Technische Universiteit
Eindhoven
University of Technology

Partial-Order Reduction for Performance Analysis of Max-Plus Timed Systems

Bram van der Sanden, Marc Geilen, Michel Reniers, and Twan Basten

ES Reports
ISSN 1574-9517

ESR-2018-01
2 February 2018

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

© 2018 Technische Universiteit Eindhoven, Electronic Systems.
All rights reserved.

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Partial-Order Reduction for Performance Analysis of Max-Plus Timed Systems

Bram van der Sanden*, Marc Geilen*, Michel Reniers*, Twan Basten*§
*Eindhoven University of Technology, Eindhoven, The Netherlands
§ESI, Eindhoven, The Netherlands
Email: b.v.d.sanden@tue.nl, m.c.w.geilen@tue.nl, m.a.reniers@tue.nl, a.a.basten@tue.nl

***Abstract*—This paper presents a partial-order reduction method for performance analysis of max-plus timed systems. A max-plus timed system is a network of automata, where the timing behavior of deterministic system tasks (events in an automaton) is captured in (max,+) matrices. These tasks can be characterized in various formalisms like synchronous data flow, Petri nets, or real-time calculus. The timing behavior of the system is captured in a (max,+) state space, calculated from the composition of the automata. This state space may exhibit redundant interleaving with respect to performance aspects like throughput or latency. The goal of this work is to obtain a smaller state space to speed up performance analysis. To achieve this, we first formalize state-space equivalence with respect to throughput and latency analysis. Then, we present a way to compute a reduced composition directly from the specification. This yields a smaller equivalent state space. We perform the reduction on-the-fly, without first computing the full composition. Experiments show the effectiveness of the method on a set of realistic manufacturing system models.**

I. Introduction and Related Work

Performance is one of the key aspects in the design of complex systems. Besides having to meet functional requirements, systems need to adhere to timing constraints, and optimize productivity, typically expressed with throughput or latency metrics. Throughput describes the system performance in the long run, for instance the number of products produced by the system per hour. Latency describes the temporal distance between certain events, for instance the time between the start and end of processing a product. Usually, system performance can only be measured at a later stage in the development process, once the system is assembled. A model-based design approach to performance engineering [1] can be used to address this issue. In such an approach, formal models capture the system behavior under the various scenarios of execution. Moreover, by adding timing information, performance analysis techniques can be used to predict the system performance at an early stage in the design process. However, in many industrial applications, the underlying timed state space of these models becomes large, and performance analysis quickly becomes a bottleneck.

In this work, we introduce a new partial-order reduction technique to speed up performance analysis of timed systems. The reduction explores only a restricted number of interleavings of concurrently enabled system operations that use different sets of system resources. The ample conditions on the reduction guarantee that the performances properties of the original model are preserved in the reduced model. Traditional partial-order techniques typically consider local properties such as deadlocks, and temporal properties formulated in logics like $\text{LTL}_{\setminus\bigcirc}$ (next-time-free Linear-time Temporal Logic) [2] and $\text{CTL}^*_{\setminus\bigcirc}$ (next-time-free Computation Tree Logic) [3].

There has been some initial work in applying partial-order reduction techniques to timed systems. Bengtsson et al. [4] apply standard partial-order reduction on timed automata for reachability analysis. These automata execute asynchronously, in their own local time scale, and synchronize their time scales on communication transitions. This work has been extended by Minea [5] to perform model checking for an extension of LTL, that can express timing relations between events. Yoneda et al. [6] investigated partial-order reduction for timed Petri nets, that allows the verification of similar timing relations. Theelen et al. [7] apply ideas from partial-order reduction on Scenario-Aware Data Flow models, where they use an independence relation among actions to resolve non-deterministic choices that have no impact on the performance metrics.

In this paper, we consider max-plus timed systems as a formal model. Such systems are described by a set of (max,+) automata [8] and a composition operator. A (max,+) automaton is a conventional automaton, where the timing semantics of each system task (event in an automaton) is described by a (max,+) matrix. Such a matrix captures the corresponding timing behavior, induced by the corresponding action execution times and action dependencies. Max-plus timed systems can express the timing semantics of a broad range of specification formalisms, such as Network Calculus [9], Real-Time Calculus [10], Synchronous Data Flow [11], Scenario-Aware Data Flow [12], and Timed-Event Graphs [13], [14], an important subclass of timed Petri nets. A broad range of industrial systems can be expressed using these formalisms.

To illustrate the concepts in this paper we use a specification framework [15] suitable for performance analysis of manufacturing systems. It allows analysis of both throughput and latency as performance metrics. System tasks are described by *activities*, which consist of a set of *actions* that execute on *resources* and dependencies among those actions. The timing behavior of each activity is captured by a (max,+) matrix. A parallel composition of (max,+) automata describes the order in which activities can be executed. As composition operator, multi-party synchronization [16] is used. From the system specification, a timed (max,+) state space is derived that captures the system behavior and the necessary timing information to evaluate system throughput and latency. We use partial-order reduction to compute a reduced composition of (max,+) automata directly from the specification. From the reduced composition a reduced state space is computed that preserves performance properties.

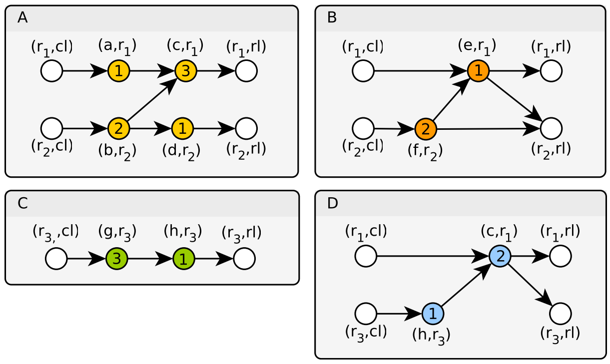
Well-known partial-order reduction techniques include the *stubborn sets* method of Valmari [17], the *persistent sets* method of Godefroid [18], and the *ample sets* method of Peled [19]. The idea in these methods is to exploit information about the independence of certain activities to reduce the size of the state space, while preserving the properties of interest. In each state, only a subset of all possible transitions is selected. In this work, we use ample sets and the extension of cluster-based ample sets [20].

In the remainder of this paper, we first formally introduce max-plus timed systems and define the properties of interest. Then, we introduce a reduction function that preserves the specified properties on the level of the state space. Next, we introduce local conditions to compute a reduced composition automaton directly from the network of (max,+) automata. The reduced state space can be computed from the reduced composition automaton. An experimental evaluation shows the effectiveness of the reduction technique.

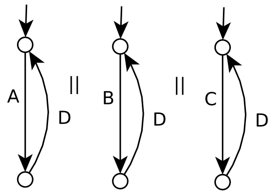
II. Max-Plus Timed Systems

In the specification framework that we use [15], a system is modeled in terms of *resources* that provide the *actions* that the system can execute. Deterministic system tasks are described by *activities*. An activity consists of a fixed set of action instances and dependencies among those action instances. A resource must be claimed before its actions can be used. After execution of the actions, the resource must be released. The timing information of each activity is captured in a (max,+) matrix. This matrix describes the release time of each system resource in terms of when the resources are available at the start of executing the activity. The availability times of resources are captured in a (max,+) vector. Given such a *resource availability vector*, we obtain the new resource availability vector after execution of some activity by multiplying with the corresponding (max,+) matrix.

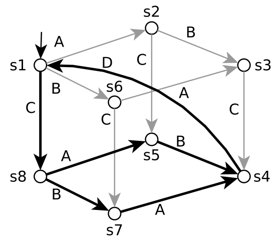
We use (max,+) algebra (see for instance [21]) to capture the timing semantics of activities in a concise way. Two essential characteristics of the execution of an activity are *synchronization*, when an action inside an activity waits until all preceding actions are finished, and *delay*, when an action execution takes an amount of time before it completes. These characteristics correspond well to the (max,+) operators *maximum* (max) and *addition* (+), defined over the set $\mathbb{R}^{-\infty} = \mathbb{R} \cup \{-\infty\}$. Operators



(a) Activities A, B, C and D. Actions are from a till h, and the action timings are given inside the nodes. Resources r_1, r_2 and r_3 are claimed (cl) and released (rl).



(b) Max-plus timed system. Activities A, B, D have reward 0; C has reward 1.



(c) Composition of the (max,+) automata. The reduced composition is shown with thick transitions.

Fig. 1. Running Example.

$$\boldsymbol{M}_A = \begin{bmatrix} 4 & 5 & -\infty \\ -\infty & 3 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix} \quad \boldsymbol{M}_B = \begin{bmatrix} 1 & 3 & -\infty \\ 1 & 3 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix}$$

$$\boldsymbol{M}_C = \begin{bmatrix} 0 & -\infty & -\infty \\ -\infty & 0 & -\infty \\ -\infty & -\infty & 4 \end{bmatrix} \quad \boldsymbol{M}_D = \begin{bmatrix} 2 & -\infty & 3 \\ -\infty & 0 & -\infty \\ 2 & -\infty & 3 \end{bmatrix}$$

Fig. 2. (max,+) matrices of activities A, B, C and D.

max and + are defined as usually in algebra, with the additional convention that $-\infty$ is the unit element of max: $\max(-\infty, x) = \max(x, -\infty) = x$, and the zero-element of +: $-\infty + x = x + -\infty = -\infty$.

Since (max,+) algebra is a linear algebra, it can be extended to matrices and vectors in the usual way. Given matrix $\boldsymbol{A}$ and matrix $\boldsymbol{B}$, we use $\boldsymbol{A} \otimes \boldsymbol{B}$ to denote the (max,+) matrix multiplication. Given $m \times p$ matrix $\boldsymbol{A}$ and $p \times n$ matrix $\boldsymbol{B}$, the elements of the resulting matrix $\boldsymbol{A} \otimes \boldsymbol{B}$ are determined by: $[\boldsymbol{A} \otimes \boldsymbol{B}]_{ij} = \max_{k=1}^{p}([\boldsymbol{A}]_{ik} + [\boldsymbol{B}]_{kj})$. For any vector $\boldsymbol{x}$, $\|\boldsymbol{x}\| = \max_i [\boldsymbol{x}]_i$ denotes the vector norm of $\boldsymbol{x}$. For vector $\boldsymbol{x}$, with $\|\boldsymbol{x}\| > -\infty$, we use $norm(\boldsymbol{x})$ to denote $\boldsymbol{x} - \|\boldsymbol{x}\|$, the normalized vector, such that $\|norm(\boldsymbol{x})\| = 0$. We use $\mathbf{0}$ to denote a vector with only zero-valued entries.

