The Combinatorics of Non-determinism*

Olivier Bodini[1], Antoine Genitrini[2], and Frédéric Peschanski[2]

1 **Laboratoire d'Informatique de Paris-Nord, CNRS UMR 7030 – Institut Galilée – Université Paris-Nord, Villetaneuse, France**
`Olivier.Bodini@lipn.univ-paris13.fr`

2 **Laboratoire d'Informatique de Paris 6, CNRS UMR 7606 and Université Pierre et Marie Curie, Paris, France**
`{Antoine.Genitrini,Frederic.Peschanski}@lip6.fr`

Abstract

A deep connection exists between the interleaving semantics of concurrent processes and increasingly labelled combinatorial structures. In this paper we further explore this connection by studying the rich combinatorics of partially increasing structures underlying the operator of non-deterministic choice. Following the symbolic method of analytic combinatorics, we study the size of the computation trees induced by typical non-deterministic processes, providing a precise quantitative measure of the so-called "combinatorial explosion" phenomenon. Alternatively, we can see non-deterministic choice as encoding a family of tree-like partial orders. Measuring the (rather large) size of this family on average offers a key witness to the expressiveness of the choice operator. As a practical outcome of our quantitative study, we describe an efficient algorithm for generating computation paths uniformly at random.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Concurrency theory, Analytic combinatorics, Non-deterministic choice, Partially increasing trees, Uniform random generation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.425

1 Introduction

The mathematical structures underlying concurrency theory are most often considered as *abstract* entities from an algebraic point of view, and are much less often scrutinized under the meticulous lens of *concrete mathematics*. From the combinatorial point of view, even very basic process operators prove quite intricate. Indeed, we show in [3] that the study of the classic merge (or interleaving) operator requires non-trivial analytic combinatorics techniques.

In this paper we study the rich combinatorics of *partially increasing structures* underlying the operator of *non-deterministic choice* [11]. Note that this is a major departure from related studies involving regular shuffle in automata theory, e.g. [12]. Although we are still far from a full-fledged process algebra, the increase of expressiveness if compared to pure merge is noticeable. When unfolded as computation trees, the resulting process behaviours prove much more difficult to deal with. We show in this paper, however, that the analytic techniques we rely on – based on the *symbolic method* of *analytic combinatorics* [9] – scale relatively well to cover this enriched model. In Section 3 we give precise – average-case – results concerning the number of computation paths induced by typical non-deterministic

[*] This research was partially supported by the CNRS project *ALPACA* (PEPS INS2I 2012–2013) and by the A.N.R. project *MAGNUM*, ANR 2010-BLAN-0204.

© Olivier Bodini, Antoine Genitrini, and Frédéric Peschanski;
licensed under Creative Commons License CC-BY

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 425–436
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

processes, hence providing a quantitative measure for the so-called “combinatorial explosion” phenomenon. As we make clear in the paper, this phenomenon is not only a worst-case situation but it is also quite perceptible in the average case.

Our quantitative study also establishes deep links between the combinatorics of increasing structures and partial orders. Indeed, we can interpret non-deterministic choice as an encoding of a family of tree-like partial orders or *tree-posets* [2]. Measuring the (indeed exponential) average size of this family on average offers a key witness to the expressiveness of the choice operator. This is discussed in Section 4.

As a practical outcome of our quantitative study, we aim at developing techniques to analyze properties of the computation trees without explicitly constructing them. As an illustration, we describe in Section 5 an efficient algorithm for the generation of non-deterministic computation paths uniformly at random directly from the syntax of process specification. This algorithms is based on a compact polynomial representation of the uniform distribution of the computation paths.

2 Processes as combinatorial structures

In this section we define the mathematical objects that represent the syntax of process specifications on the one side, and the semantics of process behaviours on the other side.

2.1 Syntax: process trees

The syntax of formal languages is most often defined as a context-free grammar yielding tree structures: (abstract) syntax trees that will be called *process trees* in what follows. For the minimalist process calculus we discuss in this paper, the (semi-formal) grammar of process trees is the following :

$$\left[\begin{array}{lcll} P & ::= & P_{\parallel} \mid P_{+} & \text{(process)} \\ P_{\parallel} & ::= & \alpha \mid \alpha.\,(P \parallel \ldots) & \text{(prefixed parallel)} \\ P_{+} & ::= & P_{\parallel} + P_{\parallel} + \ldots & \text{(non-deterministic choice)} \end{array}\right.$$

If compared to most algebraic-oriented presentations, we do not use binary constructors. The main reason is that the parallel and choice operators are associative. We could of course encode the process terms using binary syntactic trees. However, if this is quite transparent for the tree structure, the encoding would have a sizable (and somewhat gratuitous) impact on the analytic developments. For similar reasons the prefixing construct is intermixed with the merge constructor for parallel processes, which means that each branch of a choice is prefixed. Hence we consider in essence what is often called *guarded choice* in the literature. To avoid an ambiguity in the model, we also require a choice to have at least two sub-processes.

The process specification that we will use as a running example in the paper is the following one: $a.\big(\ [b.(c \parallel d)+e] \parallel f.\,([g+h.(i \parallel j)+k] \parallel l)\ \big)$

Note that we use distinct labels for actions because our purely structural study does not reflect on the identity nor the nature of the atomic actions performed by the processes. So we do not distinguish among e.g. internal vs. external choice. In fact, we study the *effect* of non-determinism (i.e. the branching in computation trees) rather than its *cause*.

