# Space-efficient scheduling of stochastically generated tasks[☆]

Tomáš Brázdil[a,1], Javier Esparza[b], Stefan Kiefer[c,2,*], Michael Luttenberger[b]

[a] *Faculty of Informatics, Masaryk University, Brno, Czech Republic*
[b] *Institut für Informatik, Technische Universität München, Germany*
[c] *Department of Computer Science, University of Oxford, UK*

**Abstract**

We study the problem of scheduling tasks for execution by a processor when the tasks can stochastically generate new tasks. Tasks can be of different types, and each type has a fixed, known probability of generating other tasks. We present results on the random variable $S^\sigma$ modeling the maximal space needed by the processor to store the currently active tasks when acting under the scheduler $\sigma$. We obtain tail bounds for the distribution of $S^\sigma$ for both offline and online schedulers, and investigate the expected value $\mathbb{E}[S^\sigma]$.

*Keywords:* stochastic models, space-efficient scheduling, multithreaded programs, branching processes

## 1. Introduction

We study the problem of scheduling tasks where every task can stochastically generate a set of new subtasks. Tasks can be of different types, and each type has a fixed, known probability of generating new subtasks.

Systems of tasks can be described using a notation similar to that of stochastic context-free grammars. For instance

$$X \xhookrightarrow{0.2} \langle X, X\rangle \qquad X \xhookrightarrow{0.3} \langle X, Y\rangle \qquad X \xhookrightarrow{0.5} \emptyset \qquad Y \xhookrightarrow{0.7} \langle X\rangle \qquad Y \xhookrightarrow{0.3} \langle Y\rangle$$

describes a system with two types of tasks. Tasks of type $X$ can generate two tasks of type $X$, one task of each type, or zero tasks with probabilities $0.2$, $0.3$, and $0.5$,

[☆] A preliminary version of this work appeared at the 37th International Colloquium on Automata, Languages and Programming, ICALP 2010.

[*] Corresponding author

*Email addresses:* xbrazdil@fi.muni.cz (Tomáš Brázdil), esparza@in.tum.de (Javier Esparza), stefan.kiefer@cs.ox.ac.uk (Stefan Kiefer), luttenbe@model.in.tum.de (Michael Luttenberger)

[1] Supported by Czech Science Foundation, grant No. P202/10/1469.

[2] Supported by the EPSRC project *Automated Verification of Probabilistic Programs* and by a postdoctoral fellowship of the German Academic Exchange Service (DAAD).

respectively (angular brackets denote multisets). Tasks of type $Y$ can generate one task, of type $X$ or $Y$, with probability $0.7$ and $0.3$. Readers familiar with process algebra will identify this notation as a probabilistic version of Basic Parallel Processes [1, 2, 3].

Tasks are executed by one processor. The processor repeatedly selects a task from a pool of unprocessed tasks, processes it, and puts the generated subtasks (if any) back into the pool. The pool initially contains one task of type $X$, and the next task to be processed is selected by a *scheduler*. We study random variables modeling the time and space needed to *completely* execute a task $\tau$, i.e., to empty the pool of unprocessed tasks assuming that initially the pool only contains task $\tau$. We assume that processing a task takes one time unit, and storing it in the pool takes one unit of memory. So the *completion time* is given by the total number of tasks processed, and the *completion space* by the maximum size reached by the pool during the computation. The completion time has been studied in [4], and so the bulk of the paper is devoted to studying the distribution of the completion space for different classes of schedulers.

Our computational model is abstract, but relevant for different scenarios. In the context of search problems, a task is a problem instance, and the scheduler is part of a branch-and-bound algorithm (see e.g. [5]). In the more general context of multithreaded computations, a task models a thread, which may generate new threads. The problem of scheduling multithreaded computations space-efficiently on *multi*processor machines has been extensively studied (see e.g. [6, 7, 8, 9]). These papers assume that schedulers know nothing about the program, while we consider the case in which stochastic information on the program behaviour is available (obtained from sampling). We restrict ourselves to the case in which a task has at most two children, i.e., all rules $X \stackrel{p}{\hookrightarrow} \langle X_1, \ldots, X_n \rangle$ satisfy $n \leq 2$. This case already allows to model the forking-mechanism underlying many multithreaded operating systems, e.g. Unix-like systems.

We study the performance of *online* schedulers, which know only the past of the computation, and compare them with the *optimal offline* scheduler, which has complete information about the future. Intuitively, this scheduler has access to an oracle that knows how the stochastic choices will be resolved. The oracle can be replaced by a machine that inspects the code of a task and determines which subtasks it will generate (if any).

We consider task systems with completion probability 1 (in the context of search problems or multithreaded computations, a termination probability different from 1 usually indicates the presence of an error). These can be further divided into those with finite and infinite expected completion time, often called *subcritical* and *critical*. Many of our results are related to the probability generating functions (pgfs) associated to a task system. The functions for the example above are $f_X(x, y) = 0.2x^2 + 0.3xy + 0.5$ and $f_Y(x, y) = 0.7x + 0.3y$, and the reader can easily guess the formal definition. The completion probability is the least fixed point of the system of pgfs [10].

Our first results (Section 3) concern the distribution of the completion space $S^{op}$ of the optimal offline scheduler $op$ on a fixed but arbitrary task system with $\boldsymbol{f}(\boldsymbol{x})$ as pgfs (in vector form). We exhibit a very surprising connection between the probabilities $\Pr[S^{op} = k]$ and the *Newton approximants* to the least fixed point of $\boldsymbol{f}(\boldsymbol{x})$ (the approximations to the least fixed point obtained by applying Newton's method for ap-

proximating a zero of a differentiable function to $\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x} = \mathbf{0}$ with seed $\mathbf{0}$). This connection allows us to apply recent results on the convergence speed of Newton's method [11, 12], leading to tail bounds of $S^{op}$, i.e., bounds on $\Pr[S^{op} \geq k]$. We then study (Section 4) the distribution of $S^{\sigma}$ for an online scheduler $\sigma$, and obtain upper and lower bounds for the performance of *any* online scheduler in subcritical systems. The proof of this result suggests two ways of improving the bounds for special classes of task systems, and special classes of schedulers. We study *continuing* task systems, which are particularly natural in the context of multithreaded computation and queueing theory, and *light-first* schedulers, in which "light" tasks (loosely speaking, tasks whose progeny becomes extinguished in a short time) are chosen before "heavy" tasks, and obtain improved tail bounds.

