in [BGPR15]. Unsurprisingly the results are very similar, although from an analytic combinatorics points of view, the details are indeed interesting. In a complementary work, we addressed the case of *non-deterministic choice* [BGP13]. More recently we started to investigate the principle of *synchronization* [AI07]. Other operators, such as *hiding*, also deserve further investigations. We also wish to further investigate the case of non-plane process trees. Although the nature of the operators does not seem to be really impacted (confirming the intuition of Flajolet and Sedgewick), the technical aspects in terms of analytic combinatorics are quite interesting. Another interesting continuation of the work would be to study the compaction of computation trees by identifying common subtrees. This would amount to study the interleaving of process trees up-to *bisimilarity*, the natural notion of equivalence for concurrent processes. Note that our algorithmic

framework would not be affected by such studies, since they do not require the explicit construction of the computation trees (whether compacted or not, plane or non-plane).

Perhaps the most significant outcome of our study is the emergence of a deep connection between concurrent processes and increasing labelling of combinatorial structures. We indeed connected the pure merge operator with increasing trees to measure the number of concurrent runs. We also define the notion of increasing admissible cuts to study the number of nodes by level in computation trees. We expect the discovery of similar increasingly labelled structures when we go deeper into concurrency theory.

From a broader perspective, we definitely see an interest in reinterpreting *semantic objects* (from logic, programming language theory, concurrency theory, etc.) under the lights of analytic combinatorics tools. Such objects (like computation trees) may be quite intricate when considered as combinatorial classes, thus requiring non-trivial techniques. This is highlighted here e.g. by the generalized hook-length formula characterizing the expected size of computation trees. Conversely, we think it is interesting to know – precisely, not just by intuition – the high-level of sharing and symmetry within computation trees. This naturally leads to practical algorithms, making us confident that real-world applications (in our case, especially related to random testing and statistical model-checking) might result from such a study.

Acknowledgements. Thanks to Matthieu Dien and Olivier Roussel for fruitful remarks about the algorithms.

The authors thank an anonymous referee for the careful reading of the manuscript and several suggestions improving the clarity of the presentation.

References

[AI07] L. Aceto and A. Ingólfsdóttir. The saga of the axiomatization of parallel composition. In *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2007.

[Atk90] M. D. Atkinson. On computing the number of linear extensions of a tree. *Order*, 7:23–25, 1990.

[BFLR11] A. Boussicault, V. Féray, A. Lascoux, and V. Reiner. Linear extension sums as valuations of cones. *Journal of Algebraic Combinatorics*, 35(4):573–610, 2011.

[BFS92] F. Bergeron, P. Flajolet, and B. Salvy. Varieties of increasing trees. In J.-C. Raoult, editor, *CAAP*, volume 581 of *Lecture Notes in Computer Science*, pages 24–48. Springer, 1992.

[BGP12] O. Bodini, A. Genitrini, and F. Peschanski. Enumeration and random generation of concurrent computations. In *23rd International Meeting on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms, (AofA)*, pages 83–96, Montreal, Canada, June 2012.

[BGP13] O. Bodini, A. Genitrini, and F. Peschanski. The combinatorics of non-determinism. In *FSTTCS'13*, volume 24 of *LIPIcs*, pages 425–436. Schloss Dagstuhl, 2013.

[BGPR15] O. Bodini, A. Genitrini, F. Peschanski, and N. Rolin. Associativity for binary parallel processes: A quantitative study. In *Algorithms and Discrete Applied Mathematics - First International Conference, CALDAM 2015, Kanpur, India, February 8-10, 2015. Proceedings*, pages 217–228, 2015.

[Bli89] W. D. Blizard. Multiset theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66, 1989.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[BW91] G. Brightwell and P. Winkler. Counting linear extensions is ♯P-Complete. In C. Koutsougeras and J. S. Vitter, editors, *STOC*, pages 175–181. ACM, 1991.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.

[Chy98] F. Chyzak. An extension of zeilberger's fast algorithm to general holonomic functions. In *Formal Power Series and Algebraic Combinatorics*, pages 172–183, 1998.

[CK98] A. Connes and D. Kreimer. Hopf algebras, renormalization and noncommutative geometry. *Comm. Math. Phys.*, 199(1):203–242, 1998.

[Com64] L. Comtet. Calcul pratique des coefficients de Taylor d'une fonction algébrique. *Enseignement Math.*, 10:267–270, 1964.

