ES

ELECTRONIC SYSTEMS

TU/e

Partial-Order Reduction for Synthesis and Performance Analysis of Supervisory Controllers

Bram van der Sanden[*§], Marc Geilen[*], Michel Reniers[*], Twan Basten[*§]
[*]Eindhoven University of Technology, Eindhoven, The Netherlands
[§]ESI (TNO), Eindhoven, The Netherlands
bram.vandersanden@tno.nl, m.c.w.geilen@tue.nl, m.a.reniers@tue.nl, a.a.basten@tue.nl

ES Reports
ISSN 1574-9517

ESR-2019-02, update 2022
4 January 2022

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

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 Synthesis and Performance Analysis of Supervisory Controllers

Bram van der Sanden*§, Marc Geilen*, Michel Reniers*, Twan Basten*§
*Eindhoven University of Technology, Eindhoven, The Netherlands
§ESI (TNO), Eindhoven, The Netherlands
Email: bram.vandersanden@tno.nl, m.c.w.geilen@tue.nl, m.a.reniers@tue.nl, a.a.basten@tue.nl

***Abstract*—A key challenge in the synthesis and subsequent analysis of supervisory controllers is the impact of state-space explosion caused by concurrency. The main bottleneck is often the memory needed to store the composition of plant and requirement automata and the resulting supervisor. Partial-order reduction (POR) is a well-established technique that alleviates this issue in the field of model checking. It does so by exploiting redundancy in the model with respect to the properties of interest. For controller synthesis, the functional properties of interest are nonblockingness, controllability, and least-restrictiveness. But also performance properties, such as throughput and latency are of interest. We propose an on-the-fly POR on the input model that preserves both functional and performance properties in the synthesized supervisory controller. This improves scalability of both synthesis and performance analysis. Experiments show the effectiveness of the POR on a set of realistic manufacturing system models**

I. Introduction

Supervisory controller synthesis [1] is a method to automatically synthesize a supervisor that restricts the behavior of a system, described by a plant, to a given requirement that describes the allowed behaviors of the plant. Standard synthesis first computes the composition of all plant and requirement automata, and subsequently prunes the state space to ensure properties like controllability and nonblockingness (explained below) of the resulting supervisor [1], [2]. A disadvantage of this synthesis is its limited scalability, caused by the memory complexity of $\mathcal{O}(|Q_P|^2 \cdot |\Sigma|)$ [1], [3], where Q_P is the set of specification states of the combined plant and requirement automata and Σ the set of events. The memory needed to store the full state space often becomes a bottleneck [4]. The size of the supervisor also directly impacts the efficiency of performance analysis, performed on the state space of the supervisor augmented with timing information.

An approach to improve scalability of synthesis and performance analysis is partial-order reduction (POR) [5], [6]. The idea of POR is to exploit redundancy in the network of automata to obtain a reduced composition, while preserving the properties of interest. In each state of the composite automaton, a subset of redundant enabled transitions is removed, in turn, reducing the number of reachable states. Synthesis can then be performed on this smaller automaton leading to a smaller supervisor, and a smaller state space for performance analysis.

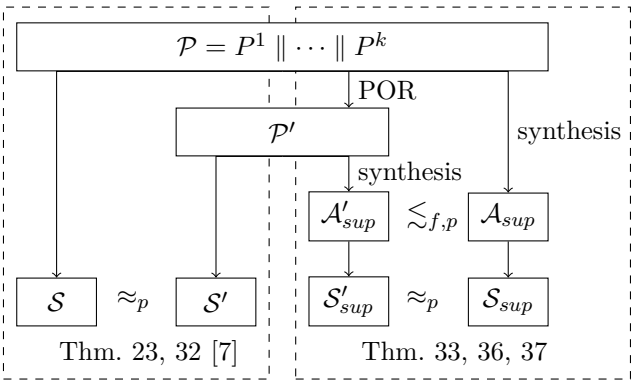


Fig. 1: Partial-order-reduction approach.

In the reduction, we want to preserve both functional and performance aspects. The functional properties of interest are controllability, nonblockingness, and least-restrictiveness. The supervisor needs to be *controllable* with respect to the plant, which means that it should not disable uncontrollable events. Furthermore, the composition of the supervisor with the plant, the *controlled system*, should be *nonblocking*, which means that from every reachable state in the controlled system, a *marked state* can be reached that typically indicates the completion of an operation. The supervisor is (non)blocking if the controlled system is (non)blocking [3]. In addition, it is required that the supervisor is *least restrictive*, which means that it restricts the system as little as possible while still being controllable and nonblocking. The performance aspects of interest are throughput and latency. *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.

Fig. 1 shows the proposed POR approach, consisting of two main results. The first result is a POR that preserves performance aspects, extending [7] with more detailed proofs and a more extensive experimental evaluation. Thm. 23 [7] shows that POR applied directly on state space $\mathcal{S}$ preserves performance aspects in reduced state space $\mathcal{S}'$, denoted by $\mathcal{S} \approx_p \mathcal{S}'$ (Def. 14). Thm. 32 [7] lifts that

result to automata models, with a POR on plant $\mathcal{P}$ that is a composition of automata[1] $P^1 \parallel \cdots \parallel P^k$, yielding a reduced plant $\mathcal{P}'$. Ref. [7] shows that this reduction can be done on the fly. In this report, we also consider the preservation of functional aspects, extending [9] with more detailed proofs. Synthesis can be applied on $\mathcal{P}$ directly to obtain a supervisor $\mathcal{A}_{sup}$ (an automaton). Alternatively, using POR, $\mathcal{P}'$ is computed on which synthesis can be applied to obtain a reduced supervisor $\mathcal{A}'_{sup}$. The conditions on the reduction to $\mathcal{P}'$ guarantee that the functional and performance properties of $\mathcal{A}_{sup}$ are preserved in $\mathcal{A}'_{sup}$, denoted $\mathcal{A}'_{sup} \lesssim_{f,p} \mathcal{A}_{sup}$ (defined precisely in Def. 30). This relation ensures that $\mathcal{A}'_{sup}$ is nonblocking if $\mathcal{A}_{sup}$ is nonblocking, $\mathcal{A}'_{sup}$ is controllable with respect to $\mathcal{P}$ if $\mathcal{A}_{sup}$ is controllable with respect to $\mathcal{P}$, and $\mathcal{A}'_{sup}$ is least-restrictive to $\mathcal{P}$ under an adapted notion of least-restrictiveness (Def. 10) that considers redundancy. The relation also guarantees that the performance aspects are preserved in the corresponding timed state spaces $\mathcal{S}_{sup}$ and $\mathcal{S}'_{sup}$, i.e., $\mathcal{S}_{sup} \approx_p \mathcal{S}'_{sup}$. Thm. 33 proves the result that synthesis after POR preserves both functional and performance aspects, with Thm. 36 proving that the reduced plant can be obtained on the fly during composition. Thm. 37 proves that our concrete POR algorithm (Alg. 1) provides a valid on-the-fly reduction to compute reduced plant $\mathcal{P}'$.

We follow [10] by taking system *activities* as the events in our models. An activity captures a functionally deterministic part of the system behavior, consisting of low-level actions operating on system resources and (acyclic) precedences between those actions. An activity may, for example, correspond to moving a robot arm from one specified position to another position, consisting of several actions on one or more motor resources in a specific order. Actions of different activities may not interfere with each other except through resource claims and releases. During the execution of an activity, the relevant resources are claimed and no interference on these resources is possible. Resources are assigned to activities in the order in which activities are executed. Activities that use different sets of resources may execute concurrently. Activities can then be treated as atomic events, abstracting from the execution orders of activity-internal concurrent actions. As shown in [10], this improves scalability of controller synthesis.

There are various ways to capture the timing behavior of supervisory controllers. Some well-known approaches are real-valued clocks as used in timed automata [11], discrete-valued clocks as used in tick-based models [12], and (max,+) algebra [13]–[15]. We use (max,+) algebra (see for instance [16]), which fits naturally with the notion of activities. A (max,+) timing matrix expresses the relation between the availability times of the system resources and the release times of the resources after executing an activity. Such a (max,+) timing model enables efficient performance analysis [15]. Supervisor synthesis on automata with activities can be done without considering their timing because activities can be treated as atomic events. Given the supervisor and the timing matrices of the activities, a timed (max,+) state space can be computed that provides the necessary timing information to evaluate system throughput and latency. Our POR technique preserves both functional and performance properties. It improves scalability of both controller synthesis and performance analysis for automata models with (max,+) timing semantics. It can also be used on conventional finite-state automata with events, by assuming activities do not claim or release resources and are assigned the empty 0×0 (max,+) timing matrix (implying that they are timeless).