*Related work.* Space-efficient scheduling for search problems or multithreaded computations has been studied in [5, 6, 7, 8, 9]. These papers assume that nothing is known about the program generating the computations. We study the case in which statistical information is available on the probability that computations split or die.

The theory of *branching processes* studies stochastic processes modeling populations whose members can reproduce or die [10, 13]. In computer science terminology, all existing work on branching processes assumes that the number of processors is *unbounded* [14, 15, 16, 17, 18, 19]. We study the 1-processor case, and to our knowledge we are the first to do so. The authors of [7] study, in a non-probabilistic setting, so-called *strict computations*, in which a task can only terminate after all the tasks it has (recursively) spawned have terminated. The optimal scheduler in this case is the *depth-first* scheduler, i.e., the one that completely executes the child task before its parent, resulting in the familiar stack-based execution. Under this scheduler our tasks are equivalent to special classes of recursive state machines [20] and probabilistic pushdown automata [21]. Recent results [22] can be used to analyze the completion space for such systems.

Last but not least, our results are strongly related to the area of probabilistic verification [23, 24]. They provide techniques and fast algorithms for the verification of properties of the form: the available memory (for storing tasks) suffices to carry out the computation with probability at least $p$ (for some given bound $p$).

*Structure of the paper.* The rest of the paper is structured as follows. The preliminaries in Section 2 formalize the notions from the introduction and summarize known results on which we build. In Section 3 we study the performance of optimal offline schedulers. Section 4 is dedicated to online schedulers. First we prove performance bounds that hold uniformly for all online schedulers, then we provide improved upper bounds for certain task systems, and then for certain schedulers. In Section 5 we obtain several results on the expected space consumption under different schedulers. Section 6 contains conclusions.

*Proofs.* The main body of the paper provides proof sketches of a number of theorems. Detailed proofs of all theorems can be found in the appendix.

## 2. Preliminaries

Let $A$ be a finite set. We regard elements of $\mathbb{N}^A$ and $\mathbb{R}^A$ as *vectors* and use boldface (like $\boldsymbol{u}, \boldsymbol{v}$) to denote vectors. The vector whose components are all 0 (resp. 1) is denoted by $\mathbf{0}$ (resp. $\mathbf{1}$). We use angular brackets to denote multisets and often identify multisets over $A$ and vectors indexed by $A$. For instance, if $A = \{X, Y\}$ and $\boldsymbol{v} \in \mathbb{N}^A$ with $\boldsymbol{v}_X = 1$ and $\boldsymbol{v}_Y = 2$, then $\boldsymbol{v} = \langle X, Y, Y \rangle$. We often shorten $\langle a \rangle$ to $a$. The set of multisets over $A$ containing at most 2 elements is denoted by $M_A^{\leq 2}$.

**Definition 1.** A *task system* is a tuple $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ where $\Gamma$ is a finite set of *task types*, $\hookrightarrow \subseteq \Gamma \times M_\Gamma^{\leq 2}$ is a set of *transition rules*, $\mathit{Prob}$ is a function assigning positive probabilities to transition rules so that for every $X \in \Gamma$ we have $\sum_{X \hookrightarrow \alpha} \mathit{Prob}((X, \alpha)) = 1$, and $X_0 \in \Gamma$ is the *initial type*.

Figure 1 (a) shows a task system with $\Gamma = \{X, Y, Z\}$. We write $X \overset{p}{\hookrightarrow} \alpha$ whenever $X \hookrightarrow \alpha$ and $\mathit{Prob}((X, \alpha)) = p$. Executions of a task system are modeled as family trees. Intuitively, a family tree is a tree whose nodes are tasks; a node is labeled with the type of its task. The initial task is the root, and the children of a task are the tasks generated by it, sorted according to some fixed total order on the task types. Formally, a *family tree* $t$ is a pair $(N, L)$ where $N \subseteq \{0,1\}^*$ is a finite binary tree (i.e. a prefix-closed finite set of words over $\{0,1\}$) and $L : N \hookrightarrow \Gamma$ is a labelling such that every node $w \in N$ satisfies one of the following conditions: $w$ is a leaf and $L(w) \hookrightarrow \emptyset$, or $w$ has a unique child $w0$, and $L(w)$ satisfies $L(w) \hookrightarrow L(w0)$, or $w$ has two children $w0$ and $w1$, and $L(w0)$, $L(w1)$ satisfy $L(w) \hookrightarrow \langle L(w0), L(w1) \rangle$ and $L(w0) \preceq L(w1)$, where $\prec$ is an arbitrary total order on $\Gamma$. Given a node $w \in N$, the subtree of $t$ rooted at $w$, denoted by $t_w$, is the family tree $(N', L')$ such that $w' \in N'$ iff $ww' \in N$ and $L'(w') = L(ww')$ for every $w' \in N'$. If a tree $t$ has a subtree $t_0$ or $t_1$, we call this subtree a *child* of $t$. (So, the term *child* can refer to a node or a tree, but there will be no confusion.) Figure 1 shows on the right a family tree of the task system on the left of the figure.

