A Quantitative Study of Pure Parallel Processes

Olivier Bodini, Antoine Genitrini, Fréderic Peschanski

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
Laboratoire d'Informatique de Paris 6,
CNRS UMR 7606 and
Université Pierre et Marie Curie,
4 place Jussieu, 75005 Paris, France.
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 very hard - at least from the point of view of computational complexity - the analysis of process behaviours e.g. by model-checking. 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) and A.N.R. project *MAGNUM*, ANR 2010-BLAN-0204.

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 continue 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 a branching operator.

The pure merge operator is a principal source of *combinatorial explosion* 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 address 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 has 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 while we go deeper into concurrency theory. We think 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.

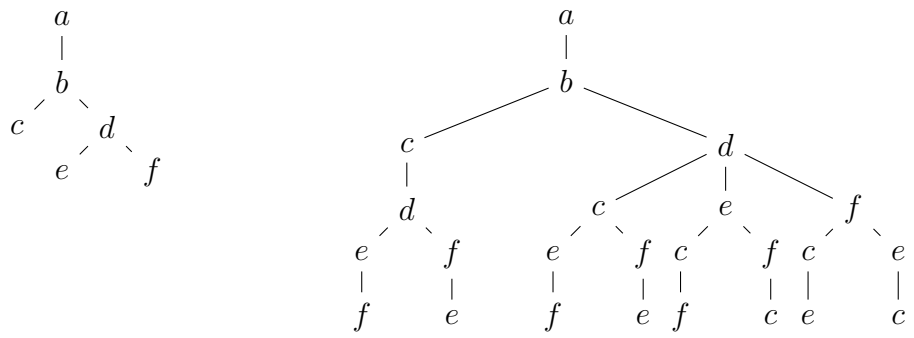


Figure 1: A syntax tree (left) and the corresponding semantic tree (right)

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 semantic 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 provide 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 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 problematic 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. The set of process specifications is the least set satisfying:

- an atomic action, denoted $a, b, \ldots$ is a process,
- the prefixing $a.P$ of an action a 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 actions or prefixed processes is a process.

Let us first remark that this grammar takes the $\parallel$ operators as associative operators and thus two of them cannot appear consecutively. Moreover, in the rest of the paper we will concentrate on *prefixed processes*. This choice does not deplete the results thanks to the bijection between prefixed processes of size n and processes without prefixed action of size $n-1$.

An example of a valid specification is:

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

which can be faithfully represented by a tree, namely a *syntax tree*, as depicted on the lefthand side of Figure 1. Such a tree can be read as a set of *precedence constraints* between atomic actions. Under these lights 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 may only happen after d.

In combinatorial terms we adopt the classical specification for *plane rooted trees* to represent the syntactic domain. The *size* of a tree is its total number of nodes. Note that we do not keep the names of actions in the process trees since they play no rôle for the pure merge operator.

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, we remind 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 a formal power series $C(z)$ such that $C(z) = \sum_{n \geqslant 0} C_n z^n$. 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]):

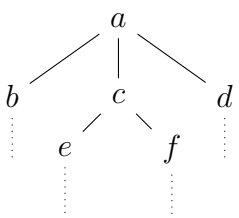
Fact 2. $C(z) = \frac{1}{2} - \frac{\sqrt{1-4z}}{2}$ *and* $C_n = \frac{4^{n-1}}{\sqrt{\pi n^3}} \left(1 + \frac{3}{8n} + \frac{25}{128n^2} + \frac{105}{1024n^3} + \frac{1659}{32768n^4} + O\left(\frac{1}{n^5}\right)\right)$.

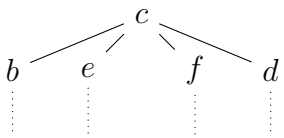
2.2 Semantic trees

The semantic domain we study is much less classical than syntax trees, although it is *still* composed of plane rooted trees. An example of a *semantic tree* is depicted on the righthand side of Figure 1. This tree represents all the possible executions – or *runs* – that may be observed for the process specified on the left. More precisely each branch of the semantic tree, e.g. $\langle a, b, c, d, e, f\rangle$, is a concurrent run (or admissible computation) of the process, and all the branches share their common prefix. In the literature such structures are also called *computation trees* [CES86].