To formalize the process specifications as combinatorial objects, we rely on the *symbolic method* of analytic combinatorics [9]. Our starting point is the following combinatorial

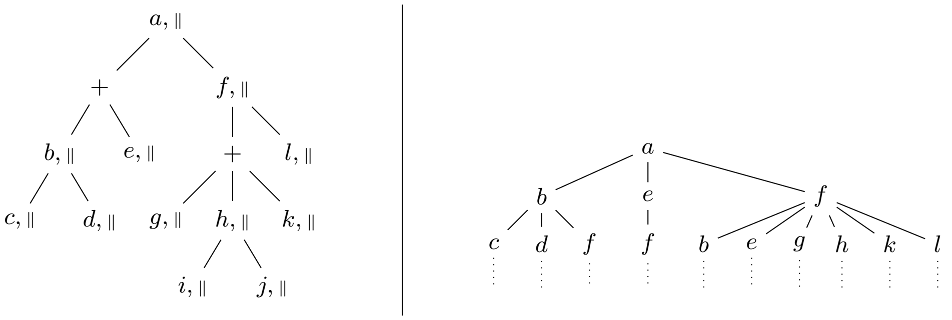

Figure 1 A process tree (left) and the first three layers of its computation tree (right).

specification of *process trees*:

$$\left[\begin{array}{lcll} \mathcal{A} & = & \mathcal{A}_{\parallel} + \mathcal{A}_{+} & \text{(process trees)} \\ \mathcal{A}_{\parallel} & = & \mathcal{Z} \times \textsc{Seq}\,(\mathcal{A}) & \text{(trees with action/parallel root, counted by } \mathcal{Z}\text{)} \\ \mathcal{A}_{+} & = & \mathcal{A}_{\parallel} \times \mathcal{A}_{\parallel} \times \textsc{Seq}(\mathcal{A}_{\parallel}) & \text{(trees with choice root)} \end{array}\right.$$

There is a strong correlation between this specification and the grammar discussed previously. The definition here is formal: this is the combinatorial class $\mathcal{A}$ of process trees. We remind the reader that a set is a combinatorial class if each element of the class has a known finite size. A further requirement is that there is only a finite number of elements in the class for a given size n. The $+$ operator on combinatorial classes denotes disjoint union. Thus a process tree is defined here as being either a merge tree in class $\mathcal{A}_{\parallel}$ or a non-deterministic choice in class A_{+}. A merge tree has a labelled root and its sub-trees form a (possibly empty) finite forest, as specified by the Seq constructor. The $\mathcal{Z}$ mark explains that the label of the root node is counted and thus the size of a merge tree is the sum of the sizes of its sub-trees plus one. Finally, the non-deterministic choices are trees with a root that is not counted for the size and a sequence of at least two merge sub-trees. Since they are not counted, the choice nodes cannot nest otherwise arbitrarily large sub-trees of the same size could be constructed. In the same spirit, the prefixing of parallel nodes with actions is justified to avoid arbitrarily large intermixes of operators. Hence, as noted previously, there is a precise combinatorial motivation for guarding the choices.

The process tree representing the example given above is depicted on the left-hand side of Figure 1. For the sake of clarity we explicitly mark the nodes with $\parallel$ and $+$ labels but only the action could be shown without any ambiguity. According to the specification for class $\mathcal{A}$ given above, this tree has size 12 (only counting the atomic actions).

Following the principles of analytic combinatorics, the combinatorial class $\mathcal{A}$ admits a counting sequence A_n consisting of the number of objects of $\mathcal{A}$ of size n. This sequence is linked to a formal power series $A(z)$ such that $A(z) = \sum_{n \geq 0} A_n z^n$. The n-th coefficient of $A(z)$ is commonly denoted by $[z^n]A(z) = A_n$. Various analysis techniques can then deployed to "dissect" such power series, e.g. study convergence, find closed formulas and derive asymptotic results. Indeed, the class $\mathcal{A}$ is a quite classical tree model, as e.g. studied in [9, chapter I] .

▶ Fact 1. The combinatorial class $\mathcal{A}$ of process trees satisfies:

$$A(z) = \frac{1}{2}\left(1 - z - \sqrt{1 - 6z + z^2}\right); \qquad A_n \sim_{n\to\infty} \sqrt{\frac{3\sqrt{2} - 4}{4\pi n^3}}\left(3 - 2\sqrt{2}\right)^{-n}.$$

The integer sequence A_n (precisely, shifted by 1) is known as *Large Schröder Numbers* and appears naturally in lattice paths' context [13]. See for example [9, p. 475], or the *Online Encyclopedia of Integer Sequence*: OEIS A006318.

2.2 Semantics: computation trees

A classical semantic domain for process behaviours is the class of *computation trees* that we intend to study as combinatorial structures, a study that will cover most of the remaining material of this paper.

