

Using Max-Plus Algebra for the Evaluation of Stochastic Process Algebra Prefixes

Lucia Cloth, Henrik Bohnenkamp, and Boudewijn Haverkort

Department of Computer Science
Laboratory for Performance Evaluation and Distributed Systems
RWTH Aachen, D-52056 Aachen
{lucia, henrik, haverkort}@cs.rwth-aachen.de
Phone: +49 241 44021432

Abstract. In this paper, the concept of complete finite prefixes for process algebra expressions is extended to stochastic models. Events are supposed to happen after a delay that is determined by random variables assigned to the preceding conditions. Max-plus algebra expressions are shown to provide an elegant notation for stochastic prefixes not containing any decisions. Furthermore, they allow for the computation of performance measures. The derivation of the so called k -th occurrence times is shown in detail.

1 Introduction

Stochastic process algebras (SPA) have become accepted languages for the description of functional and quantitative aspects of distributed systems. Their compositionality allows for easy-to-read specifications and for the reuse of modules. The evaluation and verification of SPA models is frequently based on interleaving semantics, which are prone to state-space explosion. Action delays are often restricted to exponential distributions, so that results can be obtained by Markovian analysis. Most interleaving semantics do not allow for general distributions.

In this paper, we avoid some of the above restrictions by using a true-concurrency semantics: a finite stochastic event structure prefix. Because of its non-interleaving nature, it is generally smaller than an interleaving transition system, and has an explicit notion for the concurrent execution of actions. Additionally, it allows for general distributions for the description of action delays. Our semantics for a simple stochastic process algebra is based on the complete finite prefix for (non-stochastic) process algebra introduced in [9], which in turn was inspired by a McMillan prefix for Petri nets [10] (improved by [5]). Expressions in max-plus algebra [2] are shown to be a natural description for timing properties of systems. We show that our approach is appropriate for the derivation of performance measures. Similar work can be found in [13].

The rest of this paper is organised as follows. In Section 2 we briefly review the derivation of a complete finite prefix for simple stochastic process algebra

expressions. Section 3 gives an overview of max-plus algebra, especially of max-plus matrix operations and their application to graphs and systems of linear equations. We use max-plus methods in Section 4 for the description and evaluation of prefixes. Section 5 concludes with a discussion of the results, drawbacks and further ideas. The material presented in this paper is based on [4].

2 Complete Finite Prefix for Stochastic Process Algebra

This section gives a brief outline on computing prefixes. Further details can be found in [4,9].

2.1 Stochastic Process Algebra

We use a simple SPA throughout this paper. Its syntax is defined as follows.

Definition 1. *Let a be an action, F a distribution function with $F(x) = 0$ for $x < 0$, and A a set of actions. The **stochastic process algebra expressions** are defined by the following rules:*

$$P ::= \text{stop} \mid \langle a, F \rangle.P \mid P + P \mid P \parallel_A P \mid Id \quad \diamond$$

stop is the process that does nothing. $\langle a, F \rangle.P$ is the process that executes action a after a delay that is distributed according to F . We can think of a clock that is set to a randomly chosen delay (according to F) and that counts downwards to 0, where it eventually expires. After the clock has expired, action a can be executed. The execution of an action does not consume time. The process $P_1 + P_2$ may behave either as P_1 or as P_2 . Which behaviour is chosen depends on the clocks that are running for P_1 and P_2 : the fastest wins. If the winner can not be uniquely determined, the choice is made non-deterministically. The process $P_1 \parallel_A P_2$ describes the independent parallel execution of P_1 and P_2 . Only if one of them wants to execute an action that is contained in the synchronisation set A , it has to wait until the other process becomes ready to execute this action as well. Both execute the action synchronously. An identifier (Id) is a place-holder for an expression P that is defined by an equation $Id = P$. We can instantiate a process several times, i.e., for recursive definitions.

Example 1. As running example, we consider a simple buffer with a writing and a reading agent. Data of random length is written into the buffer (**in**) and later read from the buffer (**out**). In order to reflect the random length, both operations require an exponential delay, but reading is faster than writing.

The writer thinks for a uniformly distributed amount of time (between 1 and 5 time units) and writes then something into the buffer:

$$\text{Writer} = \langle \text{thinkW}, \text{uniform}([1, 5]) \rangle. \langle \text{in}, \text{exp}(5) \rangle. \text{Writer}$$

The reader waits exactly 2 time units before it reads from the buffer:

$$\text{Reader} = \langle \text{thinkR}, \text{det}(2) \rangle. \langle \text{out}, \text{exp}(8) \rangle. \text{Reader}$$

with

$$\text{Buffer} = \langle \text{in}, \text{det}(0) \rangle . \langle \text{out}, \text{det}(0) \rangle . \text{Buffer}.$$

The entire system is described by

$$\text{System} = \text{Writer} \parallel_{\{\text{in}\}} \text{Buffer} \parallel_{\{\text{out}\}} \text{Reader}.$$

Note that the buffer itself does not delay the `Writer` nor the `Reader` upon `in` and `out` actions, respectively, since its delay is 0 time units with probability 1.