[Com74] L. Comtet. *Advanced Combinatorics: The Art of Finite and Infinite Expansions*. Reidel, 1974.

[DHNT11] G. Duchamp, F. Hivert, J.-C. Novelli, and J.-Y. Thibon. Noncommutative symmetric functions vii: Free quasi-symmetric functions revisited. *Annals of Combinatorics*, 15:655–673, 2011.

[Die89] P. F. Dietz. Optimal algorithms for list indexing and subset rank. In F. Dehne, J.-R. Sack, and N. Santoro, editors, *Algorithms and Data Structures*, volume 382 of *Lecture Notes in Computer Science*, pages 39–46. Springer Berlin Heidelberg, 1989.

[DPRS12] A. Darrasse, K. Panagiotou, O. Roussel, and M. Soria. Biased Boltzmann samplers and generation of extended linear languages with shuffle. In *23rd International Meeting on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms, (AofA)*, pages 125–140, Montreal, Canada, June 2012.

[Drm09] M. Drmota. *Random trees*. Springer, Vienna-New York, 2009.

[FGT92] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

[FS09] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[FS13] P. Flajolet and R. Sedgewick. *An Introduction to the Analysis of Algorithms (2nd Edition)*. Addison Wesley, 2013.

[GDG+08] M.-C. Gaudel, A. Denise, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet. Coverage-biased random exploration of models. *Electronic Notes in Theoretical Computer Science*, 220(1):3 – 14, 2008. Proceedings of the Fourth Workshop on Model Based Testing (MBT 2008).

[GS05] R. Grosu and S. A. Smolka. Monte carlo model checking. In *TACAS'05*, volume 3440 of *LNCS*, pages 271–286. Springer, 2005.

[Hub06] M. Huber. Fast perfect sampling from linear extensions. *Discrete Mathematics*, 306(4):420–428, 2006.

[Kla97] M. Klazar. Twelve countings with rooted plane trees. *European Journal of Combinatorics*, 18(2):195–210, 1997.

[KMPW14] J. S. Kim, K. Mészáros, G. Panova, and D. B. Wilson. Dyck tilings, increasing trees, descents, and inversions. *J. Comb. Theory, Ser. A*, 122:9–27, 2014.

[Knu98] D. E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[MM78] A. Meir and J. W. Moon. On the altitude of nodes in random trees. *Canad. J. Math.*, 30:997–1015, 1978.

[MZ08] M. Mishna and M. Zabrocki. Analytic aspects of the shuffle product. In *STACS*, pages 561–572, 2008.

[OR95] F. Olken and D. Rotem. Random sampling from databases: a survey. *Statistics and Computing*, 5:25–42, 1995.

[PS70] G. Pólya and G. Szegö. *Aufgaben und Lehrsätze aus der Analysis I. (4th ed.)*. Springer, 1970.

[PS79] E.M. Palmer and A.J. Schwenk. On the number of trees in a random forest. *Journal of Combinatorial Theory, Series B*, 27(2):109 – 121, 1979.

[PWZ96] M. Petkovsek, H. S. Wilf, and D. Zeilberger. *A=B*. A. K. Peters, Wellesley, MA, 1996.

[Pó37] G. Pólya. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica*, 68(1):145–254, 1937.

[Sha55] D. Shanks. Nonlinear transformations of divergent and slowly convergent sequences. *Journal of Mathematics and Physics*, 34:1–42, 1955.

[Sta80] R. P. Stanley. Differentiably finite power series. *European Journal of Combinatorics*, 1(2):175 – 188, 1980.

[SZ94] B. Salvy and P. Zimmermann. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2):163–177, 1994.

[Was02] W. Wasow. *Asymptotic Expansions for Ordinary Differential Equations*. Dover phoenix editions. Dover, 2002.

[WE80] C. K. Wong and M. C. Easton. An efficient method for weighted sampling without replacement. *SIAM J. Comput.*, 9(1):111–113, 1980.

[Zei90] D. Zeilberger. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics*, 32(3):321–368, 1990.

A Weighted random sampling in dynamic multisets

This appendix discusses the problem of random sampling elements according to their respective weight in a multiset. Moreover, the multiset must be dynamic in that the cardinality of elements can be changed on-the-fly. This problem represents a basic algorithmic component in the random generation of concurrent runs (cf. Section 6). To our knowledge, this has not been addressed precisely in the literature (some basic information can be found in [WE80, OR95]).

