Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

A Construction and Minimization Service for Continuous Probability Distributions*

Reza Pulungan[1], **Holger Hermanns**[2]

[1] Jurusan Ilmu Komputer dan Elektronika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada, Yogyakarta, Indonesia, e-mail: pulungan@ugm.ac.id
[2] Department of Computer Science, Saarland University, Saarbrücken, Germany, e-mail: hermanns@cs.uni-saarland.de

The date of receipt and acceptance will be inserted by the editor

Abstract. The universe of acyclic continuous-time Markov chains can provide arbitrarily close approximations of any continuous probability distribution. We span this universe by a compositional construction calculus for acyclic phase-type distributions. The calculus draws its expressiveness from a single operator, yet the calculus is equipped with further convenient operators, namely convolution, maximum, and minimum. However, the size of the chains constructed in this way can grow rapidly. We therefore link our calculus to a compositional minimization algorithm that whenever applied almost surely yields a chain with the least possible size. The entire approach is available in the form of an easy-to-use web service. The paper describes the architecture of this service in detail, and reports on experimental evidence demonstrating its usefulness.

Key words: phase-type distributions – acyclic – minimization – maximum – minimum – convolution – Erlang

1 Introduction

Continuous probability distributions are natural means to model statistically varying phenomena. In full generality, it is difficult to store and manipulate continuous probability distributions. In this respect, phase-type distributions stand out, because they can be represented by finite-state graph models, in fact by absorbing continuous-time Markov chains. At the same time, they can be used to approximate any continuous probability distribution with arbitrary precision. The same holds true for the class of acyclic phase-type distributions, where the underlying graph is acyclic. Pragmatically speaking, these kinds of models thus provide a good compromise between expressiveness and model representability. Furthermore, they are closed under widely used stochastic operations such as maximum, minimum, and convolution. Using these operations may however drastically increase the model sizes.

This paper presents an approach enabling the specification and manipulation of acyclic phase-type distributions, in such a way that their representations virtually never occupy larger sizes than what is ultimately needed. The backbone of this approach is an algorithm that almost surely (*i.e.*, with probability 1) finds the smallest possible representation of a given acyclic phase-type distribution, and it does so in cubic time. The algorithm is embedded in a simple yet expressive calculus of delays, enabling the user to specify complex delay dependencies with the aid of convenient operations. The calculus, called CCC, is parsimonious in the sense that it has no actions or synchronization mechanisms and no recursion operation. CCC can easily be embedded in a standard process calculus providing these missing features. This can follow the lines of interactive Markov chains [13] or Markov automata [30], but also PEPA [15] or timed automata with a stochastic semantics [9]. All these calculi include the possibility to express exponentially distributed (non-synchronising) delays. These can be rightaway enhanced to acyclic phase-type distributions. CCC is made available as a web service ensuring almost sure minimality of the state-space sizes. This is an important asset for the overall scalability of any kind of analysis performed with these models.

* This work has been supported by the DFG as part of the SFB/TR 14 "Automatic Verification and Analysis of Complex Systems" (AVACS), by the DFG/NWO Bilateral Research Programme ROCKS, and by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS) and no. 318490 (SENSATION). Part of this work was done while Reza Pulungan was a guest of the Saarbrücken Graduate School of Computer Science funded by the German "Exzellenzinitiative des Bundes und der Länder".
Correspondence to: Reza Pulungan

Contributions. This paper expands the conference publication [29], in which we presented the algorithm to almost surely minimize any given acyclic phase-type representation to its minimal representation, by adding the following contributions:

1. We define a stochastic process calculus that can be used to conveniently generate and manipulate acyclic phase-type distributions representing delays of processes in a compositional manner.
2. We present a complete and stable implementation of a tool chain, which encapsulates the minimization algorithm and the stochastic calculus.
3. For the end user, we provide a convenient and easy-to-use interface to the tool chain through a web service and a web page.
4. We provide a case study to demonstrate the applicability of the calculus and the minimization algorithm.

Organization of the paper. The paper is organized as follows: Section 2 reviews several important concepts from continuous-time Markov chains and phase-type distributions. Section 3 then introduces the calculus and its properties. Section 4 discusses how and to what extend representation minimality can be achieved. In Section 5 we discuss the implementation and web-service specifics of our approach, followed (Section 6) by an exemplary case illustrating its usefulness. Section 7 finally concludes the paper.

2 Distributions, Canonicity and Stochastic Operations

This section reviews continuous-time Markov chains and phase-type distributions, together with important notions including size, order, degree, and minimal representation. Afterwards, two canonical forms for acyclic phase-type representations and three common stochastic operations on them are presented.

2.1 Phase-Type Distributions

A finite *continuous-time Markov chain* (CTMC) is a tuple $\mathcal{M} = (\mathcal{S}, \mathbf{Q}, \vec{\pi})$, where $\mathcal{S} = \{s_1, s_2, \ldots, s_n, s_{n+1}\}$ is a countable set of states, $\mathbf{Q}\colon (\mathcal{S} \times \mathcal{S}) \to \mathbb{R}$ is an infinitesimal *generator matrix*, and $\vec{\pi}\colon \mathcal{S} \to [0,1]$ is the initial probability distribution on $\mathcal{S}$. Intuitively, for any two distinct states $s, s' \in \mathcal{S}$, $\mathbf{Q}(s,s')$ specifies the *rate* of the transition from s to s'. This means that the probability that a state change (*i.e.*, a transition) occurs from s to s' within t time units is $1 - \exp(-\mathbf{Q}(s,s')t)$, namely it is governed by the *exponential distribution* with rate $\mathbf{Q}(s,s')$. By definition it holds that $\mathbf{Q}(s,s') \geq 0$ for all $s \neq s'$, and $\mathbf{Q}(s,s) = -\sum_{s \neq s'} \mathbf{Q}(s,s')$. The negative of the diagonal value, $E(s) = -\mathbf{Q}(s,s)$, is called the *exit rate* of state s.

If state s_{n+1} is absorbing (*i.e.*, $E(s_{n+1}) = 0$) and all other states s_i, for $1 \leq i \leq n$, are transient (*i.e.*, there is a nonzero probability that s_i will never be visited once it is left), the infinitesimal generator matrix of the CTMC can be written as:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{A} & \vec{A} \\ \vec{0} & 0 \end{bmatrix}.$$

Matrix $\mathbf{A}$ is called a *PH-generator* and, since the first n states in the CTMC are transient, it is non-singular. Vector $\vec{A}$ is a column vector where its component $\vec{A}_i$, for $1 \leq i \leq n$, represents the transition rate from s_i to the absorbing state. Let $\vec{1}$ be a column vector whose components are all 1. Note that since $\mathbf{Q}$ is a generator matrix (*i.e.*, each of its row sums up to 0), $\mathbf{A}\vec{1} = -\vec{A}$.

The CTMC $\mathcal{M}$ is fully specified by generator matrix $\mathbf{Q}$ and initial probability vector $\vec{\pi} = [\vec{\alpha}, \alpha_{n+1}]$, where $\vec{\alpha}$ is an n-dimensional row vector corresponding to the initial probabilities of the transient states, and α_{n+1} is the probability to be immediately in the absorbing state. The probability distribution of the time to absorption in such a CTMC is called a *phase-type* (PH) distribution [20]. In this paper, all PH distributions are required to be *non-defective*. This means that they do not have mass at $t = 0$, which technically means that $\alpha_{n+1} = 0$ and vector $\vec{\alpha}$ is stochastic. So we exclude distributions having a nonzero probability of stopping in zero time. The pair $(\vec{\alpha}, \mathbf{A})$ is called the *representation* of the PH distribution and $\mathsf{PH}(\vec{\alpha}, \mathbf{A})$ is used to denote the PH distribution that has representation $(\vec{\alpha}, \mathbf{A})$. The dimension of $\mathbf{A}$ is called the *size of the representation*. All PH representations we are dealing with are assumed to be irreducible. A representation is *irreducible* if for the specified initial probability distribution all transient states are visited with non-zero probability.

A PH distribution is completely characterized by its (cumulative) distribution function:

$$F(t) = 1 - \vec{\alpha}\exp(\mathbf{A}t)\ \vec{1}, \quad t \geq 0.$$

An alternative characterisation is given by its Laplace-Stieltjes transform (LST):

$$\begin{aligned} \tilde{F}(s) &= \int_{-\infty}^{\infty} \exp(-st) dF(t) \\ &= \vec{\alpha}(s\mathbf{I} - \mathbf{A})^{-1}\vec{A} + \alpha_{n+1}, \quad s \in \mathbb{R}_+, \end{aligned} \tag{1}$$

where $\mathbf{I}$ is the identity matrix of dimension n. An LST is a bijective transformation of a function to a complex domain, resolving that function into its moments. In particular, if X is a continuous random variable with cumulative distribution function $F(t)$, then the i-th moment of X is given by:

$$\mathbb{E}[X^i] = (-1)^i \frac{d^i \tilde{F}(s)}{ds^i}\bigg|_{s=0}.$$

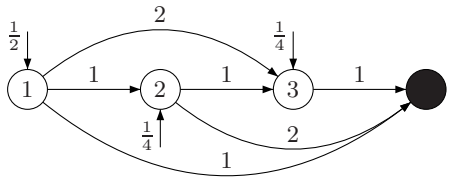

Fig. 1. An acyclic PH distribution

The LST in Equation (1) is a rational function. When expressed in irreducible ratio, the degree of its denominator, which is called the *algebraic degree of the distribution*, is no more than n. A PH distribution has more than one irreducible representation [20,22]. The size of a *minimal irreducible representation*, namely that with the fewest possible number of states, is called the *order of the PH distribution*. O'Cinneide in [22] showed that the algebraic degree of a PH distribution is the lower bound of its order.