From an algebraic point of view, computation trees can be characterized quite concisely, for example using the following inference rules:

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\,(act) \qquad \frac{P_i \xrightarrow{\alpha_i} P'_i \quad 1 \leq i \leq n}{P_1 + \ldots + P_i + \ldots + P_n \xrightarrow{\alpha_i} P'_i}\,(sum)$$

$$\frac{P_i \xrightarrow{\alpha_i} P'_i \quad 1 \leq i \leq n}{P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_n \xrightarrow{\alpha_i} P_1 \parallel \ldots \parallel P'_i \parallel \ldots \parallel P_n}\,(par)$$

In combinatorics, these rules are interpreted as the transformation of a syntactic process tree T into a computation tree, denoted $[\![T]\!]$, with quite a constrained structure. In the special case of a process without any occurrence of the choice operator – as thoroughly studied in [3] – then each branch of the corresponding computation tree is a full traversal of the initial process. With the choice operator, one must first select exactly one branch for each choice – an operation we call a *global choice* – and only then traverse the resulting "choice-free" process. On the right-hand side of Figure 1 we depict the first three layers of the computation tree corresponding to our example process. Each node of the tree corresponds to a labelled transition that can be proved by the inference rules above. We further abstract from the process states since the transitions capture the required information. Note that the computation tree of Figure 1 is not represented entirely since it has a total of 1120 leaves! The calculation is detailed in Section 5.2.

There is a fruitful reinterpretation of this semantic construction in terms of partial orders. With pure parallel processes, the process trees can be seen as tree-like partial orders or *tree-posets*, as studied in e.g. [2]. The associated computation trees then encode the sets of their *linear extensions*, i.e. the strict orderings induced by the posets. In the presence of the non-deterministic choice, the process trees encode a family of tree-posets (one for each possible global choice) and the associated computation tree is the combination of all their linear extensions. For example, if we take the process tree of Figure 1, then $\langle a, e, f, l, g\rangle$ is a valid linear extension, whereas $\langle a, f, l, e, b, d, c\rangle$ is not because b and e occur in distinct branches of a choice node, and should thus be mutually exclusive. Also, $\langle a, e, l, f, g\rangle$ is invalid since f precedes l in the partial order.

3 Quantitative study I – number of computation paths

The understanding of the computation trees generated by non-deterministic processes as combinatorial objects requires a meaningful notion of size. The measure that conveys the most important information about the computation trees is their number of leaves, or equivalently the number of computation paths. Indeed, this measure directly relates (asymptotically, by a constant factor) to most other natural measures such as the total number of internal nodes (this is discussed at length in [3]).

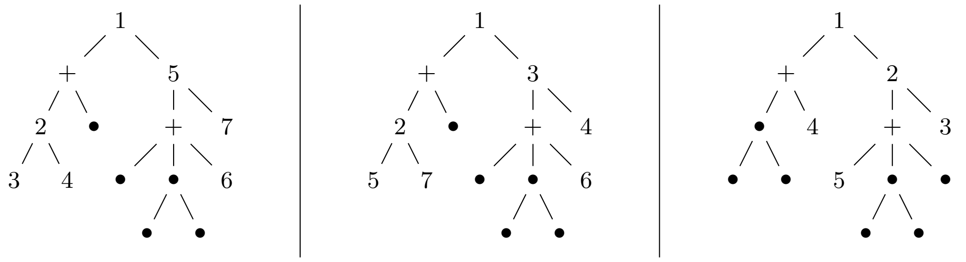

Figure 2 Three partially increasing trees based on the process tree of Figure 1.

3.1 Partially increasing trees

Consider T a process tree of size n. In [3] we show that the corresponding computation tree $[\![T]\!]$ has as many computation paths as the number of distinct ways to label the process tree T with increasing labels in range $[1..n]$.

If we add the choice operator, then the isomorphism with increasing trees is less direct since only the nodes that have been selected in a global choice must be labelled: exactly one branch for each choice node. All the other branches must be unselected in the resulting total number of possible computation paths. For this, we relax the total increasing labelling by allowing unlabelled nodes in the counting. To illustrate this combinatorial model, three distinct partial labellings for our running example are depicted on Figure 2. For example the leftmost case corresponds to a possible increasing labelling when the b and k branches are taken (a global choice labelled $\{b, k\}$). The unselected branches are labelled by the silent mark $\bullet$. Note that there are 1117 other possibilities of such partially increasing trees to count the total number of computation paths induced by the process tree of Figure 1.

The symbolic method can once again be used to formally define the combinatorial class of the partially increasing trees, denoted by $\mathcal{B}$, as follows:

$$\left[\begin{array}{lcl} \mathcal{B} & = & \mathcal{B}_{\parallel} + \mathcal{B}_{+} \\ \mathcal{B}_{\parallel} & = & \mathcal{W}^{\Box_{\mathcal{W}}} \star \mathcal{Z} \times \textsc{Seq}(\mathcal{B}) \\ \mathcal{B}_{+} & = & (\mathcal{B}_{\parallel} \times \mathcal{A}_{\parallel} \times \textsc{Seq}(\mathcal{A}_{\parallel})) + (\mathcal{A}_{\parallel} \times \textsc{Seq}(\mathcal{A}_{\parallel}) \times \mathcal{B}_{\parallel} \times \textsc{Seq}(\mathcal{A}_{\parallel})) \end{array}\right.$$