In this paper, we use the running example shown in Fig. 1. This max-plus timed system consists of activities A, B, C and D (see Fig. 1a), and three (max,+) automata (see Fig. 1b). Fig. 1c shows the composition of the (max,+) automata. Each activity has a corresponding (max,+) matrix that captures the timing behavior, shown in Fig. 2. Each matrix row represents the symbolic release time of a

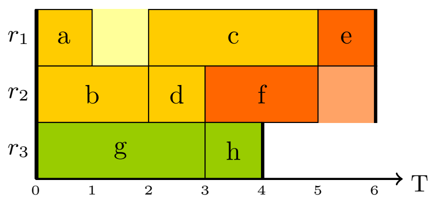


Fig. 3. Gantt chart of activity sequence ABC when all resources are initially available.

resource in terms of all system resources, which is denoted by set $\mathcal{R}$. Let R be a function that maps each activity to the set of resources the activity uses. As an example, consider the first row of matrix $\boldsymbol{M}_A$. This row describes the release time of resource $r_1 \in \mathcal{R}$, expressed in terms of when resources r_1, r_2 and r_3 are available at the start of executing A. In the execution of activity A, there is a timing delay of 4 time units between the claiming of resource r_1 and the subsequent release of resource r_1. Similarly, a delay of 5 is present between the claiming of resource r_2 and the release of r_1. There is no dependency between the availability times of resource r_3 and the release of resource r_1, indicated by $-\infty$. This can also be seen in the structure of activity A (Fig. 1a), since resource r_3 is not involved in the execution. For details on how to compute the (max,+) matrices of activities, see [15].

The timing evolution of the system is expressed using (max,+) matrix multiplication. Assume that all resources are initially available, captured in vector $\mathbf{0}$. The new availability times of the resources after executing activity A with corresponding matrix $\boldsymbol{M}_A$ are computed as follows:

$$\boldsymbol{M}_A \otimes \mathbf{0} = \begin{bmatrix} \max(4+0, 5+0, -\infty+0) \\ \max(-\infty+0, 3+0, -\infty+0) \\ \max(-\infty+0, -\infty+0, 0+0) \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 0 \end{bmatrix}.$$

Resources r_1 and r_2 are available again after 5 and 3 time units respectively. The availability time of resource r_3 stays 0, since it is not used by activity A.

The timing semantics of an activity sequence is defined in terms of repeated matrix multiplication. As an example, consider the execution of activity sequence ABC, shown in Fig. 3. The new resource availability vector after execution of this sequence is computed as follows:

$$\boldsymbol{M}_C \otimes \boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \mathbf{0} = [6, 6, 4]^\intercal.$$

To capture all possible activity orderings in the system, we use (max,+) automata.

Definition 1 ((max,+) automaton (adapted from [8]))**.** *A* (max,+) automaton $\mathcal{A}$ *is a tuple* $\langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$ *where* S *is a finite set of states,* $\hat{s} \in S$ *is the initial state,* $\mathcal{A}ct$ *is a nonempty set of activities, function* $reward : \mathcal{A}ct \to \mathbb{R}^{\geq 0}$ *quantifies the amount of progress per activity, function* M *maps each activity to its associated (max,+) matrix of size* $|\mathcal{R}| \times |\mathcal{R}|$*, and* $T \subseteq S \times \mathcal{A}ct \times S$ *is the transition relation. Let* $s \xrightarrow{A} s'$ *be a shorthand for* $\langle s, A, s'\rangle \in T$*. It is assumed that* $\mathcal{A}$ *is deterministic, which means that for any* $s, s', s'' \in S$ *and* $A \in \mathcal{A}ct$*,* $s \xrightarrow{A} s'$ *and* $s \xrightarrow{A} s''$ *imply* $s' = s''$*.*

Let $\mathcal{A} = \langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$ be a (max,+) automaton. An activity $A \in \mathcal{A}ct$ is said to be *enabled* in a state $s \in S$ if $s \xrightarrow{A} s'$ for some $s' \in S$. Set $enabled(s) = \{A \in \mathcal{A}ct \mid \exists s' : s \xrightarrow{A} s'\}$ contains all activities enabled in s. State s is a *deadlock* state if $enabled(s) = \emptyset$. Since $\mathcal{A}$ is deterministic, for any activity $A \in enabled(s)$, there is a unique A-successor of s, denoted by $A(s)$. For an activity sequence $A_1 \ldots A_n$, the resulting state is defined inductively as $(A_1)(s) = A_1(s)$ if $A_1 \in enabled(s)$, and $(A_1 \ldots A_n A_{n+1})(s) = A_{n+1}((A_1 \ldots A_n)(s))$ if $A_{n+1} \in enabled((A_1 \ldots A_n)(s))$. Otherwise, $(A_1 \ldots A_{n+1})(s)$ is undefined.

A (max,+) automaton is an ω-automaton [22] that accepts infinite ω-words over $\mathcal{A}ct$. There are no specific acceptance conditions on these words, so any infinite word that conforms to a sequence of transitions starting in the initial state is accepted. A possible behavior of the (max,+) automaton is described in a run. An *infinite* run ρ of $\mathcal{A}$ is an infinite, alternating sequence of states and activities:

$$\rho = s_0 A_1 s_1 A_2 s_2 A_3 \ldots \text{ such that } s_0 = \hat{s} \\ \text{and } s_{i+1} = A_{i+1}(s_i) \text{ for all } i \geq 0.$$

Given run ρ, let $\rho[..i]$ denote the prefix $s_0 A_1 \ldots s_i$, and let $\rho[i, j] = s_i A_{i+1} \ldots s_j$ denote a run fragment from state s_i until s_j. Furthermore, let $\rho[i]$ denote state s_i in ρ.

We use the term (max,+) automaton to emphasize that the timing semantics of the automaton is expressed in (max,+) algebra. Note that the original definition in [8] does not consider rewards, but this extension is considered for instance in Weakly-Consistent Scenario-Aware Data Flow [23]. It allows for a refined, explicit specification of progress. A max-plus timed system is defined as a composition of (max,+) automata. In our framework we use multi-party synchronization as composition operator, which is defined in the following way.

Definition 2 (Multi-party synchronization)**.** *Given (max,+) automata* $\mathcal{A}_1 = \langle S_1, \hat{s}_1, \mathcal{A}ct_1, reward_1, M_1, T_1\rangle$ *and* $\mathcal{A}_2 = \langle S_2, \hat{s}_2, \mathcal{A}ct_2, reward_2, M_2, T_2\rangle$*, we define the multi-party synchronization* $\mathcal{A}_1 \parallel \mathcal{A}_2 = \langle S_1 \times S_2, \langle \hat{s_1}, \hat{s_2}\rangle, \mathcal{A}ct_1 \cup \mathcal{A}ct_2, reward_1 \cup reward_2, M_1 \cup M_2, T_{12}\rangle$*, where*

$$T_{12} = \begin{cases} \langle s_1, s_2\rangle \xrightarrow{A}_{12} \langle s_1', s_2'\rangle & \text{if } A \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2, \\ & \quad s_1 \xrightarrow{A}_1 s_1', s_2 \xrightarrow{A}_2 s_2' \\ \langle s_1, s_2\rangle \xrightarrow{A}_{12} \langle s_1', s_2\rangle & \text{if } A \in \mathcal{A}ct_1 \setminus \mathcal{A}ct_2, s_1 \xrightarrow{A}_1 s_1' \\ \langle s_1, s_2\rangle \xrightarrow{A}_{12} \langle s_1, s_2'\rangle & \text{if } A \in \mathcal{A}ct_2 \setminus \mathcal{A}ct_1, s_2 \xrightarrow{A}_2 s_2'. \end{cases}$$

Definition 3 (Max-plus timed system)**.** *A* max-plus timed system $\mathcal{M}$ *is described by* $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ *with (max,+) automata* $\mathcal{A}_i$ *and multi-party synchronization operator* $\parallel$*. It is assumed that all matrices have the same dimensions of* $|\mathcal{R}| \times |\mathcal{R}|$*, and that the reward functions agree for the same activities.*

The composition of all the individual automata is again an automaton. Fig. 1c shows the composition of the automata shown in Fig. 1b. Each (max,+) automaton can be interpreted as a normalized (max, +) state space that captures all the accepted runs, and contains all the necessary information to evaluate performance properties.

Definition 4 (Normalized (max,+) state space (adapted from [12])). *Given (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$ with matrices of size $|\mathcal{R}| \times |\mathcal{R}|$, we define the normalized* (max, +) *state space $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$ as follows:*

- *set $C = S \times \mathbb{R}^{-\infty^{|\mathcal{R}|}}$ of configurations that consists of a state and a normalized (resource availability) vector;*
- *initial configuration $\hat{c} = \langle \hat{s}, \mathbf{0}\rangle$;*
- *a labeled transition relation $\Delta \subseteq C \times \mathcal{A}ct \times C$ that consists of the transitions in the set $\{\langle\langle s, \boldsymbol{\gamma}\rangle, A, \langle s', norm(\boldsymbol{\gamma}')\rangle\rangle \mid s \xrightarrow{A} s' \wedge \boldsymbol{\gamma}' = M(A) \otimes \boldsymbol{\gamma}\}$;*
- *function w_1 that assigns a weight $w_1(c, A, c') = reward(A)$ to each transition $\langle c, A, c'\rangle \in \Delta$;*
- *function w_2 that assigns a weight $w_2(c, A, c') = \|M(A) \otimes \boldsymbol{\gamma}\|$ to each transition $(c, A, c') \in \Delta$. This weight indicates the total added execution time to the complete schedule.*

We define the set of enabled activities and runs in a (max,+) state space in a similar way as in a (max,+) automaton. The state space of $\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ is computed in two steps. First, we compute the composition, and subsequently we compute the corresponding state space. Fig. 4 shows the state space of the max-plus timed system of Fig. 1. In some cases, this state space might be infinite [12]. The state space is guaranteed to be finite, if for every activity sequence u allowed by the (max,+) automaton and any $k \geq 0$, there is some $m > k$ such that the matrix $\boldsymbol{M}_{u(k)} \otimes \cdots \otimes \boldsymbol{M}_{u(m-1)}$ contains no entries $-\infty$ [12]. We can make the following observation.

Proposition 5. *Given (max,+) automaton $\mathcal{A}$ and corresponding (max,+) state space $\mathcal{S}$, $\mathcal{S}$ is finite iff each resource is used by at least one activity in any cycle in $\mathcal{A}$.*

In the (max,+) automaton shown in Fig. 1c, each cycle involves activities A, B, C, and D. Together, these activities use all three resources r_1, r_2, and r_3. Therefore, the corresponding state space shown in Fig. 4 is finite.

The behavior of a (max,+) automaton $\mathcal{S}$ is captured by set $\mathcal{R}(\mathcal{S})$ of all allowed runs. A run $\rho \in \mathcal{R}(\mathcal{S})$ is an infinite, alternating sequence of configurations and activities:

$$\rho = c_0 A_1 c_1 A_2 c_2 A_3 \ldots \text{ such that } c_0 = \hat{c} \\ \text{and } c_{i+1} = A_{i+1}(c_i) \text{ for all } i \geq 0.$$

Given run ρ, we define run prefix $\rho[..i] = c_0 A_1 \ldots c_i$, run fragment $\rho[i, j] = c_i A_{i+1} \ldots c_j$ from configuration c_i until c_j, and $\rho[i] = c_i$. We also define vector $\bar{\boldsymbol{\gamma}}_n = (\bigotimes_{k=1}^{n} M(A_k)) \otimes \mathbf{0}$, which is the resulting resource availability vector after executing activities $A_1 \ldots A_n$ without normalization. These vectors can be derived from the normalized (max,+) state space.