The simplest of PH distributions is the family of exponential distributions. Indeed, this family of distributions is the building block of PH distributions. Maier and O'Cinneide [18] show that exponential distributions together with finite mixture, convolution, and recursion operations can generate the whole of PH distributions. A convolution of several exponential distributions with the same rate is called an Erlang distribution.

Another interesting subset of PH distributions is the family of *acyclic PH* (APH) *distributions*. The family can be identified by the fact that the graph representations of their state spaces are acyclic; see Fig. 1 for an example. Both families of exponential and Erlang distributions are subsets of APH distributions. An *acyclic minimal representation* of an APH distribution is an APH representation with the least number of states. The *acyclic order* of an APH distribution is the size of its acyclic minimal representation [24]. An APH distribution is called *acyclic ideal* if and only if its acyclic order is equal to its algebraic degree. Thus, an acyclic-ideal distribution has a minimal representation that is acyclic. In contrast, a non-acyclic-ideal APH distribution has even smaller representations that possibly contain cycles.

O'Cinneide [23] proved a theorem that characterizes APH distributions in terms of the properties of their density functions and LSTs. In particular he showed that an LST is the LST of an APH distribution if and only if all of its poles are real. Thus, any general PH representation—possibly containing cycles—represents an APH distribution (and hence has an acyclic representation) whenever all of the poles of its LST are real numbers.

Notably, both APH and PH distributions are topologically dense [16]. This implies that any continuous distribution can be approximated arbitrarily closely by an APH distribution or a PH distribution.

2.2 Canonicity

We now discuss two different canonical forms of APH representations, each with a simple and easy-to-understand structure. Each of them is "canonical" in the sense that it (if viewed as a graph) is unique up to isomorphism. Every APH representation can be transformed into either of them without altering its stochastic behavior, *i.e.*, its distribution.

Ordered Bidiagonal Representation. A PH-generator of the form:

$$\begin{bmatrix} -\lambda_1 & \lambda_1 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & \lambda_2 & \cdots & 0 \\ 0 & 0 & -\lambda_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -\lambda_n \end{bmatrix}$$

is called a *bidiagonal generator*, and it is denoted by $\mathbf{Bi}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. The first canonical form is obtained if the negative diagonal entries, the exit rates, of the bidiagonal generator are in ascending order.

Theorem 1. *Let $(\vec{\alpha}, \mathbf{A})$ be an acyclic phase-type representation, $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of $-\mathbf{A}$, and, without loss of generality, assume that $\lambda_n \geq \lambda_{n-1} \geq \ldots \geq \lambda_1 > 0$. Then there exists a unique bidiagonal representation, the* ordered bidiagonal representation*, $(\vec{\beta}, \mathbf{Bi}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ *such that:*

$$\mathsf{PH}(\vec{\beta}, \mathbf{Bi}(\lambda_1, \lambda_2, \ldots, \lambda_n)) = \mathsf{PH}(\vec{\alpha}, \mathbf{A}).$$

This representation has a simple structure: the states are ordered by their exit rates, each of them has only one transition to its neighbour, and the initial distributions spans the entire state space; see Fig. 2(a) for a graph visualization.

An efficient algorithm, called the *spectral polynomial algorithm* (SPA) [11], can be used to construct the canonical ordered bidiagonal representation of any given APH representation. SPA has complexity $\mathcal{O}(n^3)$, where n is the size of the given APH representation.

Cox Representation. A *Dirac* distribution is a probability distribution that assigns full probability to a single outcome. Let $0 \leq p_i < 1$, for $1 \leq i \leq n-1$. A PH-generator of the form:

$$\begin{bmatrix} -\lambda_1 & p_1\lambda_1 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & p_2\lambda_2 & \cdots & 0 \\ 0 & 0 & -\lambda_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -\lambda_n \end{bmatrix}$$

is called a *Cox generator*, and it is denoted by $\mathbf{Cx}([\lambda_1, p_1], [\lambda_2, p_2], \ldots, \lambda_n)$. Here, every state, apart from the absorbing state, has a transition to the next

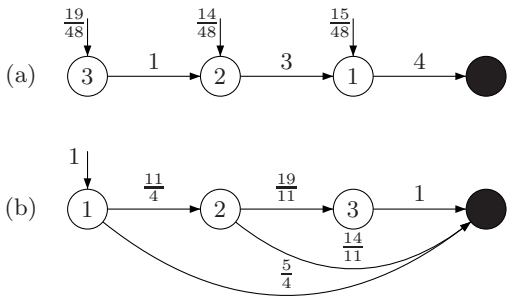

Fig. 2. (a) An ordered bidiagonal and (b) a Cox representation of the acyclic PH representation from Fig. 1

state, possibly a transition to the absorbing state, and no other transitions. The second canonical form, the *Cox representation*, is obtained by a Cox generator with descending exit rates and a Dirac initial distribution to the highest exit rate state. The name is due to David R. Cox [7], who coined this representation; see Fig. 2(b) for a graph visualization.

Theorem 2 ([8]). *Let* $(\vec{\beta}, \mathbf{Bi}(\lambda_1, \lambda_2, \ldots, \lambda_n))$ *be an ordered bidiagonal representation, and let vector* $\vec{\delta} = [1, 0, \ldots, 0]$. *Then there exists a unique* Cox representation, $(\vec{\delta}, \mathbf{Cx}([\lambda_n, x_n], [\lambda_{n-1}, x_{n-1}], \ldots, \lambda_1))$ *such that:*

$$\mathsf{PH}(\vec{\delta}, \mathbf{Cx}([\lambda_n, x_n], [\lambda_{n-1}, x_{n-1}], \ldots, \lambda_1)) = \mathsf{PH}(\vec{\beta}, \mathbf{Bi}(\lambda_1, \lambda_2, \ldots, \lambda_n)).$$

The vector $\vec{x}$ determining the Cox representation is derived from vector $\vec{\beta}$ by:

$$x_i = 1 - \beta_i \prod_{j=i+1}^{n} \frac{1}{x_j}, \quad \text{for } 2 \leq i \leq n. \tag{2}$$

Example 1. Consider the APH representation depicted in Fig. 1. The three non-absorbing states have exit rates 4, 3, and 1, respectively. Fig. 2(a) depicts the corresponding ordered bidiagonal representation. The branching structure of the original representation is now reflected (though in a non-obvious way) by the initial probability distribution of the ordered bidiagonal representation. Fig. 2(b) depicts the corresponding Cox representation, obtained from the ordered bidiagonal representation by instantiating Equation (2).

Notably, both canonical forms are characterized by $2n-1$ real parameters. For the ordered bidiagonal representation, this is $\vec{\beta}$ (where the last value can be dropped since $\sum_1^n \beta_i = 1$) and $\vec{\lambda}$, for the Cox representation, $\vec{x}$ and $\vec{\lambda}$.

2.3 Three Stochastic Operations

We now consider three very common and convenient operations on continuous probability distributions. Let X_1 and X_2 be two independent random variables with distribution functions $F_1(t)$ and $F_2(t)$, respectively; and let the random variables $X_{\text{con}} = X_1 + X_2$, $X_{\max} = \max\{X_1, X_2\}$, and $X_{\min} = \min\{X_1, X_2\}$ be the summation (convolution), maximum, and minimum of X_1 and X_2, respectively. The random variables X_{con}, $X_{\max}$, and $X_{\min}$, by definition, have distribution functions $F_{\text{con}}(t) = \int_0^t F_1(t-x)F_2(x)dx$, $F_{\max}(t) = F_1(t)F_2(t)$, and $F_{\min}(t) = 1 - (1 - F_1(t))(1 - F_2(t))$, respectively.