A.1 Dynamic multiset basics

In this section we recall a few concepts and basic notations of multisets, the reader may consult e.g. [Bli89] for a more thorough treatment. A finite multiset (or bag) M can be defined formally as a function from a *carrier set* $\overline{M}$ to positive integers, more precisely an injective function $M \subset \overline{M} \to \mathbb{N}$. As an example we consider a multiset $M_0 = \{\!\{a^2, b^3, c^1\}\!\}$ with carrier set $\overline{M_0} = \{a, b, c\}$. In the common functional notation, we would denote $M_0 = \{(a, 2), (b, 3), (c, 1)\}$. Each positive integer associated to an element is called its *weight* (i.e. number of "occurrences") in the multiset. The weight of and element $\alpha \in \overline{M}$ is denoted $M(\alpha)$. And by a slight abuse of notation we write $\alpha \in M$ *iff* $M(\alpha) \geqslant 1$. For example, the element a has weight 2in M_0, thus $M_0(a) = 2$ and of course $a \in M_0$. The notation $\alpha \notin M$ may either denote $\alpha \notin \overline{M}$ or $M(\alpha) = 0$. This slight ambiguity has interesting algorithmic implications.

The cardinal or *total weight* of M is $|M| = \sum_{\alpha \in M} M(\alpha)$, for example $|M_0| = 2+3+1 = 6$ whereas for the carrier set we have $|\overline{M_0}| = 3$.

Given a multiset M the two operations we are interested in are:

- a random sampler $\mathsf{sample}(M)$ that generates an element $\alpha \in M$ at random with probability $\frac{M(\alpha)}{|M|}$.
- an update operation $\mathsf{update}(M, \alpha, k)$ that produces a multiset M' such that $\forall \beta \in M, \beta \neq \alpha \implies M'(\beta) = M(\beta)$ and $M'(\alpha) = k$.

Remark that if $M' = \mathsf{update}(M, \alpha, 0)$ we do have $M'(\alpha) = 0$ hence $\alpha \notin M'$ but it is left unspecified whether $\alpha \in \overline{M'}$ or not.

A.2 A naive random sampler

Probably the fastest way to implement the sample operation is to represent a multiset M with an array of length $n = |M|$. Formally, this defines a finite sequence, i.e. a function $\sigma_M : [1..n] \to |M|$. Consider $M = \{\!\{a_1^{k_1}, \ldots, a_n^{k_n}\}\!\}$ an arbitrary multiset. The cardinality of M is $|M| = \sum_{j=1}^{n} k_j$ and its carrier set is $\overline{M} = \{\alpha_1, \ldots, \alpha_n\}$. For the sake of simplicity and without any loss of generality, we assume an implicit strict order $\alpha_1 < \ldots < \alpha_n$. Now we define σ_M such that $\forall i \in [1..n],\ \forall j \in \left[\sum_{p=1}^{i-1} k_p + 1 .. \sum_{p=1}^{i-1} k_p + k_i\right],\ \sigma_M(j) = \alpha_i$, and

Algorithm 3: naive random sampler for dynamic multisets.

Data: $M = \{\!\{a_1^{k_1}, \ldots, a_n^{k_n}\}\!\}$
Result: β an element of M taken with probability $\frac{M(\beta)}{|M|}$

$\rho :=$ a uniform random integer taken in range $[1..n]$;
return $\beta = \sigma_M(\rho)$;

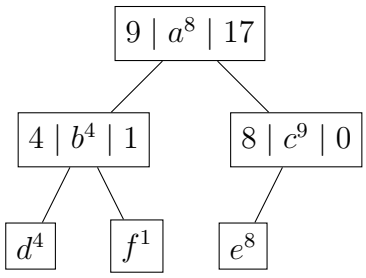


Figure 7: A partial sum tree for multiset $M_2 = \{\!\{a^8, b^4, c^9, d^4, f^1, e^8\}\!\}$.

everywhere else σ_M is undefined. For example σ_{M_0} is $\{1 \mapsto a, 2 \mapsto a, 3 \mapsto b, 4 \mapsto b, 5 \mapsto b, 6 \mapsto c\}$ which we may also denote $\langle a, a, b, b, b, c\rangle$.