□

2.2 Condition Event Structures

The base structure for prefixes are *condition event structures*. Events describe the possible occurrences of actions. Each possible occurrence of an action is denoted by an unique event. We will here identify events by their corresponding action name, possibly adding a subscript in order to form a unique name.

Definition 2. A **stochastic condition event structure** is a tuple $(D, E, \sharp, \prec, l_{\mathcal{F}})$ where

- D is a set of **conditions**,
- E is a set of **events**,
- $\sharp \subset D \times D$ is the **choice relation** (symmetric and irreflexive)
- $\prec \subset (D \times E) \cup (E \times D)$ is the **flow relation**,
- $l_{\mathcal{F}} : D \rightarrow DF$ is a mapping from conditions to distribution functions. \diamond

Note that neither D nor E has to be finite. For technical reasons, we assume E to contain a bottom event \perp that denotes the start of the modelled system behaviour.

Stochastic condition event structures have a simple graphical representation. Conditions are drawn as circles, labelled with their names and distributions. Events simply appear with their names. The flow relation is represented by arrows, the choice relation by lines with a \sharp on them.

An event structure models all possible behaviours of a system. With the start of the system, we also set imaginary clocks assigned to the conditions directly following the bottom event \perp , i.e., for all $d \in D : \perp \prec d$. If for any event the clocks of all directly preceding conditions have expired, this event occurs immediately, starting the clocks of the succeeding conditions, and so on.

The occurrence of an event e inhibits forever the occurrence of all events being in conflict with e .

Downwards closed sets of pairwise non-conflicting events are called *configurations*. The *local configuration* $[e]$ of an event e is defined by

$$[e] := \{e' \in E \mid e' \prec^* e\}.$$

\emptyset is the local configuration of \perp . Corresponding to configurations are *cuts*: a cut is the set of conditions “following” a configuration. Each configuration corresponds to a cut and vice versa. After the occurrence of an event, usually some time has to pass before the next occurrence of an event. Consequently, cuts represent *states* in which the system may spend time.

2.3 Unfolding

The algorithm in [9] for the construction of a condition event structure for a process algebra expression is derived from the unfolding algorithm for Petri nets [10]. So we refer to the resulting event structure as *unfolding* as well. As shown in [4], the unfolding algorithm in [9] can directly be used to derive unfoldings for SPA expression, yielding stochastic event structures.

The unfolding algorithm, which we do not describe in detail here, “unfolds” the SPA expression, generating conditions and events step by step. The conditions are additionally labelled with so-called *components* (via a mapping l_C). Components are basically prefixed stochastic process algebra expressions, equipped with a notion for the synchronisation context in which they occur.

Definition 3. *Let P be a stochastic process algebra expression, a an action, and A a set of actions and F a distribution function. A **component** C is defined by*

$$C ::= \text{stop} \mid \langle a, F \rangle.P \mid C \parallel_A \mid \parallel A C. \quad \diamond$$

Stochastic process algebra expression are decomposed into components (roughly by splitting parallel and choice expressions).

Components are assigned to conditions. The choice relation between components, derived during the decomposition of SPA expressions, determines the choice relation between the respective conditions.

For sets of components a structured operational semantics (SOS) can be defined [9]. Starting from a set of initial components, all possible transitions (labelled with actions) are derived according to this SOS. For each transition, a new event and its successor conditions (with appropriately assigned components) is introduced in the unfolding. Hence, each occurrence of an action is transformed into a unique event. The whole unfolding is created by successive application of this derivation of transitions from “unused” components in the unfolding.

If an action is executed by a single component, the corresponding event will have exactly one preceding condition; if it is the result of a synchronisation, it will have several predecessors. Apart from the explicit choice, described by the choice relation on components, there can also be choice as a result of synchronisation on common labels that can be recognised by the existence of conditions with more than one succeeding event. In general (for recursive SPA expressions) there is an infinite number of possible occurrences of actions, and so the resulting stochastic condition event structure will be an infinite structure.

States. A cut in the unfolding corresponds to a set of conditions, and so it does (via the mapping l_C) to a set of components. These sets will be such that they correspond to a valid stochastic process algebra expression that is reachable (via the classical interleaving semantics) from the original SPA expression. The expressions can be assembled by undoing the decomposition. They are called the *states* of the cut (resp. of the corresponding configuration). In [9] it is shown that *exactly* the reachable states are represented in the unfolding.