Theorem 3. *[20, Theorem 2.2.9] Let* $(\vec{\alpha}, \mathbf{A})$ *and* $(\vec{\beta}, \mathbf{B})$ *be the representations of PH distributions* $F(t)$ *and* $G(t)$ *of size* m *and* n, *respectively. Then:*

(a) their convolution $\text{con}(F, G)$ *is a PH distribution with representation* $(\vec{\delta}, \mathbf{D})$ *of size* $m + n$, *where:*

$$\vec{\delta} = [\vec{\alpha}, \alpha_{m+1}\vec{\beta}] \quad \textit{and} \quad \mathbf{D} = \begin{bmatrix} \mathbf{A} & \vec{A}\vec{\beta} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}. \tag{3}$$

(b) their minimum $\min(F, G)$ *is a PH distribution with representation* $(\vec{\delta}, \mathbf{D})$ *of size* mn, *where:*[1]

$$\vec{\delta} = \vec{\alpha} \otimes \vec{\beta} \quad \textit{and} \quad \mathbf{D} = \mathbf{A} \oplus \mathbf{B}. \tag{4}$$

(c) their maximum $\max(F, G)$ *is a PH distribution with representation* $(\vec{\delta}, \mathbf{D})$ *of size* $mn + m + n$, *where:*

$$\vec{\delta} = \left[\vec{\alpha} \otimes \vec{\beta},\ \beta_{n+1}\vec{\alpha},\ \alpha_{m+1}\vec{\beta}\right] \quad \textit{and}$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{A} \oplus \mathbf{B} & \mathbf{I}_A \otimes \vec{B} & \vec{A} \otimes \mathbf{I}_B \\ \mathbf{0} & \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B} \end{bmatrix}. \tag{5}$$

3 CCC Syntax and Semantics

In this section, we develop a simple yet convenient stochastic delay calculus that we use to generate and manipulate APH representations. We call the calculus the *Cox & Convenience Calculus* or, for short, CCC.

A delay in our calculus reflects the completion time of some activity. As basic building blocks we use exponential distributions. More complex delays are built by composing basic delays using several operations. In fact, a single operator is enough to obtain a completeness result w.r.t. APH distributions. Further operators are added for mere convenience. They correspond to the operations discussed in Section 2.3.

The calculus is parsimonious in the sense that it has no actions or synchronization mechanisms and no recursion operation. CCC can easily be embedded in a standard process calculus providing these missing features. For the setting considered here, all processes, from the basic to the most complex, are acyclic, and just represent completion times of activities. The semantics of CCC maps on absorbing CTMCs representing the APH distributions that govern the completion times.

[1] See Appendix A for the formal definition of Kronecker product operator $\otimes$ and Kronecker sum operator $\oplus$.

Cox. The simplest building block that can be expressed in CCC is an exponentially distributed delay, expressed simply by 'λ', with $\lambda \in \mathbb{R}_+$. The intended semantics is an (unspecified) activity that induces a delay distributed according to an exponential distribution with rate λ and then terminates.

One of the advantages of the Cox representations is the fact that each of them starts in a single state, which is important in developing the delay calculus. Our delay calculus will generate APH representations in Cox forms.

CCC allows us to generate the whole set of Cox representations from the building blocks by using a single additional operator, $\lhd$, which is inspired by the disabling operator, as found, for instance, in LOTOS [3]. Disabling takes the form $\mu \lhd \lambda.P$, where delay μ forces a preemptive termination, unless delay λ finishes first. Concretely, delay $\lambda.P$ behaves as delay P after an exponentially distributed duration with rate λ. Now, μ and $\lambda.P$ disable each other, in the sense that the termination of delay μ cancels delay $\lambda.P$ and terminates, while the occurrence of delay λ cancels delay μ and continues with delay P. The implied semantics is a race between the two exponential distributions, and the time until the race is determined therefore corresponds to an exponential distribution with rate $\mu + \lambda$.

Example 2. Consider the Cox representation depicted in Fig. 2(c). In CCC, the Cox representation can be expressed by the following delay:

$$P = \frac{5}{4} \lhd \frac{11}{4}.\left(\frac{14}{11} \lhd \frac{19}{11}.1\right).$$

Convenience. The second part of the delay calculus is the "convenience" part, namely the part that helps us manipulate APH representations. In Section 2.3, we have reviewed three common operations on APH representations. They constitute the construction means we capture by the calculus. We thus define three composition operators to carry out these three operations, namely *sequential composition* (;), *parallel composition* ($\|$), and *choice* ($\oplus$).

Here we exploit a natural correspondence between the convolution of two APH distributions and the sequential composition of their delays (;). Delay $P;Q$ describes the *sequential composition* of delays P and Q in that particular order. This delay behaves as P until P terminates. Upon the termination of P, it then behaves as Q. The overall behavior of $P;Q$ is therefore determined by the sum of the completion times of P and Q.

Similarly, the maximum of two PH distributions corresponds to the parallel composition ($\|$) of their delays. This is rooted in the memoryless property of the underlying CTMCs, which makes it possible to use a simple interleaving semantics of their delays. The precise correspondence will be the subject of Theorem 6. Delay $P\|Q$ describes the *parallel composition* of delays P and Q. This delay behaves as P and as Q, at the same time. For the delay to terminate, both delays P and Q must terminate. Hence, the completion time of delay $P\|Q$ corresponds to the largest completion time of its components. In other words, the overall behavior of $P\|Q$ is determined by the maximum of the completion times of P and Q.

In Theorem 6, we will also show that the minimum of two APH distribution corresponds to the choice $\oplus$ of the two delays. The choice implements a race between two delays which is decided only once the first delay terminates. Delay $P \oplus Q$, describes the *choice* of delays P and Q. This delay behaves either as P or as Q, whichever terminates first. Since a delay describes its random completion time, we can think of delay $P \oplus Q$ as the race between the two delays: whichever has the least completion time wins. Therefore, when either of them terminates, the delay itself terminates. The overall behavior of $P \oplus Q$ is then determined by the minimum of the completion times of P and Q.

Syntax. Each term built according to the following grammar is a CCC *delay*:

$$P \quad ::= \quad \lambda \quad | \quad \mu \lhd \lambda.P \quad | \quad P;P \quad | \quad P \oplus P \quad | \quad P\|P$$

where $\lambda \in \mathbb{R}_+$ and $\mu \in \mathbb{R}_{\geq 0}$ are *rates*. We use $P, P_1, P_2, \ldots, P', Q, R, \ldots$ to range over delays, and $\lambda, \lambda_1, \lambda_2, \ldots, \mu, \nu, \ldots$ to range over rates.

We fix the precedence of the operators in the language as follows: disabling ($\lhd$), sequential (;), choice ($\oplus$), and parallel ($\|$) operators. Thus, the disabling operator takes precedence over sequential, choice, and parallel operators, sequential operator takes precedence over choice and parallel operators, and so on. To circumvent these default precedence rules, we may use parentheses. Furthermore, the operators described above are binary operators. Unparenthesized repeated applications of the same operator are assumed to have a left-associative evaluation order: $P\|Q\|R$ is evaluated as $(P\|Q)\|R$.

Semantics. The semantics of CCC is remarkably simple. As for many stochastic process calculi [15,14,2,26], it comes in two stages. In the first stage, the entire language is mapped onto an acyclic labelled transition system in the style of decorated transition semantics inspired by [14,26], while the second stage interprets this as an absorbing CTMC. We could have opted for a direct semantics, harvesting work on concise stochastic calculi semantics [17,21,6].

MTS Semantics. Let *Lab* be the set of strings defined by the grammar:

$$w \ ::= \ \varepsilon \mid \lhd_l .w \mid \lhd_r .w \mid \oplus_l .w \mid \oplus_r .w \mid \|_l.w \mid \|_r.w$$

where ε is the empty string, and '.' is the string concatenation operator. The subscripts l and r stand for left and right, respectively. The strings will be used to differentiate different transitions having the same rate between two processes.

Table 1. Structural operational semantics for CCC

(0)	$\bullet \| \bullet \equiv \bullet$	(1)	$\dfrac{}{\lambda \xrightarrow{\lambda,\varepsilon} \bullet}$
(2.a)	$\dfrac{}{\mu \lhd \lambda.P \xrightarrow{\mu,\lhd_l} \bullet}$	(2.b)	$\dfrac{}{\mu \lhd \lambda.P \xrightarrow{\lambda,\lhd_r} P}$
(3.a)	$\dfrac{P \xrightarrow{\lambda,w} P'}{P;Q \xrightarrow{\lambda,w} P';Q}\ (P' \neq \bullet)$	(3.b)	$\dfrac{P \xrightarrow{\lambda,w} \bullet}{P;Q \xrightarrow{\lambda,w} Q}$
(4.a)	$\dfrac{P \xrightarrow{\lambda,w} P'}{P \oplus Q \xrightarrow{\lambda,\oplus_l.w} P' \oplus Q}\ (P' \neq \bullet)$	(4.b)	$\dfrac{P \xrightarrow{\lambda,w} \bullet}{P \oplus Q \xrightarrow{\lambda,\oplus_l.w} \bullet}$
(4.c)	$\dfrac{Q \xrightarrow{\lambda,w} Q'}{P \oplus Q \xrightarrow{\lambda,\oplus_r.w} P \oplus Q'}\ (Q' \neq \bullet)$	(4.d)	$\dfrac{Q \xrightarrow{\lambda,w} \bullet}{P \oplus Q \xrightarrow{\lambda,\oplus_r.w} \bullet}$
(5.a)	$\dfrac{P \xrightarrow{\lambda,w} P'}{P \| Q \xrightarrow{\lambda,\|_l.w} P' \| Q}$	(5.b)	$\dfrac{Q \xrightarrow{\lambda,w} Q'}{P \| Q \xrightarrow{\lambda,\|_r.w} P \| Q'}$

A *Markov Transition System* (MTS) is a tuple $\mathfrak{M} = (\mathsf{CCC}, P_0, \longrightarrow)$, where:

- CCC is the set of all CCC delays,
- P_0 is the initial delay, and
- $\longrightarrow \subseteq (\mathsf{CCC} \times (\mathbb{R}_+ \times Lab) \times \mathsf{CCC})$ is the least relation satisfying the derivation rules listed in Table 1.