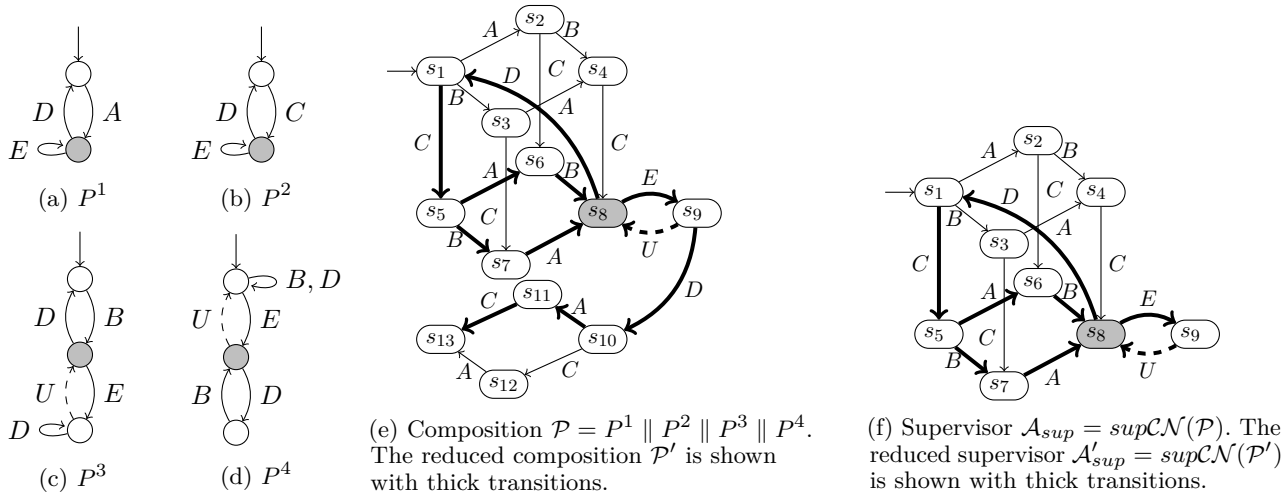
The report is structured as follows. Section II introduces the framework that we use to capture a (max,+) timed system. Section III defines the functional and performance aspects considered in the POR. Section IV describes the conditions that are needed on a (max,+) state-space reduction to preserve performance aspects. Section V lifts these conditions to the level of automata, and it considers supervisor synthesis prior to performance analysis, adding conditions to also preserve functional aspects. Section VI then introduces an on-the-fly reduction that uses local conditions to compute a reduced automaton directly from a composition of automata. The experimental evaluation in Section VII shows the effectiveness of our POR technique. Related work is described in Section VIII and Section IX concludes. Proofs can be found in the appendix.

II. Modeling

Consider a running example with activities A, B, C, D, E, and U. To capture all possible activity orderings in the system, we use (max,+) automata. A (max,+) automaton is a conventional finite-state automaton, where the timing semantics of each activity is described by a (max,+) matrix, as explained below. The (max,+) automata of the running example are shown in Fig. 2. We use a representation of (max,+) automata that extends the definition of [15] with rewards and we restrict ourselves to a setting with deterministic (max,+) automata. This corresponds to the construction discussed in Section VI in [15], in which a regular language constraining the possible sequences of events is combined with a (max,+) automaton defining the timing constraints of the events using the Hadamard product. In our representation, the automata encode the possible sequences of activities (the regular language of events) and the timing constraints are encoded using the classical matrix representation of (max,+) automata.

Definition 1 ((max,+) automaton (adapted from [15])). *A* (max,+) automaton $\mathcal{A}$ *is a tuple* $\langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$ *where* S *is a finite set of states,* $\hat{s} \in S$ *is the initial state,* $S^m \subseteq S$ *is the set of* marked states*,* $\mathcal{A}ct$ *is a nonempty set of activities,* $reward : \mathcal{A}ct \to \mathbb{R}^{\geq 0}$ *quantifies the progress per activity,* M *maps each activity to its associated (max,+) matrix, and* $T \subseteq S \times \mathcal{A}ct \times S$ *is the transition relation. We assume*

[1]For the purpose of supervisory controller synthesis, we assume that all requirements are translated into plant automata using a *plantify* transformation [8], such that we can treat all automata in the same way.



(a) P^1 (b) P^2 (c) P^3 (d) P^4

(e) Composition $\mathcal{P} = P^1 \parallel P^2 \parallel P^3 \parallel P^4$. The reduced composition $\mathcal{P}'$ is shown with thick transitions.

(f) Supervisor $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$. The reduced supervisor $\mathcal{A}'_{sup} = sup\mathcal{CN}(\mathcal{P}')$ is shown with thick transitions.

Fig. 2: Running example: plants P^1, P^2, P^3, and P^4, and the composition $\mathcal{P}$. Transitions of uncontrollable activities are denoted with dashed arrows. Marked states are indicated in gray.

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

$$\boldsymbol{M}_D = \begin{bmatrix} 2 & -\infty & 3 \\ -\infty & 0 & -\infty \\ 2 & -\infty & 3 \end{bmatrix} \quad \boldsymbol{M}_E = \begin{bmatrix} 0 & -\infty & -\infty \\ -\infty & 0 & -\infty \\ -\infty & -\infty & 5 \end{bmatrix} \quad \boldsymbol{M}_U = \begin{bmatrix} 2 & -\infty & -\infty \\ -\infty & 0 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix}$$

Fig. 3: (max,+) matrices of activities A, B, C, D, E and U.

that $\mathcal{A}$ is deterministic; for any $s, s', s'' \in S$ and $A \in \mathcal{A}ct$, $\langle s, A, s' \rangle \in T$ and $\langle s, A, s'' \rangle \in T$ imply $s' = s''$.

Both plants and supervisors are described as (max,+) automata. Marked states define a notion of completion of the plant for supervisor synthesis, as illustrated below.

Let set $\mathcal{A}ct^*$ contain all finite *strings* over $\mathcal{A}ct$, including the empty string ε. We write $s_1 \xrightarrow{A} s_2$ if $\langle s_1, A, s_2 \rangle \in T$. The transition relation is extended to $\mathcal{A}ct^*$ in the relation $\rightarrow^*$ by letting $s \xrightarrow{\varepsilon}{}^* s$ for all $s \in S$, and for all $\alpha \in \mathcal{A}ct^*$, $A \in \mathcal{A}ct$, $s, s' \in S$, $s \xrightarrow{\alpha A}{}^* s'$ if $s \xrightarrow{\alpha}{}^* s''$ and $s'' \xrightarrow{A} s'$ for some $s'' \in S$. We write $s \rightarrow^* s'$ if $s \xrightarrow{\alpha}{}^* s'$ for some $\alpha \in \mathcal{A}ct^*$. 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 $A(s)$. For activity sequence $A_1 \dots A_n$, the resulting state is defined inductively as $\varepsilon(s) = s$ and $(A_1 \dots A_n A_{n+1})(s) = A_{n+1}((A_1 \dots A_n)(s))$ for $n \geq 0$ if $A_{n+1} \in enabled((A_1 \dots A_n)(s))$ and $(A_1 \dots A_n)(s)$ is defined.

A possible behavior of an automaton is described in a *path*. A(n) (in)finite path p of $\mathcal{A}$ is a(n) (in)finite, alternating sequence of states and activities: $p = s_0 A_1 s_1 A_2 s_2 A_3 \dots$ such that $s_0 = \hat{s}$ and $s_{i+1} = A_{i+1}(s_i)$ for all $i \geq 0$. A finite path always ends in a state. Given path p and $i \geq 0$, $p[..i]$ denotes prefix $s_0 A_1 \dots s_i$, and $p[i, j] = s_i A_{i+1} \dots s_j$ with $0 \leq i \leq j$ is the path fragment from state s_i up to s_j. Further, $p[i]$ denotes state s_i in p.

Definition 2 (Synchronous composition of (max,+) automata)**.** *Given automata $\mathcal{A}_1 = \langle S_1, \hat{s_1}, S_1^m, \mathcal{A}ct_1, reward_1, M_1, T_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, \hat{s_2}, S_2^m, \mathcal{A}ct_2, reward_2, M_2, T_2 \rangle$, with for each activity $A \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2$, $reward_1(A) = reward_2(A)$ and $M_1(A) = M_2(A)$, the synchronous composition $\mathcal{A}_1 \parallel \mathcal{A}_2 = \langle S_1 \times S_2, \langle \hat{s_1}, \hat{s_2} \rangle, S_1^m \times S_2^m, \mathcal{A}ct_1 \cup \mathcal{A}ct_2, reward_1 \cup reward_2, M_1 \cup M_2, T_{12} \rangle$, with*

$\langle s_1, s_2 \rangle \xrightarrow{A}_{12} \langle s_1', s_2' \rangle$ *if* $A \in \mathcal{A}ct_1 \cap \mathcal{A}ct_2, s_1 \xrightarrow{A}_1 s_1', s_2 \xrightarrow{A}_2 s_2'$;

$\langle s_1, s_2 \rangle \xrightarrow{A}_{12} \langle s_1', s_2 \rangle$ *if* $A \in \mathcal{A}ct_1 \setminus \mathcal{A}ct_2, s_1 \xrightarrow{A}_1 s_1'$;

$\langle s_1, s_2 \rangle \xrightarrow{A}_{12} \langle s_1, s_2' \rangle$ *if* $A \in \mathcal{A}ct_2 \setminus \mathcal{A}ct_1, s_2 \xrightarrow{A}_2 s_2'$.

Fig. 2e shows the synchronous composition of $P^1 \parallel P^2 \parallel P^3 \parallel P^4$ (composition is associative). An activity is disabled in a state, if it is disabled in the current state of one of the automata that has the activity in its alphabet. For example, D is initially disabled because plant P^3 initially disables activity D. The synchronous composition of deterministic automata is again deterministic.