For a tree T and one of its node v, the *sub-tree* roooted in v from T is the tree of all the descendants of v in T. To describe the recursive construction of the semantic 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 root-labels of 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 whose children are, from left to right, $T(v_1), \ldots, T(v_{i-1}), T(v_{i_1}), \ldots, T(v_{i_m}), T(v_{i+1}), \ldots, T(v_r)$ (where $T(\nu)$ denotes the sub-tree whose root is ν and $v_{i_1}, \ldots, v_{i_m}$ are the root-labels of 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 label of the root of the i-th child (here c). Now, the interleaving operation follows a straightforward recursive scheme.

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

- *if T is a leaf, then $\textsc{Sem}(T)$ is T,*
- *if T has root t and r children ($r \in \mathbb{N} \setminus \{0\}$), then $\textsc{Sem}(T)$ is the plane tree with root t and children, from left to right, $\textsc{Sem}(T \lhd 1), \ldots, \textsc{Sem}(T \lhd r)$.*

The mapping between the syntax trees on the one side, and the semantic trees on the other side is trivially one-to-one. Figure 2 depicts the enumeration of the first syntax trees (by size n) together with the corresponding semantic tree.

We note that the semantic trees are *balanced* (i.e. all their leaves belong to the same level), and even more importantly and that their height is $n-1$, where n is the size of the associated process tree. This is obvious since each branch of a semantic tree corresponds to a complete traversal of the syntax tree. Thus there are as many semantic trees of height $n-1$ as there are 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 semantic tree is lower than the number of

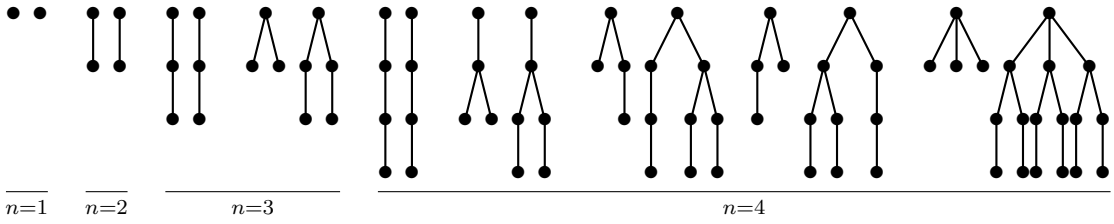


Figure 2: Enumerating behaviours (semantic trees) from process specifications (syntax trees).

nodes at the next level. Thus, the width of the semantic tree corresponds to its number of leaves.

The following observation bears witness to the high level of redundancy exhibited by semantic trees.

Proposition 5. *The knowledge of a single branch of a semantic tree is sufficient to recover the corresponding syntax tree.*

To go slightly further into the details, we may indeed exhibit a familiy of inverse functions from singled-out semantic tree branches to syntax trees. These inverse functions exploit the concept of a *degree-sequence*, defined as follows.

Definition 6. *A* ***degree-sequence*** $(u_p)_{p\in\{1,\dots,n\}}$ *is a sequence of non-negative integers of length* n *that satisfies:*

$$u_1 > 0; \qquad \forall p > 1,\ u_p \geqslant u_{p-1} - 1; \qquad u_n \text{ is the single term equal to 0.}$$

The degrees of the nodes from the root to a leaf in any branch of a semantic tree is a degree-sequence.

Proposition 7. *Let* (u_p) *be a degree-sequence of length* n *that is linked to the leftmost branch of a semantic tree* S*. Let us define the new sequence* (v_p) *such that:*

$$v_1 = u_1; \qquad \forall p > 1,\ v_p = u_p - u_{p-1} + 1.$$

We build a tree T *of size* n *such that the sequence* (v_n) *corresponds to the degrees of each node of the tree, ordered by the prefix traversal. The semantic image of* T *is the tree* S*.*

An important remark is that we only considered the leftmost branch of the semantic tree to construct the corresponding degree-sequence, from which we recover the initial syntax tree. It is interesting to note that the leftmost branch of the semantic tree encodes a Łukasiewicz word which is directly related to the degree-sequence of the prefix traversal [FS09, p. 74–75].

We can show, in fact, that the initial tree can be recovered by considering *any* of the branches of its semantic tree, not just the leftmost one. Each branch corresponds to a

degree sequence visiting the nodes of the initial tree by a specific traversal. For example, if the leftmost branch encodes the prefix traversal; the rightmost branch enumerates its mirror: the postfix traversal. Last but not least, the set of degree-sequences of length n is only of cardinality C_n, so the semantic trees are highly *symmetrical* in that many branches must be defined by the same degree-sequence.