We define a function $\Pr$ which, loosely speaking, assigns to a family tree $t = (N, L)$ its probability (see *Assumptions* below). Assume that the root of $t$ is labeled by $X$. If $t$ consists only of the root, and $X \overset{p}{\hookrightarrow} \emptyset$, then $\Pr[t] = p$; if the root has only one child (the node 0) labeled by $Y$, and $X \overset{p}{\hookrightarrow} Y$, then $\Pr[t] = p \cdot \Pr[t_0]$; if the root has two children (the nodes 0 and 1) labeled by $Y$ and $Z$, and $X \overset{p}{\hookrightarrow} \langle Y, Z \rangle$, then $\Pr[t] = p \cdot \Pr[t_0] \cdot \Pr[t_1]$. We denote by $\mathcal{T}_X$ the set of all family trees whose root is labeled by $X$, and by $\Pr_X$ the restriction of $\Pr$ to $\mathcal{T}_X$. We drop the subscript of $\Pr_X$ if $X$ is understood.

For every task system $\Delta$, we define its *probability generating function (pgf)* as the function $\boldsymbol{f} : \mathbb{R}^\Gamma \to \mathbb{R}^\Gamma$ where for every $X \in \Gamma$

$$\boldsymbol{f}_X(\boldsymbol{v}) = \sum_{X \overset{p}{\hookrightarrow} \langle Y,Z \rangle} p \cdot \boldsymbol{v}_Y \cdot \boldsymbol{v}_Z + \sum_{X \overset{p}{\hookrightarrow} \langle Y \rangle} p \cdot \boldsymbol{v}_Y + \sum_{X \overset{p}{\hookrightarrow} \emptyset} p \,.$$

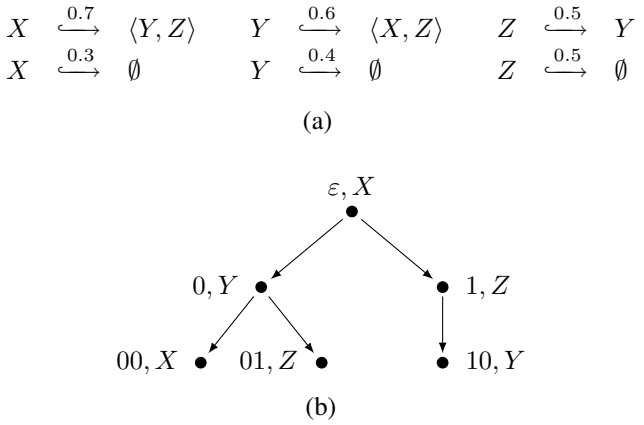


Figure 1: (a) A task system. (b) A family tree.

**Example 1.** Figure 1 shows (a) a task system with $\Gamma = \{X, Y, Z\}$; and (b) a family tree $t$ of the system with probability $\Pr[t] = 0.7 \cdot 0.6 \cdot 0.3 \cdot 0.5 \cdot 0.5 \cdot 0.4$. The name and label of a node are written next to it. The pgf is given by

$$\boldsymbol{f}(X, Y, Z) = \begin{pmatrix} \boldsymbol{f}_X(X, Y, Z) \\ \boldsymbol{f}_Y(X, Y, Z) \\ \boldsymbol{f}_Z(X, Y, Z) \end{pmatrix} = \begin{pmatrix} 0.7 \cdot Y \cdot Z + 0.3 \\ 0.6 \cdot X \cdot Z + 0.4 \\ 0.5 \cdot Y + 0.5 \end{pmatrix}$$

with task types used as corresponding variables on $\mathbb{R}$.

*Assumptions.* Throughout the paper we assume that a task system $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ satisfies the following two conditions for every type $X \in \Gamma$: (1) $X$ is *reachable* from $X_0$, meaning that some tree in $\mathcal{T}_{X_0}$ contains a node labeled by $X$, and (2) $\Pr[\mathcal{T}_X] = \sum_{t \in \mathcal{T}_X} \Pr[t] = 1$. So we assume that $(\mathcal{T}_X, \Pr_X)$ is a discrete probability space with $\mathcal{T}_X$ as set of elementary events and $\Pr_X$ as probability function. This is the formal counterpart to assuming that every task is completed with probability 1.

**Proposition 1.** *It can be decided in polynomial time whether assumptions (1) and (2) are satisfied.*

PROOF. (1) is trivial. It is well known (see e.g. [10]) that (2) holds if and only if the least nonnegative fixed point of $\boldsymbol{f}$ equals $\mathbf{1}$, which is decidable in polynomial time [20, 25]. □

*Derivations and schedulers.* Let $t = (N, L)$ be a family tree. A *state* of $t$ is a subset of $N$ in which no node is a proper prefix of another node (graphically, no node is a proper descendant of another node). The elements of a state are called *tasks*. If $s$ is a state and $w \in s$, then the *$w$-successor of $s$* is the uniquely determined state $s'$ defined as follows: if $w$ is a leaf of $N$, then $s' = s \setminus \{w\}$; if $w$ has one child $w0$, then $s' = (s \setminus \{w\}) \cup \{w0\}$;

if $w$ has two children $w0$ and $w1$, then $s' = (s \setminus \{w\}) \cup \{w0, w1\}$. We write $s \Rightarrow s'$ if $s'$ is the $w$-successor of $s$ for some $w$. A *derivation of* $t$ is a sequence $s_1 \Rightarrow \ldots \Rightarrow s_k$ of states such that $s_1 = \{\epsilon\}$ and $s_k = \emptyset$. A *scheduler* is a mapping $\sigma$ that assigns to a family tree $t$ a derivation $\sigma(t)$ of $t$. If $\sigma(t) = (s_1 \Rightarrow \ldots \Rightarrow s_k)$, then for every $1 \leq i < k$ we denote by $\sigma(t)[i]$ a task of $s_i$ such that $s_{i+1}$ is the $\sigma(t)[i]$-successor of $s_i$. Intuitively, $\sigma(t)[i]$ is the task of $s_i$ scheduled by $\sigma$. This definition allows for schedulers that know the tree, and so how future tasks will behave. In Section 4 we define and study online schedulers which only know the past of the computation. Notice that schedulers are deterministic (non-randomized).