The specification above uses two distinct counting variables: $\mathcal{Z}$ for all the nodes and $\mathcal{W}$ only for the increasingly labelled nodes. A partially increasing tree in $\mathcal{B}$ may either be with a parallel or a choice root node, respectively in class $\mathcal{B}_{\parallel}$ and $\mathcal{B}_{+}$. The root node of a parallel node has two counts: one (totally) increasing in $\mathcal{W}$ and one for the total size in $\mathcal{Z}$ (including the unselected sub-trees marked with $\bullet$ in Figure 2). Its sub-trees consists of partially increasing trees in $\mathcal{B}$. The *box* notation $\mathcal{W}^{\Box_{\mathcal{W}}}$ specifies that the labelling counted by $\mathcal{W}$ must be increasing. This box operator introduces non-trivial differential functional equations, cf. [9, p. 139] for further details. The choice nodes can be formed out of two possibilities depending on where we select the branch in the semantics. In the first case, the leftmost branch of the choice has been selected, and thus below we need a partially increasing parallel tree in $\mathcal{B}_{\parallel}$ (we remind the reader that choice nodes cannot nest directly). The rest of the sub-trees are in $\mathcal{A}_{\parallel}$, which means that they are not labelled. The other possibility is that the branch we select is not the first one. This decomposition is required because a choice node must have at least two sub-trees to avoid any ambiguity with parallel nodes with only one sub-tree.

The most notable characteristic of partially increasing trees is that their specification mixes the box operator together with the unlabelled variant of the classical combinatorial constructs. To our knowledge, this is the first study of such a mixed labelled/unlabelled combinatorial class, which represents, we believe, a contribution in the field of analytic combinatorics.

3.2 Average case analysis

The combinatorial class of partially increasing trees can be directly translated to the following system of generating functions:

$$\left[\begin{array}{ll} B_{\|}(z,w) & = \int_0^w \frac{z}{1-B_{\|}(z,t)-B_{+}(z,t)}\, dt \\ B_{+}(z,w) & = \frac{B_{\|}(z,w)\cdot A_{\|}(z)}{1-A_{\|}(z)} \cdot \left(1 + \frac{1}{1-A_{\|}(z)}\right). \end{array}\right.$$

Here, the generating functions are bivariate with variable z for marking all the nodes and w marking only the increasingly labelled ones. We see here also the interpretation of the box operator as an integral. As usual with the symbolic method, the passage from the specification to the generating functions is completely automatic.

This system of generating functions is not trivial to study in analytic combinatorics. Resolving it requires the use of *holonomy theory* and related advanced techniques. In fact a computer algebra system must be used because some calculations are very intricate. We do however get a workable result.

▶ **Theorem 2.** *The generating function that enumerates the accumulated number of computation paths induced by all the process trees of a given size n is:*

$$\hat{B}(z) = \int_{w=0}^{\infty} (B_{\|}(z,w) + B_{+}(z,w)) \exp(-w) dw.$$

This function satisfies a linear homogeneous differential equation[1] *of order* 9 *whose coefficients are polynomials with highest degree* 15.

Now, using the holonomic equation of $\hat{B}(z)$, we can compute very efficiently its first terms (several thousands of terms can be obtained in a few seconds). For a more general complexity result, we are now interested in the asymptotic behaviour of the coefficients of $\hat{B}(z)$.

▶ **Theorem 3.** *Assuming the Hayman-admissibility of the function $\hat{B}$, the asymptotic of the average number of computation paths induced by all the process trees of a given size n is:*

$$\left(\frac{(6-4\sqrt{2})\,n}{e}\right)^n \left(\frac{e\sqrt{2\pi}}{\sqrt{3\sqrt{2}-4}} n^{1/2} + \frac{1}{48}\frac{e\sqrt{2\pi}\,(12\sqrt{2}-7)}{(3\sqrt{2}-4)^{3/2}} n^{-1/2} + O(n^{-3/2})\right).$$

Roughly speaking, a function $C(z)$ is *Hayman-admissible* if the distribution of law $(\mathbb{P}(N = n) = a_n z^n / C(z))_z$ tends to a Gaussian distribution when z tends to infinity. The Hayman-admissibility of $\hat{B}$ is unequivocal, although the complete proof requires the difficult Wasow's theory [14]. Beyond H-admissibility, the proof of the theorem is based on saddle point analysis, cf. [9, Section V] for similar technical developments.

In the case of pure merge trees, we show in [3] that the average number of computation paths is $(n-1)!/2^n = \Theta(\sqrt{n}(0.5 \cdot n/e)^n)$. In comparison, the asymptotics for the process trees with choice nodes is approximately $\Theta(\sqrt{n}(0.34315 \cdot n/e)^n)$. It is interesting to note that

[1] The differential equation cited in Theorem 2 is not informative in itself, it is thus left omitted.