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 semantic 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 semantic 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

Our study begins by a simple observation that connects the pure merge operator to the set of *linear extensions* of tree-like partial orders or *tree-posets* [Atk90].

Definition 8. *Let T be the syntax tree of a process, and A its set of actions. We define the poset $(A, \prec)$ such that $a \prec b$ iff a is the label of a node that is the parent of a node with label b in T. The linear extensions of $(A, \prec)$ is the set of all the strict orderings $(A, <)$ that respect the partial ordering.*

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

Proposition 9. *Let T be the syntax tree of process, $(A, \prec)$ the associated tree-poset. Then:*

- *Each branch of $\textsc{Sem}(T)$ encodes a distinct strict ordering of A that respects $(A, \prec)$,*
- *If a strict ordering $(A, <)$ respects $(A, \prec)$ then it is encoded by a given branch of $\textsc{Sem}(T)$.*

This observation is quite trivial, and can be justified by the fact that each branch of $\textsc{Sem}(T)$ encodes a distinct traversal of T. For example in Figure 1 the leftmost branch of the semantic tree is the linear extension $a < b < c < d < e < f$, that indeed fulfills the tree-poset.

Under these new order-theoretic lights, we can exhibit a deep connection between the number of concurrent runs of a syntax tree T and the number of ways to label it in a

strictly increasing way. Indeed, as already observed in [KMPW12], the linear extensions of tree-posets are in one-to-one correspondence with *increasing trees* [BFS92, Drm09].

Definition 10. *An **increasing tree** is a labelled plane rooted tree such that the sequence of labels along any branch starting at the root is increasing.*

For example, to label the tree of Figure 1, a would take the label 1, b then takes the label 2. Then the label of c must belong to $\{3, 4, 5, 6\}$, which would then it induces constraints on the other nodes. Finally, only 8 labelled trees are increasing trees among the 6! possible unconstrained labellings.

Increasing plane rooted trees satisfy 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]):

Fact 11. *The number of increasing plane rooted 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 we obtain our first significant measure.

Theorem 12. *The mean number of concurrent runs induced by syntax trees of size n is:*

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

This result is obtained from Fact 11 by taking the average number of increasing trees of size n, and the asymptotics is based on 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. 67]:

Fact 13. *The number ℓ_T of increasing trees built on a plane rooted tree T is:*

$$\ell_T = \frac{|T|!}{\prod_{S \text{ sub-tree of } T} |S|},$$

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

Corollary 14. *The number of concurrent runs of 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. 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 12 is the usual way to measure in average. Nevertheless, a small number of compact syntax trees (such as a root followed by (n-1) sons) produces a huge number of runs and unbalance the mean. So, a natural way to avoid such 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 15. *The geometric mean number of concurrent runs built on process trees of size n satisfies:*

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

$$\textit{where}^2\ L(1/4) = \textstyle\sum_{n>1} \log n \cdot C_n \cdot 4^{-n} \approx \mathbf{0.5790439}217 \pm 5 \cdot 10^{-9}.$$

This growth appears to us as less important than what we conjectured with the arithmetic mean, although it is still very large. For both means, the result is indeed quite far from the upper bound $(n-1)!$.

Proof. First we need to obtain a recurrence formula based on the hook length formula. Let us give the following observation:

$$\prod_{S \text{ sub-tree of } T} |S| = |T| \cdot \prod_{R \text{ child of the root of } T} \left(\prod_{S \text{ sub-tree of } R} |S| \right).$$

Now, by Fact 13 we deduce the next recursive equation:

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

Since the geometric mean of ℓ_T is related to the arithmetic mean of $\ln(\ell_T)$, we introduce the sequence $w_T = \log(\ell_T/|T|!)$ and its generation function $W(z) = \sum_T w_T z^{|T|}$. Using the latter recursive formula on ℓ_T, we deduce:

$$W(z) = -L(z) + \sum_T \sum_{R \text{ child of the root of } T} w_R z^{|T|},$$

where $L(z) = \sum_T \log|T| z^{|T|} = \sum_{n\geqslant 1} C_n \log(n) z^n$ and $C(z) = \sum_n C_n z^n$ is the generating function enumerating all trees.
By partitioning trees T according to their number of root-children, we get

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

[2]For approximate constants, the exact digits are written in bold type.