The expression $\bullet$ in Table 1, is a terminal symbol describing a terminated behavior. As a process, $\bullet$ does nothing. Notably, $\bullet$ is not part of the CCC syntax, which corresponds to our restriction to non-defective distributions.

An element of the set $\longrightarrow$ is called a transition. The transitions among processes are obtained by applying the given set of the derivation rules. For reading convenience, we write $P \xrightarrow{\lambda,w} Q$ to denote the existence of a transition from state P to Q with rate λ and label w, instead of $(P, (\lambda, w), Q) \in \longrightarrow$.

The formal (first stage) semantics of language CCC is given in the style of *Structural Operational Semantics* (SOS) [25,1]. The intuitive interpretation described before is formalized by means of SOS derivation rules depicted in the table. Among the SOS rules of language CCC listed in Table 1 are three axioms, namely rules (1), (2.a), and (2.b). Axiom (2.a) specifies that for all processes of the form $\mu \lhd \lambda.P$, transition $\mu \lhd \lambda.P \xrightarrow{\mu,\lhd_l} \bullet$ is valid. The remaining rules in the table are derivation rules; take for instance rule (4.a). This rule states that given that process P' is not equal to $\bullet$, whenever $P \xrightarrow{\lambda,w} P'$ is a valid transition, then so is the transition $P \oplus Q \xrightarrow{\lambda,\oplus_l.w} P' \oplus Q$. The other rules are interpreted in a similar fashion.

Each transition in an MTS has two labels: the rate of the corresponding transition and an additional label. The additional label is necessary to allow us to differentiate transitions that otherwise would be indistinguishable, namely to explicitly produce different transitions if there exist different derivations with the same rate between two processes. For instance, the process $\lambda \oplus \mu$ has two transitions to process $\bullet$, one with rate λ and the other with rate μ. In this case, the additional label is not required. However, in the case of a process of the form $\lambda \oplus \lambda$, the additional label is necessary to uphold that there are indeed two different derivations for rate λ, namely the transitions:

$$\lambda \oplus \lambda \xrightarrow{\lambda,\oplus_l} \bullet \quad \text{and} \quad \lambda \oplus \lambda \xrightarrow{\lambda,\oplus_r} \bullet.$$

This technical means to overcome the restriction of multiple transitions with the same rate (or in other contexts, action or label) between any two processes was first described in [4], and has been widely used, for instance in [14].

In addition to the operational semantic rules, a single structural congruence rule is required. Structural congruences are common, for instance, in π-calculus [19] variations. The structural congruence rule is:

$$\bullet \| \bullet \equiv \bullet. \tag{6}$$

This rule allows us to collect terminated delays. It is essential for a proper functioning of rules (3.b), (4.b), and (4.d), and the second-stage semantics.

Reachability. Reachability is defined as usual: A *finite path* σ of length $n \in \mathbb{Z}_{\geq 0}$ from P_0 in $\mathfrak{M}$ is an alternating sequence of delays and labels:

$$P_0(\lambda_1, w_1)P_1(\lambda_2, w_2)P_2 \ldots P_{n-1}(\lambda_n, w_n)P_n,$$

such that, for all $1 \leq i \leq n$, $P_{i-1} \xrightarrow{\lambda_i,w_i} P_i$. The path *ends in* delay P_n. A path $\sigma' = P_0$ is of length 0; starts from P_0 and ends in P_0. A finite path is *maximal* if it is not a proper prefix of another finite path. A delay $P' \in \mathsf{CCC}$ is *reachable* from $P \in \mathsf{CCC}$, if there is $n \in \mathbb{Z}_{\geq 0}$ and a path σ of length n such that $P_0 = P$ and $P_n = P'$.

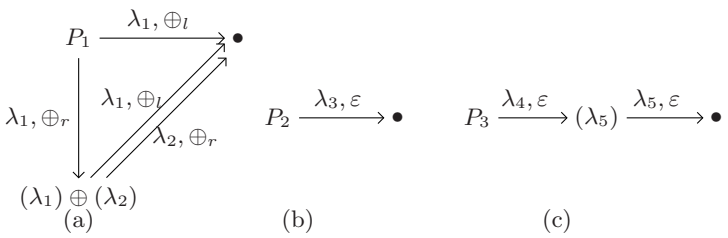

Fig. 3. The MTS semantics of (a) P_1, (b) λ_3 and (c) $\lambda_4; \lambda_5$

If delay P' is reachable from delay P we write $P \Longrightarrow P'$. We use $Reach(P)$ to denote the set of all those delays in CCC that are reachable from P, *i.e.*, $Reach(P) = \{P' \in \mathsf{CCC} \mid P \Longrightarrow P'\}$.

Example 3. Consider a delay $P \in \mathsf{CCC}$ defined as $P = P_1; \lambda_3 \| \lambda_4; \lambda_5$, where $P_1 = \lambda_1 \oplus \lambda_1; \lambda_2$. Delay P is the parallel composition of two sequentially composed delays. Fig. 3 depicts the semantics of P_1 and the other component delays. The semantics of delay P is depicted in Fig. 4. Both Fig. 3 and Fig. 4 depict only the reachable fragments of each delay.

Absorbing CTMC Semantics. In the second stage of the semantics, we define an interpretation of the MTS we obtained in the first stage in terms of an absorbing CTMC. This is achieved by a function $\gamma \colon (\mathsf{CCC} \times \mathsf{CCC}) \to \mathbb{R}_{\geq 0}$ that accumulates the rate of all transitions between two delays in the MTS:

$$\gamma(P, Q) = \sum_{(\lambda, w) \in \{(\lambda, w) \mid P \xrightarrow{\lambda, w} Q\}} \lambda.$$

Function γ is necessary because the operational semantics may justify more than one transition between two distinct delays, in contrast to a CTMC.

The *absorbing CTMC semantics* of a delay $P \in \mathsf{CCC}$, with MTS semantics $(\mathsf{CCC}, P, \longrightarrow)$, is $\mathcal{M}_P = (\mathcal{S}, \mathbf{Q}, \vec{\pi})$, where:

- the state space $\mathcal{S} = Reach(P)$,
- the infinitesimal generator matrix $\mathbf{Q}(P, Q) = \gamma(P, Q)$ if $P \neq Q$ and $-\sum_{R \neq P} \mathbf{Q}(P, R)$ if $P = Q$, and
- the initial probability distribution $\vec{\pi}$ is a Dirac initial distribution to delay P.

Once we have the semantics of a CCC delay in terms of an MTS, we can map the obtained MTS onto an absorbing CTMC. While the MTS $\mathcal{M}_P$ associated with a CCC delay is uncountably large, the size of the state space of $\mathcal{M}_P$ is $|Reach(P)|$.

Example 4. Continuing from Example 3, the absorbing CTMC semantics of the MTS depicted in Fig. 4 is shown in Fig. 5. For reading convenience, we have renamed all reachable MTS delays in the CTMC figure, while keeping their relative position.

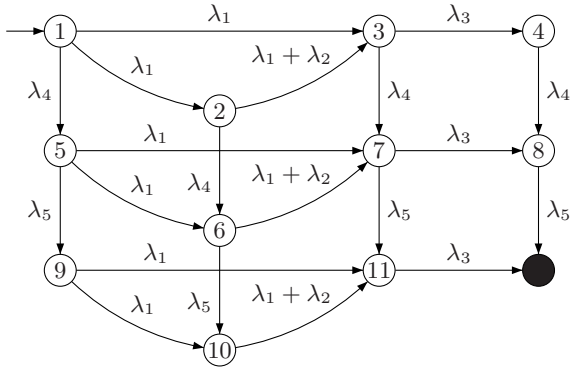

Fig. 5. The absorbing CTMC semantics of delay $P_1.P_2 \| P_3$

Delays and PH Distributions. We now discuss whether, as we would expect, the CTMC semantics of any delay $P \in \mathsf{CCC}$ always represents an APH distribution, and vice versa. The following lemmas are easily established by structural induction.