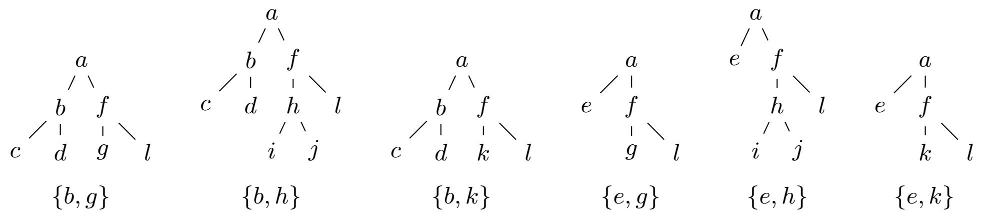

Figure 3 The global choices induced by the process tree of Figure 1.

the latter is exponentially smaller, which gives us a quantitative measure of the "cutting" effect induced by the choice operator. However, the reduction in size is not large enough for any algorithm requiring the explicit construction of the computation trees (even in some compacted form) to be of practical interest in general. Put in other terms, the combinatorial explosion phenomenon is thus not just a worst-case situation, but it is also quite perceptible in the average case.

4 Quantitative study II – average number of choices

In the previous section we obtained a precise although not really surprising negative result about the average number of computation paths. We now aim at measuring the choice itself. As explained previously, the choice operator – when interpreted globally – can be seen as encoding a family of computation trees resulting from choice-free process trees. In this section we study the average size of this family. This provides a rather precise characterization of the expressive power of the choice operator: the amount of information it encodes.

4.1 Generalized hook length formula

Let T be a process tree. A *global choice* of T is obtained by selecting exactly one sub-tree for each choice node of T. In Figure 3 we describe the set of all possible global choices for the process tree of Figure 1. There are indeed 6 possible global choices depending on the pairs of branches selected for the two choice nodes. Each choice can be identified by the root labels of the two selected sub-trees.

The important question of interest is the number of such possible global choices that can be expanded from typical process trees. Indeed, if we find that there are only a few possible choices on average, like for the example in Figure 1, then one might expand these choices and apply the efficient algorithms developed in [3] to analyze the computation trees without constructing them explicitly.

As a matter of fact, there is a tight connection between the possible number of such global choices, for a given process tree T, and the number of leaves of its computation tree $[\![T]\!]$ that we quantified in the previous section.

▶ **Lemma 4.** (Generalized hook length formula) *Let T be a process tree. The number ℓ_T of computation paths in $[\![T]\!]$ is given by the following formula:*

$$\ell_T = \sum_{C \text{ global choice of } T} \frac{|C|!}{\prod_{S \text{ sub-tree of } C} |S|}.$$

This theorem is justified as follows. If we make a global choice in a process tree (or if there is no choice to make), then we obtain a pure merge tree and in this case the number of computation paths is given by the "standard" hook length formula (cf. [3]). Then by summing over all the possible global choices we obtain the desired result.

4.2 Choice expansion

Our objective is to count, for average process trees, the number of summands in the formula of Lemma 4. To this end we start with the following specification:

$$\left[\begin{array}{lcl} \bar{\mathcal{A}} & = & \bar{\mathcal{A}}_{\parallel} + \bar{\mathcal{A}}_{+} \\ \bar{\mathcal{A}}_{\parallel} & = & \mathcal{Z} \times \mathcal{Y} \times \textsc{Seq}\left(\bar{\mathcal{A}}\right) \\ \bar{\mathcal{A}}_{+} & = & \bar{\mathcal{A}}_{\parallel} \times \mathcal{A}_{\parallel} \times \textsc{Seq}(\mathcal{A}_{\parallel}) + \mathcal{A}_{\parallel} \times \textsc{Seq}(\mathcal{A}_{\parallel}) \times \bar{\mathcal{A}}_{\parallel} \times \textsc{Seq}(\mathcal{A}_{\parallel}) \end{array}\right.$$

Each tree in class $\bar{\mathcal{A}}$ is either a parallel or a choice tree. In the case of a parallel node, there are two distinct counts: $\mathcal{Z}$ counts the number of parallel nodes as previously, and $\mathcal{Y}$ only counts the nodes that are part of a given global choice. The sub-trees of parallel nodes are zero or more trees in $\bar{\mathcal{A}}$. In order to obtain a non ambiguous specification for $\bar{\mathcal{A}}_{+}$, we split it in two parts: either the first branch is in $\bar{\mathcal{A}}_{\parallel}$, in which case the second and possibly the others are in $\mathcal{A}$, or it is another branch that belongs to $\bar{\mathcal{A}}_{\parallel}$. In both cases, the choice nodes are not counted. Their sub-trees are formed by a single sub-tree in $\bar{\mathcal{A}}$, i.e. counted by both $\mathcal{Z}$ and $\mathcal{Y}$, and all the other sub-trees only counted by $\mathcal{Z}$ by combinatorial class $\mathcal{A}$ of "normal" computation trees, as defined in Section 2.

Working with the generating functions, we obtain the following result:

▶ **Theorem 5.** *The average number k of choices in a process tree of size n is, asymptotically, such that there are two constants A and B with $k = A \cdot B^n$. We have the following estimates*[2] *for the constants: $A \approx 1.4408$ and $B \approx 1.11062$. Moreover, the average size of a choice in a process tree of size n is, asymptotically, equal to $C \cdot n$ where $C \approx 0.49636$.*

This shows that on average the number of choices induced by non-deterministic processes is exponential. Thus, even if it is clearly more efficient to resolve the choices first and then work on the choice-free systems, this does not yield tractable algorithms in general, in the average (and not just the worst) case. From another perspective, this result appears to us as a fairly good quantitative witness of the expressiveness of the non-deterministic choice operator. The syntactic choice construct indeed provides a particularly succinct encoding at the semantic level of an arbitrarily large family of global choices.