Theorem 6. *Let $\mathcal{S}$ be a (max,+) state space, and $\rho = c_0 A_1 c_1 A_2 c_2 A_3 \ldots$ be a run in $\mathcal{S}$. Then, for each $n \geq 0$ it holds that $\bar{\boldsymbol{\gamma}}_n = \sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + \boldsymbol{\gamma}_n$.*

Proof. Proof by induction over n. First consider the base case $n = 0$. Then, $\bar{\boldsymbol{\gamma}}_0 = \boldsymbol{\gamma}_0 = \mathbf{0}$. Now, consider the induction step. As induction hypothesis assume $\bar{\boldsymbol{\gamma}}_n = \sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + \boldsymbol{\gamma}_n$. Then:

$$\begin{aligned}
&\sum_{k=0}^{n} w_2(c_k, A_{k+1}, c_{k+1}) + \boldsymbol{\gamma}_{n+1} \\
&= \sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + w_2(c_n, A_{n+1}, c_{n+1}) + \boldsymbol{\gamma}_{n+1} \\
&= \{\text{Def. 4}\} \\
&\quad \sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + \|M(A_{n+1}) \otimes \boldsymbol{\gamma}_n\| + \\
&\quad M(A_{n+1}) \otimes \boldsymbol{\gamma}_n - \|M(A_{n+1}) \otimes \boldsymbol{\gamma}_n\| \\
&= \sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + M(A_{n+1}) \otimes \boldsymbol{\gamma}_n \\
&= \{c + \boldsymbol{M} \otimes \boldsymbol{\gamma} = \boldsymbol{M} \otimes (\boldsymbol{\gamma} + c)\} \\
&\quad M(A_{n+1}) \otimes \left(\sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + \boldsymbol{\gamma}_n \right) \\
&= \{\text{induction hypothesis}\} \\
&\quad M(A_{n+1}) \otimes \left((\bigotimes_{k=1}^{n} M(A_k)) \otimes \mathbf{0} \right) \\
&= (\bigotimes_{k=1}^{n+1} M(A_k)) \otimes \mathbf{0} \\
&= \bar{\boldsymbol{\gamma}}_{n+1}
\end{aligned}$$

□

Example 7. *Consider the normalized (max,+) state space shown in Fig. 4, and the execution of activity sequence ABC starting from the initial state. This corresponds to some run ρ that starts with run fragment $\rho[0, 3] =$*

$$\langle s_1, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \rangle \xrightarrow{A,5} \langle s_2, \begin{bmatrix} 0 \\ -2 \\ -5 \end{bmatrix} \rangle \xrightarrow{B,1} \langle s_3, \begin{bmatrix} 0 \\ 0 \\ -6 \end{bmatrix} \rangle \xrightarrow{C,0} \langle s_4, \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} \rangle.$$

The vector in each configuration without normalization can now be computed using Theorem 6; $\bar{\boldsymbol{\gamma}}_0 = \boldsymbol{\gamma}_0 = \mathbf{0}$ and

$$\begin{aligned}
\bar{\boldsymbol{\gamma}}_1 &= \mathbf{0} + 5 + [0, -2, 5]^\intercal = [5, 3, 0]^\intercal \\
\bar{\boldsymbol{\gamma}}_2 &= \mathbf{0} + 5 + 1 + [0, 0, -6]^\intercal = [6, 6, 0]^\intercal \\
\bar{\boldsymbol{\gamma}}_3 &= \mathbf{0} + 5 + 1 + 0 + [0, 0, -2]^\intercal = [6, 6, 4]^\intercal.
\end{aligned}$$

Fig. 3 shows the same availability times of 6,6, and 4 for resources r_1, r_2, and r_3 after executing ABC.

III. Partial-order reduction for performance analysis

In this section we present a partial-order reduction technique that preserves throughput and latency properties. We first define throughput and latency.

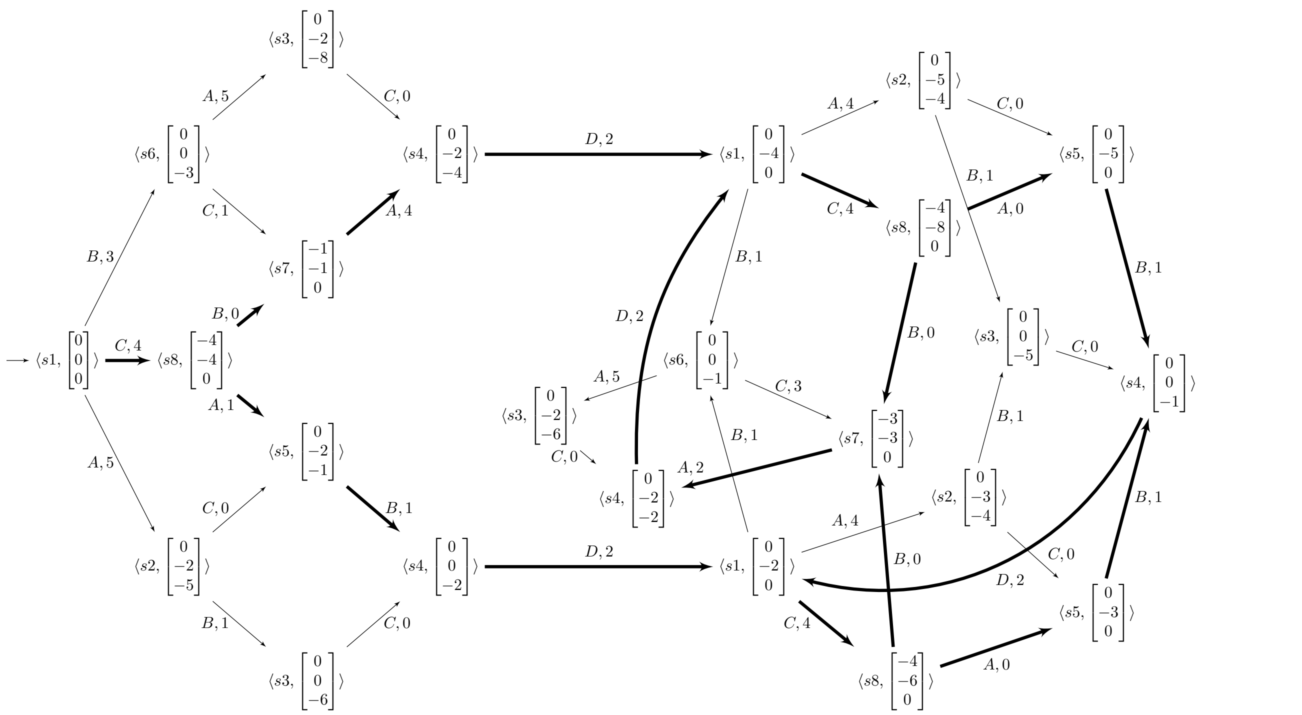


Fig. 4. Normalized (max,+) state space of the (max,+) automaton shown in Fig. 1c. The reduced state space is shown with thick transitions. Transitions are annotated with the corresponding activity and w_2 value. Activity C has reward 1, and activities A, B, and D have reward 0.

Throughput: We quantify the throughput of a run as the ratio between the total reward (sum of w_1 weights) and the total execution time (sum of w_2 weights).

Definition 8 (Ratio value of a run)**.** *The ratio of a run $\rho = c_0 A_1 c_1 A_2 c_2 A_3 \ldots$ is the ratio of the sums of weights w_1 and w_2, defined as follows*

$$\mathsf{Ratio}(\rho) = \limsup_{l \to \infty} \frac{\sum_{i=0}^{l} w_1(c_i, A_{i+1}, c_{i+1})}{\sum_{i=0}^{l} w_2(c_i, A_{i+1}, c_{i+1})}.$$

We define the ratio value of a run fragment $\rho[i, j]$ as

$$\mathsf{Ratio}(\rho[i, j]) = \frac{\sum_{k=i}^{j} w_1(c_k, A_{k+1}, c_{k+1})}{\sum_{k=i}^{j} w_2(c_k, A_{k+1}, c_{k+1})}.$$

The system throughput is determined by the possible ratio values over all infinite runs on some state space $\mathcal{S}$. We can quantify a guarantee on the throughput of the system by the *minimum ratio value* achieved by any of those runs:

$$\tau_{min}(\mathcal{S}) = \min_{\rho \in \mathcal{R}(\mathcal{S})} \mathsf{Ratio}(\rho).$$

If $\mathcal{S}$ is finite, each infinite run eventually reaches a recurrent configuration. Each reachable simple cycle in this state space allows for a periodic execution of the system. Since $\mathcal{S}$ has a finite number of simple cycles (no repetition of transitions is allowed), we can determine the minimum ratio value of the graph from a *minimum cycle ratio* (MCR) analysis [24]. The minimum cycle ratio (MCR) over all the cycles in $\mathcal{S}$, say $cycles(\mathcal{S})$, is defined in the following way:

$$\mathsf{MCR}(\mathcal{S}) = \min_{c \in cycles(\mathcal{S})} \mathsf{Ratio}(c) = \tau_{min}(\mathcal{S}).$$

Example 9 (Cycle ratio)**.** *Consider the normalized (max,+) state space $\mathcal{S}$ shown in Fig. 4. Recall that activity C has a reward of 1, and activities A, B, and D have a reward of 0. In this way, the ratio relates to the number of C occurrences per time unit. The minimum cycle ratio $\mathsf{MCR}(\mathcal{S}) = 3/8$, which can for instance be found in the following cycle corresponding to the execution of $(CBAD)^\omega$:*

$$\langle s_1, \begin{bmatrix} 0 \\ -4 \\ 0 \end{bmatrix} \rangle \xrightarrow{C,4} \langle s_8, \begin{bmatrix} -4 \\ -8 \\ 0 \end{bmatrix} \rangle \xrightarrow{B,0} \langle s_7, \begin{bmatrix} -3 \\ -3 \\ 0 \end{bmatrix} \rangle \xrightarrow{A,2} \langle s_4, \begin{bmatrix} 0 \\ -2 \\ -2 \end{bmatrix} \rangle.$$

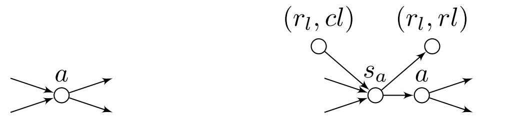
$D, 2$

The other periodic executions where B precedes A, i.e. $(BACD)^\omega$ and $(BCAD)^\omega$, have the same minimum cycle ratio value.