Now, by symmetry of the trees R_i, we get:

$$W(z) = -L(z) + \sum_{r\geqslant 1} r \cdot \sum_{R_1,\ldots,R_r} w_{R_1} z^{1+\sum_{j=1}^{r}|R_j|} = -L(z) + zU(z)\sum_{r\geqslant 1} rC^{r-1}(z).$$

We recognise $\sum_{r\geqslant 1} rC^{r-1}(z) = (1-C(z))^{-2}$, thus,

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

In order to obtain the geometric mean width Γ_n, we first extract the n-th coefficient of the previous product. Then we apply the exponential function on the result. We then multiply it by $n!$ and take the C_n-th root of the result. Finally $\bar{\Gamma}_n$ is equal to this result divided by n.

In order to approximate for $[z^n]W(z) = W_n$, we give an approximation $A(z)$ of $L(z)$:

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

where γ is Euler's constant. By using the two first terms in the development of Catalan numbers (see Fact 2) and formulas [FS09, p. 388], we obtain $L_n = A_n + O(4^n\ n^{-5/2}\ \ln n)$. Consequently,

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

Thus,

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

Finally we conclude:

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

□

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 semantic trees as non-plane (or unordered) rooted trees.

Thankfully the non-plane analogous of the Catalan number is well known (cf. [FS09, p. 475–477]):

Fact 16. *The specification of unlabelled non-plane rooted trees is* $\mathcal{T} = \mathcal{Z} \times \mathrm{MSet}\,\mathcal{T}$. *The number* T_n *of such trees of size* n *is:*

$$T_n \sim \frac{\gamma}{2\sqrt{\pi n^3}} \eta^{-n},$$

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

Compared to 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. Luckily, the increasing variant on non-plane trees have been studied in the model of *increasing Cayley trees* [FS09, p. 526–527]:

Fact 17. *The specification of increasing non-plane rooted trees is* $\mathcal{I} = \mathcal{Z} \ {}^{\square}\star\, \mathrm{Set}(\mathcal{I})$. *The number* I_n *of such trees of size* n *is:*

$$I_n = (n-1)!.$$

Theorem 18. *The mean number of concurrent runs built on non-plane syntax trees of size* n *is:*

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

where η *and* γ *are introduced in Fact 16.*

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 equivalent, 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 asymptotics 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 semantic trees. This study puts into light a new – and, we think, interesting – combinatorial class: the model of *increasing admissible cuts* (of plane 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 that enables the decomposition of a semantic tree by levels. Based on this construction, we study experimentally the level decomposition of semantic trees corresponding to syntax trees of a size 40 (which yields semantic trees with more than 10^{28} nodes !). Finally, we discuss the mean number of nodes by level, which is obtained by counting increasing admissible cuts. This provides a fairly precise characterization of the typical shape for process behaviours.

4.1 Increasing admissible cuts

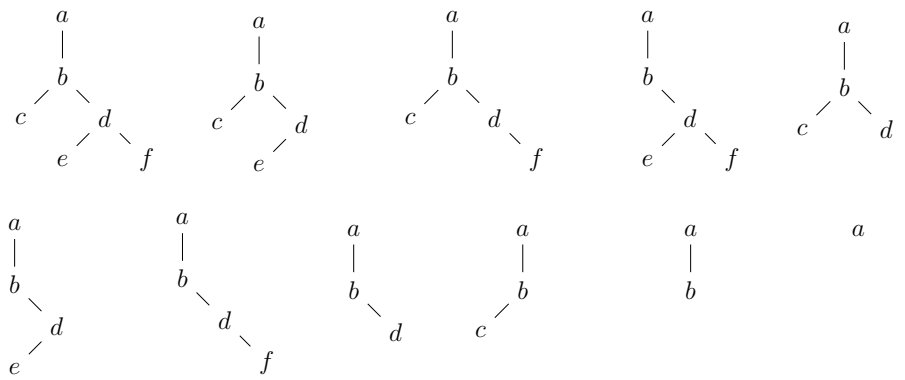


Figure 3: The admissible cuts of the syntax tree of Figure 1.

The notion of *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 19. *Let T be a 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 3 depicts the set of 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 20. *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 of size k.*

Proof. We proceed by finite induction on k. For $k = 1$ there is a single run prefix of length $k = 1$ with the root of T and the corresponding admissible cut is the root node, which only has one increasing labelling. 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 hypothesis of induction, any run prefix σ_k of length k is encoded by a given admissible cut of size k. Let us denote by $S(\sigma_k)$ 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 corresponds to a node in T that is a direct child of one of the nodes of $S(\sigma_k)$. Thus we obtain a unique $S(\sigma_{k+1})$ as $S(\sigma_k)$ 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 both first admissible cuts of size 4 depicted on Figure 3.