Activities abstract from low-level activity-internal action executions. For our intended supervisor synthesis and performance analysis, we only need the timing of the claims and releases of resources. So we abstract from the low-level actions. The (max,+) matrices of the activities of the running example are shown in Fig. 3. Each matrix row represents the release time of a resource in terms of all the availability times of resources.

To illustrate, consider matrix $\boldsymbol{M}_A$ of activity A. The first row describes the release time of resource r_1 in terms of when resources r_1, r_2 and r_3 are available at the start of executing A. The execution of A implies a time delay of 4 time units between the claiming of r_1 and its subsequent release. Similarly, a delay of 5 occurs between the claiming of r_2 and the release of r_1. There is no dependency on the availability of r_3, indicated by $-\infty$. We define a (global)

resource set $Res = \{r_i \mid 1 \leq i \leq s\}$ where s is the number of rows and columns of the matrices. In our running example, we have $Res = \{r_1, r_2, r_3\}$. Function R maps each activity to the set of resources used. For activity A with (max,+) matrix $\boldsymbol{M}_A$, $r \notin R(A)$ iff $[\boldsymbol{M}_A]_{r,r} = 0$, $[\boldsymbol{M}_A]_{i \neq r,r} = -\infty$ and $[\boldsymbol{M}_A]_{r,j \neq r} = -\infty$. For example, given $\boldsymbol{M}_B$ in Fig. 3, $R(B) = \{r_1, r_2\}$. As $r_3 \notin R(B)$, r_3 does not influence the result of multiplying a vector with $\boldsymbol{M}_B$.

The two essential characteristics in an activity execution are *synchronization*, for example when an action needs to wait until resources are available, and *delay*, to model the execution time of actions on resources. These characteristics correspond to the (max,+) operators *maximum* (max) and *addition* (+), defined over the set $\mathbb{R}^{-\infty} = \mathbb{R} \cup \{-\infty\}$. These operators are defined as usual, 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 $m \times p$ matrix $\boldsymbol{A}$ and $p \times n$ matrix $\boldsymbol{B}$, $\boldsymbol{A} \otimes \boldsymbol{B}$ denotes $(\max, +)$ matrix multiplication, resulting in matrix $\boldsymbol{A} \otimes \boldsymbol{B}$ with elements $[\boldsymbol{A} \otimes \boldsymbol{B}]_{ij} = \max_{k=1}^{p}([\boldsymbol{A}]_{ik} + [\boldsymbol{B}]_{kj})$. Adding a constant c to matrix $\boldsymbol{A}$ yields a new matrix $\boldsymbol{A} + c$ with $[\boldsymbol{A} + c]_{ij} = [\boldsymbol{A}]_{ij} + c$. For any vector $\boldsymbol{x}$ of size n, $\|\boldsymbol{x}\| = \max_{i=1}^{n}[\boldsymbol{x}]_i$ denotes the vector norm of $\boldsymbol{x}$. For vector $\boldsymbol{x}$, with $\|\boldsymbol{x}\| > -\infty$, $norm(\boldsymbol{x})$ denotes $\boldsymbol{x} - \|\boldsymbol{x}\|$, the normalized vector, with $\|norm(\boldsymbol{x})\| = 0$. $\boldsymbol{0}$ denotes a vector with only zero entries.

Resource availability times are captured in a (max,+) vector, typically denoted γ. Given such a *resource availability vector*, we obtain the new resource availability vector after executing an activity by multiplying it with the corresponding (max,+) matrix. Timing evolution is therefore expressed using (max,+) vector-matrix multiplication. When all resources are initially available at time 0, captured in vector $\boldsymbol{0}$, the new availability times of the resources after executing A are computed as follows:

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

Resources r_1 and r_2 are available again after 5 and 3 time units, respectively. Resource r_3 is not present in $R(A)$, but a row is present for r_3 to carry over its time stamp. Its availability time stays 0, since it is not used by A. The timing semantics of an activity sequence is defined in terms of repeated matrix multiplication. For example, the resource availability after executing activity sequence ABC given vector $\boldsymbol{0}$ (see Fig. 4) is computed as $\boldsymbol{M}_C \otimes \boldsymbol{M}_B \otimes \boldsymbol{M}_A \otimes \boldsymbol{0} = [6, 6, 4]^\intercal$. This representation of activity timing generalizes the well-known heaps-of-pieces model [17], [18], where pieces have a rigid structure (i.e. have a fixed shape). In our approach however, the (max,+) matrices encode flexible pieces, as illustrated in Fig. 4 for A.

We define the input model for our POR as a composition of (max,+) automata. The composition of all the individual

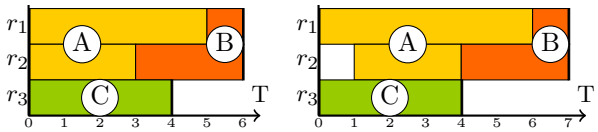


Fig. 4: Gantt chart of activity sequence ABC when all resources are initially available (left) and when resource r_2 is available after 1 time unit (right).

automata is again an automaton. We assume that the matrices of the constituent automata all have the same size to ensure that they can be multiplied. Additional resources can be added to a matrix by adding a new row and column for the resource and having $-\infty$ in all the new positions, except on the diagonal where the value is 0.

Definition 3 ((max,+) timed system)**.** *A* (max,+) timed system $\mathcal{M}$ *is described by* $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ *with (max,+) automata* $\mathcal{A}_i = \langle S_i, \hat{s}_i, S_i^m, \mathcal{A}ct_i, reward_i, M_i, T_i \rangle$ *with* $1 \leq i \leq n$*. We assume that all matrices have size* $|Res| \times |Res|$ *and that for all* $1 \leq i, j \leq n$ *and each activity* $A \in \mathcal{A}ct_i \cap \mathcal{A}ct_j$*,* $reward_i(A) = reward_j(A)$ *and* $M_i(A) = M_j(A)$*.*

A (max,+) automaton can be interpreted as a normalized $(\max, +)$ state space that captures each path of the automaton as a *run*, and contains all the necessary information for evaluation of performance properties. The state space records normalized resource availability vectors, with transitions between them. Each configuration $c = \langle s, \boldsymbol{\gamma} \rangle$ in the state space consists of a state s of the (max,+) automaton and a normalized resource availability vector $\boldsymbol{\gamma}$. Fig. 6 shows the normalized (max,+) state spaced of a simplified running example, discussed in more detail below.

Definition 4 (Normalized (max,+) state space (adapted from [19]))**.** *Given (max,+) automaton* $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T \rangle$ *with matrices of size* $|Res| \times |Res|$*, 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* $C = S \times \mathbb{R}^{-\infty^{|Res|}}$ *of configurations;*
- *initial configuration* $\hat{c} = \langle \hat{s}, \boldsymbol{0} \rangle$*;*
- *labeled transition relation* $\Delta \subseteq C \times \mathcal{A}ct \times C$ *that consists of the transitions in the set*
  $\{\langle \langle s, \boldsymbol{\gamma} \rangle, A, \langle s', norm(\boldsymbol{\gamma}') \rangle \rangle \mid s \xrightarrow{A} s' \wedge \boldsymbol{\gamma}' = M(A) \otimes \boldsymbol{\gamma}\}$ *(where* $norm(\boldsymbol{\gamma}') = \boldsymbol{\gamma}' - \|\boldsymbol{\gamma}'\|$*);*
- *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* $\langle c, A, c' \rangle \in \Delta$ *with* $c = \langle s, \boldsymbol{\gamma} \rangle$ *indicating the time passed during execution.*

The set of enabled activities and runs in a normalized (max,+) state space $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2 \rangle$ is defined in a similar way as in a (max,+) automaton for paths. A(n) (in)finite run ρ of $\mathcal{S}$ is a(n) (in)finite, alternating sequence of configurations and activities: $\rho = c_0 A_1 c_1 A_2 c_2 A_3 \ldots$ such that $c_0 = \hat{c}$ and $c_{i+1} = A_{i+1}(c_i)$ for all $i \geq 0$. 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 up to c_j, and

$\rho[i] = c_i$. Vector $\bar{\gamma}_n = (\bigotimes_{k=1}^{n} M(A_k)) \otimes \mathbf{0}$ gives the resulting resource availability vector after executing activities $A_1 \cdots A_n$ (without normalization). It can be derived from the normalized (max,+) state space through summation of encountered w_2 values and the final normalized vector.

Proposition 5. *Let $\mathcal{S}$ be a normalized (max,+) state space with run $\rho = c_0 A_1 c_1 A_2 c_2 A_3 \ldots$ such that $c_i = \langle s_i, \gamma_i \rangle$ for each i. Then, for all $n \geq 0$, $\bar{\gamma}_n = \sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + \gamma_n$.*

In the remainder, we restrict ourselves to the reachable part of the normalized (max,+) state space and we assume that this is finite. The reachable part might be infinite [19]. It is guaranteed to be finite, however, if for each simple cycle (no repetition of transitions is allowed) in the (max,+) automaton with corresponding activity sequence $A_1 \ldots A_k$, the matrix $\boldsymbol{M}_{A_k} \otimes \ldots \otimes \boldsymbol{M}_{A_1}$ is irreducible (meaning that its associated digraph is strongly connected). This can be verified by finding the simple cycles [20] and checking that the corresponding matrices have no entries $-\infty$. Practically, this means that the claim of any resource is linked to the claim of each other resource via the resource dependencies over each cycle in the automaton. In practical systems, resources typically do not operate fully independently.