Latency: In general, latency is the time delay between a stimulus and its effect. In the context of max-plus timed systems, we define the *latency* in terms of the temporal distance that separates the resource availability times of a resource at the start of two activities A_{src} and A_{snk}. In the state space, consider some run $\rho = c_0 A_1 c_1 A_2 \ldots$ with $c_i = \langle s_i, \boldsymbol{\gamma}_i \rangle$ containing run fragment $\rho[i, j+1] = c_i A_{i+1} \ldots c_j A_{j+1} c_{j+1}$, with $A_{i+1} = A_{src}$ and $A_{j+1} = A_{snk}$. Then we define the start-to-start latency λ between the resource availability times of resource r in $\boldsymbol{\gamma}_i$ and $\boldsymbol{\gamma}_j$ as

$$\lambda(\rho, i, j, r) = [\bar{\boldsymbol{\gamma}}_j]_r - [\bar{\boldsymbol{\gamma}}_i]_r.$$

Example 10 (Latency)**.** *Consider again the execution of activity sequence $A \cdot B \cdot C$ starting from configuration c_0 in the (max,+) state space shown in Fig. 4. Suppose we want to compute the start-to-start latency between the resource*



(a) Before transformation (b) After transformation

Fig. 5. Adaptation of an activity to measure the start time of action a in terms of the availability time of resource r_l.

availability times of r_1 *in* $\bar{\gamma}_0$ *(start of activity A) and* $\bar{\gamma}_2$ *(start of activity C). Recall from Example 7 that* $\bar{\gamma}_0 = \mathbf{0}$ *and* $\bar{\gamma}_2 = [6, 6, 0]^\intercal$*. The latency is now computed as*

$$\lambda(\rho, 0, 2, r_1) = [\bar{\gamma}_2]_{r_1} - [\bar{\gamma}_0]_{r_1} = [\begin{bmatrix}\mathbf{6}\\6\\0\end{bmatrix}]_{r_1} - [\begin{bmatrix}\mathbf{0}\\0\\0\end{bmatrix}]_{r_1} = 6.$$

The above focuses on resources. The temporal distance between the start of two *actions* can also be determined from the resource availability times in the state space, by slightly adapting the corresponding activities that contain the actions. Assume action instances a and f in activities A_{src} and A_{snk}. To determine the start-to-start latency between a and f, we slightly transform activities A_{src} and A_{snk}. We illustrate the approach for A_{src}, the transformation for A_{snk} is analogous. First, we add a new action s_a to A_{src}, using the same resource as a, that does not take time. We remove the incoming dependencies from a and add them to s_a. Then we add a dependency from s_a to a. Next, we add a new resource r_l and a dependency from the claim node of resource r_l to s_a, and from s_a to the release node of r_l. In this way, we encode the start time of action a in terms of the resource availability of resource r_l. The transformation is illustrated graphically in Fig. 5.

We assume that the occurrences of A_{src} and A_{snk} activities are related. In any run, for any $k > 0$, the k-th occurrence of A_{src} is paired with the k-th occurrence of A_{snk}. We refer to such a pair of related activities as a source-sink pair. Let $\mathsf{getOccurrence}(\rho, A, k)$ be a function that returns the index of the k-th occurrence of activity A in run ρ. The *start-to-start latency* for resource r in ρ with source-sink pair $A_{i+1} = A_{src}$ and $A_j = A_{snk}$ in run fragment $\rho[i, j+1]$, is now equal to $\lambda(\rho, i, j, r)$. The maximum start-to-start latency in a run is now obtained by looking at all source-sink pairs:

$$\begin{aligned}\lambda_{max}(\rho, &A_{src}, A_{snk}, r) = \sup_{k>0} \lambda_k(\rho) \textbf{ where}\\ \lambda_k(\rho) &= \lambda(\rho, i, j, r),\\ &i = \mathsf{getOccurence}(\rho, A_{src}, k), \text{ and}\\ &j = \mathsf{getOccurence}(\rho, A_{snk}, k).\end{aligned}$$

Definition 11 (Latency)**.** *Given normalized (max,+) state space* $\mathcal{S}$*, the maximum start-to-start latency of resource* r *with source-sink pair* A_{src}*,* A_{snk} *in* $\mathcal{S}$ *is found by taking the maximum latency over all runs in the state space:*

$$\lambda_{max}(\mathcal{S}) = \sup_{\rho \in \mathcal{R}(\mathcal{S})} \lambda_{max}(\rho, A_{src}, A_{snk}, r).$$

Ratio Independence: In the state space, there can be redundancy with respect to multiple runs that have the same ratio value. Part of this redundancy is caused by the interleaving of activities that have no mutual influence. We reduce the size of the state space by removing redundant interleaving of ratio-independent activities.

Definition 12 (Ratio independent)**.** *Let* $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$ *be a (max,+) state space,* $c \in C$ *be a configuration, and* $A, B \in enabled(c)$ *be activities enabled in* c*. Activities* A *and* B *are* ratio independent *in* c *iff if they satisfy the following conditions:*

1) *if* $A, B \in enabled(c)$*, then* $B \in enabled(A(c))$*,* $A \in enabled(B(c))$*, and* $AB(c) = BA(c)$*;*
2) $w_i(c, A, A(c)) + w_i(A(c), B, AB(c)) = w_i(c, B, B(c)) + w_i(B(c), A, BA(c))$ *for* $i \in \{1, 2\}$*;*
3) $R(A) \cap R(B) = \emptyset$*.*

Two activities are ratio dependent *if they are not ratio independent.*

The first property is the classical notion of *independence*: in every configuration where A and B are both enabled, the execution of one activity cannot disable the other activity, and the resulting configuration after executing both activities in any order is the same. The second property requires that the sum of the weights w_1 and w_2 of the corresponding transitions of A and B is the same. The third property requires that activities A and B do not share resources.

Reduced state space: In the remainder of this section, we formalize an ample reduction on a (max,+) state space that preserves throughput and latency.

Definition 13 (State space reduction function)**.** *A reduction function reduce for a (max,+) state space* $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$ *is a mapping from* C *to* $2^{\mathcal{A}ct}$ *such that* $reduce(c) \subseteq enabled(c)$ *for each configuration* $c \in C$*. We define the reduction of* $\mathcal{S}$ *induced by reduce as the smallest (max,+) state space* $\mathcal{S}' = \langle C', \hat{c}', \mathcal{A}ct', \Delta', M', w_1', w_2'\rangle$ *that satisfies the following conditions:*

- $C' \subseteq C,\ \hat{c}' = \hat{c},\ \mathcal{A}ct' = \mathcal{A}ct,\ \Delta' \subseteq \Delta,\ M' = M$*;*
- *for every* $c \in C'$ *and* $A \in reduce(c)$*,* $(c, A, A(c)) \in \Delta'$*,* $w_1'(c, A, A(c)) = w_1(c, A, A(c))$*, and* $w_2'(c, A, A(c)) = w_2(c, A, A(c))$*.*

Definition 14 (Ample conditions state space)**.** *Let ample be a reduction function on a (max,+) state space that satisfies the following conditions:*

(R1) Non-emptyness condition: *if* $enabled(c) \neq \emptyset$*, then* $ample(c) \neq \emptyset$*.*

(R2) Ratio-dependency condition: *For any configuration* $c_0 \in C'$ *and run* $c_0A_1c_1A_2 \ldots A_mc_m$ *with* $m \geq 1$ *in* $\mathcal{S}$*, if activity* A_m *and some activity in* $ample(c_0)$ *are ratio dependent in* c_0*, then there is an index* i *with* $1 \leq i \leq m$ *with* $A_i \in ample(c_0)$*.*

Example 15. *Consider initial configuration* $c_0 = \langle s1, \mathbf{0}\rangle$ *in the state space shown in Fig. 4. Activities A and B are ratio dependent in* c_0 *(they do not satisfy condition 1 in*

Def. 12) and ratio independent with activity C. Ample sets $\{A, B\}$ and $\{C\}$ both satisfy conditions (R1) and (R2).

Condition (R2) implies that starting from some configuration c_i, any activity in $ample(c_i)$ remains enabled as long as no activity in $ample(c_i)$ has been executed.

Lemma 16 (adapted from [25, Lemma 8.15])**.** *Let $\rho[i, j] = c_i B_i c_{i+1} \ldots B_j c_j$ be a run fragment in $\mathcal{S}$. If $ample(c_i)$ satisfies condition (R2) and $\{B_i, \ldots, B_j\} \cap ample(c_i) = \emptyset$, then all activities $A \in ample(c_i)$ are ratio-independent of $\{B_i, \ldots, B_j, \}$. In addition, we have $A \in enabled(c_k)$ for $i < k \leq j$.*

Conditions (R1) and (R2) ensure that for each run in the (max,+) state space we can find an equivalent run in the reduced (max,+) state space. First we define equivalence of run prefixes. Two run prefixes are equivalent iff their corresponding activity sequences can be obtained from each other by repeatedly commuting adjacent ratio-independent activities.

Definition 17. *Activity sequences $v, w \in \mathcal{A}ct^*$ are considered equivalent [26], denoted $v \equiv w$, iff there exists a list of activity sequences $u_0, u_1, \ldots u_n$, where $u_0 = v$, $u_n = w$, and for each $0 \leq i < n$, $u_i = \bar{u}AB\hat{u}$ and $u_{i+1} = \bar{u}BA\hat{u}$ for some $\bar{u}, \hat{u} \in \mathcal{A}ct^*$ and ratio-independent activities $A, B \in \mathcal{A}ct$.*

Given prefix $\rho[..m] = c_0 A_1 \ldots A_m c_m$ of some run ρ, let ρ^m denote the activity sequence $A_1 \ldots A_m$.

Definition 18. *Prefixes $\rho[..m]$ and $\sigma[..m]$ of runs ρ and σ are equivalent, denoted $\rho[..m] \equiv \sigma[..m]$, iff $\rho^m \equiv \sigma^m$.*

Throughput is defined as a limit on prefix ratios of infinite runs. To define equivalence of runs in terms of throughput, we need to consider run prefixes with a bounded difference in activities following those prefixes.

Definition 19 (Equivalence of runs)**.** *Let ρ and σ be two runs. We define $\rho \succeq \sigma$ iff there exists a $c \in \mathbb{N}$ such that for all $n \geq 0$ it holds that $\rho \succeq_n \sigma$, where $\rho \succeq_n \sigma$ is defined if there exists some $k \geq n$, run prefixes $\rho[..k]$ and $\hat{\rho}[..k]$ with $\hat{\rho}^k \equiv \rho^k$ such that $\hat{\rho}^k = \sigma^n \cdot \tau$ for some τ, and $k - n \leq c$. Runs ρ and σ are equivalent, denoted $\rho \equiv \sigma$, iff $\rho \succeq \sigma$ and $\sigma \succeq \rho$.*

If a reduction satisfies conditions (R1) and (R2), then for each run in the full (max,+) state space we can find an equivalent run in the reduced (max,+) state space.

Theorem 20 (Equivalent runs)**.** *Let $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2 \rangle$ be a finite normalized (max,+) state space, and $\mathcal{S}'$ be the reduced (max,+) state space induced by reduction function ample. If ample satisfies conditions (R1) and (R2), then for each run $\rho \in \mathcal{R}(\mathcal{S})$, there exists a run $\sigma \in \mathcal{R}(\mathcal{S}')$ with $\rho \equiv \sigma$.*

Proof. Let $\rho \in \mathcal{R}(\mathcal{S})$ be some run in $\mathcal{S}$. Now, we need to show that there exists an equivalent run $\sigma \in \mathcal{R}(\mathcal{S}')$ such that $\rho \succeq \sigma$ and $\sigma \succeq \rho$.

First, we show that there exists a run $\sigma \in \mathcal{R}(\mathcal{S}')$ such that $\rho \succeq \sigma$ with $c = |C|$. To this end, we first define σ^n recursively as follows:

$$\sigma^0 = \varepsilon$$
$$\sigma^{n+1} = \sigma^n \cdot A, \text{where } A \in ample(\rho[n]) \text{ is the first such activity in } \rho[n..].$$

We now prove that such an activity A can be found in $\rho[n..] = c_n B_1 c_{n+1} B_2 \ldots$, with $c_n = \rho[n]$. Let $\mathcal{R}' \subseteq \mathcal{R}$ denote the set of resources used by activities in $ample(c_n)$, i.e. $\mathcal{R}' = \bigcup_{A \in ample(c_n)} R(A)$. Since the normalized (max,+) state space is finite, eventually we always reach a cycle. By Corollary 5, there is at least one activity on this cycle that uses one of the resources in $\mathcal{R}'$ that is also used by some activity $A \in ample(c_n)$. Let $A = B_m$ be the first such activity.