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

Proposition 21. *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 trivial order-theoretic argument. Each admissible cut is a tree-poset and thus the number of run it encodes is the number of its linear extensions. □

For example, there are three admissible cuts of size 4 in Figure 3. The first one admits two increasing labellings and the other ones have a single labelling. This gives $2+1+1 = 4$ 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 semantic tree. And this of course generalizes: the number of run prefixes of length k corresponds to the number of nodes at level $k-1$ in the semantic tree.

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

Corollary 22. *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 13). 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 working an exact formula for the mean number of nodes by level, we can take advantage of Corollary 22 to compute the shape of some typical semantic trees.

4.2.1 Experimental study

Our experiments consists 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 syntax tree with a size n very large, given the following result.

Observation 23. *The mean number $\bar{m}_n$ of admissible cuts of trees of size n satisfies:*

$$\bar{m}_n \sim_{n\to\infty} \frac{1}{\sqrt{15}} \left(\frac{5}{2}\right)^{2n}.$$

[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 $M_n = \sum_{T;|T|=n} \sum_{S \text{ adm. cut of } T} 1$. The tag $\mathcal{Z}$ marks the nodes of the tree carrying the admissible cut. The generating function $C(z)$ enumerates all trees. The specification of $\mathcal{M}$ is $\mathcal{Z} \times Seq(\mathcal{M} \cup \mathcal{C})$. In fact, an admissible cut is a root and a sequence of children that are either admissible cuts, or trees that corresponds to a branch of the original tree that has entirely disappeared. Consequently, $M(z)$ satisfies the following equation that can be easily solve:

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

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. By using transfer lemmas [FS09, p. 392], we get the asymptotic behaviour. □

Using the same kind of method than one of the following (Section 5), we can exhibit the P-recurrence satisfied by the cumulative number of admissible cuts m_n:

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

with $m_0 = 0, m_1 = 1, m_2 = 2$ and $m_3 = 7$. This sequence is registered by OEIS at A007852.

As a consequence we must be particularly careful when computing the shape of a semantic tree in practice using our generalization of the hook-length formula for increasing admissible cuts. However, 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: $\bar{W}^0_{40} > 1.48 \cdot 10^{36}$!

For the two syntax trees depicted on the left of Figure 4, the shape of the corresponding semantic tree is depicted on the right. We use a logarithmic scale of the horizontal axis so that the exponential fringes become lines. The semantic 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 theoretical 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. We also remark that both the trees have a semantic size that is smaller than the average. To analyse this particularity, we have sampled more than fifty typical process trees of size 40 (of course 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 have a semantic-tree size that is much smaller than the average,

[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.

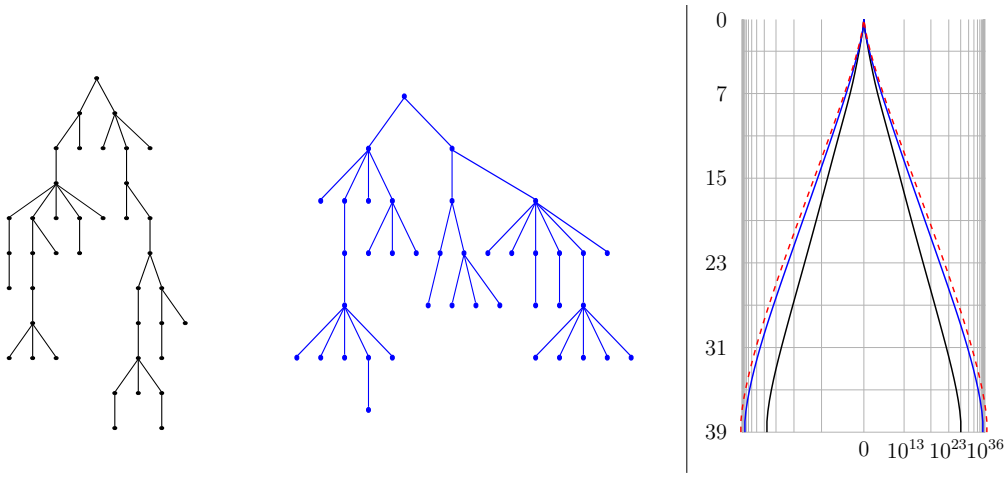


Figure 4: Typical trees of size 40 and their semantic-tree profile behind the average profile.