The naive random sampler is described by Algorithm 3. For a multiset M, we first pick a uniform random integer in range $[1..|M|]$. This way, we select the ρ-th position in the sequence σ_M with probability $\frac{1}{|M|}$. Now let i such that $\beta = \sigma_M(\rho) = \alpha_i \in M$. By definition of σ_M we have $|\{j \mid \sigma_M(j) = \beta\}| = \sum_{p=1}^{i-1} k_p + k_i - \sum_{p=1}^{i-1} k_p - 1 + 1 = k_i = M(\alpha_i)$. Thus the probability of picking element $\beta = \sigma_M(\rho)$ is $\frac{1}{|M|} \times M(\beta) = \frac{M(\beta)}{|M|}$ which is as required.

The complexity of the sampling algorithm corresponds to the uniform random sampling of an integer in range $|1..n|$ for a multiset of total weight n, plus a single access to the array σ_M which is in general performed in constant time. The space complexity is linear in n since we must record σ_M with its $|M|$ elements, which is not very good since the weight of a given element can be arbitrarily high. Moreover, the update operation is not very efficient for the same reason.

A.3 A more efficient random sampler based on partial sum trees

We now describe a random sampler that has far better space requirements – in the order of $|\overline{M}|$ – and also enjoys a much more efficient update operation. The main idea is to exploit a representation based on *partial sum trees* [Die89].

In Figure 7 we give a possible partial sum tree (PST) representation of the multiset $M_2 = \{\!\{a^8, b^4, c^9, d^4, f^1, e^8\}\!\}$. The idea is to represent a multiset M as a binary tree with nodes labelled with three informations: the total weight of the left and right subtrees as

well as a unique element α of M together with its weight $M(\alpha)$.

Algorithm 4: partial sum tree (PST) based random sampler for dynamic multisets.

Data: T_M a PST for a multiset $M = \{\!\{a_1^{k_1}, \ldots, a_n^{k_n}\}\!\}$
Result: β an element of M taken with probability $\frac{M(\beta)}{|M|}$

$\rho :=$ a uniform random integer taken in range $[1..|M|]$;
return $\beta =$ `dispatch(`T_M`,` ρ`)`;

where:

`dispatch(`$\big[L \mid \alpha^k \mid R\big]$`,` ρ`)` is
  if $\rho \leqslant |L|$ **then**
    return `dispatch(`T_L`,` ρ`)`
  else if $\rho - |L| \leqslant k$ **then**
    return α
  else
    return `dispatch(`T_R`,` $\rho - (|L| + k)$`)`

The random sampler based on the PST representation is described by Algorithm 4. The first step is the same as in the naive algorithm: pick an integer ρ uniformly at random in the range $[1..|M|]$. The second part is a simple recursive dispatch within the tree T_M depending only on the value of ρ. If ρ is less than the total weight $|L|$ of the left-subtree, denoted T_L, of T_M then we pick the element in this left-subtree. If otherwise ρ is in the range $[|L|..|L| + k]$ then we pick-up the root element α. Otherwise, we pick the element in the right-subtree T_R without forgetting to update ρ as $\rho - (|L| + k)$ in the recursive calls.

Proposition 47. *If we assume the tree T_M to be well-balanced, the worst-case time complexity of the PST-based random sampler is $O(\log |\overline{M}|)$. The update operation inherits the same worst-case complexity. Moreover the PST itself occupies space of order $\Theta(|\overline{M}|)$ in memory.*

The well-balanced assumption is easy to obtain in practice, either by relying on implicitly well-balanced tree models e.g. AVL or red-black trees, or by simply constructing the PST in a deterministically well-balanced way (e.g. with a bit flag in each node, flipped after each insertion). So if compared to the naive algorithm and its constant-time sampling, the PST algorithm is far better in terms of memory usage. The most prominent advantage is a now very efficient update operation: we just need to update the left and right sums in the nodes from the updated node to the root of the tree (trivially also in order $O(\log n)$ for a well-balanced tree).

The proof for the correctness property is now slightly more involved.

Proposition 48. *Let M a multiset. The PST random sampler returns an element $\alpha \in M$ with probability $\frac{M(\alpha)}{|M|}$.*

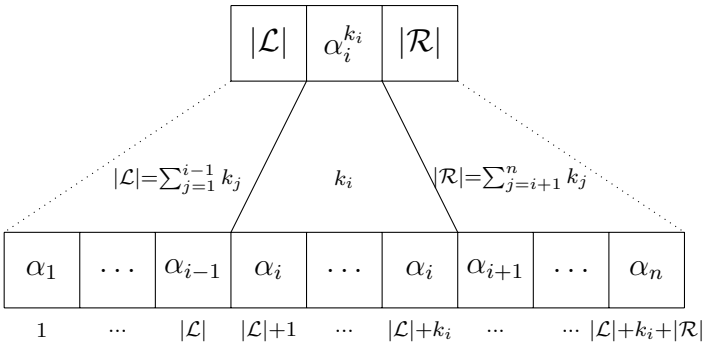


Figure 8: Mapping from a node of the partial sum tree to its corresponding inorder sequence.