Let σ^n for each $n \geq 0$ be constructed using this procedure, and let $\sigma = \bigsqcup_{n \geq 0} \sigma^n$. We now prove that $\rho \succeq_n \sigma$ for all $n \geq 0$ by induction on n. As a base case, consider $n = 0$. Then, obviously $\sigma^0 = \rho^0 = \varepsilon$ satisfies $\rho \succeq_0 \sigma$. As induction hypothesis assume that $\rho \succeq_n \sigma$. This means that there exists a $k \geq n$, and $\hat{\rho}^k \equiv \rho^k$ such that $\hat{\rho}^k = \sigma^n \cdot \tau$ for some τ, and $k - n \leq |C|$. In the construction, we find some $l \geq n + 1$, such that $\sigma^{n+1} = \sigma^n \cdot A$, $\tau' = B_1 \ldots B_{m-1}$ and $\tilde{\rho}^l = \sigma^{n+1} \cdot \tau'$. By Lemma 16, $A = B_m \in ample(c_n)$, and B_m is ratio-independent with $B_1 \ldots B_{m-1}$, which means that $AB_1 \ldots B_{m-1} \equiv B_1 \ldots B_{m-1}A$. The value of m (and subsequently also the length of τ') is bounded by the maximum simple cycle length in the state space, i.e. $m \leq |C|$. Since $\sigma^n \equiv \hat{\rho}^n$ and $AB_1 \ldots B_{m-1} \equiv B_1 \ldots B_{m-1}A$, we have that $\tilde{\rho}^l \equiv \rho^l$. By the principle of induction, $\rho \succeq_n \sigma$ for all $n \geq 0$.

Since $\rho \succeq_n \sigma$ for all $n \geq 0$, run σ satisfies $\rho \succeq \sigma$. The fact that $\sigma \succeq \rho$, follows from the observation that each run $\sigma \in \mathcal{R}(\mathcal{S}')$ is also a run in $\mathcal{R}(\mathcal{S})$. Therefore, we have shown that for each run $\rho \in \mathcal{R}(S)$, there exists a run $\sigma \in \mathcal{R}(S')$ such that $\rho \equiv \sigma$. □

Theorem 21 (Equivalent runs have the same throughput)**.** *Let ρ and σ be runs. If $\rho \equiv \sigma$, then $\mathsf{Ratio}(\rho) = \mathsf{Ratio}(\sigma)$.*

Proof. Since $\rho \equiv \sigma$, by Def. 19, we have $\rho \succeq_n \sigma$ for $n \geq 0$. This means that there exists a $k \geq n$, and run $\hat{\rho}$ such that $\hat{\rho}^k = \sigma^n \cdot \tau$ for some τ, and $k - n \leq c$.

The maximum difference between $w_i(\rho^n)$ and $w_i(\sigma^n)$ for $i \in \{1, 2\}$ and any $n \geq 0$ is bounded by the maximum total w_i sum of suffix τ, whose length is bounded by c. Let k_i denote the maximum total w_i sum of τ, i.e. the maximum total w_i sum over all simple cycles in the graph:

$$w_i(\rho^n) \leq w_i(\sigma^n) \leq w_i(\rho^n) + k_i \text{ for some } k_i \geq 0.$$

Since $\limsup_{n \to \infty} w_i(\rho^n) = \infty$ for $i \in \{1, 2\}$, the constant k_i can be ignored, and we obtain the following result:

$$\mathsf{Ratio}(\rho) = \limsup_{n \to \infty} \frac{w_1(\rho^n)}{w_2(\rho^n)} = \limsup_{n \to \infty} \frac{w_1(\sigma^n)}{w_2(\sigma^n)} = \mathsf{Ratio}(\sigma).$$

□

Theorem 22 (Equivalent runs have the same latency)**.** *Let $\rho, \sigma \in \mathcal{R}(\mathcal{S})$. Let A_{src} and A_{snk} be any source-sink pair, and* *let r be the resource for which we want to calculate the start-to-start latency. If $\rho \equiv \sigma$, then $\lambda_{max}(\rho, A_{src}, A_{snk}, r) = \lambda_{max}(\sigma, A_{src}, A_{snk}, r)$.*

Proof. Consider any source-sink pair instance k in run ρ with latency $\lambda_k(\rho) = \lambda(\rho, i, j, r) = [\bar{\gamma}_j]_r - [\bar{\gamma}_i]_r$ for some $0 \leq i < j$. Furthermore, let $\lambda_k(\sigma) = \lambda(\sigma, m, n, r)$ for some $0 \leq m < n$.

The order of A_{src} and A_{snk} activities in σ is the same as in run ρ, since $R(A_{src}) \cap R(A_{snk}) \neq \emptyset$ and activities A_{src} and A_{snk} are ratio-dependent in any configuration. The corresponding run fragments $\rho[i, j]$ and $\sigma[m, n]$ start with the k-th occurrence of A_{src} and end with the k-th occurrence of A_{snk}.

Let $l = max(j, n)$. Since $\rho \equiv \sigma$, also $\rho \succeq_l \sigma$ and there exists some $\hat{\rho}$ and $k \geq l$ such that $\rho^k \equiv \hat{\rho}^k = \sigma^l \cdot \tau$ for some τ. Prefix $\hat{\rho}^k$ is obtained from ρ by repeatedly commuting ratio-independent activities. We need to show that each such swap has no influence on the latency of source-sink pair instance k. Let A, B be any pair of such activities, ratio-independent in some configuration c_s.

First, consider the case where both A and B are different from A_{src} and A_{snk}. Let $\bar{\gamma}_j$ be the resource vector of interest. Assume that $s \leq j$, since for $s > j$, obviously the swap has no impact on the value. Since A, B are ratio-independent, their matrices commute, and therefore the value of $\bar{\gamma}_j$ remains the same:

$$\begin{aligned}\bar{\gamma}_j &= \bigotimes_{k=1}^{j} M(A_k) \otimes \mathbf{0}\\ &= \bigotimes_{k=s+2}^{j} M(A_k) \otimes M(A_{s+1}) \otimes M(A_s) \otimes \bigotimes_{k=1}^{s-1} M(A_k) \otimes \mathbf{0}\\ &= \bigotimes_{k=s+2}^{j} M(A_k) \otimes M(A_s) \otimes M(A_{s+1}) \otimes \bigotimes_{k=1}^{s-1} M(A_k) \otimes \mathbf{0}.\end{aligned}$$

Now assume a swap of activities A_{src} and B, ratio-independent in configuration c_s. Assume that $A_{src} = A^{\rho}_{s+1}$ in run ρ, and $B = A^{\sigma}_{s+1}$ in run σ. We need to show that $[\bar{\gamma}_s]_r = [\boldsymbol{\gamma}^{\bar{\sigma}}_{s+1}]_r$, where $\boldsymbol{\gamma}^{\bar{\sigma}}_{s+1}$ is the resource vector after executing B from configuration c_s. Since A_{src} and B are ratio-independent, resource r is not used by B. This means that the resource availability time before and after executing B stays the same for resource r. This implies that $[\bar{\gamma}_s]_r = [\boldsymbol{\gamma}^{\bar{\sigma}}_{s+1}]_r$. A similar reasoning can be used for the case where A_{snk} and B are swapped, and when B is executed first in run ρ instead of in run σ. Since the resource availability vectors corresponding to the start of the k-th occurrences of A_{src} and A_{snk} are the same, we conclude that $\lambda_k(\rho) = \lambda_k(\sigma)$ for any $k \geq 0$. It follows that $\lambda_{max}(\rho, A_{src}, A_{snk}, r) = \lambda_{max}(\sigma, A_{src}, A_{snk}, r)$. ☐

From Theorems 20-22, it immediately follows that a safe reduction preserves throughput and latency aspects.

Corollary 23. *Let $\mathcal{S}$ be a finite normalized (max,+) state space, and $\mathcal{S}'$ the reduced (max,+) state space induced by reduction function ample. If ample satisfies conditions (R1) and (R2), then $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$ and $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$.*

Reduced (max,+) automata composition: We want to perform the partial-order reduction at the level of (max,+) automata, rather than at the level of the (max,+) state space. To achieve this goal, we introduce the notion of resource independence at the level of a (max,+) automaton. This notion is used in the ample conditions on reductions of a (max,+) automaton.

Definition 24 (Resource-independent activities)**.** *Given (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$ and state $s \in S$, activities $A, B \in enabled(s)$ are* resource independent *in s if they satisfy the following conditions:*

- *$B \in enabled(A(s))$ and $A \in enabled(B(s))$;*
- *$AB(s) = BA(s)$;*
- *$R(A) \cap R(B) = \emptyset$.*

Two activities are resource dependent*, if they are not resource independent.*

If two activities are resource independent in some state in the (max,+) automaton, then they are also *ratio* independent in the corresponding configurations in the underlying state space. When two activities are resource independent, it holds that $R(A) \cap R(B) = \emptyset$. In this case their corresponding (max,+) matrices commute, which means that $\boldsymbol{M}_A \otimes \boldsymbol{M}_B = \boldsymbol{M}_B \otimes \boldsymbol{M}_A$. As a result, the resulting normalized vector after multiplication is the same, and the sum of the weights w_1 and w_2 is the same, independent of the execution order.

If two activities are resource independent, then they are ratio independent in the underlying state space.

We show the relation between resource independence and ratio independence in a number of steps. First we show that if for activities A and B it holds that $R(A) \cap R(B) = \emptyset$, then their corresponding (max,+) matrices *commute*.

Definition 25 (Commuting matrices)**.** *Two (max,+) matrices $\boldsymbol{M}_A$ and $\boldsymbol{M}_B$ are said to* commute *if $\boldsymbol{M}_A \otimes \boldsymbol{M}_B = \boldsymbol{M}_B \otimes \boldsymbol{M}_A$.*

Lemma 26. *Let A and B be activities with corresponding matrices $\boldsymbol{M}_A$ and $\boldsymbol{M}_B$. If $R(A) \cap R(B) = \emptyset$, then $\boldsymbol{M}_A \otimes \boldsymbol{M}_B = \boldsymbol{M}_B \otimes \boldsymbol{M}_A$.*

If the (max,+) matrices of two activities commute, then the resulting normalized vector in the (max,+) state space is the same.