5 Uniform random generation of computation paths

We describe in this section an algorithm to generate non-deterministic computation paths uniformly at random for a fixed non-deterministic process. This provides a basic building block for e.g. (uniform) random testing or statistical model checking (cf. [7]). As our quantitative study makes clear, the approaches of (1) constructing the computation trees or (2) expanding the non-deterministic choices first, both yield impractical algorithms even in the average case. Thankfully the symbolic method of analytic combinatorics leads to a more tractable approach.

[2] In Theorem 5 we have presented approximations instead of exact expressions for the constants, for the sake of brevity. In fact we have calculated the exact values but their expression is very intricate and does not bring any essential information.

5.1 Polynomial representation

Let T be a process tree, $T_1, T_2, \ldots$ its children and $\mathrm{act}(T)$ the root action of T. We specify, in the sense of the symbolic method, the set $\mathcal{S}(T)$ of all runs of T:

$$\left[\begin{array}{ll} \mathcal{S}(T) & = \mathcal{S}^{\|}(T) + \mathcal{S}^{+}(T) \\ \mathcal{S}^{\|}(T) & = \mathcal{Z}^{\Box}_{\mathrm{act}(T)} \star \mathrm{S}\textsc{eq}_i(\mathcal{S}(T_i)) \\ \mathcal{S}^{+}(T) & = \mathcal{S}^{\|}(T_1) \times \mathcal{S}(T_2)^{\|} \times \ldots \end{array}\right.$$

This is very similar to the specification of the semantic trees except that each parallel node is counted by a dedicated variable $\mathcal{Z}_{\mathrm{act}(T)}$ that records the action performed at the root of T. If a parallel node has a root action labelled a then the variable is denoted $\mathcal{Z}_a$. In fact, we consider the process T as a combinatorial class itself.

For our running example (cf. Figure 1), we get the following specification:

$$\mathcal{S}(T) = \mathcal{Z}_a^{\Box} \star \left(\left(\mathcal{Z}_b^{\Box} \star (\mathcal{Z}_c \times \mathcal{Z}_d) + \mathcal{Z}_e\right) \times \mathcal{Z}_f^{\Box} \star \left(\left(\mathcal{Z}_g + \mathcal{Z}_h^{\Box} \star (\mathcal{Z}_i \times \mathcal{Z}_j) + \mathcal{Z}_k\right) \times \mathcal{Z}_l\right)\right).$$

This is almost a *paraphrase* of the process tree under study. The idea, then, is to apply the *recursive method* of uniform random generation developed by Nijenhuis and Wilf [15] and latter generalized to combinatorial classes (cf. the papers [10, 16]). However this is not possible in a direct way, because, to our knowledge the box operator has not been fully integrated in these methods. For instance, even if we obtain a correct selection of the $\mathcal{Z}$'s, this would not tell us in which order the action labels must be taken. Thus we propose to adapt the recursive method by decomposing the approach in two distinct phases: (1) the selection of a global choice and (2) the generation of a computation path. For the first phase, we do not require to mark all the action labels but only those involving a non-deterministic choice, i.e. actions just below choice nodes. We thus apply a simple substitution on the specification, denoted as follows:

$$\mathcal{S}'(T) = S(T)\{\mathcal{Z}/\mathcal{Z}_v, y_w\mathcal{Z}/\mathcal{Z}_w\} \text{ label } w \text{ is at the root of a plus branch and } v \text{ otherwise.}$$

We thus keep a unique mark y_w for each branch the choices, and we anonymize all the other actions. Applied to our example, we obtain:

$$\mathcal{S}'(T) = \mathcal{Z}^{\Box} \star \left(\left(y_b\mathcal{Z}^{\Box} \star \mathcal{Z}^2 + y_e\mathcal{Z}\right) \times \mathcal{Z}^{\Box} \star \left(\left(y_g\mathcal{Z} + y_h\mathcal{Z}^{\Box} \star \mathcal{Z}^2 + y_k\mathcal{Z}\right) \times \mathcal{Z}\right)\right).$$

We now recall the principles of the recursive method, albeit adapted to fit the special requirements for the box operator. The generating function corresponding to a weighted specification $\mathcal{S}'$ is a polynomial defined inductively.

▶ Definition 6. Let $S'(T)$ be a weighted choice specification for a process tree T. The *polynomial* of T is $\mathscr{P}_{S'(T)}(x)$ in the single variable x depending on the parameters y_v's of $S'(T)$ and constructed according to the following rules:

$$\left[\begin{array}{ll} \mathscr{P}_{\mathcal{Z}^{\Box}\star\mathrm{SEQ}(S'_i)}(x) = & \int_0^x \prod_i \mathscr{P}_{S'_i}(t)dt \\ \mathscr{P}_{\mathrm{SEQ}_{\geq 2}(y_{v_i}\times S'_i)}(x) = & \sum_i y_{v_i} \cdot \mathscr{P}_{S'_i} \end{array}\right.$$