III. Properties to be preserved

In this section, we introduce the properties that we want to preserve; controllability, nonblockingness, and least-restrictiveness as functional properties and latency and throughput as performance properties.

A. Controllability, nonblockingness, least-restrictiveness

In supervisory control, the activity alphabet $\mathcal{A}ct$ is partitioned into set $\mathcal{A}ct_c$ of *controllable* activities and set $\mathcal{A}ct_u$ of *uncontrollable* activities. In the running example, A, B, C, D, and E are controllable and U is uncontrollable.

To define least-restrictiveness, we need the notions of subautomata and union of automata. A *subautomaton* is obtained as a reduction of a given (max,+) automaton, where a subset of the states and transitions is preserved.

Definition 6 (Subautomaton)**.** *Let $\mathcal{A}_i = \langle S_i, \hat{s}, S_i^m, \mathcal{A}ct, reward, M, T_i \rangle$ for $i \in \{1, 2\}$ be two (max,+) automata with the same alphabet $\mathcal{A}ct$ (and derived reward function and timing matrices) and initial state $\hat{s}$. Automaton $\mathcal{A}_1$ is a* subautomaton *of $\mathcal{A}_2$, denoted $\mathcal{A}_1 \preceq \mathcal{A}_2$, iff $S_1 \subseteq S_2$, $T_1 \subseteq T_2$, and $S_1^m = S_1 \cap S_2^m$. The latter ensures that marking in $\mathcal{A}_1$ is consistent with marking in $\mathcal{A}_2$.*

Definition 7 (Union of (max,+) automata)**.** *Let $\mathcal{A}_i = \langle S_i, \hat{s}, S_i^m, \mathcal{A}ct, reward, M, T_i \rangle$, $i \in I$ be a set of automata all having the same activity alphabet, reward function, timing matrices, and initial state. The union of these (max,+) automata is defined as $\bigcup_{i \in I} \mathcal{A}_i = \langle \bigcup_{i \in I} S_i, \hat{s}, \bigcup_{i \in I} S_i^m, \mathcal{A}ct, reward, M, \bigcup_{i \in I} T_i \rangle$.*

The behavior of an *uncontrolled system* is represented by a plant $\mathcal{P}$. A supervisor $\mathcal{A}_{sup}$ is added to ensure that the *controlled system*, formed by $\mathcal{P} \parallel \mathcal{A}_{sup}$, is *nonblocking* [1].

Definition 8 (Nonblockingness)**.** *A (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T \rangle$ is* nonblocking *iff, for every state $s \in S$ such that $\hat{s} \rightarrow^* s$, s is a nonblocking state. A state $s \in S$ is a* nonblocking *state iff $s \rightarrow^* s^m$ for some marked state $s^m \in S^m$; otherwise it is a* blocking *state.*

Plant $\mathcal{P}$ in Fig. 2e is blocking, since states s_{10}, s_{11}, s_{12}, and s_{13} are blocking. $\mathcal{A}_{sup}$ in Fig. 2f is nonblocking, as it is always possible to return to s_8. This shows how marked states can be used to ensure a notion of completion.

We assume that $\mathcal{A}_{sup} \preceq \mathcal{P}$, since the computed supervisor is often a restriction of the plant by disabling transitions that lead to undesired behavior, violating the nonblocking requirement or the controllability requirement, defined below. Since $\mathcal{A}_{sup} \preceq \mathcal{P}$, $\mathcal{P} \parallel \mathcal{A}_{sup} = \mathcal{A}_{sup}$, which means that $\mathcal{P} \parallel \mathcal{A}_{sup}$ is nonblocking iff $\mathcal{A}_{sup}$ is nonblocking.

A supervisor is required to be *controllable* with respect to the plant it needs to control, such that it does not disable any uncontrollable activity that the plant defines in any state reachable in the controlled system.

Definition 9 (Controllability)**.** *Let $\mathcal{P} = \langle S_\mathcal{P}, \hat{s}_\mathcal{P}, S_\mathcal{P}^m, \mathcal{A}ct_\mathcal{P}, reward_\mathcal{P}, M_\mathcal{P}, T_\mathcal{P} \rangle$ and $\mathcal{A} = \langle S_\mathcal{A}, \hat{s}_\mathcal{A}, S_\mathcal{A}^m, \mathcal{A}ct_\mathcal{A}, reward_\mathcal{A}, M_\mathcal{A}, T_\mathcal{A} \rangle$ be (max,+) automata, with $\mathcal{A}ct_u \subseteq \mathcal{A}ct_\mathcal{P} \cup \mathcal{A}ct_\mathcal{A}$ the set of uncontrollable activities. $\mathcal{A}$ is* controllable with respect to $\mathcal{P}$ *iff, for every string $\alpha \in (\mathcal{A}ct_\mathcal{P} \cup \mathcal{A}ct_\mathcal{A})^*$, every state $s \in S_\mathcal{A}$, and every $U \in \mathcal{A}ct_u$ such that $\hat{s}_\mathcal{A} \xrightarrow{\alpha}_\mathcal{A}^* s$ and $\hat{s}_\mathcal{P} \xrightarrow{\alpha U}_\mathcal{P}^* s'$ for some $s' \in S_\mathcal{P}$, it holds that $s \xrightarrow{U}_\mathcal{A} s''$ for some $s'' \in S_\mathcal{A}$. Any state s of $\mathcal{A}$ that satisfies this property is called a* controllable *state; otherwise it is* uncontrollable.

As an example, consider string $CABE$ and path $s_1 \xrightarrow{CABE}_{\mathcal{A}'_{sup}}^* s_9$ in supervisor $\mathcal{A}'_{sup}$ shown in Fig. 2f. In plant $\mathcal{P}$, the execution of string $CABE$, leads to state s_9. In state s_9 in $\mathcal{P}$, uncontrollable activity U is enabled. In $\mathcal{A}'_{sup}$, activity U is also present in state s_9 in $\mathcal{A}'_{sup}$. Since this also holds for the other strings and uncontrollable activities, $\mathcal{A}'_{sup}$ is controllable with respect to $\mathcal{P}$.

The union of controllable and nonblocking subautomata of a given automaton is again controllable and nonblocking [8]. A subautomaton is least-restrictive iff it is the union of all the subautomata of $\mathcal{P}$ that satisfy both properties.

Definition 10 (Least-restrictiveness)**.** *Let $\mathcal{A}, \mathcal{A}'$ be (max,+) automata. Define predicate $CN(\mathcal{A}', \mathcal{A})$ to be true iff $\mathcal{A}' \preceq \mathcal{A}$ and $\mathcal{A}'$ is nonblocking and controllable with respect to $\mathcal{A}$. The supremal controllable and nonblocking subautomaton of $\mathcal{A}$ is $sup\mathcal{CN}(\mathcal{A}) = \bigcup_{\mathcal{A}', s.t.\ CN(\mathcal{A}', \mathcal{A})} \mathcal{A}'$. $\mathcal{A}'$ is* least restrictive *with respect to $\mathcal{A}$ iff $\mathcal{A}' = sup\mathcal{CN}(\mathcal{A})$.*

Both $\mathcal{A}_{sup}$ and $\mathcal{A}'_{sup}$ in Fig. 2f are nonblocking and controllable with respect to $\mathcal{P}$. Given $\mathcal{P}$, calculating $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$ is called *synthesis*. Synthesis algorithms implement fixed-point computations [1], [2], where in each iteration *bad* states are removed. Bad states are those states that are blocking or that lead to a blocking state through uncontrollable activities. These bad states are identified and removed iteratively, until no more such states remain in the resulting supervisor.

B. Throughput and latency

The performance of a supervisor is quantified using throughput and latency metrics, defined on the corresponding normalized (max,+) state space. Throughput is defined as the ratio between the cumulative reward and the cumulative execution time of a run. For throughput analysis, we only consider infinite runs. Latency is the temporal distance that separates the resource availability times of a resource at the start of two activity instances in a run. Latency analysis can be performed on both finite and infinite runs. For readability, we assume for performance analysis that a supervisor is *total*, meaning that each reachable state has at least one outgoing transition. A non-total supervisor has a throughput lower bound of zero. The presented latency analysis applies to non-total supervisors as well. As supervisor synthesis guarantees that from each state a marked state is reachable, the supervisor is total if each marked state has at least one outgoing transition. A total supervisor can be seen as an ω-automaton [21] accepting infinite words over $\mathcal{A}ct$. There are no specific acceptance conditions, so any infinite word starting in the initial state is accepted. Marked states are not used to define acceptance conditions. For performance analysis of a supervisor, all infinite runs in the timed states space $\mathcal{S}$ are considered, contained in set $\mathcal{R}(\mathcal{S})$.

We define *throughput* of a run in a normalized (max,+) state space as the ratio between the cumulative reward (sum of w_1 weights in the state space) and the cumulative execution time (sum of w_2 weights in the state space).