Lemma 27. *Consider (max,+) matrices $\boldsymbol{M}_A, \boldsymbol{M}_B$ and vector $\boldsymbol{\gamma}$. If $\boldsymbol{M}_A$ and $\boldsymbol{M}_B$ commute, then the resulting normalized vector after multiplication in the normalized (max,+) state space is the same.*

Proof.

$$\begin{aligned}
& norm(\boldsymbol{M}_B \otimes norm(\boldsymbol{M}_A \otimes \boldsymbol{\gamma})) \\
= \; & \boldsymbol{M}_B \otimes (\boldsymbol{M}_A \otimes \boldsymbol{\gamma} \; - \; \|\boldsymbol{M}_A \otimes \boldsymbol{\gamma}\|) \\
& - \big\|\boldsymbol{M}_B \otimes (\boldsymbol{M}_A \otimes \boldsymbol{\gamma} \; - \; \|\boldsymbol{M}_A \otimes \boldsymbol{\gamma}\|)\big\| \\
= \; & \boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \; - \; \|\boldsymbol{M}_A \otimes \boldsymbol{\gamma}\| \\
& - \|\boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \boldsymbol{\gamma}\| + \|\boldsymbol{M}_A \otimes \boldsymbol{\gamma}\| \\
= \; & \boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \; - \; \|\boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \boldsymbol{\gamma}\| \\
= \; & \{\text{using } \boldsymbol{M}_A \otimes \boldsymbol{M}_B = \boldsymbol{M}_B \otimes \boldsymbol{M}_A\} \\
& \boldsymbol{M}_A \otimes \boldsymbol{M}_B \otimes \boldsymbol{\gamma} \; - \; \|\boldsymbol{M}_A \otimes \boldsymbol{M}_B \otimes \boldsymbol{\gamma}\| \\
= \; & \{ \text{ same steps in reverse direction } \} \\
& norm(\boldsymbol{M}_A \otimes norm(\boldsymbol{M}_B \otimes \boldsymbol{\gamma})).
\end{aligned}$$

□

Given resource-independent activities, the sum of the weights for both w_1 and w_2 after execution are the same, independent of the execution order.

Lemma 28. *Let c be a configuration and A, B be resource-independent activities. Then, the sum of weights w_1 and w_2 of the paths after execution of both A and B starting from c is the same, independent of the relative order of A and B.*

Proof. For weight w_1: trivial. For weight w_2:

$$\begin{aligned}
& w_2(c, A, A(c)) + w_2(A(c), B, AB(c)) \\
= \; & \| \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \| + \| \boldsymbol{M}_B \otimes norm(\boldsymbol{M}_A \otimes \boldsymbol{\gamma}) \| \\
= \; & \| \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \| + \| \boldsymbol{M}_B \otimes (\boldsymbol{M}_A \otimes \boldsymbol{\gamma} - \| \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \|) \| \\
= \; & \{\text{using } \boldsymbol{M}_A \otimes (\boldsymbol{\gamma} - c) = \boldsymbol{M}_A \otimes \boldsymbol{\gamma} - c\} \\
& \| \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \| + \| \boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \boldsymbol{\gamma} - \| \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \| \| \\
= \; & \{\text{using } \| \boldsymbol{\gamma} - c \| = \| \boldsymbol{\gamma} \| - c\} \\
& \| \boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \boldsymbol{\gamma} \| \\
= \; & \| \boldsymbol{M}_A \otimes \boldsymbol{M}_B \otimes \boldsymbol{\gamma} \| \\
= \; & \{ \text{ same steps in reverse direction } \} \\
& w_2(c, B, B(c)) + w_2(B(c), A, BA(c)).
\end{aligned}$$

□

Theorem 29. *Given are a (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$ and state $s \in S$ with activities $A, B \in enabled(s)$. Consider any configuration $c = \langle s, \boldsymbol{\gamma}\rangle$ in the underlying normalized (max,+) state space. If A and B are resource-(in)dependent in s, then they are ratio-(in)dependent in c.*

Proof. Assume that A and B are resource independent in s. To prove that A and B are ratio independent in c, we show that the three conditions stated in Def. 12 hold. The first part of condition 1 follows directly by independence of A and B. The second part requires a unique configuration $c' = \langle s', \boldsymbol{\gamma}'\rangle$ after executing A and B in arbitrary order. The fact that the same state s' is reached follows from the independence of A and B, and that the same resource availability vector $\boldsymbol{\gamma}'$ is reached follows from Lemma 27. Condition 2 requires that the sum of the weights for both w_1 and w_2 is the same, independent of the execution order. This follows from Lemma 28. Condition 3 requires that A and B have no resources in common, which follows directly from the definition of resource independence.

The proof for the case that A and B are resource dependent is analogous. □

A reduction function on a (max,+) automaton is defined in the following way.

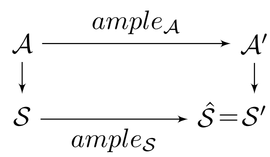
Definition 30 ((max,+) automaton reduction function)**.** *A reduction function reduce for a (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$ is a mapping from S to $2^{\mathcal{A}ct}$ such that $reduce(s) \subseteq enabled(s)$ for each state $s \in S$. We define the reduction of $\mathcal{A}$ induced by reduce as the smallest (max,+) automaton $\mathcal{A}' = \langle S', \hat{s}', \mathcal{A}ct', reward', M', T'\rangle$ that satisfies the following conditions:*

- *$S' \subseteq S$, $\hat{s}' = \hat{s}$, $\mathcal{A}ct' = \mathcal{A}ct$, $T' \subseteq T$;*
- *for every $s \in S'$ and $A \in reduce(s)$, $\langle s, A, A(s)\rangle \in T'$, $reward'(A) = reward(A)$, and $M'(A) = M(A)$.*

Given reduction function $ample_{\mathcal{A}}$ on a (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$, we define reduction function $ample_{\mathcal{S}}$ on the corresponding (max,+) state space $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$ in the following way: for any $c \in C$ with $c = \langle s, \gamma\rangle$ and $s \in S$, define $ample_{\mathcal{S}}(c) = ample_{\mathcal{A}}(s)$.

Lemma 31. *Let $\mathcal{A} = \langle S, \hat{s}, \mathcal{A}ct, reward, M, T\rangle$ be any (max,+) automaton with corresponding state space $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$. Let $ample_{\mathcal{A}}$ be an ample reduction on $\mathcal{A}$ that yields $\mathcal{A}'$ with corresponding state space $\mathcal{S}'$. Let $ample_{\mathcal{S}}$ be an corresponding ample reduction on $\mathcal{S}$ that yields state space $\hat{\mathcal{S}}$, then $\hat{\mathcal{S}} = \mathcal{S}'$.*

Proof. To illustrate the approach, consider the following figure.



For each $s \in S$, it holds that $enabled_{\mathcal{S}'}(s) = ample_{\mathcal{A}}(s)$. For each configuration $c \in C$ with $c = \langle s, \gamma\rangle$ we have $enabled_{\mathcal{S}'}(c) = enabled_{\mathcal{A}'}(s)$ by Def. 4. Also, $enabled_{\mathcal{S}}(c) = enabled_{\mathcal{A}}(s)$ by Def. 4, and $enabled_{\hat{\mathcal{S}}}(c) = ample_{\mathcal{S}}(c) = ample_{\mathcal{A}}(s)$ by definition of $ample_{\mathcal{S}}$. This implies that $enabled_{\hat{\mathcal{S}}}(c) = enabled_{\mathcal{S}'}(c)$ for each configuration $c \in C$. Since the set of enabled activities is the same, the set of configurations and transitions in both $\hat{\mathcal{S}}$ and $\mathcal{S}'$ is also the same. Also $\mathcal{A}ct' = \hat{\mathcal{A}ct} = \mathcal{A}ct$, $M' = \hat{M} = M$, and $w_i' = \hat{w}_i = w_i$ for $i \in \{1, 2\}$. This proves that $\hat{\mathcal{S}} = \mathcal{S}'$. □

To preserve latency and throughput properties, we impose the following ample conditions on a reduction function of a (max,+) automaton.

Definition 32 (Ample conditions (max,+) automaton)**.** *Let ample be a reduction function on a (max,+) automaton that satisfies the following conditions:*

(A1) Non-emptyness condition: *if* $enabled(s) \neq \emptyset$, *then* $ample(s) \neq \emptyset$.

(A2) Dependency condition: *For any state* $s_0 \in S'$ *and run* $s_0 A_1 s_1 A_2 \ldots A_m s_m$ *with* $m \geq 1$ *in* $\mathcal{A}$, *if activity* A_m *and some activity in* $ample(s_0)$ *are resource dependent in* s_0, *then there is an index* i *with* $1 \leq i \leq m$ *with* $A_i \in ample(s_0)$.

Example 33. *Consider state* s_1 *in the (max,+) automaton shown in Fig. 1c. Activities* A *and* B *are resource dependent in* s_1 *and resource independent with activity* C. *Ample sets* $\{A, B\}$ *and* $\{C\}$ *both satisfy conditions (A1) and (A2).*

Lemma 34. *If ample reduction* $ample_{\mathcal{A}}$ *on* $\mathcal{A}$ *satisfies conditions (A1) and (A2), then the corresponding ample reduction* $ample_{\mathcal{S}}$ *on the corresponding state space* $\mathcal{S}$ *satisfies (R1) and (R2).*

Proof. Condition (R1) follows directly from (A1). By Lemma 29, if activities are resource (in)dependent in s, then they are also ratio-(in)dependent in any c with $c = \langle s, \gamma \rangle$. Combined with the definition of $ample_{\mathcal{S}}$, (R2) now also follows from (A2). □

An ample reduction on a (max,+) automaton preserves the minimum throughput and maximum latency values.

Theorem 35. *Let* $\mathcal{A}$ *be any (max,+) automaton with corresponding state space* $\mathcal{S}$. *Let ample be an ample reduction on* $\mathcal{A}$ *satisfying conditions (A1) and (A2) that yields* $\mathcal{A}'$ *with corresponding state space* $\mathcal{S}'$. *Then* $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$ *and* $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$.

Proof. Let $ample_{\mathcal{S}}$ be the corresponding ample reduction on $\mathcal{S}$ that yields state space $\hat{\mathcal{S}}$. By Lemma 31, $\hat{\mathcal{S}} = \mathcal{S}'$. Since $ample_{\mathcal{A}}$ satisfies conditions (A1) and (A2), by Lemma 34, $ample_{\mathcal{S}}$ satisfies conditions (R1) and (R2). By Corollary 23, this implies that $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$ and $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$. Therefore, if $ample_{\mathcal{A}}$ satisfies (A1) and (A2), then the minimum throughput and maximum latency values are preserved in the reduced state space $\mathcal{S}'$. □

IV. On-The-Fly Reduction

The previous section gives sufficient conditions for a reduction function on a (max,+) automaton to preserve performance properties in the corresponding (max,+) state space. For an efficient reduction, we avoid first computing the full composition of the (max,+) automata. Rather, we use sufficient local conditions on the network of (max,+) automata to compute a reduced composition on-the-fly.