**Example 2.** A scheduler may schedule the tree $t$ in Figure 1 as follows: $\{\varepsilon\} \Rightarrow \{0,1\} \Rightarrow \{0,10\} \Rightarrow \{0\} \Rightarrow \{00,01\} \Rightarrow \{01\} \Rightarrow \{\}$. The scheduler which always picks the least unprocessed task w.r.t. the lexicographical order on $\{0,1\}^*$ (an online scheduler), schedules $t$ differently, as follows: $\{\varepsilon\} \Rightarrow \{0,1\} \Rightarrow \{00,01,1\} \Rightarrow \{01,1\} \Rightarrow \{1\} \Rightarrow \{10\} \Rightarrow \{\}$.

*Time and space.* Given $X \in \Gamma$, we define a random variable $T_X$, the *completion time of* $X$, which assigns to a tree $t \in \mathcal{T}_X$ its number of nodes. Assuming that tasks are executed for one time unit before its generated subtasks are returned to the pool, $T_X$ corresponds to the time required to completely execute $X$. Our assumption (2) guarantees that $T_X$ is finite with probability 1, but its expectation $\mathbb{E}[T_X]$ may or may not be finite. A task system $\Delta$ is called *subcritical* if $\mathbb{E}[T_X]$ is finite for every $X \in \Gamma$. Otherwise it is called *critical*. If $\Delta$ is subcritical, then $\mathbb{E}[T_X]$ can be easily computed by solving a system of linear equations [4]. The notion of criticality comes from the theory of branching processes, see e.g. [10, 13]. Here we only recall the following results:

**Proposition 2 ([10, 20]).** *Let $\Delta$ be a task system with pgf $\boldsymbol{f}$. Denote by $\boldsymbol{f}'(\mathbf{1})$ the Jacobian matrix of partial derivatives of $\boldsymbol{f}$ evaluated at $\mathbf{1}$. If $\Delta$ is critical, then the spectral radius of $\boldsymbol{f}'(\mathbf{1})$ is equal to 1; otherwise it is strictly less than 1. It can be decided in polynomial time whether $\Delta$ is critical.*

The proposition essentially follows from statements in [10, 20], but we provide an explicit proof, in order to make the paper more self-contained.

PROOF. One can show (see e.g. [21]) that $\mathbb{E}[T_X]$ is the $X$-component of the least non-negative fixed point of $\boldsymbol{f}'(\mathbf{1})\boldsymbol{x} + \mathbf{1}$, i.e., the $X$-component of the (componentwise) least vector $\boldsymbol{x} \in [0,\infty]^\Gamma$ with $\boldsymbol{x} = \boldsymbol{f}'(\mathbf{1})\boldsymbol{x} + \mathbf{1}$. This least fixed point is given by $\sum_{i=0}^{\infty} (\boldsymbol{f}'(\mathbf{1}))^i \mathbf{1}$, a series that may or may not converge. It is a standard fact (see e.g. [26]) that the series converges iff $\rho(\boldsymbol{f}'(\mathbf{1})) < 1$ holds for the spectral radius $\rho(\boldsymbol{f}'(\mathbf{1}))$ of $\boldsymbol{f}'(\mathbf{1})$.

Assume first that $\Delta$ is subcritical. Then the above series must converge, so we have $\rho(\boldsymbol{f}'(\mathbf{1})) < 1$ in this case. Now assume that $\Delta$ is critical. Then the above series must diverge, so we have $\rho(\boldsymbol{f}'(\mathbf{1})) \geq 1$. On the other hand, in [12, 20] it is shown that $\rho(\boldsymbol{f}'(\mathbf{1})) \leq 1$. (More precisely, it is shown there that $\rho(\boldsymbol{f}'(\boldsymbol{y})) < 1$ holds for $\boldsymbol{y}$ that are strictly less than the least fixed point of $\boldsymbol{f}$. By continuity of eigenvalues, $\rho(\boldsymbol{f}'(\boldsymbol{y})) \leq 1$ also holds for the least fixed point of $\boldsymbol{f}$ which is $\mathbf{1}$ according to the proof of Proposition 1.) Hence we have $\rho(\boldsymbol{f}'(\mathbf{1})) = 1$.

In order to decide on the criticality, it thus suffices to decide whether the spectral radius of $\boldsymbol{f}'(\mathbf{1})$ is $\geq 1$. This condition holds iff $\boldsymbol{f}'(\mathbf{1})\boldsymbol{x} \geq \boldsymbol{x}$ holds for a nonnegative, nonzero vector $\boldsymbol{x}$ (see e.g. Thm. 2.1.11 of [27] and cf. [20]). This can be checked in polynomial time with linear programming. □

**Example 3.** For the task system of Figure 1, the spectral radius of $\boldsymbol{f}'(\mathbf{1})$ is $\approx 0.97$. Hence it is subcritical.

A state models a pool of tasks awaiting to be scheduled. We are interested in the maximal size of the pool during the execution of a derivation. So we define the random *completion space* $S^\sigma_X$ as follows. If $\sigma(t) = (s_1 \Rightarrow \ldots \Rightarrow s_k)$, then $S^\sigma_X(t) := \max\{|s_1|, \ldots, |s_k|\}$, where $|s_i|$ is the cardinality of $s_i$. Sometimes we write $S^\sigma(t)$, meaning $S^\sigma_X(t)$ for the type $X$ labelling the root of $t$. If we write $S^\sigma$ without specifying the application to any tree, then we mean $S^\sigma_{X_0}$.