2.4 Prefix

For finite state SPA processes, the possible behaviour is already represented in a finite prefix of the unfolding. In this section we comment on the construction of the prefix.

Events are made comparable by a so-called *adequate order* \sqsubset such that (roughly) $e' \sqsubset e$ if e' is encountered earlier in the unfolding process. There is a local configuration $[e]$ for all events e in the unfolding and consequently we have local states, written $St([e])$. We use the idea of states assigned to events for the construction of a finite prefix: Whenever we find an event e , such that there exists an event e' with $e' \sqsubset e$ and $St([e]) = St([e'])$ we call it a *cutoff* event. We do not consider the unfolding beyond the cuts of cutoff events. The resulting prefix is finite and complete (although not always the smallest possible).

The events in the prefix can be seen as representatives of whole classes of events. They represent the finite number of possible actions, the occurrences of which determine the event classes. These classes may have an infinite number of elements.

Example 2. In Figure 1 the unfolding and the complete finite prefix for the SPA expression **System** of Example 1 is depicted. In order to distinguish different occurrences of an action, events have action names with unique subscripts. We name conditions with capital letters. \square

3 Max-Plus Algebra

3.1 Introduction

The so-called max-plus algebra has been developed for the description and evaluation of discrete-event systems (DES). A complete survey can be found in [2]. Stochastic DES are treated in [12]. The max-plus algebra is an algebraic ring comprising the set of real numbers (extended by $-\infty$) as carrier set and \oplus and \otimes as operations. \oplus is interpreted as max and \otimes as + (we also write \oslash for $-$). In our models we express time by random variables that are mappings from some probability space into the set of real numbers. By $\mathbf{0}$ we denote the random variable which is $-\infty$ with probability 1; it is the zero-element for the maximum operation \oplus . With $\mathbf{1}$ we denote the random variable which is 0 with probability 1; it is the unit-element for \otimes .

3.2 Matrices and Weighted Directed Graphs

The operations \oplus and \otimes can be extended to vectors and matrices whose elements are drawn from the max-plus algebra.

Definition 4. A **directed graph** (*digraph*) $G = (V, E)$ consists of a set $V = \{1, 2, \dots, n\}$ of nodes (*vertices*) and a set $E \subseteq V \times V$ of arcs (*edges*).

G is a *max-plus weighted digraph*, if it comes with a mapping w that assigns to each arc a max-plus random variable. \diamond

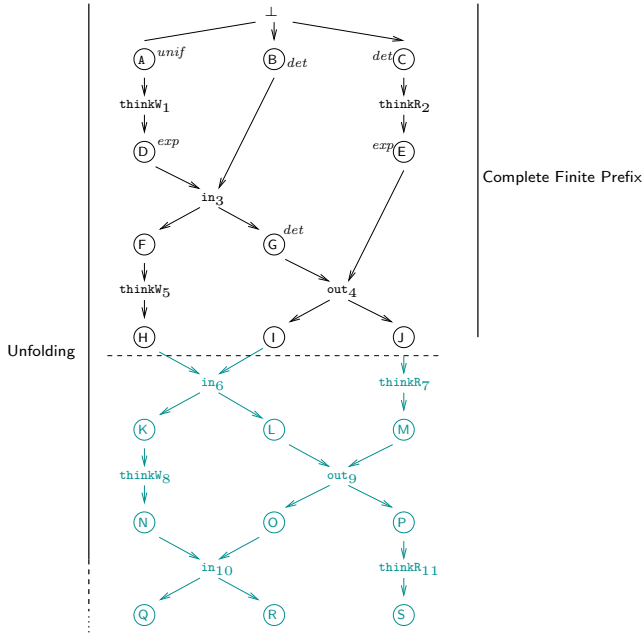


Fig. 1. Unfolding and Complete Finite Prefix for $\text{Writer} \parallel_{\text{in}} \text{Buffer} \parallel_{\text{out}} \text{Reader}$

An alternative representation for a graph is an *adjacency matrix* A , where

$$A_{ij} := \begin{cases} w((i, j)), & \text{if } (i, j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

Of course it is also possible to construct a graph to a given $n \times n$ max-plus matrix. Squaring the adjacency matrix A results in A^2 , which represents the graph with an arc between two nodes n_1 and n_2 into the resulting graph, if there is a path of length two in the graph described by A from node n_1 to node n_2 . Its weight is given by the sum of the original weights. In the *star* of a matrix, paths of arbitrary length are subsumed.