Note that in particular we have $\mathscr{P}_{Z^{\Box}\star\emptyset}(x) = x$. For our example we obtain:

$$\begin{aligned} \mathscr{P}(x) \quad &= \int_0^x \left[y_b \cdot \textstyle\int_0^t u^2du + y_e \cdot t\right] \times \left[\int_0^t \left(y_g \cdot u + y_h \cdot \textstyle\int_0^u v^2dv + y_k \cdot u\right) \times u\, du\right] dt \\ &= \tfrac{x^9}{405} \cdot y_by_h + \tfrac{x^7}{315} \cdot (5y_b(y_g + y_k) + 3y_ey_h) + \tfrac{x^5}{15} \cdot y_e(y_g + y_k) \end{aligned}$$

If compared to the specification $\mathcal{S}'(T)$ the polynomial $\mathscr{P}(x)$ provides a decomposition of the available global choices in terms of size given by the degree of each monomial in x. In our example, there are global choices of size 9, 7 and 5. The weight of each choice in term of the distribution on the computation paths corresponds to the coefficient for each monomial.

The algorithm to generate the polynomial $\mathscr{P}(x)$ only requires a single traversal of the initial process tree. Using a compact representation (based on directed acyclic graphs for sharing common sub-terms), the size of the polynomial is at most $\Theta(n^2)$. This is because in the worst case the maximum degree of this polynomial is n (the size of the process tree), and each of the coefficients contains at most n occurrences of the parameters y_v's.

5.2 Sampling algorithm

Algorithm 1: weighted random generation of a global choice

```
**Data**: a process tree $T$ and its polynomial $\mathscr{P}_T(x) = \sum_d \frac{x^d}{C_d}\lambda_d$
**Result**: a set $C = \{v \mid v \text{ a label of } T\}$ identifying a global choice in $T$
$\pi_T := \{n_d = \Gamma(\frac{x^d}{C_d}\lambda_d\{y_v \leftarrow 1 \mid v \text{ a label of } T\})\}$
Pick $m \in [1; \sum_d \pi_T(n_d)]$ at random
Select $d$ s.t. $S < m \leq S + n_d$ for $n_d \in \pi_T$ and $S = \sum_{d'<d} n_{d'}$ for $n_d, n_{d'} \in \pi_T$
**return** $V(\lambda_d) \overset{\text{def}}{=} \begin{cases} \{v\} \text{ if } \lambda_d = y_v \\ \bigcup_i \{V(\lambda_i)\} \text{ if } \lambda_d = \prod_i \lambda_i \\ V(\lambda_j) \text{ if } \lambda_d = \sum_i \lambda_i \text{ for } j \text{ identifying } w_j \end{cases}$
with $w_j$ taken randomly in $W \overset{\text{def}}{=} \{w_i \mid w_i = \lambda_i\{y_v \leftarrow 1 \mid v \text{ a label}\},\ \lambda = \sum_i \lambda_i\}$
```

Based on the polynomial representation discussed previously, Algorithm 1 is used to sample a global choice with the correct relative weight according to the uniform distribution of computation paths. We illustrate this process with the same example as previously. In the first step of the algorithm we use the polynomial $\mathscr{P}(x)$ to construct π_T, an integer partition of the total number of computation paths of the process tree T. Each element of the partition is calculated by taking the Γ-transform (in x) of a monomial (of a given degree d; $\Gamma(\alpha_d \cdot x^d) = \alpha_d \cdot d!$) where all the constants y_v's are simply replaced by 1's. For our example we obtain the partition $\{n_9 = 896, n_7 = 208, n_5 = 16\}$ (thus the total number of computation paths is $\sum_d \pi_T(n_d) = 1120$). In the next step, we sample an integer m in $[1; 1120]$, for example $m = 900$. Consequently, we select the monomial in x^7 (choices of size 7) with weight 208 in the partition π_T. We are thus considering the coefficient $\lambda_7 = 5y_b(y_g + y_k) + 3y_e y_h$ in the root polynomial.

To compute $V(\lambda_7)$ in the next step, we must eliminate the outermost sum, which corresponds to the third (and most complex) case in the computation. For this we use the distribution W as defined in the algorithm. Each summand is associated in distribution W to a given weight w_i obtained by substituting all the y_v's of the choices of a given degree d and summand λ_i. One of the w_i's is taken arbitrarily (because the choice is non-deterministic). In our example, suppose we have $w_1 = 10$ for the left side and $w_2 = 3$ on the right side for the two summands of λ_7. To choose uniformly one of the two summands, we can pick a random integer in $[1; 13]$, for example 9. We thus select the left summand $5y_b(y_g + y_k)$. The top-most product is eliminated by simply computing the V of its operands. For the left operand, the label b is selected and in the right operand the sum is eliminated as already explained. If at the end we select the label k then we obtain $V(\lambda_7) = \{b, k\}$ which identifies a single choice in the starting process tree T. This global choice corresponds to the tree named $\{b, k\}$ in Figure 3. Since we obtain a pure merge tree, the dynamic multiset random sampler of [3] can be used to obtain a computation path uniformly at random.