Lemma 1. *For every $P \in \mathsf{CCC}$:*

1. $P \Longrightarrow \bullet$,
2. $|Reach(P)|$ *is finite,*
3. *all maximal paths from P end in $\bullet$, and*
4. *there is an upper bound on the length of any path from P.*

The proof of Lemma 1.3 makes use of the structural congruence (Equation. (6)), and the observation that operator $\|$ is the only static operator of the calculus. Lemma 1.4 ensures that indeed there is no infinite path in the semantics of the language. Altogether, these lemmas allow us to establish that CCC is indeed a calculus for APH distributions.

Theorem 4. *For every $P \in \mathsf{CCC}$, $\mathcal{M}_P = (\mathcal{S}, \mathbf{Q}, \vec{\pi})$ is a CTMC underlying an acyclic phase-type representation with a Dirac initial distribution.*

So we can identify any CCC delay P with the PH distribution associated with $\mathcal{M}_P$, from now on denoted $\mathsf{PH}(P)$. While this theorem can be seen as a closure property of the calculus, the following theorem provides a completeness result, namely that any APH distribution can be generated by CCC.

Theorem 5. *For any acyclic phase-type distribution* APH *there is a delay P such that* $\mathsf{APH} = \mathsf{PH}(P)$.

Proof. We can transform the APH representation associated with any such APH to a Cox form by using the spectral polynomial algorithm (see Section 2.2) and Theorem 2. Our restriction to non-defective distributions will be inherited by the Cox representation which thus has a Dirac initial distribution to the highest rated state. It is now straightforward to encode the bidiagonal structure of the Cox representation into a right-bracketed nesting of disabling-operators $\lhd$ and a terminating rate. □

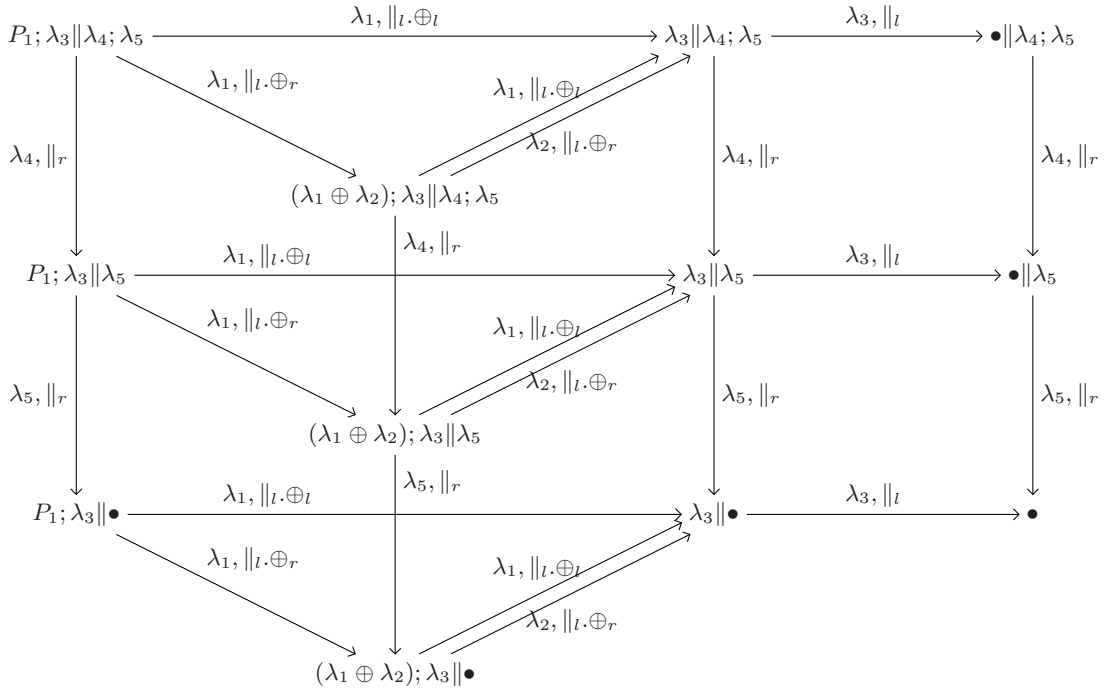

Fig. 4. The MTS semantics of $P_1;\lambda_3\|\lambda_4;\lambda_5$

An illustrative example of the last step has been given in Example 2. So, in fact the disabling operator $\lhd$ (plus one terminating rate) is enough to encode any APH distribution into CCC. Dropping the operator $\lhd$ from the calculus renders the above result impossible. The other binary operators of the CCC are added for convenience; they do not enhance expressive power. In the following, we verify that their intuitive interpretation is matched by the semantics.

Theorem 6. *For all delays $P, Q \in CCC$:*

1. $\mathrm{con}(\mathsf{PH}(P), \mathsf{PH}(Q)) = \mathsf{PH}(P;Q)$,
2. $\min(\mathsf{PH}(P), \mathsf{PH}(Q)) = \mathsf{PH}(P \oplus Q)$, *and*
3. $\max(\mathsf{PH}(P), \mathsf{PH}(Q)) = \mathsf{PH}(P\|Q)$.

The proof of Theorem 6 can be found in Appendix B.3 of [27]. This theorem induces a sound denotational semantics for CCC by the clauses in Theorem 3 for con(), max(), and min(), together with setting the denotational semantics of $\mu \lhd \lambda.P$ to APH representation $(\vec{e}_2, \mathbf{A}_2)$, such that:

$$\mathbf{A}_2 = \begin{bmatrix} -(\mu+\lambda) & \lambda\vec{e}_1 \\ \vec{0} & \mathbf{A}_1 \end{bmatrix},$$

where $\mathbf{A}_1$ is the PH-generator of the denotational semantics of P, and $\vec{e}_1$ and $\vec{e}_2$ are row vectors of appropriate dimensions representing Dirac initial distributions to the first component.

4 Minimality by Construction

While CCC provides expressive and convenient means to describe delay distributions, we are in practice often facing an explosion in size, especially rooted in the semantics of maximum ($\|$) and minimum ($\oplus$) (regardless of whether we use denotational or operational semantics). They grow as the product of their component sizes, while the convolution (;) grows as their sum. This phenomenon is not uncommon especially for concurrent system representations. Luckily, we can equip CCC with effective means to compress the resulting sizes considerably in almost all cases. This is rooted in a polynomial-time algorithm [29] that reduces the size of any component CCC delay.

Given an arbitrary APH representation $(\vec{\alpha}, \mathbf{A})$ as input, the algorithm returns an ordered bidiagonal representation, denoted $\mathsf{Red}(\vec{\alpha}, \mathbf{A})$, having the same PH distribution, with a representation size being at most the size of the original one. The reduction achievable goes beyond concepts like lumpability [5], since the algorithm exploits properties of the Laplace-Stieltjes transform. In very brief terms, the algorithm checks, for each matrix row and column, certain dependencies in the matrix of its ordered bidiagonal representation of the distribution (obtained by running the SPA algorithm a priori). These dependencies are expressible in terms of linear equation systems. If satisfied, a state can be identified as being removable, and is subsequently removed from the representation. This process is then iterated until no further state is identified as being removable.

Algorithm 1

```
1: function RedAcycRep(α⃗, A)
2:     (β⃗, Bi) ← Spa(α⃗, A)
3:     n ← SizeOf(β⃗, Bi)
4:     i ← 2
5:     while i ≤ n do
6:         if Removable(i, (β⃗, Bi)) then
7:             (β⃗, Bi) ← Remove(i, (β⃗, Bi))
8:             n ← n − 1
9:         else
10:            i ← i + 1
11:        end if
12:    end while
13:    return (β⃗, Bi)
14: end function
```

Algorithm 1 takes as input an acyclic PH representation, and outputs its reduced ordered bidiagonal representation. Function $\textsc{Spa}(\cdot)$, in the algorithm, transforms an acyclic PH representation to its ordered bidiagonal representation using the spectral polynomial algorithm [11]. Function $\textsc{SizeOf}(\cdot)$ returns the size of the given representation. Function $\textsc{Removable}(\cdot)$ and function $\textsc{Remove}(\cdot)$ check for each state whether it is removable, and if so, remove it from the representation.