Definition 5. For A an adjacency matrix, A^* is defined by the max-plus sum, that is the maximum, of all its powers:

$$A^* := \bigoplus_{k=0}^{\infty} A^k.$$

◇

In an acyclic graph, there is a maximum length for all paths, and so it is possible to compute the star with a finite number of operations:

$$A^* = \bigoplus_{k=0}^n A^k, \text{ for some } n < \infty.$$

3.3 Systems of Linear Equations

We can formulate systems of linear equations in max-plus algebra, as we can do in the field of real numbers. The well-known techniques for the solution of systems of linear equations depend on the inverse of \oplus . Unfortunately, max has no inverse. Consequently, we are not able to solve general systems of linear equations in the max-plus algebra. However, there is a special case that is well suited for our purposes [2], as we will see in Section 4.

Theorem 1. *Consider a system of linear equations as follows: $T = T \otimes A \oplus S$, where T is the solution vector we are looking for, A is a matrix and S a given vector of suitable dimensions. If A is strictly upper triangular, the solution is given by $T = S \otimes A^*$.*

Proof. By insertion of $S \otimes A^*$ in the equation $T = T \otimes A \oplus S$. ◇

4 Application to Prefixes

We use the max-plus methods introduced in the previous section for the notation and evaluation of prefixes.

4.1 Occurrence Times and Linear Max-Plus Expressions

The flow relation of the event structure prefixes describes the causal dependencies between events. Conditions are equipped with random variables that describe the delay between events. Combining dependencies and delays, we can determine a random variable $O(e)$ describing the time the event e is bound to occur.

We can derive these occurrence times recursively, starting with the bottom event \perp which we assume to happen at time $O(\perp) = 0 = \mathbb{1}$. Then, we look at immediate successor events of \perp : their occurrence times are determined by the maximum delay assigned to the conditions between \perp and themselves (plus the occurrence time of \perp , which is zero). We introduce for each condition d a random variable X_d distributed according to $l_{\mathcal{F}}(d)$. Then we can express occurrence times for all events: we add the random variable of the appropriate conditions to the occurrence times of the predecessors and then take the maximum.

Definition 6. *The occurrence time $O(e)$ of an event e is defined recursively as $O(\perp) := \mathbb{1}$ and $O(e) := \bigoplus_{e' \prec_d \prec_e} O(e') \otimes X_d$ for $e \neq \perp$, respectively.* ◇

The resulting expressions are linear in the max-plus algebra.

Example 3. For the unfolding in Figure 1 we have the following occurrence times:

$$\begin{aligned}
 O(\text{thinkW}_1) &= O(\perp) \otimes X_A \\
 O(\text{thinkR}_2) &= O(\perp) \otimes X_C \quad \text{deterministic} \\
 O(\text{in}_3) &= (O(\text{thinkW}_1) \otimes X_D) \oplus (O(\perp) \otimes X_B) \\
 O(\text{thinkW}_5) &= O(\text{in}_3) \otimes X_F \\
 O(\text{out}_4) &= (O(\text{in}_3) \otimes X_G) \oplus (O(\text{thinkR}_2) \otimes X_E) \\
 O(\text{thinkR}_7) &= O(\text{out}_4) \otimes X_J
 \end{aligned}$$

$O(\text{thinkW}_1)$ is a uniform random variable with expectation 3; $O(\text{in}_3)$ is the sum of a uniform and an exponential random variable, its expectation is 3.2. $O(\text{thinkR}_7)$ is the sum of a deterministic random variable and the maximum of sums of random variables. Its expectation is 5.6857. \square

4.2 Representation of Prefixes

In their graphical form, prefixes have a close resemblance to max-plus weighted graphs. We now show how to construct a graph (and the corresponding matrix) for a stochastic event structure prefix.

We interpret events as the nodes of a graph. The flow relation seen at event level represents the arcs, i.e., if there is a condition d such that $e_1 \prec d \prec e_2$, then there is an arc from e_1 to e_2 , denoted (e_1, e_2) . The weight of this arc is given by the delay assigned to d .

Definition 7. For a finite stochastic condition event structure $(D, F, \#, \prec, l_{\mathcal{F}})$ the corresponding weighted digraph G is defined as follows:

- $V = F$,
- $(e_1, e_2) \in E : \iff \exists d \in D : e_1 \prec d \prec e_2$
and $w((e_1, e_2)) := X_d$ with $X \sim l_{\mathcal{F}}(d)$. \diamond

Note that we are loosing information: the conflict between two events is no longer expressed!

Example 4. The graph (left) and matrix (right) of Figure 2 represent the prefix of Figure 1. \square

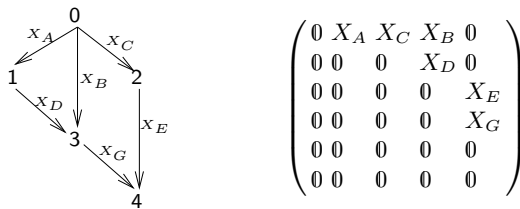


Fig. 2. Graph and Matrix for the Prefix of Figure 1

4.3 The Repeating Part

In the preceding section we have shown how max-plus matrices can be used for the representation of prefixes, although they do not reflect the conflict situations

between events. Therefore, we have to restrict ourselves to *decision-free* prefixes, i.e., prefixes without conflicts between events; all events must eventually occur. For the corresponding SPA expressions this means that they are not allowed to contain the choice operator and that they have to avoid synchronisations on common labels.