▶ **Theorem 7.** *Let T a process tree of size n. A computation path can be generated uniformly at random in $O(n^2)$ arithmetic operations and with $O(n^2)$ space complexity.*

The global choice is obtained from Algorithm 1 in time linear in the size of the polynomial $\mathscr{P}(x)$ (represented as a DAG) because each coefficient contains at most one occurrence of a parameter y_v. Once the choice is sampled, the generation of the linear extension from a choice-free process is achieved in time $O(n \log n)$ where n is the size of the sampled choice. Thus, the overall complexity of the proposed random generator is dominated by the construction of the polynomial $\mathscr{P}(x)$, which already involves $O(n^2)$ operations.

Random generation is only one of various questions we can answer thanks to the symbolic representation of the problem. For example, we can count the number of computation paths in quadratic time. Using a similar process, we can also compute the probability of a given computation prefix, which would be useful to guide the search of counter-example in statistical model-checking. The paper [3] gives the detail about these algorithmic variations, in the choice-free context.

6 Related work

The algebraic properties of "pure" non-deterministic processes have been extensively studied, cf. e.g. [1, 5]. However, the underlying *concrete* combinatorial objects remain mostly undiscovered.

Related forms of choice and shuffle (or parallel) operators naturally appear in the framework of regular languages. In terms of analytic combinatorics the regular shuffle on disjoint alphabets has already been studied with basic quantitative results in e.g. [8], and more thorough properties studied in the work of Mishna and Zabrocki [12]. They prove, interestingly, that the algebraic properties of the shuffle directly translates to the nature of the generating functions that encodes the enumeration problems. However the connection with the non-deterministic choice studied in the present paper is rather loose. First in regular language the non-deterministic automata can be determinized, which gives a too abstract view of concurrent processes in the general case. This indeed amounts to consider trace-equivalence *versus* bisimilarity when comparing languages vs. processes [11]. In both case a form of abstraction is proposed, although it is far more radical in the case of regular languages because one can simply forget about the tree structure of the process behaviours.

The application of uniform random generation of execution paths to software testing is discussed in [6]. The major difference is that the random generation is based on the semantic structure whereas in our approach only the (exponentially smaller) syntactic structure needs to be constructed.

The complexity of counting of linear extensions for arbitrary partial orders is ♯P-complete as shown by [4]. In [3] we show that for tree-like partial orders it is linear (whereas the previously known algorithm was quadratic [2]). With non-deterministic choices the present paper shows that a quadratic worst-case algorithm exists and we conjecture this is also a lower bound. We emphasize the fact that the algorithm computes the number of linear extensions for a potentially exponential number of tree-like partial orders.

7 Conclusion

The quantitative study of the non-deterministic choice operator is an important milestone towards our goal of reinterpreting concurrency theory from the analytic combinatorics point of view. In the next step in our study, we shall investigate various forms of synchronization, in general corresponding to reflecting the action labels within the semantics. This calls for a *non-strict* variant of increasing structures. This in turn would allow a deeper study of the *causes of non-determinism*, e.g. the combinatorial interpretations of internal choice vs. external choice.

Another interesting continuation of the work is to study the compaction of the computation trees by identifying common subtrees. This would amount to study the semantic trees up-to *bisimilarity*. Note that our algorithmic framework would not be affected by such study, since the explicit construction of the semantic trees (whether compacted or not) is not required.

Finally, as pointed out by a kind reviewer of the paper, the process trees we consider are finite trees whereas in practice more expressive classes must be considered. An important goal we seek next is to apply our algorithmic framework in the context of e.g. regular processes. We do think an approach in the spirit of partial-order unfoldings of processes is feasible.

References

1 L. Aceto. On relating concurrency and non-determinism. In *MFPS*, volume 598 of *LNCS*, pages 376–402. Springer, 1991.

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

3 O. Bodini, A. Genitrini, and F. Peschanski. Enumeration and random generation of concurrent computations. In *DMTCS AofA'12 proceedings*, pages 83–96, 2012.

4 G. Brightwell and P. Winkler. Counting linear extensions is #P-complete. In *STOC*, pages 175–181, 1991.

5 S. Christensen. *Decidability and decomposition in process algebras.* PhD thesis, University of Edinburgh, 1993.

6 A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet. Coverage-biased random exploration of large models and application to testing. *STTT*, 14(1):73–93, 2012.

7 M.-C. Gaudel et al. Coverage-biased random exploration of models. In *ETAPS Workshop on Model Based Testing*, volume 220, pages 3–14. ENTCS, 2008.

8 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.

9 P. Flajolet and R. Sedgewick. *Analytic Combinatorics.* Cambridge Univ. Press, 2009.

10 P. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theor. Comput. Sci.*, 132(2):1–35, 1994.

11 R. Milner. *A Calculus of Communicating Systems.* Springer Verlag, 1980.

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

13 R. P. Stanley. *Enumerative Combinatorics, Vol. 2.* Cambridge Univ. Press, 2001.

14 W. Wasow. *Asymptotic Expansions for Ordinary Differential Equations.* Dover phoenix editions. Dover, 2002.

15 H. S. Wilf and A. Nijenhuis. *Combinatorial algorithms : An update.* CBMS-NSF. Philadelphia, Pa. Society for Industrial and Applied Mathematics, 1989.

16 P. Zimmermann. Random generation of unlabelled combinatorial structures. Technical report, Algorithms Seminar, 1993–1994, volume 2381. INRIA, 1994.