To give some insight into how they work, we recall that the Laplace-Stieltjes transform (cf. Equation (1)) of an exponential distribution with rate λ is $\tilde{F}(s) = \frac{\lambda}{s+\lambda}$. Let $L(\lambda) = \frac{s+\lambda}{\lambda}$, *i.e.*, the reciprocal of the LST of the exponential distribution. We call a single expression of $L(\cdot)$ an *L-term*. The LST of an ordered bidiagonal representation $(\vec{\beta}, \mathbf{Bi}(\lambda_1, \lambda_2, \ldots, \lambda_n))$ can then be written as:

$$\begin{aligned}\tilde{F}(s) &= \frac{\vec{\beta}_1}{L(\lambda_1)\ldots L(\lambda_n)} + \frac{\vec{\beta}_2}{L(\lambda_2)\ldots L(\lambda_n)} + \ldots + \frac{\vec{\beta}_n}{L(\lambda_n)} \\ &= \frac{\vec{\beta}_1 + \vec{\beta}_2 L(\lambda_1) + \ldots + \vec{\beta}_n L(\lambda_1)L(\lambda_2)\ldots L(\lambda_{n-1})}{L(\lambda_1)L(\lambda_2)\ldots L(\lambda_n)}.\end{aligned} \tag{7}$$

Note that the LST expression in Equation (7) may not be in irreducible ratio form. The LST is produced in such a way that the denominator polynomial corresponds exactly to the sequence of the total outgoing rates of the ordered bidiagonal representation. Hence, the degree of the denominator polynomial is equal to the size of the ordered bidiagonal representation. Observing the equation, we see that in order to remove a state from the ordered bidiagonal representation, we have to find a common L-term in both the numerator and denominator polynomials. Removing such a common L-term from the numerator and denominator polynomials means removing the corresponding state from the representation. However, such a removal of a state can only be carried out if after the removal, the initial probability distribution $\vec{\beta}$ is redistributed in a correct way. This means the new initial probability distribution, say $\vec{\delta}$, must be a sub-stochastic vector.

Thus in order to reduce the size of an ordered bidiagonal representation, we need to check two conditions, namely the divisibility of the numerator polynomial and the sub-stochasticity of the resulting initial probability vector. For this, we let $R(s)$ denote the numerator polynomial of the LST in Equation (7):

$$R(s) = \vec{\beta}_1 + \vec{\beta}_2 L(\lambda_1) + \ldots + \vec{\beta}_n L(\lambda_1)\ldots L(\lambda_{n-1}).$$

The root of an L-term $L(\lambda_i)$ is $-\lambda_i$. Hence, polynomial $R(s)$ is divisible by $L(\lambda_i)$ if one of the roots of $R(s)$ is also $-\lambda_i$, or simply if $R(-\lambda_i) = 0$. This is what function $\textsc{Removable}(\cdot)$ in Algorithm 1 checks, it returns $\textsc{True}$ if and only if $R(-\lambda_i) = 0$.

Now, assume that $R(-\lambda_i) = 0$ and, for brevity of presentation, let $\mathbf{Bi}_1 := \mathbf{Bi}(\lambda_1, \ldots, \lambda_n)$ and $\mathbf{Bi}_2 := \mathbf{Bi}(\lambda_1, \ldots, \lambda_{i-1}, \lambda_{i+1}, \ldots, \lambda_n)$. We thus are looking for vector $\vec{\delta}$ satisfying:

$$\mathsf{PH}(\vec{\delta}, \mathbf{Bi}_2) = \mathsf{PH}(\vec{\beta}, \mathbf{Bi}_1).$$

or equivalently:

$$\vec{\delta}\exp(\mathbf{Bi}_2 t)\vec{1} = \vec{\beta}\exp(\mathbf{Bi}_1 t)\vec{1}, \qquad t \geq 0.$$

To obtain $\vec{\delta}$ from $\vec{\beta}$, $n-1$ equations relating their components are needed. For this, one might think of evaluating the last equation at $n-1$ different t values to obtain a system of equations determining $\vec{\delta}$. But this way is not recommendable, since it involves matrix exponentiations. Instead we look at the derivatives of the distribution functions: For a PH representation $(\vec{\alpha}, \mathbf{A})$ and $j \geq 0$, the j-th derivative is:

$$\frac{d^j}{dt^j}F(t) = -\vec{\alpha}\mathbf{A}^j\exp(\mathbf{A}t)\vec{1},$$

Evaluating these derivatives at $t = 0$ allows us to avoid computing the exponential of matrices. Hence, vector $\vec{\delta}$ can be computed by solving:

$$\vec{\delta}\mathbf{Bi}_2^j\vec{1} = \vec{\beta}\mathbf{Bi}_1^j\vec{1}, \qquad 0 \leq j \leq n-2. \tag{8}$$

Once this system of equations is solved, the sub-stochasticity of vector $\vec{\delta}$ can be determined simply by verifying that all of its entries are nonnegative real numbers. Function $\textsc{Remove}(\cdot)$ in Algorithm 1 performs this computation.

In fact, the bidiagonality of the matrices allows further optimisations. We observe that for any bidiagonal PH-generator of dimension d, $\mathbf{Bi}[s_i, s_i] = -\mathbf{Bi}[s_i, s_{i+1}]$, for $1 \leq i < d$. Since both PH-generators in the system of equations are bidiagonal, we can prove the following lemma.

Lemma 2. *Equations* (8) *can be transformed into:*

$$\mathbf{A}[\delta_1, \ldots, \delta_{i-1}]^\top = \vec{b}, \tag{9}$$

where $\mathbf{A}$ *is an upper triangular matrix of dimension* $i-1$.

We have mentioned that the complexity of SPA(·) is $\mathcal{O}(n^3)$. Function REMOVABLE(·) can be performed with linear complexity, while constructing the system of linear equations (cf. Equation (9)) and solving it in function REMOVE(·) has complexity $\mathcal{O}(n^3)$.

Overall, the algorithm has complexity $\mathcal{O}(n^3)$, where n is the number of states of the original representation. It can be applied to arbitrary APH distributions, and hence to arbitrary CCC delays or their constituents. In fact, this can turn an exponential growth (in the number of composed components) of the matrix size into a linear growth, as we will demonstrate in the case study section. However, the algorithm is not guaranteed to produce minimal PH representations [29]. But it does so if the distribution is *acyclic ideal*. Recall that an acyclic-ideal PH distribution has an acyclic order that equals its algebraic degree.

Lemma 3. *[29, Lemma 10] Let* $(\vec{\alpha}, \mathbf{A})$ *be an APH representation and* $\mathsf{PH}(\vec{\alpha}, \mathbf{A})$ *acyclic ideal. The size of* $\mathsf{Red}(\vec{\alpha}, \mathbf{A})$ *is equal to the algebraic degree of* $\mathsf{PH}(\vec{\alpha}, \mathbf{A})$.

So, if we are able to stay inside the acyclic-ideal subset of APH distributions, we can always apply Red(.) and thereby keep the representation minimal. Thus, we are left with the problem to understand how likely it is that we are dealing with acyclic-ideal distributions when juggling with the CCC calculus.

We first provide partial answers: Exponential and Erlang distributions are acyclic ideal. The convolution, maximum, and minimum of Erlang distributions are acyclic ideal [28]. In full generality however, the binary operators of CCC do not, but do almost always, preserve acyclic ideality.

Theorem 7. *[29, Lemma 14] Convolution, maximum, and minimum operations are acyclic-ideal preserving almost everywhere.*

Thus, given two arbitrary acyclic-ideal APH distributions, regardless of the size of their representations, we cannot be certain, but can be *almost certain* that the APH distribution of their convolution, maximum, or minimum is also acyclic ideal, and hence is reducible to minimal size, by applying Red(.). We are almost certain, in the sense that, the measure of the set of acyclic-ideal distributions, where the resulting APH representation of the convolution, maximum, or minimum is not reducible to minimal size, is zero. Notably, Red(.) is always guaranteed to reduce a given acyclic representation to a minimal acyclic representation, still there is a marginal chance that smaller cyclic representations exist.

Given a process $P \in \mathsf{CCC}$, Algorithm 1 can be used to minimize the state space of its associated APH representation. The algorithm works on the corresponding ordered bidiagonal representation. Once this representation is obtained, we can transform it to its Cox representation. This Cox representation basically forms a process

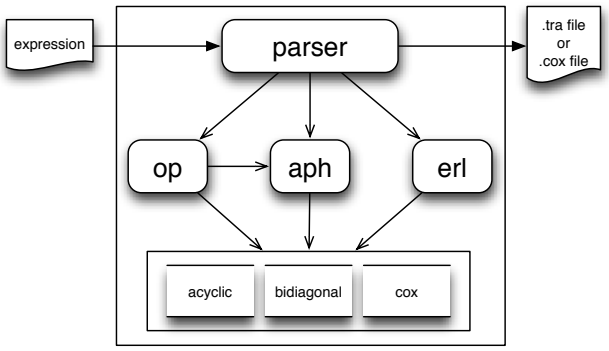

Fig. 6. Architecture of APHMIN

$\mathsf{Red}(P) \in \mathsf{CCC}$, the minimized process of P. Now, the following lemma is straightforward.

Lemma 4. *For all processes* $P \in \mathsf{CCC}$, $\mathsf{PH}(\mathsf{Red}(P)) = \mathsf{PH}(P)$.

Hence, Algorithm 1 can be used to minimize the size of processes in each composition step. By Theorem 7, the application of the convolution, minimum, or maximum operations on APH representations when accompanied by the minimization algorithm produces representations that are almost surely minimal. Therefore, the minimization algorithm can be used to keep the processes obtained after sequential, choice, and parallel compositions almost surely minimal.

5 Tool Implementation

This section describes the software service wrapping the CCC calculus and the minimization engine.