**Example 4.** The schedulers of Example 2 have completion space 2 and 3, respectively.

*Running examples.* Throughout the paper we use two task systems as running examples. The first one is the task system of Figure 1, which we analyze numerically. The second one is actually a family of task systems: the family $X \overset{p}{\hookrightarrow} \langle X, X\rangle,\ X \overset{q}{\hookrightarrow} \emptyset$ with pgf $f(x) = px^2 + q$, where $0 < p \leq 1/2$ is a parameter and $q = 1 - p$. This family is very simple, which allows to interpret our results analytically. Notice that the probability $p$ satisfies $p \leq 1/2$: For $1/2 < p \leq 1$ the least fixed point of $f$ is $q/p < 1$, and so the system does not terminate with probability 1. For $p \leq 1/2$ the least fixed point of $f$ is 1, and the system is critical for $p = 1/2$, and subcritical for $p < 1/2$.

## 3. Optimal (Offline) Schedulers

Let $S^{op}$ be the random variable that assigns to a family tree the minimal completion space of its derivations. We call $S^{op}(t)$ the *optimal completion space* of $t$. The optimal scheduler assigns to each tree a derivation with optimal completion space. In the multithreading scenario, it corresponds to a scheduler that can inspect the code of a thread and decide whether it will spawn a new thread or not. Note that, although the optimal scheduler "knows" how the stochastic choices are resolved, the optimal completion space $S^{op}(t)$ is still a random variable, because it depends on a random tree. The following proposition characterizes the optimal completion space of a tree in terms of the optimal completion space of its children.

**Proposition 3.** *Let $t$ be a family tree. Then*

$$S^{op}(t) = \begin{cases} \min\left\{ \begin{array}{l} \max\{S^{op}(t_0)+1, S^{op}(t_1)\}, \\ \max\{S^{op}(t_0), S^{op}(t_1)+1\} \end{array} \right\} & \text{if } t \text{ has two children } t_0, t_1 \\ S^{op}(t_0) & \text{if } t \text{ has exactly one child } t_0 \\ 1 & \text{if } t \text{ has no children.} \end{cases}$$

PROOF. The only nontrivial case is when $t$ has two children $t_0$ and $t_1$. Consider the following schedulings for $t$, where $i \in \{0,1\}$: Execute first all tasks of $t_i$ and then all tasks of $t_{1-i}$; within both $t_i$ and $t_{1-i}$, execute tasks in optimal order. While executing $t_i$, the root task of $t_{1-i}$ remains in the pool, and so the completion space is $s(i) = \max\{S^{op}(t_i)+1, S^{op}(t_{1-i})\}$. Since the optimal scheduler can choose the value of $i$ that minimizes $s(i)$, we have $S^{op}(t) \leq \min\{s(0), s(1)\}$.

It remains to argue why the scheduler cannot save space by interleaving the schedulings for $t_0$ and $t_1$. Consider an optimal scheduling of $t$. Assume that the derivation of the $t_0$-tree is completed first. Then at least one task from $t_1$ terminates only after the derivation of $t_0$ is completed, so this scheduling needs space of at least $S^{op}(t_0)+1$. Since any scheduling of $t$ needs space of at least $S^{op}(t_1)$, we have $S^{op}(t) \geq s(0)$. Assume now that the derivation of the $t_1$-tree is completed first. Then, similarly, we have $S^{op}(t) \geq s(1)$. So, we obtain $S^{op}(t) \geq \min\{s(0), s(1)\}$, and, with the inequality above, $S^{op}(t) = \min\{s(0), s(1)\}$. □

Given a type $X$, we are interested in the probabilities $\Pr[S^{op}_X \leq k]$ for $k \geq 1$. Proposition 3 yields a recurrence relation which at first sight seems difficult to handle. However, using results of [28, 29] we can exhibit a surprising connection between these probabilities and the pgf $\boldsymbol{f}$.

Let $\boldsymbol{\mu}$ denote the least fixed point of $\boldsymbol{f}$ and recall from the proof of Proposition 1 that $\boldsymbol{\mu} = \mathbf{1}$. Clearly, $\mathbf{1}$ is a zero of $\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}$. It has recently been shown that $\boldsymbol{\mu}$ can be computed by applying to $\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}$ Newton's method for approximating a zero of a differentiable function [20, 11]. More precisely, $\boldsymbol{\mu} = \lim_{k\to\infty} \boldsymbol{\nu}^{(k)}$ where

$$\boldsymbol{\nu}^{(0)} = \mathbf{0} \quad \text{and} \quad \boldsymbol{\nu}^{(k+1)} = \boldsymbol{\nu}^{(k)} + (I - \boldsymbol{f}'(\boldsymbol{\nu}^{(k)}))^{-1}\left(\boldsymbol{f}(\boldsymbol{\nu}^{(k)}) - \boldsymbol{\nu}^{(k)}\right)$$

and $\boldsymbol{f}'(\boldsymbol{\nu}^{(k)})$ denotes the Jacobian matrix of partial derivatives of $\boldsymbol{f}$ evaluated at $\boldsymbol{\nu}^{(k)}$ and $I$ the identity matrix. Surprisingly, the sequence of approximants computed by Newton's method provides exactly the information we are looking for:

**Theorem 1.** $\Pr[S^{op}_X \leq k] = \boldsymbol{\nu}^{(k)}_X$ *for every type* $X$ *and every* $k \geq 0$.

*Proof sketch.* We illustrate the proof idea on the one-type task system with the pgf $f(x) = px^2 + q$, where $q = 1-p$. Notice that the "vectors" $f$ and $\nu^{(k)}$ and the "matrix" $f'$ have a single entry only, so they can be treated as numbers. Let $\mathcal{T}_{\leq k}$ and $\mathcal{T}_{=k}$ denote the sets of trees $t$ with $S^{op}(t) \leq k$ and $S^{op}(t) = k$, respectively. We show $\Pr[\mathcal{T}_{\leq k}] = \nu^{(k)}$ for all $k$ by induction on $k$. The case $k = 0$ is trivial. Assume that $\nu^{(k)} = \Pr[\mathcal{T}_{\leq k}]$ holds for some $k \geq 0$. We prove $\Pr[\mathcal{T}_{\leq k+1}] = \nu^{(k+1)}$. Notice that

$$\nu^{(k+1)} := \nu^{(k)} + \frac{f(\nu^{(k)}) - \nu^{(k)}}{1 - f'(\nu^{(k)})} = \nu^{(k)} + (f(\nu^{(k)}) - \nu^{(k)}) \cdot \sum_{i=0}^{\infty} f'(\nu^{(k)})^i.$$