The traditional on-the-fly method of Peled [19] selects the enabled activities $enabled_i(s) = enabled(s) \cap \mathcal{A}ct_i$ of some automaton $\mathcal{A}_i$ as ample set, while exploring a state $s = \langle s_1, s_2, \ldots, s_n \rangle$. The ample condition requires that all locally enabled activities in $enabled(s_i)$ of this automaton $\mathcal{A}_i$ are resource-independent with all activities in the other automata. Otherwise, all enabled activities in s are selected. In an experimental evaluation, we found that this approach did not yield any reduction on the models described in Section V. Therefore, we consider a generalization of the approach to clusters, as in [20]. The cluster-inspired ample reduction selects a safe cluster in each state in the composition. Given max-plus timed system $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ and $\mathcal{A} = \{\mathcal{A}_i \mid 1 \leq i \leq n\}$, we define a *cluster* $\mathcal{C} \subseteq \mathcal{A}$. Let $\mathcal{A}ct(\mathcal{C}) = \bigcup_{\mathcal{A}_i \in \mathcal{C}} \mathcal{A}ct_i$ denote the set of activities that occur in $\mathcal{C}$. The set of enabled activities that is selected in state s induced by a cluster $\mathcal{C}$ is $enabled_{\mathcal{C}}(s) = enabled(s) \cap \mathcal{A}ct(\mathcal{C})$. We define a projection to consider the local state $\pi_{\mathcal{C}}(s)$ in a cluster $\mathcal{C} \subseteq \mathcal{A}$ in the following way:

$$\pi_{\mathcal{A}_i}(s) = s_i$$
$$\pi_{\mathcal{C}}(s) = \langle \pi_{\mathcal{A}_{c_1}}(s), \ldots, \pi_{\mathcal{A}_{c_k}}(s) \rangle \text{ where } \mathcal{C} = \{\mathcal{A}_{c_1}, \ldots, \mathcal{A}_{c_k}\}, \text{ and for all } 1 \leq j < k, c_j < c_{j+1}.$$

Given local state $\pi_{\mathcal{C}}(s)$, $enabled(\pi_{\mathcal{C}}(s))$ denotes the set of activities that are enabled in the composition of precisely the automata in $\mathcal{C}$. Note that $enabled_{\mathcal{C}}(s) \subseteq enabled(\pi_{\mathcal{C}}(s))$, since the latter might contain activities that are enabled in the local state of the cluster-composition, but disabled in the global composition due to an automaton outside the cluster that disables the activity. We only consider independence of activities among automata, and not within the same automaton. The former can be checked locally, whereas the latter requires an exploration on the internal transition structure. We treat activities inside the same automaton as resource dependent.

Definition 36 (Cluster safety)**.** *Let* $\mathcal{C} \subseteq \mathcal{A}$ *be any cluster, and* s *be a state in the composition. Cluster* $\mathcal{C}$ *is safe in* s *if the following conditions are satisfied.*

(C1) *if* $enabled(s) \neq \emptyset$, *then* $enabled_{\mathcal{C}}(s) \neq \emptyset$;

(C2.1) *for any* $A \in enabled_{\mathcal{C}}(s)$ *and* $B \in \mathcal{A}ct(\mathcal{A}) \backslash \mathcal{A}ct(\mathcal{C})$, $R(A) \cap R(B) = \emptyset$;

(C2.2) *for any* $A \in enabled(\pi_{\mathcal{C}}(s))$, *if* $A \in \mathcal{A}ct_i$ *then* $\mathcal{A}_i \in \mathcal{C}$.

Condition (C2.1) requires that each enabled activity in $enabled_{\mathcal{C}}(s)$ does not share resources with any activity outside $\mathcal{A}ct(\mathcal{C})$. Condition (C2.2) requires that each activity in $enabled(\pi_{\mathcal{C}}(s))$ does not occur outside of the cluster. Together, these two conditions ensure that no activity $A \in \mathcal{A}ct(\mathcal{A}) \setminus enabled_{\mathcal{C}}(s)$, dependent on some activity in $enabled_{\mathcal{C}}(s)$, becomes enabled by executing only activities outside the cluster. We define a cluster-inspired ample reduction through a safety condition M on a max-plus timed system in the following way.

Definition 37 (Cluster-inspired ample reduction)**.** *A cluster-inspired ample reduction function ample for a max-plus timed system* $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ *is a mapping from* $S = S_1 \times \ldots \times S_n$ *to* $2^{\mathcal{A}ct}$ *such that* $ample(s) = enabled_{\mathcal{C}}(s)$ *for some cluster* $\mathcal{C} \subseteq \mathcal{A}$, *and satisfies the following condition:*

(M) Cluster-safety condition: *for any state* s, $ample(s) = enabled_{\mathcal{C}}(s)$ *where* $\mathcal{C}$ *is safe in* s.

The reduction of $\mathcal{M}$ *is defined using Def. 30.*

Theorem 38. *Let ample be a cluster-inspired ample reduction on $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$. Then ample satisfies conditions (A1) and (A2).*

Proof. Consider any state s in the composition of $\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$. First consider the case where $enabled(s) = \emptyset$. Then by Def. 37 $ample(s) = \emptyset$, and conditions (A1) and (A2) are satisfied. Now assume that $enabled(s) \neq \emptyset$, and let $ample(s) = enabled_{\mathcal{C}}(s)$, where $\mathcal{C}$ is any safe cluster in s. Since cluster $\mathcal{C}$ is safe, by Def. 36 it satisfies conditions (C1), (C2.1) and (C2.2). We need to show that $enabled_{\mathcal{C}}(s)$ satisfies conditions (A1) and (A2).

Condition (A1) follows directly from the definition. We prove condition (A2) by contraposition. Assume that condition (A2) does not hold. This means that there exists a finite run fragment $\rho = s \xrightarrow{A_1} s_1 \xrightarrow{A_2} s_2 \xrightarrow{A_3} \ldots s_{n-1} \xrightarrow{A_n}$, where $A_1 \ldots A_{n-1}$ are resource-independent with $ample(s) = enabled_{\mathcal{C}}(s)$, and A_n is resource-dependent with some activity in $enabled_{\mathcal{C}}(s)$.

Since A_n is resource-dependent with some activity in $enabled_{\mathcal{C}}(s)$, by condition (C2.1), $A_n \in \mathcal{A}ct(\mathcal{C})$. Moreover, we have $A_n \notin enabled_{\mathcal{C}}(s)$. Since activities $A_1 \ldots A_{n-1}$ are resource-independent with $ample_{\mathcal{C}}(s)$, $A_1 \ldots A_{n-1} \in \mathcal{A}ct(\mathcal{A}) \setminus \mathcal{A}ct(\mathcal{C})$ and they do not affect the state of $\mathcal{C}$, which means that $\pi_{\mathcal{C}}(s)$ does not change in the first $n-1$ steps. As $A_n \in enabled(s_{n-1})$, $A_n \in enabled(\pi_{\mathcal{C}}(s))$. We also have that $A_n \notin enabled_{\mathcal{C}}(s)$, since otherwise $A_n \in enabled_{\mathcal{C}}(s)$, contradicting our assumption. This means that A_n becomes enabled in $\pi_{\mathcal{C}}(s)$ by executing one of the activities in set $A_1, \ldots, A_{n-1}$. Since $A_1, \ldots, A_{n-1}$ are activities outside of $\mathcal{C}$, there must be some A_i with $1 \leq i \leq n-1$ that enabled A_n, which can only happen if A_n occurs outside of cluster $\mathcal{C}$ by definition of synchronous composition. This contradicts condition (C2.2). □

In each state in the composition, there possibly exist multiple safe clusters that can be chosen. Heuristics can be used to select a cluster that likely leads to a large reduction. In our experiments, we use a heuristic that chooses a smallest safe cluster in each state. This heuristic often performs well [27], because it allows to prune most enabled transitions. Starting from each enabled activity in s as a candidate, we construct a safe cluster. After trying all candidates, we select a cluster $\mathcal{C}$ with the smallest $enabled_{\mathcal{C}}(s)$ set. This approach is best suited when the enabled sets are small. If there are many states with large enabled sets, then a heuristic could be used to compute only one safe cluster starting from activities that occur in only few automata.

Algorithm 1 shows the algorithm to compute a safe cluster in a state. The algorithm checks for each activity enabled in the current cluster $\mathcal{C}$ whether condition (C2.1) or (C2.2) is violated. The algorithm starts with initial *candidate* activity A. If A is enabled in the composition, we add all automata that contain A and add an automaton for each dependent activity outside the current cluster. This ensures that condition (C2.1) is satisfied for activity A and the cluster obtained after executing lines 5-9. If A is enabled in the composition of automata in the cluster,

Algorithm 1. Algorithm to compute a safe cluster.

1: **proc** COMPUTECLUSTER(s, $candidate$)
2: $A \leftarrow candidate$; $\mathcal{C} \leftarrow \emptyset$; $processed \leftarrow \emptyset$
3: **while** $A \neq \bot$ **do**
4: $processed \leftarrow processed \cup \{A\}$
5: **if** $A \in enabled(s)$ **then**
6: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathcal{A}_i \mid A \in \mathcal{A}ct_i\}$
7: **for** $B \in \{D \in \mathcal{A}ct \mid R(D) \cap R(A) \neq \emptyset\}$ **do**
8: **if** $B \notin \mathcal{A}ct(\mathcal{C})$ **then**
9: $\mathcal{C} \leftarrow \mathcal{C} \cup [\mathcal{A}_i \mid B \in \mathcal{A}ct_i].first()$
10: **if** $A \notin enabled(s) \wedge A \in enabled(\pi_{\mathcal{C}}(s))$ **then**
11: **for** $\mathcal{A}_i \in \mathcal{A}$ **do**
12: **if** $A \in \mathcal{A}ct_i \wedge \mathcal{A}_i \notin \mathcal{C} \wedge A \notin enabled(s_i)$ **then**
13: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathcal{A}_i\}$
14: break;
15: **if** $\left|enabled(\pi_{\mathcal{C}}(s)) \setminus processed\right| > 0$ **then**
16: $A \leftarrow [enabled(\pi_{\mathcal{C}}(s)) \setminus processed].first()$
17: **else**
18: $A \leftarrow \bot$
19: **return** $\mathcal{C}$

but not in the full composition, then we add an automaton that causes A to be disabled in the full composition. This ensures that condition (C2.2) is satisfied for A for the cluster obtained after executing lines 10-14. We use the notation $[\mathcal{A}_i \mid B \in Act_i]$ to denote a list comprehension, and function $first()$ picks the first element from a list. After handling the activity, we check whether there are other activities that are locally enabled in the new cluster and not yet processed (line 15-18). We continue until all locally enabled activities are processed. The algorithm is repeatedly called for each activity in $enabled(s)$ as a candidate, and afterwards the cluster with the smallest set of enabled activities is chosen.

Theorem 39. *Let s be a state in the composition, and A be the candidate activity.* COMPUTECLUSTER(s, A) *returns a cluster that is safe in s.*

Proof. Consider activity A and let $\mathcal{C}_k$ denote the value of $\mathcal{C}$ at line 6, and $\mathcal{C}_{k+1}$ the new cluster after executing lines 4-18. We show that conditions (C2.1) and (C2.2) hold for A in $\mathcal{C}_{k+1}$ and any superset of $\mathcal{C}_{k+1}$. We consider two cases:

- case $A \in enabled(s)$: $\mathcal{C}_{k+1}$ contains all automata $\mathcal{A}_i$ with $A \in \mathcal{A}ct_i$ (added at line 6), which means that condition (C2.2) is satisfied for A. Since there are no automata outside $\mathcal{A}_i$ with A, condition (C2.2) holds also for any superset of $\mathcal{C}_{k+1}$. After executing lines 7-9, there is no resource-dependent activity outside cluster $\mathcal{C}_{k+1}$, which means that condition (C2.1) is satisfied. It suffices to add only one automaton with such an activity to the cluster to satisfy condition (A2.1), since then this resource-dependent activity becomes part of the cluster. Condition (C2.1) is also satisfied in any superset of $\mathcal{C}_{k+1}$.
- case $A \notin enabled(s) \wedge A \in enabled(\pi_{\mathcal{C}}(s))$: after executing lines 11-14, condition (C2.1) trivially holds since $A \notin enabled(s)$. Since $A \notin enabled(s)$ and $A \in enabled(\pi_{\mathcal{C}}(s))$, there exists at least one automaton $\mathcal{A}_i \notin \mathcal{C}_k$ where $A \notin enabled(\pi_i(s))$. Let

	automata composition				normalized (max,+) state space					
	full		reduced		full		reduced			
	$\|S\|$	$\|T\|$	$\|S\|$	$\|T\|$	$\|C\|$	$\|\Delta\|$	$\|C\|$		$\|\Delta\|$	
TW1	1280	2170	1260	2130	3585	6120	3555	0.8%	6060	1.0%
TW2	63	128	34	44	379	756	149	60.7%	207	72.6%
TW3	343	759	151	209	4949	10655	1507	69.5%	2261	78.8%
TW4	318	768	139	202	18792	40957	5280	71.9%	7785	81.0%

TABLE I
REDUCTIONS ACHIEVED ON THE TWILIGHT SYSTEM MODELS.