Definition 11 (Throughput)**.** *The ratio of a run (fragment) $\rho = c_0A_1c_1A_2c_2A_3\ldots$ is the ratio of the sums of weights w_1 and w_2, defined as follows*

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

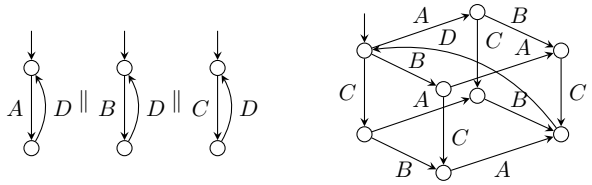
The throughput guarantee corresponds to the minimum ratio value *achieved by any of those runs:*

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

Since the reachable part of $\mathcal{S}$ is finite, infinite runs pass recurrent configurations infinitely often. Thus infinite runs are composed of simple cycles of the state space. The minimum ratio value of the state space is hence determined by the (reachable) simple cycle with the lowest ratio value, since this behavior can be continuously repeated in a run. This cycle thus provides a lower bound on the throughput (hence a throughput guarantee). Since $\mathcal{S}$ has a finite number of simple cycles, we can determine the minimum ratio value of the graph from a *minimum cycle ratio* (MCR) analysis [22].

Proposition 12 (Adapted from Prop. 1 in [15])**.** *$\tau_{min}(\mathcal{S}) = \inf_{cycle\in cycles(\mathcal{S})} \mathit{Ratio}(cycle) = MCR(\mathcal{S})$, where $cycles(\mathcal{S})$ denotes all cycles in $\mathcal{S}$ and $MCR(\mathcal{S})$ is the MCR of $\mathcal{S}$.*

To illustrate the MCR, consider the normalized (max,+) state space shown in Fig. 6, obtained from the composition



(a) $\bar{P}^1$, $\bar{P}^2$, and $\bar{P}^3$. (b) $\bar{\mathcal{P}} = \bar{P}^1 \parallel \bar{P}^2 \parallel \bar{P}^3$.

Fig. 5: Running example with (max,+) automata $\bar{P}^1$, $\bar{P}^2$, and $\bar{P}^3$, and composition $\bar{\mathcal{P}}$.

of plants $\bar{P}^1$, $\bar{P}^2$, and $\bar{P}^3$ shown in Fig. 5. We use this small model such that we can show the full (max,+) state space. In this state space, activities A, B and D have reward 0 (weight w_1), and activity C has reward 1. Then, the MCR is $1/8$, with a total reward of 1 and a total execution time of 8, which can for instance be found in the cycle corresponding to the execution of $(CBAD)^\omega$:

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

In this cycle, the first transition represents the execution of C on resource r_3, computed through matrix multiplication with $\boldsymbol{M}_C$; it adds 4 times units to the total execution time (weight w_2), and results in vector $[0,-4,4]^\intercal$. Since the vector is normalized, 4 time units are deducted from each value. The other transitions can be computed similarly. Other periodic executions where B precedes A, i.e. $(BACD)^\omega$ and $(BCAD)^\omega$, have the same MCR value.

Latency is the time delay between a stimulus and its effect. A well-known related notion in the field of production systems is makespan, where the stimulus is the start of the schedule and the effect is the completion of the product. In the context of (max,+) timed systems, we define *latency* in terms of the temporal distance that separates the resource availability times of a resource at a designated source activity A_{src} and sink activity A_{snk}. In the state space, consider some run $\rho = c_0A_1c_1A_2\ldots$ with $c_i = \langle s_i, \boldsymbol{\gamma}_i\rangle$ containing run fragment $\rho[i, j+1] = c_iA_{i+1}\ldots c_jA_{j+1}c_{j+1}$, with $A_{i+1} = A_{src}$ and $A_{j+1} = A_{snk}$. 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 $\lambda(\rho, i, j, r) = (\bar{\boldsymbol{\gamma}}_j)_r - (\bar{\boldsymbol{\gamma}}_i)_r$.

As an example, consider the execution of activity sequence ACB starting from the initial configuration in the normalized (max,+) state space shown in Fig. 6. This corresponds to run fragment $\rho =$

$$\langle s_1, \begin{bmatrix}0\\0\\0\end{bmatrix}\rangle \xrightarrow{A,5} \langle s_2, \begin{bmatrix}0\\-2\\-5\end{bmatrix}\rangle \xrightarrow{C,0} \langle s_5, \begin{bmatrix}0\\-2\\-1\end{bmatrix}\rangle \xrightarrow{B,1} \langle s_4, \begin{bmatrix}0\\0\\-2\end{bmatrix}\rangle.$$

Say that we want to compute the start-to-start latency between the resource availability times of r_2 in $\bar{\boldsymbol{\gamma}}_0$ (start of activity A) and in $\bar{\boldsymbol{\gamma}}_2$ (start of activity B). First, we compute the vectors without normalization from the state

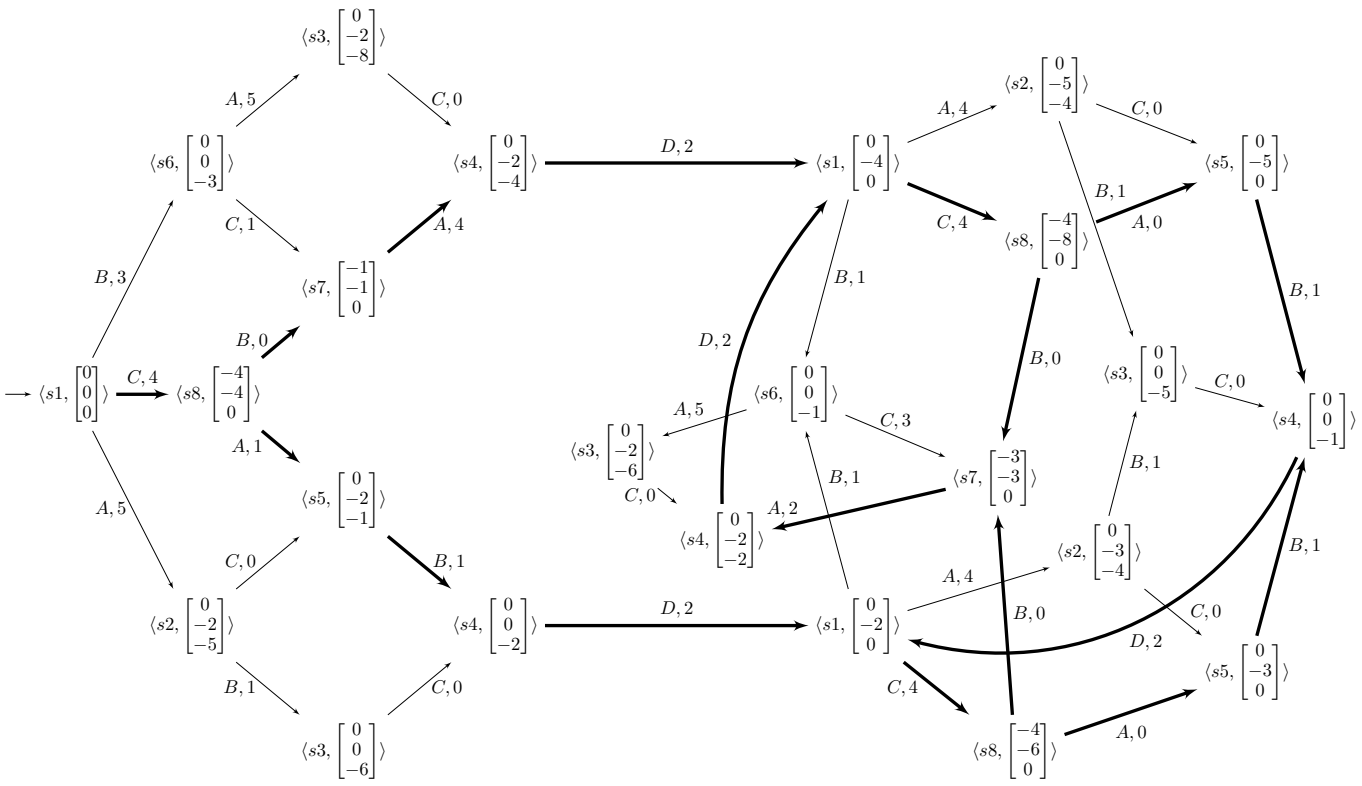


Fig. 6: Normalized (max,+) state space of $\bar{\mathcal{P}}$ (Fig. 5b). The reduction is shown with thick transitions. Transitions are annotated with the activity and w_2 value (added execution time).

space using Prop. 5 to find $\bar{\boldsymbol{\gamma}}_0 = \boldsymbol{\gamma}_0$ and $\bar{\boldsymbol{\gamma}}_2 = [5, 3, 4]^\intercal$. Then, we compute the latency:

$$\lambda(\rho, 0, 2, r_2) = (\bar{\boldsymbol{\gamma}}_2)_{r_2} - (\bar{\boldsymbol{\gamma}}_0)_{r_2} = \begin{bmatrix} 5 \\ \mathbf{3} \\ 4 \end{bmatrix}_{r_2} - \begin{bmatrix} 0 \\ \mathbf{0} \\ 0 \end{bmatrix}_{r_2} = 3.$$

We assume that the occurrences of A_{src} and A_{snk} activities are related. In any run, for any $k \geq 1$, 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 $\mathsf{getOccurrence}(\rho, A, k)$ denote 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 $A_{j+1} = A_{snk}$ in run fragment $\rho[i, j+1]$, is equal to $\lambda(\rho, i, j, r)$. The maximum start-to-start latency in a run is obtained by looking at all source-sink pairs:

$$\lambda_{max}(\rho, A_{src}, A_{snk}, r) = \sup_{k \geq 1} \lambda_k(\rho),$$

where $\lambda_k(\rho) = \lambda(\rho, i, j, r)$, $i = \mathsf{getOccurence}(\rho, A_{src}, k)$, and $j = \mathsf{getOccurence}(\rho, A_{snk}, k)$.