Let $\mathcal{B}^{(0)}_{k+1}$ be the set of trees that have two children both of which belong to $\mathcal{T}_{=k}$, and, for every $i \geq 0$, let $\mathcal{B}^{(i+1)}_{k+1}$ be the set of trees with two children, one belonging to $\mathcal{T}_{\leq k}$, the other one to $\mathcal{B}^{(i)}_{k+1}$. By Proposition 3 we have $\mathcal{T}_{\leq k+1} = \bigcup_{i\geq 0} \mathcal{B}^{(i)}_{k+1}$. We prove

$\Pr\left[\mathcal{B}_{k+1}^{(i)}\right] = f'(\nu^{(k)})^i\,(f(\nu^{(k)}) - \nu^{(k)})$ by an (inner) induction on $i$, which completes the proof. For the base $i = 0$, let $\mathcal{A}_{\leq k}$ be the set of trees with two children in $\mathcal{T}_{\leq k}$; by induction hypothesis we have $\Pr[\mathcal{A}_{\leq k}] = p\nu^{(k)}\nu^{(k)}$. In a tree of $\mathcal{A}_{\leq k}$ either (a) both children belong to $\mathcal{T}_{=k}$, and so $t \in \mathcal{B}_{k+1}^{(0)}$, or (b) at most one child belongs to $\mathcal{T}_{=k}$. By Proposition 3, the trees satisfying (b) belong to $\mathcal{T}_{\leq k}$. In fact, a tree of $\mathcal{T}_{\leq k}$ either satisfies (b) or it has one single node. Since the probability of the tree with one node is $q$, we get $\Pr[\mathcal{A}_{\leq k}] = \Pr\left[\mathcal{B}_{k+1}^{(0)}\right] + \Pr[\mathcal{T}_{\leq k}] - q$. Applying the induction hypothesis again we obtain $\Pr\left[\mathcal{B}_{k+1}^{(0)}\right] = p\nu^{(k)}\nu^{(k)} + q - \nu^{(k)} = f(\nu^{(k)}) - \nu^{(k)}$. For the induction step, let $i \geq 0$. Divide $\mathcal{B}_{k+1}^{(i+1)}$ into two sets, one containing the trees whose left (right) child belongs to $\mathcal{B}_{k+1}^{(i)}$ (to $\mathcal{T}_{\leq k}$), and the other the trees whose left (right) child belongs to $\mathcal{T}_{\leq k}$ (to $\mathcal{B}_{k+1}^{(i)}$). Using both induction hypotheses, we get that the probability of each set is $p\nu^{(k)}f'(\nu^{(k)})^i(f(\nu^{(k)}) - \nu^{(k)})$. So $\Pr\left[\mathcal{B}_{k+1}^{(i+1)}\right] = (2p\nu^{(k)}) \cdot f'(\nu^{(k)})^i(f(\nu^{(k)}) - \nu^{(k)})$. Since $f(x) = px^2 + q$ we have $f'(\nu^{(k)}) = 2p\nu^{(k)}$, and so $\Pr\left[\mathcal{B}_{k+1}^{(i+1)}\right] = f'(\nu^{(k)})^{i+1}(f(\nu^{(k)}) - \nu^{(k)})$ as desired. □

**Example 5.** Computing Newton approximants for the task system from Figure 1 and applying Theorem 1 yields

$$\begin{array}{lcllcl} \Pr[S_X^{op} \leq 0] & = & 0 & \Pr[S_X^{op} \leq 3] & \approx & 0.88 \\ \Pr[S_X^{op} \leq 1] & = & 0.3 & \Pr[S_X^{op} \leq 4] & \approx & 0.96 \\ \Pr[S_X^{op} \leq 2] & \approx & 0.70 & \Pr[S_X^{op} \leq 5] & \approx & 0.99\,. \end{array}$$

**Example 6.** Consider the task system $X \overset{p}{\hookrightarrow} \langle X, X\rangle,\ X \overset{q}{\hookrightarrow} \emptyset$ with pgf $f(x) = px^2 + q$, where $0 < p \leq 1/2$ and $q = 1 - p$. Using Newton approximants we obtain the recurrence relation

$$\Pr[S^{op} \geq k+1] = \frac{p \cdot \Pr[S^{op} \geq k]^2}{1 - 2p + 2p \cdot \Pr[S^{op} \geq k]}$$

for the distribution of the optimal scheduler. In particular, for the critical value $p = 1/2$ we get $\Pr[S^{op} \geq k] = 2^{1-k}$.

Theorem 1 allows to compute the probability mass function of $S^{op}$. As a Newton iteration requires $\mathcal{O}(|\Gamma|^3)$ arithmetical operations for a task system with set of types $\Gamma$, we obtain the following corollary, where by the unit cost model we refer to the model in which arithmetic operations have cost 1, independently of the size of the operands [30].