Finite prefixes are constructed by finding cutoff events in the unfolding of a stochastic process algebra expression. To any such cutoff event e in an unfolding there is another event e' that is smaller w.r.t. the chosen adequate order ($e' \sqsubset e$) and has the same local state ($St(e') = St(e)$). As a consequence, for an event e^* succeeding the local configuration of e' , there is an e^* succeeding the local configuration of e and $St(e^*) = St(e^*)$. Thus, by repetition of this argument we can state that the events of the prefix between e' and e , i.e., the set $[e] \setminus [e']$, represents event classes with an infinite number of members. Since in decision-free systems all events must eventually happen, the occurrence of representatives of these classes are observed infinitely often in the evolution of the system.

Definition 8. Let *Cutoff* be the set of all cutoff events in a given decision-free prefix $(D, E, \emptyset, \prec, l_{\mathcal{F}}, l_{\mathcal{C}})$ and *Cutoff'* the set of those preceding events that have the same local state. Then the set of **repeating events** of the prefix is defined as

$$E^r := \bigcup_{e \in \text{Cutoff}} [e] \setminus \bigcup_{e' \in \text{Cutoff}'} [e'].$$

The set of **repeating conditions** is given by $D^r := \{d \in D \mid \exists e \in E^r : d \prec e \vee e \prec d\}$. The **repeating part** of the prefix is given by $(D^r, E^r, \emptyset, \prec \upharpoonright (D^r \times E^r \cup E^r \times D^r))$. \diamond

Example 5. Figure 3 (a) shows the repeating part of the prefix of Figure 1. \square

4.4 The k -th Occurrence Time

When does the k -th representative of a certain infinite event class occur? As all time intervals in our system are described by random variables, the instant can also be expressed as a random variable.

To make the formal treatment easier, we identify the events (or more precise, the event classes) E^r of the repeating part with natural numbers starting with 1, in such a way that $<$ on the natural numbers respects the partial order on events. With respect to the graphical representation, we do a *topological sorting* of events; for the running example we actually have already done this with the subscripts added to action names.

Definition 9. For event class i , $T_i(k)$ is the instant of time when the k -th representative of this class occurs. $T(k)$ is then a max-plus vector containing the k -th occurrence times for all event classes. \diamond

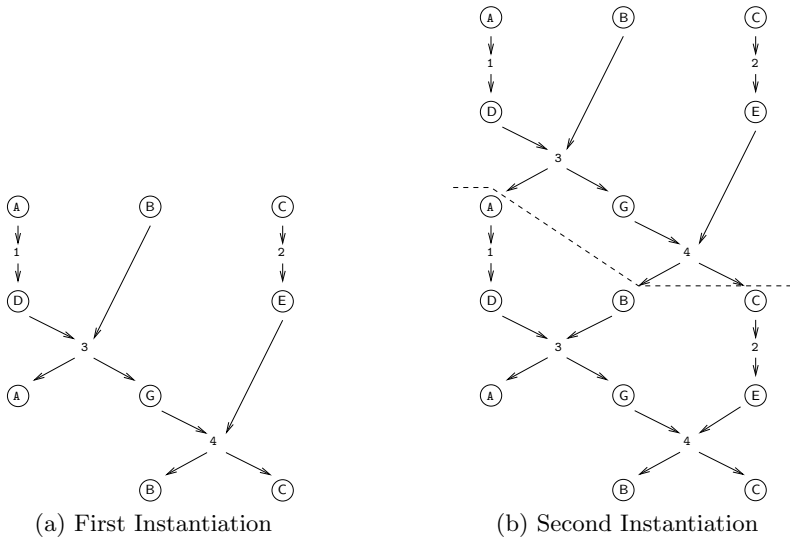


Fig. 3. Repeating Part of the Prefix

$T(1)$ is easily determined. We simply take the original prefix and compute the occurrence times for all events in the repeating part. Put in the order given by the topological sorting, we obtain $T(1)$.

Theorem 2. Let e_i be the minimal element of event class i according to \prec^* . The first occurrence time of class i is then given by the occurrence time of e_i :

$$T_i(1) = O(e_i).$$

Proof. If $e_i \prec^* e$, then e_i must occur before e . Because of being minimal, e_i is the first representative of class i to happen. \diamond

For our example

$$T(1) = (X_A, X_C, X_A \otimes X_D, X_A \otimes X_D \oplus X_C \otimes X_E).$$

But how do we derive $T(k)$ for $k > 1$? We can append a new instantiation of the repeating part to the prefix by identifying (final) conditions of the prefix with equally labelled (initial) conditions of the repeating part (cf. Figure 3 (b)). Doing this infinitely often, we obtain the whole infinite unfolding.