Overview and Implementation. We have implemented CCC as a means to compose, generate and manipulate APH representations, together with the minimization algorithm in a toolsuite called APHMIN. Fig. 6 depicts the architecture of the tool. APHMIN accepts as input any expression written in a prefix-notation version of CCC, and returns a .tra or .cox file containing the resulting minimized APH representation. The expression follows the grammar:

$$P ::= \exp(\lambda) \mid \mathrm{cox}(\mu, \lambda, P) \mid \mathrm{con}(P, P) \mid \min(P, P) \mid \max(P, P) \mid \mathrm{erl}(n, \lambda).$$

The correspondence between this grammar and the original CCC grammar is straightforward, with cox() representing $\lhd$. For enhanced convenience in building basic delays, we have added $\mathrm{erl}(n, \lambda)$ to generate an Erlang distribution of shape n and rate λ (expressible in CCC by n-fold sequential composition of rate λ).

APHMIN is assembled from four components: parser, erl, op, and aph. Component parser parses the input expression and invokes erl, op, and aph to evaluate the

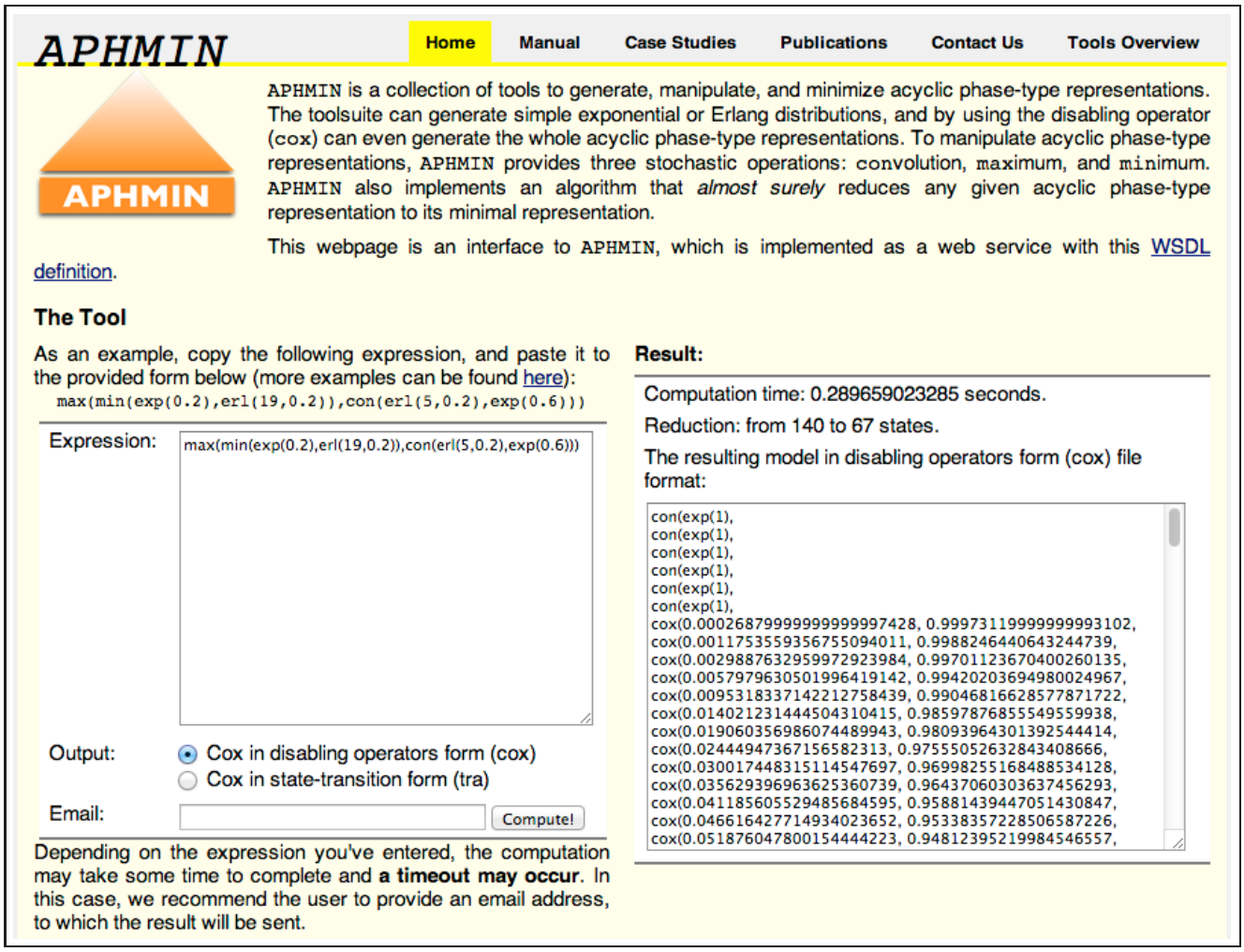

Fig. 7. The web-based interface of APHMIN

expression. Component erl is used to generate basic distributions: exponential and Erlang. Component op implements the disabling operator together with the three stochastic operations on two given APH representations. Component aph implements the minimization algorithm as well as transformations amongst the various forms of representations—acyclic, bidiagonal, and Cox. After executing an operation, op always calls aph to minimize the result of the operation. The last three components use .tra files as inputs and outputs. They also make use of three sparse-matrix-like data structures—acyclic, bidiagonal, and Cox—to store APH representations in various forms. The toolsuite is written in C and C++.

The toolsuite is wrapped in a simple web service, which supports only a single operation called `aphmin`, and deployed using the SOAP 1.1 protocol over HTTP. The request to the service takes a string containing the expression to be parsed by APHMIN, a string containing an email address to which the result will be sent if the user so wishes, and a string indicating the prefered format of the output—"`tra`" (as simple list of transitions) or "`cox`" (in Cox form as nested disabling operators). The latter format adheres to the above grammar, and hence enables direct cut-and-paste reuse of the tool output as part of a larger CCC expression. The response of the service consists of a float number representing the computation time required by APHMIN to produce the final result, two integer numbers representing the non-minimized size (which can be calculated by repeated uses of Theorem 3) and the actual minimized size of the resulting APH representation, and a string containing the final APH representation in the form chosen. The WSDL definition of the service can be found at `http://ada.cs.uni-saarland.de/aphminserver.php?wsdl`

Look and Feel. For the end user, we have wrapped the APHMIN web service into a web-based interface invoking the service. We use a web service and a web-based interface to ease first experimental evaluations of the approach without installation overhead. After positive evaluation, the tool is made available in binary form upon request to academic users. The interface (Fig. 7) comprises a form field where the user is expected to provide a CCC expression, the output format prefer-

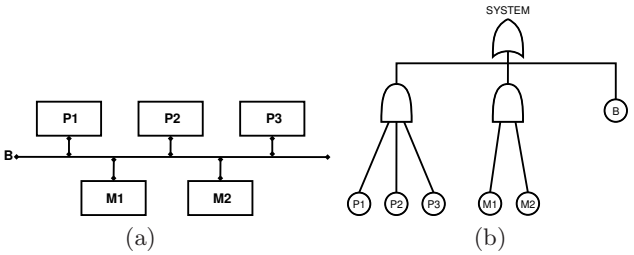

Fig. 8. (a) Three processors, two memories, and a bus (3P2M) model. (b) The fault tree of the 3P2M model

ence, and an email address. The response of the service will be displayed on the same page. Beside the resulting APH representation, the user also receives information about the computation time required and the sizes of non-minimized and minimized APH representation. The web-based interface is available at `http://depend.cs.uni-saarland.de/tools/aphmin`

6 Case Study.

We now illustrate the convenience and the effectiveness of the proposed approach by means of an exemplary case study. The size of the case to be discussed is far out of reach of traditional approaches to numerical Markov model analysis. We however circumvent the state space explosion via a minimality-by-construction approach rooted in APHMIN.

As a case study, we analyze the reliability of a processors-and-memories (3P2M) system modelled by a static fault tree [12,10]. Fig. 8(a) depicts a high-level view of the system. It consists of three identical processors (P1–P3) and two identical memory modules (M1 and M2), all connected by a bus (B). The processors and the memory modules work independently of each other. All components, including the bus, have a limited lifetime, and so they may fail. The lifetime, and hence the time to failure of each component is governed by a continuous probability distribution. The system fails when either all processors fail or all memory modules fail or the bus fails.

The fault tree model of the 3P2M system is shown in Fig. 8(b). A fault tree model consists of basic events and gates. A basic event represents the failure of some basic, indivisible component, while a gate determines the relationship and interdependency between basic events. In standard fault trees, there are three types of gates: OR, AND, and VOTING gates. The OR and AND gates are the standard logic gates.

There is a striking relation between CCC and fault trees: If considering the timed interpretation of static fault trees, OR and AND gates correspond exactly to the minimum and maximum operators on the time to failure distributions of the components [31], and hence they also correspond to the choice ($\oplus$) and the parallel ($\|$) operators of CCC, respectively.