($\approx 4.06 \cdot 10^{36}$). 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 a only a very few special syntax trees accounts for the largest increase of the semantic size. These are probably process trees whose 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 (the one whose semantic-tree size is 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: a close formula for the mean number of nodes at each level of a semantic tree.

Theorem 24. *The mean number of nodes at level $n-i-1$, for $i \in \{0, \ldots, n-1\}$ in a semantic 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 corollary of Corollary 22 gives the cumulated number W_n^i of nodes at level $n-i-1$ in semantic trees issued of syntax trees of size n to be equal to the sum on all increasingly labelled admissible cuts of size $n-i$ from syntax trees of size n. 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 sub-trees. By the reverse process, i.e. by plugging sequences of trees to a fixed tree (that corresponds

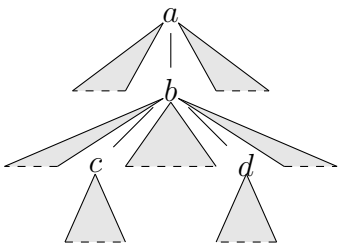


Figure 5: An admissible cut and the places where it can be enriched.

to an admissible cut), we obtain the set of trees which admit that admissible cut. On Figure 5, 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 $2n - 2i - 1$. Thus we conclude:

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

by using Definition 1 and Fact 11. 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 the "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 4. Let us first determinate the limit curve for the average shape of an semantic tree. We renormalise the values $\bar{W}_n^i$ as $f(c, n) = \ln(\bar{W}_n^{\lfloor cn \rfloor})$ and we evaluate an asymptotic of $f(c, n)$ when n tends to infinity. An easy calculation shows that, for $\frac{1}{n} \ll c \ll 1 - \frac{1}{n}$ (i.e. $\frac{1}{n} = o(c)$ and $1 - c = o(\frac{1}{n})$, we have :

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

In particular, as n tends to infinity, on every compact $[a, b]$ such $0 < a < b < 1$, the function in c, $f(c, n)$ tends uniformly to a line $(1 - c)n \ln(n)$. Moreover, if we keep the second order terms, we then obtain a curve which is totally relevant with the Figure 4.

Now, we are interested to the behaviour in the neighbourhood of the extremities of $[0,1]$. We can study the asymptotic of $\bar{W}_n^i$ 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 extremities.

5 Expected size of process behaviours

In this section we study in more details the average size of semantic trees (i.e. the mean number of nodes). In a first part we provide a first approximation based on the Theorem 24 of the previous section. Then, we make a conjecture regarding the non-plane case, whose proof require a deeper study that goes beyond the scope of this paper. Finally, we characterize the average size in a more precise way, through a linear recurrence that is obtained by various means. 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 a precise asymptotics of the size of the semantic trees.

5.1 First approximation of mean size

Our initial approximation of the mean size of the semantic trees is based on the level decomposition of Section 4, where we give a closed formula for the mean width of nodes $\bar{W}_n^i$ at level $n-i-1$ for semantic trees corresponding to syntax trees of size n, as Theorem 24. We first give, as a technical lemma, an inequality involving $\bar{W}_n^i$.

Lemma 25.

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

Proof. We first deal with some normalized expression of Theorem 24:

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

Obviously, we get the stated lower bound 1. Let us go on with the simplification. 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:

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

So we obtain the stated upper bound. □

Theorem 26. *The mean size $\bar{S}_n$ of a semantic tree induced by a process tree of size n admits the following asymptotics:*

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

Proof. Using Lemma 25 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 (Taylor-Lagrange formula):

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

so both bounds tends 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.

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

□

Corollary 27. *Let f be a function in n that tends to infinity with n. Let a_n be the average number of nodes of the semantic tree induced by all the syntax tree of size n and l_n the average number of nodes belonging to the $f(n)$ last levels. Then, l_n/a_n tends to 1 when n tends to infinity.*

The proof is analogous as the previous one using the Taylor-Lagrange formula. The unique constraint for f is that it tends to infinity, but it can grow as slow as we want. For example, asymptotically almost all nodes of the average semantic 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 analogous of powers of the Catalan generating function (see proof of Theorem 24). Here, in the case of non-plane trees this corresponds to the powers of unlabelled non-plane rooted trees. Although many results about forest 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 28. *The mean size $\bar{U}_n$ of a semantic tree induced by a process (unlabelled non-plane rooted) tree of size n admits the following asymptotics:*

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

where η and γ are introduced in Fact 16.