The events of this new instantiation constitute the second occurrences of the infinite event classes. One can easily calculate their occurrence times. But there is a formal problem: we must distinguish between the random variables of conditions with identical labels from the first and the second instantiation. We do this by introducing explicitly the parameter k : $X_d(k)$ denotes the random variable associated to d in the k -th instantiation of the repeating part.

We already know $T(1)$ when computing $T(2)$. Similar to the derivation of ordinary occurrence times, we are able to describe $T_i(2)$ by max-plus expressions concerning the direct predecessors of i . In general, $T_i(2)$ depends on $T_j(2)$ for its predecessor events j in the second instance of the repeating part. If any of the conditions preceding i is initial in the repeating part, $T_i(2)$ depends also on $T(1)$. For our example

$$\begin{aligned} T_1(2) &= T_3(1) \otimes X_A(2), \\ T_2(2) &= T_4(1) \otimes X_C(2), \\ T_3(2) &= T_1(2) \otimes X_D(2) \oplus T_4(1) \otimes X_B(2), \\ T_4(2) &= T_3(2) \otimes X_G(2) \oplus T_2(2) \otimes X_E(2). \end{aligned}$$

A sharp look reveals that this is a system of linear equations in the max-plus algebra. It can be rewritten in matrix form:

$$T(2) = \left(T(2) \otimes \begin{pmatrix} \mathbf{0} & \mathbf{0} & X_D(2) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & X_E(2) \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & X_G(2) \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \right) \oplus \left(T(1) \otimes \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ X_A(2) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & X_C(2) & X_B(2) & \mathbf{0} \end{pmatrix} \right)$$

$T(1)$ is already known, so the second part of the sum forms a vector, hence the equation is in the general form described in Theorem 1.

The results for $T(2)$ can be generalised. The vector $T(k+1)$ depends on itself for “internal” event classes, and on $T(k)$ for “initial” event classes.

Definition 10. *The internal dependencies within the $(k+1)$ -th instantiation of the repeating part are expressed by*

$$A_{ij}(k+1) := \begin{cases} X_d(k+1), & \text{if } i \prec d \prec j, \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

The dependencies between the k -th and the $(k+1)$ -th instantiation of the repeating part are given by

$$B_{ij}(k+1) := \begin{cases} X_d(k+1), & \text{if } d \text{ initial, } d \prec j, d' \text{ final, } i \prec d', l_C(d) = l_C(d'), \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

With these definitions we get

Theorem 3.

$$T_i(k+1) = \left(\bigoplus_{j=1}^n T_j(k+1) \otimes A_{ij}(k+1) \right) \oplus \left(\bigoplus_{j=1}^n T_j(k) \otimes B_{ij}(k+1) \right)$$

In matrix form:

$$T(k+1) = \left(T(k+1) \otimes A(k+1) \right) \oplus \left(T(k) \otimes B(k+1) \right) \quad \diamond$$

Due to the acyclic topology of the repeating part and the adequate labelling of event classes, the $A(k)$ are strictly upper triangular matrices. Hence, according to Theorem 1, we state

Theorem 4.

$$T(k+1) = T(k) \otimes B(k+1) \otimes A(k+1)^*, \quad \text{for } k > 1.$$

Proof. Application of Theorem 1. ◇

The calculation of $T(k+1)$ now simply depends on $T(k)$. Starting with $T(1)$, which is given by the occurrence times, all $T(k)$ can be determined recursively, step by step. For short, set $C(k) := B(k) \otimes A(k)^*$, such that the equation becomes

$$T(k+1) := T(k) \otimes C(k+1).$$

$B(k)$ contains the dependencies of all event classes to the previous occurrence of final events. $A(k)^*$ cumulates the internal delays such that they are expressed directly from one event to another without intermediate event. $C(k) = B(k) \otimes A(k)^*$ expresses the delay between the terminal events of instantiation number k and the $(k+1)$ -th occurrence of all event classes. Thus, only the rows corresponding to terminal event classes contain entries different from $\mathbf{0}$ at all.