Proof. Let $M = \{\!\{\alpha_1^{k_1}, \ldots, \alpha_n^{k_n}\}\!\}$ a multiset and let denote by T_M the its partial sum tree representation. If we assume an implicit order $\alpha_1 < \ldots < \alpha_i < \ldots < \alpha_n$ then the inorder sequence[5] of M is exactly σ_M as defined previously. More generally, a given node $N = (L, \alpha_i^{k_i}, R)$ of the representation corresponds to a multiset N, and we denote σ_N its inorder sequence. The tree T_L is the left subtree corresponding to a multiset L with inorder sequence σ_L of elements from $\sigma_N(1)$ to $\sigma_N(\sum_{j=1}^{i-1} k_j)$. The root node of T_N represents the element α_i with cardinality k_i which maps to the sub-sequence σ_{α_i} of elements from $\sigma_N(\sum_{j=1}^{i-1} k_j + 1)$ to $\sigma_N(\sum_{j=1}^{i} k_j)$. Finally T_R is the right subtree corresponding to multiset R with inorder sequence σ_R of elements from $\sigma_N(\sum_{j=1}^{i} k_j + 1)$ to $\sigma_N(\sum_{j=1}^{n} k_j)$. This mapping is depicted on Figure 8. Thus the left subtree T_L represents the multiset $L = \{\!\{\alpha_1^{k_1}, \ldots, \alpha_{i-1}^{k_{i-1}}\}\!\}$ and T_R represents the multiset $R = \{\!\{\alpha_{i+1}^{k_{i+1}}, \ldots, \alpha_n^{k_n}\}\!\}$.

Now we demonstrate the property that at node $N = (L, \alpha_i^{k_i}, R)$ the call dispatch(T_N,ρ) with ρ taken randomly in $[1..|N|]$ yields an element $\alpha \in N$ with probability $\frac{N(\alpha)}{|N|}$. We proceed by induction on the tree structure (or size). Suppose $\rho \in [1..|L|]$ then :

- if T_N is a leaf $(\emptyset, \alpha^k, \emptyset)$ then $\rho \in [1..k]$ with $k = N(\alpha)$ and $\sigma_N = [\alpha, \ldots, \alpha]$ (i.e. $ran(\sigma) = \{\alpha\}$) . Then the probability of choosing α is $\frac{N(\alpha)}{|N|} = \frac{k}{k} = 1$.

- if $T_N = (L, \alpha_i^{k_i}, R)$ is an internal node, i.e. $L \cup R \neq \emptyset$ then we show the property to hold for T_N if we assume it holds for the subtrees T_L and T_R :

  - if $1 \leqslant \rho \leqslant |L|$ then dispatch(T_N,ρ)=dispatch(T_L,ρ) and the property holds by hypothesis of induction.
  - if $|L| + 1 \leqslant \rho \leqslant |L| + k_i$ then we select element α_i with probability $\frac{k_i}{|L|+k_i+|R|} = \frac{N(\alpha_i)}{|N|}$
  - if $|L| + k_i \leqslant \rho \leqslant |N|$ then dispatch(T_N,ρ)=dispatch(T_R,ρ') with $\rho' = \rho - (|L| + k_i)$ and since $\rho' \in [1..|R|]$ the property holds by hypothesis of induction.

[5]The inorder sequence of a binary tree is the sequence of its elements ordered according to the inorder traversal of the tree.

In consequence, if $\rho \in [1..|M|]$ then dispatch(T_M,ρ) yields an element α with probability $\frac{M(\alpha)}{|M|}$. □

Note that the proof exploits the fact that the mapping from a multiset M to its representative sequence σ_M is deterministic. While imposing a total order on the elements α_i's provides a simpler solution – the representative being the inorder - ordered sequence – it is by no means a strict requirement. The proof can easily - albeit uninterestingly - adapt to any permutation in the selected order.