5.3 The mean size as a linear recurrence

In this section, we focus on the asymptotics of the average size $\bar{S}_n$ of the semantic trees induced by syntax trees of size n. Our goal is to obtain more precise approximations than Theorem 26 using different analytic combinatorics techniques. Indeed, we present three distinct ways to establish our main result: a linear recurrence that precisely capture the desired quantity. These results are deeply related to the holonomy property of the generating functions into consideration. Thus, a priori, in the non-plane case, the functions are not holonomic and consequently such proofs could not be adapted.

Theorem 29. *The mean size $\bar{S}_n$ of a semantic tree induced by a tree of size n follows the P-recurrence:*

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

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, at A216234. It consists to S_n: the cumulated sizes of semantic trees issued of process trees of size n.

Proof. A first approach to prove this theorem is based on creative telescoping. This proof is a direct consequence of the level decomposition detailed in Section 4. It is clearly simple both in terms of the technical mathematics involved and the level of computer assistance required for the demonstration. In particular, it needs no proof in the sense that all the steps are totally automatized in classical computer algebra systems (such a 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 on an average semantic tree:

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

This sum can be expressed in terms of hypergeometric functions:

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

Now, by using the package Mgfun of Maple [Chy98], we extract, by creative telescoping [PWZ96, Zei90], the stated P-recurrence for $\bar{S}_n$. □

However, we can prove Theorem 29 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.

Finally, the last approach is based on the concept *guess-and-proof*: 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 asymptotics in a relatively effortless way.

Theorem 30. *The mean size $\bar{S}_n$ of a semantic tree induced by a tree of size n admits the following precise asymptotics:*

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

Proof. We can derive more directly this result from the hypergeometric expression. 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 $\frac{n!}{2^{n-1}}\, {}_1F_1(-2\,n+1;\ -n+1/2;\ 1/2)$. Let us observe that the next hypergeometric function ${}_1F_1(-2\,n+1;\ -n+1/2;\ x)$ admits the following expansion:

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

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

Let us remark that it also possible to reach this asymptotic from 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 semantic 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 $\bar{S}_n 2^{n-1}/n!$ which is also holonomic and verifies:

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

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

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

with $R_0=0, R_1=1$ and $R_2=2$. Now, we can easily prove that this recurrence is convergent. Indeed, the recurrence 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 26 shows that the series converge 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.

Another way consists in predicting that the asymptotic expansion of $R(z)$ as z tends to the 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) to conclude. □

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 uniform probability of a concurrent run prefix. 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 semantic trees. An important remark is that these algorithms continue to apply whether we consider the plane or the non-plane case, only the average quantities are impacted.

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 semantic 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 31. *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 prefix traversal of T.*

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

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 32.

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

The proof directly derives from the hook-length formula (cf. Fact 13).

Corollary 33. *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 34. *Let T be a process tree. The probability of a prefix run $\sigma = \langle \alpha_1, \ldots, \alpha_p \rangle$ in the shuffle 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 shuffle of T
$\rho_\sigma := 1$
$i := 1$
for *i from 1 to $p-1$* **do**
  $\rho_\sigma := \rho_\sigma \times \frac{|T(\alpha_{i+1})|}{n-i}$
  $i := i+1$
return ρ_σ

From Corollaries 33 and 34 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 semantic 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 35. *Algorithm 1 computes ρ_σ 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 semantic tree). Then we have the following result.

Proposition 36. *The number of concurrent runs of a process tree T is $1/\rho_\sigma$ when σ is any complete traversal of T.*

As a matter of fact, Algorithm 1 provides us *for free* a way to compute from a syntax tree T the number of concurrent runs ℓ_T in the corresponding semantic tree. For this we simply have to compute the probability of a full run σ (we might select an arbitrary traversal of T) and then we obtain $\ell_T = 1/\rho_\sigma$.

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

Corollary 37. *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. The problem has a trivial solution if we work on a semantic 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 semantic tree. The challenge, thus, is to find a solution which does not require the explicit construction of the semantic 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 sub-quadratic 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 sub-tree 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 sub-tree 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 (with probability 1 thus cardinality n the size of the process tree T). Then it is added to σ and removed from the multiset. Finally its children are enabled with the weight as cardinality. 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 38. *At the p-th step of the algorithm, we have:*

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

Proposition 39. *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 detailed proof). By Invariant 38 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 32 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, we have the following complexity results.