Example 6. For the running example, taking into account that $X_B(k)$ and $X_G(k)$ are deterministically 0, we find:

$$C(k) = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ X_A(k) & \mathbf{0} & X_A(k) \otimes X_D(k) & X_A(k) \otimes X_D(k) \\ \mathbf{0} & X_C(k) & \mathbf{0} & X_C(k) \otimes X_E(k) \end{pmatrix} \quad \square$$

4.5 Cycle Time

Based on the k -th occurrence times of the event classes we can derive recurrence times for the different classes. As recurrence time we designate the time between the occurrence of two subsequent representatives of a single event class. It is again a random variable, describing the distribution of time. The recurrence times of all event classes are comprised in a vector $RT(k)$:

Definition 11. $RT_i(k)$ denotes the **recurrence time** between the k -th and the $(k+1)$ th occurrence of an representative of event class i :

$$RT(k) := T(k+1) \oslash T(k) \quad \diamond$$

The distributions of the recurrence times can change over time. We are interested in the long term behaviour of a system, so we take the limit of $RT(k)$ as a measure for the general recurrence times (given existence):

$$RT := \lim_{k \rightarrow \infty} RT(k).$$

The expected limiting recurrence time must be the same for all event classes. If one class were “faster” than the others, it would have to wait for its predecessors, so it would no longer be faster. Clearly this is only true for those prefixes with connected repeating part.

We take this identical limiting recurrence times as cycle time for the whole system. It denotes the time it takes to execute one instantiation of the repeating part of the system.

Definition 12. Let CT be the expected limiting recurrence time for any event class:

$$CT := E[RT_i], \quad \text{for arbitrarily chosen } 1 \leq i \leq n.$$

CT is called the **mean cycle time** of the system. ◇

Unfortunately, the calculation of the mean cycle time comes with all those problems known from the solution of task graph models [1].

Example 7. For our example, the mean cycle time for all event classes is

$$CT = E[(X_A \otimes X_D) \oplus (X_C \otimes X_E)] = 5.6857.$$

□

4.6 Condition Holding Times

So far we presented two simple performance measures that can be derived from a complete finite prefix: the k -th occurrence times of events and the mean cycle time of the repeating part. In this section we show how to use the k -th occurrence time of events for the calculation of a performance measure concerning conditions.

Conditions are assigned stochastic delays. If the system enters a condition, a clock is started according to the specified distribution function. When this clock reaches zero, the succeeding event is locally activated. But it does not need to be the case that the event is also globally activated. Sometimes the system must remain in a condition, even when the clock already has reached zero. The amount of time the system actually spends in a condition, i.e., the delay plus the possible waiting time, is called its holding time.

Definition 13. Let α be a non-terminal condition of the repeating part. Then

$$H_\alpha(k) = \begin{cases} T_{j_1}(k) \otimes T_{j_2}(k), & \text{if } j_2 \prec \alpha \prec j_1, \\ T_{j_1}(k) \otimes T_{j_2}(k-1), & \text{if } \alpha \text{ initial, } \alpha \prec j_1, j_2 \prec \alpha', l_C(\alpha') = l_C(\alpha), \end{cases}$$

denotes the **holding time** of condition α in the k -th instantiation of the repeating part. ◇

In a decision-free unfolding, each condition has exactly one predecessor and one successor event. The holding time in the k -th instantiation of a condition i is then easily determined: it is the difference of the k -th occurrence time of its

successor and the k -th occurrence of its predecessor. If the condition is initial, the $(k - 1)$ -th occurrence of the predecessor event has to be considered.

The long-run holding time of a condition α is determined by the expectation of the limit of $H_\alpha(k)$, given existence:

$$H_\alpha := E \left[\lim_{k \rightarrow \infty} H_\alpha(k) \right].$$

Example 8. Since $H_A(k) = 3$ for all k , $H_A = 3$. The sum of the mean holding times for A and D has to be equal to the mean cycle time: $H_D = CT - H_A = 5.6857 - 3 = 2.6857$. Compare this to $E[X_D] = 0.2$. The mean holding time for C is $H_C = 2$. Consequently, $H_E = 3.6857$ (compared with $E[X_E] = 0.125$).

The writer is expected to think for three time units, then he is expected to wait for the buffer. This takes (on average) 2.4857 time units. Writing into the buffer takes 0.2 time units. For the reader process it is quite similar \square

4.7 Stationary Probabilities

Prefixes are a semantics for stochastic process algebra expressions. Those expressions are composed of different agents, which interact with each other. The agents are further subdivided into components, which are then used as condition labels in the prefix. We can rebuild the agents from these labels, even if we only have the prefix. An agent is then represented by a set of conditions forming a chain w.r.t. the flow relation \prec .