**Corollary 1.** *$\Pr[S_X^{op} = k]$ can be computed in time $\mathcal{O}(k \cdot |\Gamma|^3)$ in the unit cost model.*

It is easy to see that Newton's method converges quadratically for subcritical systems (see e.g. [31]), meaning, roughly speaking, that each iteration doubles the number of accurate bits of $\Pr[S_X^{op} \geq k]$ computed so far. For critical systems, it has recently been proved that Newton's method still converges linearly [11, 12]. These results lead to tail bounds for $S_X^{op}$:

**Corollary 2.** *For any task system $\Delta$ there are real numbers $c > 0$ and $0 < d < 1$ such that $\Pr[S^{op}_X \geq k] \leq c \cdot d^k$ for all $k \in \mathbb{N}$. If $\Delta$ is subcritical, then there are real numbers $c > 0$ and $0 < d < 1$ such that $\Pr[S^{op}_X \geq k] \leq c \cdot d^{2^k}$ for all $k \in \mathbb{N}$.*

PROOF. By Theorem 1 we have $\Pr[S^{op} \geq k] = 1 - \boldsymbol{\nu}^{(k-1)}_{X_0}$. So the corollary can be understood as a statement on the convergence speed of Newton's method for solving $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$. The fact that Newton's method started at $\mathbf{0}$ converges to $\mathbf{1}$ (the least fixed point of $\boldsymbol{f}$) is shown in [20].

For the subcritical case, observe that the matrix $I - \boldsymbol{f}'(\mathbf{1})$ is nonsingular because otherwise 1 would be an eigenvalue of $\boldsymbol{f}'(\mathbf{1})$ which would, together with Proposition 2, contradict the assumption that the task system is subcritical. For nonsingular systems, it is a standard fact (see e.g. [31]) that Newton's method converges quadratically. As $\Pr[S^{op} \geq k] \leq 1 - \boldsymbol{\nu}^{(k-1)}_{X_0}$, the statement follows.

For the general case (subcritical or critical) Newton's method for solving $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$ has been extensively studied in [11, 12] and it follows from there that there is a $c_1 \in (0, \infty)$ such that $1 - \boldsymbol{\nu}^{(k)}_X \leq c_1 \cdot 2^{-k/(n2^n)}$ where $n = |\Gamma|$, implying the statement. □

## 4. Online Schedulers

From this section on we concentrate on online schedulers, which only know the past of the computation. Formally, a scheduler $\sigma$ is *online* if for every tree $t$ with $\sigma(t) = (s_1 \Rightarrow \ldots \Rightarrow s_k)$ and for every $1 \leq i < k$, the task $\sigma(t)[i]$ depends only on $s_1 \Rightarrow \ldots \Rightarrow s_i$ and on the restriction of the labelling function $L$ to $\bigcup_{j=1}^{i} s_j$.

For our results in this section it is convenient to assume that the task system is in a certain normal form, which we call *compact*.

*Compact Task Systems.* Any task system can be transformed into a so-called *compact* task system such that for every scheduler of the compact task system we can construct a scheduler of the original system yielding the same completion space up to an increase of at most $|\Gamma|$. A type $W$ is *compact* if there is a rule $X \hookrightarrow \langle Y, Z \rangle$ such that $X$ is reachable from $W$. A task system is *compact* if all its types are compact. A non-compact task system can be compacted by iteratively removing all rules with non-compact types on the left hand side, and all occurrences of non-compact types on the right hand side.

**Example 7.** Consider the following task system:

$$\begin{array}{lll} X \xhookrightarrow{0.5} \langle X, Y \rangle & Y \xhookrightarrow{0.5} \langle Y, Z \rangle & Z \xhookrightarrow{0.5} Z \\ X \xhookrightarrow{0.5} \langle X, X \rangle & Y \xhookrightarrow{0.5} Z & Z \xhookrightarrow{0.5} \emptyset. \end{array}$$

Here, only type $Z$ is not compact. After removing it, we obtain the new system:

$$\begin{array}{ll} X \xhookrightarrow{0.5} \langle X, Y \rangle & Y \xhookrightarrow{0.5} Y \\ X \xhookrightarrow{0.5} \langle X, X \rangle & Y \xhookrightarrow{0.5} \emptyset. \end{array}$$

In the reduced system, type $Y$ is not compact anymore, which necessitates a second iteration:

$$X \xhookrightarrow{0.5} X$$
$$X \xhookrightarrow{0.5} \langle X, X \rangle.$$

Note that if a scheduler schedules $Z$-tasks before $Y$-tasks and $Y$-tasks before $X$-tasks, then the pool has, at any time, at most two non-$X$-tasks.

The following proposition allows us to concentrate on compact task systems.

**Proposition 4.** *Let $\Gamma'$ denote the set of all task types removed from $\Delta$ by the above compacting procedure and let $|\Gamma'| = \ell$. If $X_0 \in \Gamma'$, then there is a scheduler $\sigma$ such that $S^\sigma \leq \ell$. Assume that $X_0 \notin \Gamma'$. Let $\Delta'$ be the compacted version of $\Delta$. Every scheduler $\sigma'$ for $\Delta'$ can be transformed into a scheduler $\sigma$ for $\Delta$ such that for all $k$*

$$\Pr\left[S^{\sigma',\Delta'} \geq k\right] \leq \Pr\left[S^{\sigma,\Delta} \geq k\right] \leq \Pr\left[S^{\sigma',\Delta'} \geq k - \ell\right].$$

*(The second superscript of $S$ indicates the task system on which the scheduler operates.)*

Computing $\sigma$ from $\sigma'$ is easy: $\sigma$ acts like $\sigma'$ but gives preferences to the types that have been (first) eliminated during the compacting procedure. Now we prove Proposition 4.