Proposition 40. *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. In the next step, we shall address other typical constructs of formalisms for concurrency, especially *non-deterministic choice* and *synchronization* [AI07]. There are indeed various forms of synchronization, in general corresponding to reflecting the action labels within the pure merge operator. 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 the semantic trees by identifying common sub-trees. 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 semantic 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 cut to study the number of nodes by level in the semantic trees. We expect the discovery of similar increasingly labelled structures while 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 semantic 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 semantic trees. Conversely, we think it is interesting to know – precisely, not just by intuition – the high-level of sharing and symmetry within semantic 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 study.

Acknowledgements. We are grateful to M. Dien and O. Roussel for fruitful remarks about the algorithms.

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.

[Bli89] Wayne 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.

[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.

[GDG+08] M.-C. Gaudel, A. Denise, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet. Coverage-biased random exploration of models. In *ETAPS Workshop on Model Based Testing*, volume 220, pages 3–14. Electr. Notes Theor. Comput. Sci., 2008.

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

[KMPW12] J. Kim, K. Mészáros, G. Panova, and D. Wilson. Dyck tilings, linear extensions, descents, and inversions. *DMTCS Proceedings*, 0(01), 2012.

[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.

[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.

[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 ordering $\alpha_1 < \ldots < \alpha_n$. Now we define σ_M such that $\forall i \in [1..n],\ \forall j \in \left[\sum_{p=1}^{i-1} k_p + 1 .. \sum_{p=1}^{i-1} k_p + k_i\right],\ \sigma_M(j) = \alpha_i$,

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 $\beta = \sigma_M(\rho)$;

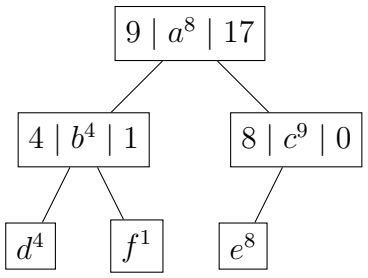


Figure 6: A partial sum tree for multiset $M_2 = \{\!\{a^8, b^4, c^9, d^4, f^1, e^8\}\!\}$.

and 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 6 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 sub-trees 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-sub-tree, denoted T_L, of T_M then we pick the element in this left-sub-tree. 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-sub-tree T_R without forgetting to update ρ as $\rho - (|L| + k)$ in the recursive calls.

Proposition 41. *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 42. *Let M a multiset. The PST random sampler returns an element $\alpha \in M$ with probability $\frac{M(\alpha)}{|M|}$.*

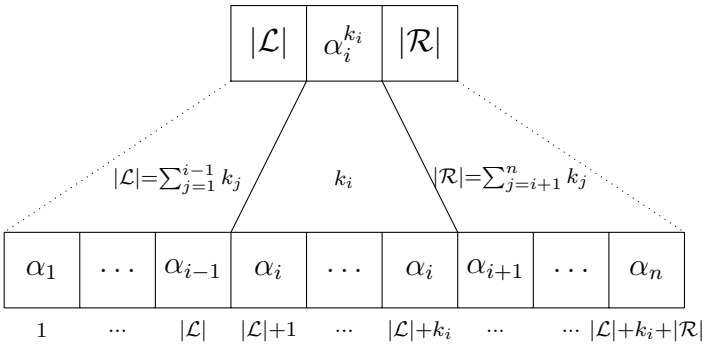


Figure 7: Mapping from a node of the partial sum tree to its corresponding infix sequence.

Proof. Let $M = \{\!\{\alpha_1^{k_1}, \ldots, \alpha_n^n\}\!\}$ a multiset and let denote by T_M the its partial sum tree representation. If we assume an implicit ordering $\alpha_1 < \ldots < \alpha_i < \ldots < \alpha_n$ then the infix 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 infix sequence. The tree T_L is the left sub-tree corresponding to a multiset L with infix 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 sub-tree corresponding to multiset R with infix 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 7. Thus the left sub-tree 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 sub-trees 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 infix sequence of a binary tree is the sequence of its elements ordered according to the infix 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 infix - ordered sequence – it is by no means a strict requirement. The proof can easily - albeit uninterestingly - adapt to any permutation in the selected order.