Example 9. In Fig. 4 the three agents of the repeating part of our example are depicted. \square

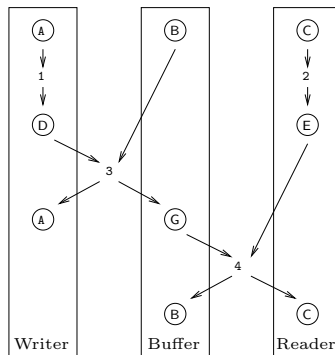


Fig. 4. The agents (Writer, Buffer and Reader) of the repeating part

For the repeating part of an agent we know the holding times of the corresponding conditions. Furthermore, we know the time the system needs for one

cycle. This is exactly the same time the agent needs for one cycle. Consequently, the sum of all holding times of one agent must be the cycle time. But this means that we are able to determine the part of time the agent spends in a particular condition. In other words, we can determine the stationary probability of a condition. In contrast to stationary probabilities of Markov chains, these probabilities are local to the agent and *do not* concern the global system state.

Definition 14. Let CT be the expected cycle time of a system and H_α the expected holding time of condition α . Then the **local stationary probability** of condition α is given by the fraction

$$\pi_\alpha := \frac{H_\alpha}{CT}. \quad \diamond$$

Example 10. For the writer and the reader of the example, we obtain the following local stationary probabilities:

- The writer is thinking with probability $\pi_A = \frac{H_A}{CT} = \frac{3}{5.6857} = 0.5276$. He is waiting for or writing in the buffer with probability $\pi_D = 0.4724$.
- The reader is thinking with probability $\pi_C = 0.3518$. He is waiting for or reading from the buffer with probability $\pi_E = 0.6482$.

5 Conclusion

In this paper, we considered complete finite stochastic event structure prefixes as true-concurrency semantics for simple SPA expressions. After a short introduction in max-plus algebra, a method for the computation of occurrence times has been presented. Using the idea of cutoff events, we found the repeating event classes of a prefix. Similar to the calculation of occurrence times, k -th occurrence times can be derived that are the basis for other performance measures.

Max-plus algebra methods provide an elegant way to describe the timing behaviour of decision-free stochastic prefixes. The timing of repeating behaviour can be determined via the solution of a system of linear equations. Unfortunately, this solution is generally a complex task, as it involves sums and maxima of random variables, that even might be dependent. So, complexity may make max-plus matrices useful only for notational purposes.

Finally, we have shown how some performance measures of SPA models with *general distributions* can be expressed in terms of the stochastic prefix and max-plus algebra.

References

1. F. Baccelli, A. Jean-Marie, and Zhen Liu. A survey on solution methods for task graph models. In N. Gotz, U. Herzog, and M. Rettetbach, editors, *Proc. of the QMIPS-Workshop on Formalism, Principles and State-of-the-art. Arbeitsberichte des IMMD 26(14)*. Universität Erlangen, 1993.

2. F. L. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity. An Algebra for Discrete Event Systems*. John Wiley & Sons, 1992.
3. E. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. A stochastic causality-based process algebra. *The Computer Journal*, 38(7):552–565, 1995.
4. L. Cloth. Complete finite prefix for stochastic process algebra. Master’s thesis, RWTH Aachen, Department of Computer Science, 2000. available at <ftp://ftp-lvs.informatik.rwth-aachen.de/pub/da/2000/cloth.ps.gz>.
5. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan’s unfolding algorithm. In T. Margaria and B. Steffen, editors, *Proc. of TACAS’96*, LNCS 1055, pages 87–106. Springer, 1996.
6. P. Glynn. A GSMP formalism for discrete event simulation. *Proc. of the IEEE*, 77(1):14–23, 1989.
7. J.-P. Katoen. *Quantitative and Qualitative Extensions of Event Structures*. PhD thesis, Universiteit Twente, 1996.
8. J.-P. Katoen, E. Brinksma, D. Latella, and R. Langerak. Stochastic simulation of event structures. In M. Ribaud, editor, *Proc. of PAPM 1996*, pages 21–40. C.L.U.T. Press, 1996.
9. R. Langerak and H. Brinksma. A complete finite prefix for process algebra. In *CAV’99*, LNCS 1663, pages 184–195. Springer, 1999.
10. K. L. McMillan. *Symbolic Model Checking*, chapter 9: A Partial Order Approach, pages 153–167. Kluwer, 1993.
11. V. Mertsotakis and M. Silva. A throughput approximation algorithm for decision free processes. In M. Ribaud, editor, *Proc. of PAPM 1996*, pages 161–178. C.L.U.T. Press, 1996.
12. G. Olsder, J. Resing, R. de Vries, M. Keane, and G. Hooghiemstra. Discrete event systems with stochastic processing times. *IEEE Transactions on Automatic Control*, 35(3):299–302, 1990.
13. T. C. Ruys, R. Langerak, J.-P. Katoen, D. Latella, and M. Massink. First passage time analysis of stochastic process algebra using partial orders. In T. Margaria and W. Yi, editors, *Proc. of TACAS 2001*, LNCS 2031, pages 220–235. Springer, 2001.