$\mathcal{C}_{k+1} = \mathcal{C}_k \cup \{\mathcal{A}_i\}$, obtained at line 13. Then $A \notin enabled(\pi_{\mathcal{C}_{k+1}}(s))$, which means that condition (C2.2) holds. Since $\mathcal{A}_i \in \mathcal{C}_{k+1}$, A will remain disabled in any superset of $\mathcal{C}_{k+1}$.

After executing lines 4-14 for activity A, conditions (C2.1) and (C2.2) hold for A and keep holding for any extension to the cluster. Upon termination of the algorithm, we obtain some cluster $\mathcal{C}_l$ where conditions (C2.1) and (C2.2) hold for each activity in $enabled(\pi_{\mathcal{C}_l}(s))$. From this it follows that $\mathcal{C}_l$ is safe in s. □

Example 40. *Consider the initial state s_1 in the composition $\mathcal{A}_1 \parallel \mathcal{A}_2 \parallel \mathcal{A}_3$ shown in Fig. 1b. We show how Algorithm 1 computes a valid ample set for s_1. Assume that A is the candidate activity. Since A is enabled, we add all automata with A (line 6) to cluster $\mathcal{C}$, i.e. $\mathcal{A}_1$. Since activity B is resource-dependent with A in s_1 and B is not yet in the cluster alphabet, we add the first automaton with B, i.e. $\mathcal{A}_2$, to $\mathcal{C}$ (line 9). There are no other activities enabled within the local state $\pi_{\mathcal{C}}(s_1)$ of the cluster, so we skip lines 11-14. Since we processed all activities and $A = \bot$, cluster $\mathcal{C}$ is returned. This yields $ample(s_1) = enabled_{\mathcal{C}}(s_1) = enabled(s_1) \cap Act(\mathcal{C}) = \{A, B, C\} \cap \{A, B, D\} = \{A, B\}$.*

Using a similar reasoning, one can validate that $\mathcal{C}_2 = \{\mathcal{A}_3\}$ (with $enabled_{\mathcal{C}_2}(s_1) = \{C\}$) is also a safe cluster in s_1. A reduced composition obtained with an ample reduction is shown in Fig. 1c. Since activity C is resource-independent with activities A and B, only one interleaving of C with A and B is explored. Both interleavings of A and B are still present, since activities A and B are resource-dependent. Fig. 4 shows the (max,+) state space after reduction.

V. EXPERIMENTAL EVALUATION

In the previous section, we described an on-the-fly reduction on the level of (max,+) automata to compute a reduced composition that preserves throughput and latency values. To test the effectiveness of the partial-order reduction on models of manufacturing systems, we created a number of variants of the Twilight system [15] model.

The Twilight system, shown in Fig. 6, processes balls that need to be drilled. First, a ball is picked up by the load robot (LR) from the input buffer (IN). Then, the ball is put on the conditioner (COND), to heat the ball to a desired temperature. The conditioner has a heater action (H) to heat a ball. Once heated, the ball is transported to the (DRILL) by the load robot or unload robot (UR) to

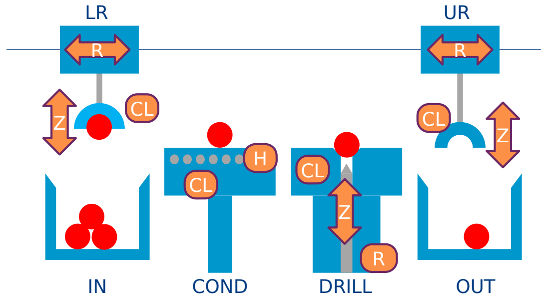


Fig. 6. Twilight manufacturing system [15] with two robots and two production stages.

drill a hole in the ball. The drill has an R-motor (R) to rotate the drill bit, and a Z-motor (Z) to move the drill bit up and down. After drilling, the unload robot puts the finished ball in the output buffer (OUT). Both robots have a clamp (CL) to pick up and hold a ball, an R-motor (R) to move along the rail, and a Z-motor (Z) to move the clamp up and down. Both processing stations have a clamp (CL) to ensure that the ball stays positioned correctly.

The system specification contains activities that describe the conditioning and drilling step, as well as activities that describe how a robot transports a product. In order to ensure safe movements, we model collision areas above the conditioner and the drill as virtual resources. The (max,+) automata describe constraints on valid activity orderings. The best-case and worst-case throughput and latency values are analyzed on the (max,+) state space of the composition of these automata.

We examine four variants of the Twilight system. The first variant (TW1) is the model described above, of which the full model is given in [15]. The life cycle and location of each product is explicitly modeled. In TW2, we remove these product-location and life-cycle automata, and instead use a set of automata that ensure that products are always moved forward in the production process. The TW2 model is more compact, leads to faster analysis, and is still correct. In TW3, we extend TW2 with a polish station, where each product undergoes the polish step and drill step after the condition step but not in a fixed order. In TW4 we fix the order so that a product is always first conditioned, then drilled, and then polished.

Table I shows that a reduction is achieved in all models. The reduction for TW1 is very small (0.8%) compared to the reductions of the other models (60.7%, 69.5&, and 71.9% for TW2, TW3, and TW4). The reduction for model TW1 is very small, because there is a lot of event synchronization by the product-location and life-cycle automata. Recall condition (C2.2), that requires that each enabled activity in the local state of a safe cluster must be independent with activities outside the cluster. During state-space exploration of model TW1, the algorithm often needs to add product-location or life-cycle automata to the cluster to satisfy this condition (C2.2); this limits reduction possibilities. The reduction for TW2, TW3, and TW4 is much larger, since we do not explicitly model the product-location and life-cycle automata. Note that the TW2 state space also before reduction is already much smaller than the TW1 state space. The TW2 model is therefore arguably a model better suitable for performance analysis than the original TW1 model.

VI. Conclusion

In this paper, we presented a new partial-order reduction technique to speed up performance analysis of max-plus timed systems. The technique is inspired by existing cluster-based ample set reduction for non-timed systems. It tries to compute a smaller state space by exploiting the structure of the concurrent (max,+) automata, and information about resource sharing. We derived conditions to compute a reduced composition, from which a reduced state space can be calculated. In this reduced state space, performance properties (throughput and latency) of the system model are preserved. The experimental evaluation shows that the partial-order reduction technique can be successfully used to reduce the size of the state space for a set of example models of manufacturing systems. The reductions that can be achieved are highly dependent on the structure of the input model, the amount of synchronization on activities among automata, and the extent to which activities use the same resources. In our models, the partial-order reduction technique successfully exploits redundant interleaving related to processing stations that can perform operations on products in parallel, and movements of the robots that can be executed simultaneously.

Acknowledgment

This research is supported by the Dutch NWO-TTW, carried out as part of the Robust Cyber-Physical Systems (RCPS) program, project number 12694.

References

[1] M. Hendriks, J. Verriet, T. Basten *et al.*, "Performance engineering for industrial embedded data-processing systems," in *Proc. of Int. Conf. on Product-Focused Software Process Improvement (PROFES)*, 2015, pp. 399–414.

[2] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems.* New York, NY, USA: Springer-Verlag New York, Inc., 1992.

[3] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[4] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, "Partial order reductions for timed systems," in *Proc. of Int. Conf. on Concurrency Theory (CONCUR)*, D. Sangiorgi and R. de Simone, Eds. Berlin, Heidelberg: Springer, 1998, pp. 485–500.

[5] M. Minea, "Partial order reduction for model checking of timed automata," in *Proc. of Int. Conf. on Concurrency Theory (CONCUR).* London, UK: Springer, 1999, pp. 431–446.

[6] T. Yoneda and B.-H. Schlingloff, "Efficient verification of parallel real–time systems," *Formal Methods in System Design*, vol. 11, no. 2, pp. 187–215, 1997.

[7] B. Theelen, M. Geilen, and J. Voeten, "Performance model checking scenario-aware dataflow," in *Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS)*, U. Fahrenberg and S. Tripakis, Eds. Berlin, Heidelberg: Springer, 2011, pp. 43–59.

[8] S. Gaubert, "Performance evaluation of (max,+) automata," *IEEE Trans. on Automatic Control*, vol. 40, no. 12, Dec 1995.

[9] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Berlin, Heidelberg: Springer-Verlag, 2001.

[10] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, vol. 4, 2000, pp. 101–104 vol.4.

[11] M. Geilen, "Synchronous dataflow scenarios," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 16:1–16:31, Jan. 2011.

[12] M. Geilen and S. Stuijk, "Worst-case performance analysis of Synchronous Dataflow scenarios," *Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 125–134, 2010.

[13] J. Campos, G. Chiola, J. M. Colom, and M. Silva, "Properties and performance bounds for timed marked graphs," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 39, no. 5, pp. 386–401, May 1992.

[14] B. Heidergott, G. J. Olsder, and J. Van Der Woude, *Max Plus at Work.* Princeton University Press, 2014.

[15] B. van der Sanden, J. Bastos, J. Voeten *et al.*, "Compositional specification of functionality and timing of manufacturing systems," in *Forum on specification & Design Languages (FDL) 2016*, 2016, pp. 1–8.

[16] B. van der Sanden, M. Reniers, M. Geilen *et al.*, "Modular model-based supervisory controller design for wafer logistics in lithography machines," in *Int. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 416–425.

[17] A. Valmari, "A stubborn attack on state explosion," *Formal Methods in System Design*, vol. 1, no. 4, pp. 297–322, 1992.

[18] P. Godefroid, *Using partial orders to improve automatic verification methods.* Berlin, Heidelberg: Springer, 1991, pp. 176–185.

[19] D. Peled, "Combining partial order reductions with on-the-fly model-checking," *Formal Methods in System Design*, vol. 8, no. 1, pp. 39–64, 1996.

[20] T. Basten, D. Bošnački, and M. Geilen, "Cluster-based partial-order reduction," *Automated Software Engineering*, vol. 11, no. 4, pp. 365–402, 2004.

[21] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity: an algebra for discrete event systems.* John Wiley and Sons, 1992.

[22] W. Thomas, "Automata on infinite objects," *Handbook of theoretical computer science, Volume B*, pp. 133–191, 1990.

[23] M. Geilen, J. Falk, C. Haubelt *et al.*, "Performance analysis of weakly-consistent scenario-aware dataflow graphs," *Journal of Signal Processing Systems*, vol. 87, no. 1, pp. 157–175, Apr 2017.

[24] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM-TODAES*, vol. 9, no. 4, pp. 385–418, 2004.

[25] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[26] A. W. Mazurkiewicz, "Trace theory," in *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, 1986, pp. 279–324.

[27] J. Geldenhuys, H. Hansen, and A. Valmari, *Exploring the Scope for Partial Order Reduction.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 39–53.