Definition 13 (Latency)**.** *Given normalized (max,+) state space $\mathcal{S}$, the maximum start-to-start latency of $\mathcal{S}$ with resource r and source-sink pair A_{src}, A_{snk} is found by taking the maximum latency over all runs in the state space:*

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

IV. Normalized (max,+) State Space Reduction

Performance of the supervisory controller is analyzed on the corresponding normalized (max,+) state space. In this section, we consider necessary conditions on a reduction function on state spaces to preserve throughput and latency. In the next section, we lift these conditions to the level of a (max,+) automaton and add conditions to preserve functional aspects in the reduced supervisor. The reduced (max,+) state space should be performance-equivalent to the full (max,+) state space, defined as follows.

Definition 14 (State-space performance equivalence)**.** *Normalized (max,+) state spaces $\mathcal{S}$ and $\mathcal{S}'$ are performance-equivalent, denoted $\mathcal{S} \approx_p \mathcal{S}'$ iff $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$ and $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$.*

A state space may have multiple equivalent runs with the same ratio value caused by the interleaving of *ratio-independent* activities that have no mutual influence. Such runs have the same throughput and latency values, as shown below. The first property of ratio-independent activities, say A and B, 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 summed weights w_1 and w_2 of the corresponding transitions of A and B are the same. The third property requires that A and B do not share resources. As an example, consider the initial configuration in Fig. 6, where activities A and C are ratio-independent.

Definition 15 (Ratio-independent)**.** *Let $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct,$*

$\Delta, M, w_1, w_2\rangle$ *be a normalized (max,+) state space,* $c \in C$ *a configuration, and* $A, B \in enabled(c)$ *activities enabled in* c*. Activities* A *and* B *are* ratio-independent *in* c *iff they satisfy the following conditions*[2]*:*

1) $B \in enabled(A(c))$, $A \in enabled(B(c))$, *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 *in configuration* c *iff they are not ratio-independent in* c*.*

To illustrate, consider configuration $c_0 = \langle s1, \mathbf{0}\rangle$ in the state space shown in Fig. 6. Activities A and B are ratio-dependent in c_0 (they do not satisfy condition 1 in Def. 15) and ratio-independent with activity C.

To formalize the equivalence on runs, we first define the equivalence of run prefixes. Two run prefixes are equivalent iff their corresponding activity sequences can be obtained from each other by repeatedly swapping adjacent ratio-independent activities. Given prefix $\rho[..m] = c_0 A_1 \dots A_m c_m$ of some run ρ, let $Act(\rho[..m])$ denote the activity sequence $A_1 \dots A_m$.

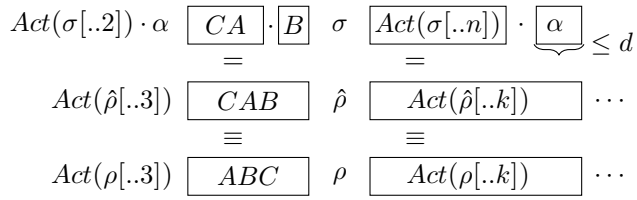
Definition 16 (String equivalence)**.** *Let* $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$ *be a normalized (max,+) state space. Strings* $\alpha, \beta \in \mathcal{A}ct^*$ *are equivalent [23] in a configuration* c*, denoted* $\alpha \equiv_c \beta$*, iff there exists a list of strings* $\upsilon_0, \upsilon_1, \dots, \upsilon_n$*, where* $\upsilon_0 = \alpha$*,* $\upsilon_n = \beta$*, and for each* $0 \le i < n$*,* $\upsilon_i = \bar{\upsilon}AB\hat{\upsilon}$ *and* $\upsilon_{i+1} = \bar{\upsilon}BA\hat{\upsilon}$ *for some* $\bar{\upsilon}, \hat{\upsilon} \in \mathcal{A}ct^*$ *and* $A, B \in \mathcal{A}ct$ *such that* A *and* B *are ratio-independent in* $\bar{\upsilon}(c)$*.*

Definition 17 (Prefix equivalence)**.** *Prefixes* $\rho[..m]$ *and* $\sigma[..m]$ *of runs* ρ *and* σ *starting in configuration* c *are equivalent in configuration* c*, denoted* $\rho[..m] \equiv_c \sigma[..m]$*, iff* $Act(\rho[..m]) \equiv_c Act(\sigma[..m])$*.*

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 equivalent run prefixes (in the sense of Def. 17) with a bounded difference in the number of activities following those prefixes. This bounded difference ensures that the resulting weight difference can be ignored in the limit.

Definition 18 (Run equivalence)**.** *Let* ρ *and* σ *be two runs in* $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$*. We define* $\rho \succeq \sigma$ *iff there exists a* $d \in \mathbb{N}$ *such that for all* $n \ge 0$ *it holds that* $\rho \succeq_n^d \sigma$*. We define* $\rho \succeq_n^d \sigma$ *iff there exists some* $k \ge n$*, a run* $\hat{\rho} \in \mathcal{R}(\mathcal{S})$ *with run prefix* $\hat{\rho}[..k]$ *with* $Act(\hat{\rho}[..k]) \equiv_{\hat{c}} Act(\rho[..k])$ *such that* $Act(\hat{\rho}[..k]) = Act(\sigma[..n]) \cdot \alpha$ *for some activity sequence* α*, and* $k - n \le d$*. Runs* ρ *and* σ *are equivalent, denoted* $\rho \equiv \sigma$*, iff* $\rho \succeq \sigma$ *and* $\sigma \succeq \rho$*.*

Two finite runs ρ and σ are equivalent if $\rho \succeq_n^d \sigma$ for $0 \le n \le |\rho|$ and $\sigma \succeq_n^d \rho$ for $0 \le n \le |\sigma|$, where $d = max(|\sigma|, |\rho|)$. As an example, consider run ρ with activity sequence $(ABCD)^\omega$ in the full state space in Fig. 6. We want to construct an equivalent run σ in the reduced state space, say $(CABD)^\omega$. This run is equivalent because C is ratio-independent of A and B. The run should satisfy $\rho \succeq_n^d \sigma$ for some $d \in \mathbb{N}$ and for all $n \ge 0$. Consider the case for $n = 2$, shown in Fig. 7a. Then, we need to match two (n) activities of ρ to the first two activities of σ in the reduced space. The prefix consisting of the first two activities of ρ, AB, is not a prefix of σ. In σ, independent activity C is performed first. Run $\hat{\rho}$, equivalent (in the sense of Def. 17) to ρ and identical to σ for $n = 2$ activities and $d = 1$, can be constructed by moving C to the front. ρ and σ must be such that this can be done for any $n \ge 0$. The general case is shown in Fig. 7b. It is crucial for the preservation of throughput that the length k that one needs to consider in ρ to find the first n activities of σ exceeds n by maximally a finite amount d, independent of n.



(a) Example $\rho \succeq_2^1 \sigma$. (b) Illustration of $\rho \succeq_n^d \sigma$.

Fig. 7: Illustration of Def. 18.

The following two lemmas show that equivalent runs indeed have the same throughput and latency values.

Lemma 19 (Equivalent runs have the same throughput)**.** *Let* $\rho, \sigma \in \mathcal{R}(\mathcal{S})$ *be runs in* $\mathcal{S}$*. If* $\rho \equiv \sigma$*, then* $\mathit{Ratio}(\rho) = \mathit{Ratio}(\sigma)$*.*

Lemma 20 (Equivalent runs have the same latency)**.** *Let* $\rho, \sigma \in \mathcal{R}(\mathcal{S})$ *be two runs in* $\mathcal{S}$*. 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* $\lambda_{max}(\rho, A_{src}, A_{snk}, r) = \lambda_{max}(\sigma, A_{src}, A_{snk}, r)$*.*

To reduce the (max,+) state space, we introduce a reduction function and state the conditions that should be imposed on this reduction to ensure that for each run in the full (max,+) state space there exists an equivalent run in the reduced (max,+) state space. We refer to these conditions as *ample conditions* and call a reduction that satisfies the ample conditions an *ample reduction*, as in [5].