Let P_i denote a CCC delay describing the time to failure of processor Pi, for $i = 1, 2, 3$. Further, let M_i be a CCC delay describing the time to failure of memory module Mi, for $i = 1, 2$, and similarly with B, the delay describing the time to failure of the bus B. Then the fault tree model in Fig. 8(b) can be expressed by the following CCC delay:

$$(P_1 \| P_2 \| P_3) \oplus (M_1 \| M_2) \oplus B. \tag{10}$$

In the analysis of the reliability of the 3P2M fault tree model, we perform several experiments. In each experiment, the time to failure of each component is governed by Erlang distributions of a particular shape. The mean value of the Erlang distributions governing each component, however, is kept constant across the experiments, which means that the rates are adjusted accordingly. The mean failure times of each processor, each memory, and the bus are set to 5, 3.33, and 7.14 years, respectively. The first three columns of Table 2 list the parameters of the Erlang distributions used in the experiments.

Columns 4–6 of Table 2 summarize the results of the experiments. We present six model instances, where we vary the shapes of the Erlang distributions governing the basic events, ranging from 1, namely exponential distributions, to 100. The fourth column of the table (Original) provides the size of the APH representations when they are generated without any size reduction whatsoever. The fifth column (Inter.) corresponds to the size of the largest intermediate APH representations the reduction algorithm encounters while minimizing each of the model instances. Recall that after carrying out each operation in Equation (10), the reduction algorithm can be used to reduce the resulting intermediate model (by cut-and-paste, if using the web interface). In this way, the size of the results of subsequent operations can be kept small. The state spaces shown in the column are usually the intermediate results prior to the last operation, which in this case is the last choice operation. The sixth column (Final) corresponds to the size of the final APH representations. Compared to the original state spaces, the size of the final state spaces is orders of magnitude smaller. While the size of an original model grows multiplicatively in the sizes of its components, the size of a reduced representation grows additively in the sizes of its components.

The last three columns of Table 2 present details on the computation times for the models. They range from milliseconds for the smallest models to more than 60 hours for the largest ones. While this seems like an unattractively long computation time, it should be noted that the original model is far out of reach of any explicit state numerical analysis engine, while the final model is truly small. So, patience does pay off. In general, the computation time is dominated by the SPA algorithm, taking more than 95% of the total time.

Table 2. Model parameters and state space reductions for the 3P2M models

Processors	Memories	Bus	State Space			Comp. Time (sec.)		
			Original	Inter.	Final	SPA	Red.	Proc.
exp(0.2)	exp(0.3)	exp(0.14)	21	6	6	< 1	< 1	< 1
erl(5, 1)	erl(5, 1.5)	erl(5, 0.7)	37625	450	114	< 1	< 1	< 1
erl(10, 2)	erl(10, 3)	erl(10, 1.4)	1596000	1950	249	4	18	< 1
erl(20, 4)	erl(20, 6)	erl(20, 2.8)	81488000	8100	519	35	265	1
erl(50, 10)	erl(50, 15)	erl(50, 7)	$> 1.72 \cdot 10^{10}$	51750	1329	10999	800	16
erl(100, 20)	erl(100, 30)	erl(100, 14)	$> 1.05 \cdot 10^{12}$	208500	2679	219180	10086	98

7 Concluding Remarks

The paper has introduced a web service that combines an expressive and convenient approach to generate arbitrary continuous probability distributions with a way to represent the result in the most concise way—almost surely. The web service is based on a calculus of APH distributed delays, CCC. Though this calculus lacks means for interaction and recursion, these can be adapted from other calculi as desired. Another option is to keep CCC separated, but to use the APHMIN web service to arrive at the (almost surely) most concise representation of an APH distributed delay of interest. The result can indeed be imported (by encoding the absorbing CTMC) into any modelling tool supporting exponential distributions, ranging from stochastic π-calculus or Petri nets, to PEPA or IMC, and even to UPPAAL with stochastic semantics.

Acknowledgments

We would like to thank the reviewers for their valuable and detailed comments and suggestions in improving this paper.

References

1. L. Aceto, W. J. Fokkink, and C. Verhoef. *Handbook of Process Algebra*, chapter Structural operational semantics, pages 197–292. Elsevier, 2001.
2. M. Bernardo and R. Gorrieri. Extended Markovian process algebra. In U. Montanari and V. Sassone, editors, *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, volume 1119 of *Lecture Notes in Computer Science*, pages 315–330. Springer, 1996.
3. T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14:25–59, 1987.
4. G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, 11:433–452, 1988.
5. P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.
6. L. Cardelli and R. Mardare. The measurable space of stochastic processes. In *Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the*, pages 171–180, 2010.
7. D. R. Cox. A use of complex probabilities in the theory of stochastic processes. *Proceedings of the Cambridge Philosophical Society*, 51(2):313–319, 1955.
8. A. Cumani. Canonical representation of homogeneous Markov processes modelling failure time distributions. *Microelectronics and Reliability*, 2(3):583–602, 1982.
9. A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, and Z. Wang. Stochastic semantics and statistical model checking for networks of priced timed automata. *CoRR*, abs/1106.3961, 2011.
10. J. B. Dugan and S. A. Doyle. New results in fault-tree analysis. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 1–23, Las Vegas, 1996.
11. Q.-M. He and H. Zhang. Spectral polynomial algorithms for computing bi-diagonal representations for phase type distributions and matrix-exponential distributions. *Stochastic Models*, 2(2):289–317, 2006.
12. E. J. Henley and H. Kumamoto. *Probabilistic Risk Assessment: Reliability Engineering, Design, and Analysis*. IEEE Press, 1992.
13. H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002.
14. H. Hermanns and M. Rettelbach. Syntax, semantics, equivalences and axioms for MTIPP. In *Proceeding of the 2nd Workshop PAPM '94*, pages 71–87, 1994.
15. J. Hillston. *A compositional approach to performance modelling*. Cambridge University Press, 1996.
16. M. A. Johnson and M. R. Taaffe. The denseness of phase distributions. School of Industrial Engineering Research Memoranda 88-20, Purdue University, 1988.
17. B. Klin and V. Sassone. Structural operational semantics for stochastic process calculi. In R. M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2008.
18. R. S. Maier and C. A. O'Cinneide. Closure characterisation of phase-type distributions. *Journal of Applied Probability*, 29(1):92–103, 1992.
19. R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, New York, NY, USA, 1999.

20. M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Dover, 1981.
21. R. Nicola, D. Latella, M. Loreti, and M. Massink. Rate-based transition systems for stochastic process calculi. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, and W. Thomas, editors, *Automata, Languages and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 435–446. Springer Berlin Heidelberg, 2009.
22. C. A. O'Cinneide. Characterization of phase-type distributions. *Communications in Statistics: Stochastic Models*, 6(1):1–57, 1990.
23. C. A. O'Cinneide. Phase-type distributions and invariant polytopes. *Advances in Applied Probability*, 23(43):515–535, 1991.
24. C. A. O'Cinneide. Triangular order of triangular phase-type distributions. *Communications in Statistics: Stochastic Models*, 9(4):507–529, 1993.
25. G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–140, 2004.
26. C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(7):578–589, 1995.
27. R. Pulungan. *Reduction of Acyclic Phase-Type Representations*. PhD thesis, Universität des Saarlandes, Saarbruecken, Germany, 2009.
28. R. Pulungan and H. Hermanns. The minimal representation of the maximum of Erlang distributions. In *Proceedings – 14th GI/ITG Conference – Measurement, Modelling and Evaluation of Computer and Communication Systems, Dortmund, March 31 – April 2, 2008*, pages 207–221. VDE Verlag, 2008.
29. R. Pulungan and H. Hermanns. Acyclic minimality by construction—almost. In *QEST 2009, Sixth International Conference on the Quantitative Evaluation of Systems, Budapest, Hungary, 13-16 September 2009*, pages 63–72. IEEE Computer Society, 2009.
30. M. Timmer, J.-P. Katoen, J. van de Pol, and M. Stoelinga. Efficient modelling and generation of Markov automata. In M. Koutny and I. Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2012.
31. W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission, Washington, DC, 1981.

A Kronecker Product and Kronecker Sum

Let $\mathbf{A}$ be an $k \times l$ matrix and B be an $m \times n$ matrix. Then the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is an $km \times ln$ matrix given by:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \cdots & a_{1,l}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \cdots & a_{2,l}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1}\mathbf{B} & a_{k,2}\mathbf{B} & \cdots & a_{k,l}\mathbf{B} \end{bmatrix}.$$

If $\mathbf{A}$ is an $m \times m$ matrix and $\mathbf{B}$ is an $n \times n$ matrix, then the Kronecker sum $\mathbf{A} \oplus \mathbf{B}$ is an $mn \times mn$ matrix given by:

$$\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_n + \mathbf{I}_m \otimes \mathbf{B},$$

where $\mathbf{I}_k$ denotes the $k \times k$ identity matrix.