PROOF. Let $\Delta_1$ be a non-compact task system with non-compact types $\Gamma_{non}$, and let $\Delta_0$ be the (possibly non-compact) task system obtained from $\Delta_1$ by removing all rules with non-compact types on the left hand side and all occurrences of non-compact types on the right hand side of all rules, i.e., $\Delta_0$ is obtained from $\Delta_1$ by performing the first iteration of the compacting procedure. Let $\sigma_0$ be a scheduler for $\Delta_0$. Construct a scheduler $\sigma_1$ for $\Delta_1$ as follows:

> The scheduler $\sigma_1$ acts exactly like $\sigma_0$ until one or two $\Gamma_{non}$-tasks are created at which point the completion space of the derivation is increased by at most 1. Then $\sigma_1$ picks a $\Gamma_{non}$-task, say $\tau_1$. Since the $\Gamma_{non}$-types are non-compact, $\sigma_1$ can complete $\tau_1$ without further increasing the completion space. After $\tau_1$ has been finished, there may be another $\Gamma_{non}$-task left, say $\tau_2$, that was created at the time when $\tau_1$ was created. If there is such a $\tau_2$, then $\sigma_1$ completes $\tau_2$ in the same way it has completed $\tau_1$. After $\tau_1$ (and possibly $\tau_2$) have been completed, $\sigma_1$ resumes to act like $\sigma_0$.

It follows from this construction that the incorporation of the non-compact type $\Gamma_{non}$ increases the completion space of a derivation by at most 1.

A straightforward induction on this construction shows in terms of the proposition statement:

$$\Pr\left[S_X^{\sigma',\Delta'} \leq k\right] \leq \Pr\left[S_X^{\sigma,\Delta} \leq k + \ell\right] \text{ for all } X \in \Gamma \setminus \Gamma'.$$

If $X_0 \in \Gamma'$, then the above construction also works. (It extends a scheduler operating on a possibly empty task system, but this poses no problems.) So, again by induction, we obtain a scheduler $\sigma$ for $\Delta$ with $S_X^{\sigma,\Delta} \leq \ell$ for all $X \in \Gamma'$.

It remains to show the inequality $\Pr\left[S_X^{\sigma',\Delta'} \geq k\right] \leq \Pr\left[S_X^{\sigma,\Delta} \geq k\right]$; but this follows directly from the fact that $\Delta'$ is obtained by deleting rules and types from $\Delta$ and $\sigma$ is obtained by extending $\sigma'$. □

*Assumption.* In light of Proposition 4 we assume for the rest of the section that task systems are compact.

The following main theorem gives lower and upper bound on $\Pr[S^\sigma \geq k]$ which hold uniformly for all online schedulers $\sigma$. The bounds are given in terms of vectors $\boldsymbol{v}, \boldsymbol{w}$ with $\boldsymbol{f}(\boldsymbol{v}) \leq \boldsymbol{v}$ and $\boldsymbol{f}(\boldsymbol{w}) \geq \boldsymbol{w}$. We will show later (Proposition 5) how to compute such vectors.

**Theorem 2.** *Let $\Delta$ be subcritical. Let $\boldsymbol{v}, \boldsymbol{w} \in (1,\infty)^\Gamma$ be vectors with $\boldsymbol{f}(\boldsymbol{v}) \leq \boldsymbol{v}$ and $\boldsymbol{f}(\boldsymbol{w}) \geq \boldsymbol{w}$. Denote by $\boldsymbol{v}_{min}$ and $\boldsymbol{w}_{max}$ the least component of $\boldsymbol{v}$ and the greatest component of $\boldsymbol{w}$, respectively. Then*

$$\frac{\boldsymbol{w}_{X_0} - 1}{\boldsymbol{w}_{max}^{k+2} - 1} \quad \leq \quad \Pr[S^\sigma \geq k] \quad \leq \quad \frac{\boldsymbol{v}_{X_0} - 1}{\boldsymbol{v}_{min}^{k} - 1} \quad \textit{for all online schedulers } \sigma.$$

*Proof sketch.* The proof adapts a technique for the analysis of random walks going back to Feller (see [32, 33]), which uses the vector $\boldsymbol{v}$ to derive a supermartingale (and proceeds analogously with $\boldsymbol{w}$). More precisely, we use $\boldsymbol{v}$ to assign a *weight* $m^{(i)}$ to the pool of tasks at time $i$ and to choose a constant $h$, so that the following property holds: at any stopping time $\tau$ of the process, the expected value of $h^{m^{(\tau)}}$ is at most $\boldsymbol{v}_{X_0}$. This provides a "stochastic invariant" of the process, from which we can extract an upper bound for $\Pr[S^\sigma \geq k]$.

We now give some more details. For every $i \geq 1$, let $\boldsymbol{z}^{(i)}$ be the vector of random variables that measures the number of tasks of each type at time $i$. We choose $h > 1$ and $\boldsymbol{u} \in (0,\infty)^\Gamma$ such that $h^{\boldsymbol{u}_X} = \boldsymbol{v}_X$ for all $X \in \Gamma$. Now, let $m^{(i)} = \boldsymbol{z}^{(i)} \bullet \boldsymbol{u} \stackrel{def.}{=} \sum_{X \in \Gamma} \boldsymbol{z}_X^{(i)} \boldsymbol{u}_X$, i.e., $m^{(i)}$ is the random variable that measures the weight of the tasks at time $i$, when the weight of a task of type $X$ is given by $\boldsymbol{u}_X$. One can show that $h^{m^{(1)}}, h^{m^{(2)}}, \ldots$ is a supermartingale for any online scheduler $\sigma$. Define a stopping time $\tau_k := \inf\{i \geq 1 \mid m^{(i)} \in \{0\} \cup [k,\infty)\}$; i.e., $\tau_k$ is the time when the task system either terminates or has at least $k$ tasks in the pool. Using the Optional Stopping Theorem [34], we obtain

$$\mathbb{E}\left[h^{m^{(\tau_k)}}\right] \leq h^{\boldsymbol{u}_{X_0}} = \boldsymbol{v}_{X_0}\,.$$

Set $p_k := \Pr\left[m^{(\tau_k)} \geq k\right]$. Then we have

$$\boldsymbol{v}_{X_0} \geq \mathbb{E}\left[h^{m^{(\tau_k)}}\right] \geq h^0 \cdot (1 - p_k) + h^k \cdot p_k = 1 - p_k + h^k \cdot p_k$$