Definition 21 ((max,+) state-space reduction)**.** *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:*

1) $C' \subseteq C$, $\hat{c}' = \hat{c}$, $\mathcal{A}ct' = \mathcal{A}ct$, $\Delta' \subseteq \Delta$, $M' = M$*;*
2) *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 22 ((max,+) state-space ample reduction)**.** *Let* $\mathcal{S} = \langle C, \hat{c}, \mathcal{A}ct, \Delta, M, w_1, w_2\rangle$ *be a (max,+) state space. A*

[2]For state spaces generated from (max,+) automata, as in this paper, condition 2 follows from 1 and 3.

reduction function ample is an ample reduction if it satisfies the following conditions in each configuration c:

(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* $c_0 A_1 c_1 A_2 \dots A_m c_m$ *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* $A_i \in ample(c_0)$.

We refer to $ample(c)$ *as an* ample set.

Condition (R1) ensures that the reduction does not introduce deadlocks. Condition (R2) implies that starting from some configuration c, any activity in $ample(c)$ remains enabled as long as no activity in $ample(c)$ has been executed. To illustrate, consider configuration $c_0 = \langle s1, \mathbf{0} \rangle$ in the state space shown in Fig. 6. Activities A and B are ratio-dependent in c_0 (they do not satisfy condition 1 in Def. 15) and ratio-independent with activity C. Ample sets $\{A, B\}$ and $\{C\}$ both satisfy conditions (R1) and (R2).

An ample reduction ensures that throughput and latency are preserved in the reduced (max,+) state space.

Theorem 23 (Ample reduction preserves throughput and latency)**.** *Let* $\mathcal{S}$ *be a finite normalized (max,+) state space, and* $\mathcal{S}'$ *the reduced (max,+) state space induced by an ample reduction. Then* $\mathcal{S} \approx_p \mathcal{S}'$.

V. (max,+) Automaton Reduction

In the previous section, we defined the notion of ratio-independence and the ample conditions to preserve the performance aspects of the supervisor in the corresponding normalized (max,+) state space. In this section, we lift these conditions to the level of a (max,+) automaton, so that we do not have to explicitly compute the full (max,+) state space. We further also consider the functional aspects.

To reuse existing POR techniques, we want to remove the need to identify the marked states with special set S^m. We add a self-loop with a special *controllable* activity ω to marked states. Nonblockingness of $\mathcal{A}$ can then be replaced by the notion of ω-reachability of the resulting automaton $\mathcal{A}_\omega$. The fact that $\mathcal{A}$ is nonblocking iff $\mathcal{A}_\omega$ is ω-reachable follows directly from the definitions.

Definition 24 (ω-reachability)**.** *Let* $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T \rangle$ *be a (max,+) automaton. The* ω*-extension of* $\mathcal{A}$ *is the automaton* $\mathcal{A}_\omega = \langle S_\omega, \hat{s}, S^m, \mathcal{A}ct_\omega, reward_\omega, M_\omega, T_\omega \rangle$ *with* $\omega \notin \mathcal{A}ct$, $\mathcal{A}ct_\omega = \mathcal{A}ct \cup \{\omega\}$, $reward_\omega = reward \cup \{\langle \omega, 0 \rangle\}$, $M_\omega = M \cup \{\langle \omega, I(\omega) \rangle\}$, *where* $I(\omega)$ *is an identity matrix of size* $|Res| \times |Res|$ *with for each* $1 \leq i, j \leq |Res|$, $[I(\omega)]_{ij} = -\infty$ *if* $i \neq j$ *and* $[I(\omega)]_{ij} = 0$ *if* $i = j$, *and* $T_\omega = T \cup \{s^m \xrightarrow{\omega} s^m \mid s^m \in S^m\}$. $\mathcal{A}_\omega$ *is* ω-reachable *iff from each reachable state* $s \in S$ *(i.e.* $\hat{s} \rightarrow^* s$*) a state* s_ω *is reachable in which* ω *is enabled.*

We want to exploit redundancy in the plant to obtain a reduced plant, while preserving the properties of interest. Part of the redundancy is caused by the interleaving of activities that have no mutual influence on these properties. Such activities are referred to as *uncontrollable-independent*.

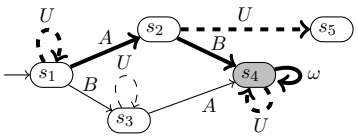


Fig. 8: Example to show the need for condition 2 of Def. 25.

Two activities A and B are uncontrollable-independent in state s if (i) they are independent in the classical sense, (ii) no uncontrollable activities are enabled in any of the states $s, A(s), B(s)$, and $AB(s)$, and (iii) they do not share resources. Condition (ii) guarantees that uncontrollable activities do not influence the execution of both activities. It ensures that both the AB and BA paths are either preserved or removed during synthesis. The need for this condition is illustrated with Fig. 8. In the example, we have paths $s_1 \xrightarrow{BA}{}^* s_4$ and $s_1 \xrightarrow{AB}{}^* s_4$ in the full automaton, and we may only have path $s_1 \xrightarrow{AB}{}^* s_4$ in the reduced automaton if the reduction considers both paths equivalent based on the independence of A and B. Synthesis on the reduced automaton yields an empty supervisor because state s_2 leads to blocking state s_5 via uncontrollable activity U. Synthesis on the full automaton yields a non-empty supervisor. Although in states s_1, s_2, s_3 and s_4, the same uncontrollable activity U is enabled, only in state s_2 it takes the system to a blocking state. Therefore, both paths cannot be considered equivalent and activities A and B cannot be considered uncontrollable-independent. Condition (iii) guarantees that the activities have no mutual influence on their timing behavior.

Definition 25 (Uncontrollable-independence)**.** *Given (max,+) automaton* $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T \rangle$ *and state* $s \in S$, *activities* $A, B \in enabled(s)$ *are* uncontrollable-independent *in* s *iff they satisfy the following conditions:*

1) $B \in enabled(A(s))$, $A \in enabled(B(s))$, *and* $AB(s) = BA(s)$;
2) $enabled(s) \cap \mathcal{A}ct_u = enabled(A(s)) \cap \mathcal{A}ct_u = enabled(B(s)) \cap \mathcal{A}ct_u = enabled(AB(s)) \cap \mathcal{A}ct_u = \emptyset$;
3) $R(A) \cap R(B) = \emptyset$.

Two activities are uncontrollable-dependent *in* s, *iff they are not uncontrollable-independent in* s.

As an example, consider state s_1 in $\mathcal{P}$ shown in Fig. 2e. Here, activities A and B are uncontrollable-dependent since they both use resources r_1 and r_2 (as shown in Fig. 3). They are uncontrollable-independent with activity C, since they do not share a resource, are independent, and do not enable an uncontrollable activity.

Uncontrollable-independence lifts the notion of ratio-independence to the level of a (max,+) automaton. If two activities A and B are uncontrollable-independent in some state in the (max,+) automaton, then they are also ratio-independent in the corresponding configurations in the underlying state space. As $R(A) \cap R(B) = \emptyset$, their corresponding (max,+) matrices commute. 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.

Lemma 26. *Given are a (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$ and a 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 uncontrollable-(in)dependent in s, then they are ratio-(in)dependent in c.*

Two paths in a (max,+) automaton are equivalent iff they can be obtained from each other by repeatedly swapping adjacent uncontrollable-independent activities.

Definition 27. *Let $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$ be a (max,+) automaton. Strings $\alpha, \beta \in \mathcal{A}ct^*$ are equivalent in a state s, denoted $\alpha \equiv_{s,u} \beta$, iff there exists a list of strings $\upsilon_0, \upsilon_1, \ldots \upsilon_n$, where $\upsilon_0 = \alpha$, $\upsilon_n = \beta$ and, for each $0 \leq i < n$, $\upsilon_i = \bar{\upsilon}AB\hat{\upsilon}$ and $\upsilon_{i+1} = \bar{\upsilon}BA\hat{\upsilon}$ for some $\bar{\upsilon}, \hat{\upsilon} \in \mathcal{A}ct^*$ and activities $A, B \in \mathcal{A}ct$ such that A and B are uncontrollable-independent in $\bar{\upsilon}(s)$. Paths $p = s \xrightarrow{\alpha}{}^* s_1$ and $p' = s \xrightarrow{\beta}{}^* s_2$ are equivalent iff $\alpha \equiv_{s,u} \beta$.*

A reduction of a (max,+) automaton is defined as follows.

Definition 28 ((max,+)-automaton reduction function)**.** *A reduction function reduce for a (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$ 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 $\mathcal{A}' = \langle S', \hat{s}, S^{m'}, \mathcal{A}ct, reward, M, T'\rangle$ that satisfies the following conditions:*

1) *$S' \subseteq S$, $S^{m'} = S^m \cap S'$, $T' \subseteq T$;*
2) *for every $s \in S'$ and $A \in reduce(s)$, $\langle s, A, A(s)\rangle \in T'$.*

A reduction on plant $\mathcal{P}$ gives a reduced plant $\mathcal{P}'$, as shown in Fig. 2e. The supervisor $\mathcal{A}'_{sup}$ synthesized for $\mathcal{P}'$ will typically not be least-restrictive with respect to $\mathcal{P}$. Therefore, we introduce a new notion of *reduced least-restrictiveness*, that defines that an equivalent path in $\mathcal{P}'$ is preserved for each path in $\mathcal{P}$ to a marked state.

Definition 29 (Reduced least-restrictiveness)**.** *Let $\mathcal{A}_1 = \langle S_1, \hat{s}, S_1^m, \mathcal{A}ct_1, reward_1, M_1, T_1\rangle$ and $\mathcal{A}_2 = \langle S_2, \hat{s}, S_2^m, \mathcal{A}ct_2, reward_2, M_2, T_2\rangle$ be two (max,+) automata such that $\mathcal{A}_1 \preceq \mathcal{A}_2$. $\mathcal{A}_1$ is reduced least-restrictive with respect to $\mathcal{A}_2$ iff for every path $\hat{s} \xrightarrow{\alpha}{}^*_{\mathcal{A}_2} s_m$ with $s_m \in S_2^m$, there exists a path $\hat{s} \xrightarrow{\beta}{}^*_{\mathcal{A}_1} s_m$ in $\mathcal{A}_1$ with $s_m \in S_1^m$ such that $\alpha \equiv_{\hat{s},u} \beta$.*

In our running example, $\mathcal{A}'_{sup}$ is reduced least-restrictive with respect to $\mathcal{A}_{sup}$, since it preserves a representative path to the marked state s_8 for all pruned paths.

We apply POR to obtain $\mathcal{P}'$ from $\mathcal{P}$. After synthesis we have supervisors $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$ and $\mathcal{A}'_{sup} = sup\mathcal{CN}(\mathcal{P}')$. Reduced supervisor $\mathcal{A}'_{sup}$ should preserve the functional aspects and performance aspects of supervisor $\mathcal{A}_{sup}$, defined by $\mathcal{A}'_{sup} \lesssim_{f,p} \mathcal{A}_{sup}$.

Definition 30. *Let $\mathcal{P}$ be a plant and $\mathcal{A}'_{sup}$ and $\mathcal{A}_{sup}$ be supervisors for $\mathcal{P}$. Let $\mathcal{S}'$ and $\mathcal{S}$ be the corresponding normalized (max,+) state spaces of $\mathcal{A}'_{sup}$ and $\mathcal{A}_{sup}$. We define $\mathcal{A}'_{sup} \lesssim_{f,p} \mathcal{A}_{sup}$ iff*

1) *if $\mathcal{A}_{sup}$ is nonblocking, then $\mathcal{A}'_{sup}$ is nonblocking,*
2) *if $\mathcal{A}_{sup}$ is controllable w.r.t. $\mathcal{P}$ then $\mathcal{A}'_{sup}$ is controllable w.r.t. $\mathcal{P}$,*
3) *$\mathcal{A}'_{sup}$ is reduced least-restrictive w.r.t. $\mathcal{A}_{sup}$, and*
4) *$\mathcal{S}' \approx_p \mathcal{S}$.*

Conditions 1-3 guarantee preservation of functional aspects. Condition 4 ensures the preservation of performance. Condition 3 does not automatically imply condition 4, as throughput is defined over infinite runs (going through simple cycles), where cycles in the state space may not have corresponding marked states in the automaton. To preserve the properties of interest, we impose the following ample conditions on a reduction of a (max,+) automaton.

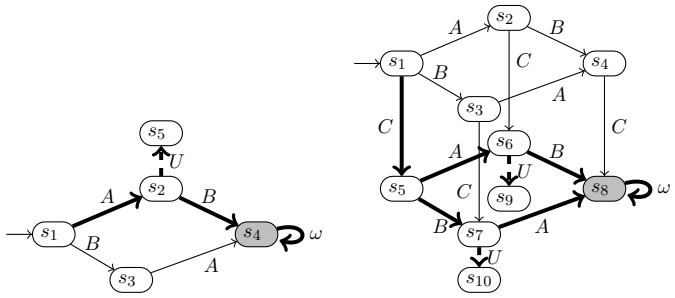
Definition 31 (Ample conditions (max,+) automaton)**.** *A reduction function ample is an ample reduction if it satisfies all the following conditions in each state s:*

- ***(A1) Non-emptyness condition:*** *if $enabled(s) \neq \emptyset$, then $ample(s) \neq \emptyset$;*
- ***(A2) Dependency condition:*** *For any path $s_0 A_1 s_1 A_2 \ldots A_m s_m$ with $s = s_0$ and $m \geq 1$ in $\mathcal{A}$, if activity A_m and some activity in $ample(s_0)$ are uncontrollable-dependent in s_0, then there is an index i with $1 \leq i \leq m$ with $A_i \in ample(s_0)$;*
- ***(A3) Controllability condition:*** *$ample(s) \supseteq enabled(s) \cap \mathcal{A}ct_u$;*
- ***(A4) Nonblockingness condition:*** *$ample(s) \supseteq enabled(s) \cap \{\omega\}$;*
- ***(A5.1) Synthesis condition 1:*** *If $A \in ample(s)$ and $enabled(A(s)) \cap \mathcal{A}ct_u \neq \emptyset$, then $ample(s) = enabled(s)$;*
- ***(A5.2) Synthesis condition 2:*** *For $A, B \in enabled(s)$ if $enabled(AB(s)) \cap \mathcal{A}ct_u \neq \emptyset$, then $A \in ample(s) \Leftrightarrow B \in ample(s)$.*

We refer to set $ample(s)$ for any state s as an ample set.

Condition (A1) ensures that deadlock-freedom is preserved. Condition (A2) ensures that starting from some state s, any activity in $ample(s)$ remains enabled as long as no activity in $ample(s)$ has been executed. Condition (A3) ensures that all uncontrollable activities in the enabled set remain in the ample set to avoid that they become disabled by the reduction. Condition (A4) ensures that in the reduced automaton marked states can still be identified. Note that ω acts like an uncontrollable activity, since it must remain in the ample set. We have chosen it to be controllable, however, as we do not want ω to have an impact on conditions (A5.1) and (A5.2), and subsequently on the reductions that can be achieved. Condition (A2) ensures that an equivalent path for any path to a marked state, where ω is enabled, is preserved in the reduced plant. Condition (A5.1) and (A5.2) ensure that if a path to a marked state is present in the supervisor, then an equivalent path is present in the reduced supervisor.

To illustrate the need for conditions (A5.1) and (A5.2), consider the plants shown in Fig. 9. Fig. 9a shows plant $\mathcal{P}_1$, where the reduction satisfies conditions (A1) till (A4) and (A5.2), but not (A5.1). Condition (A5.1) is not satisfied, since uncontrollable activity U is enabled after A, but $ample(s_1) \neq enabled(s_1)$ with $ample(s_1) = \{A\}$ and



(a) Plant $\mathcal{P}_1$ with reduced plant $\mathcal{P}_1'$ that satisfies all conditions except condition (A5.1).

(b) Plant $\mathcal{P}_2$ with reduced plant $\mathcal{P}_2'$ that satisfies all conditions except condition (A5.2).

Fig. 9: Necessity of conditions (A5.1) and (A5.2).

$enabled(s_1) = \{A, B\}$. Synthesis on $\mathcal{P}_1'$ yields an empty supervisor, whereas synthesis on $\mathcal{P}_1$ yields a supervisor with path $s \xrightarrow{BA}{}^* s_4$ to marked state s_4. If condition (A5.1) is satisfied, both A and B are in the ample set of s_1. This is needed since state s_2 becomes a bad state during synthesis, and the alternative path to s_4 is then preserved. To illustrate the need for condition (A5.2) consider plant $\mathcal{P}_2$ shown in Fig. 9b. Here, the reduction yielding $\mathcal{P}_2'$ satisfies conditions (A1) till (A5.1), but not condition (A5.2); activity $U \in enabled(BC(s_1))$, but only C is present in $ample(s_1)$, and not B. The result after synthesis on $\mathcal{P}_2$ contains the paths $s_1 \xrightarrow{A} s_2 \xrightarrow{B} s_4 \xrightarrow{C} s_8$ and $s_1 \xrightarrow{B} s_3 \xrightarrow{A} s_4 \xrightarrow{C} s_8$, whereas synthesis on the reduced plant $\mathcal{P}_2'$ yields an empty supervisor since states s_9 and s_{10} are not marked.

Using Lemma 26, it can be shown that an ample reduction at (max,+) automaton level is also an ample reduction at (max,+) state-space level, following from the fact that conditions (A1) and (A2) on the (max,+) automaton imply (R1) and (R2) in the reduction of the underlying state space. The following result then follows directly from Thm. 23.

Theorem 32. *Let $\mathcal{A}$ be a (max,+) automaton with state space $\mathcal{S}$. Let $\mathcal{S}'$ be the state space from a (max,+) automaton obtained from $\mathcal{A}$ through an ample reduction*[3]*. Then $\mathcal{S} \approx_p \mathcal{S}'$.*

Thm. 32 captures the relationship between ample reductions on state spaces and on automata. Since it does not consider the synthesis step, it cannot be used to prove the desired result that synthesis after reduction preserves both functional and performance aspects. It turns out that is does. The next theorem states this key result.

[3]In the context of [7], there is no separation between controllable and uncontrollable activities and the marking of states is ignored. Def. 31 coincides with Def. 26 in [7] when all activities are considered to be controllable and all states not marked. Then, conditions (A3), (A5.1), and (A5.2) trivially hold as $\mathcal{A}ct_u$ is empty, and (A4) holds as ω is never enabled. Similarly, the notion of uncontrollable-independent activities (Def. 25) coincides with resource-independent activities (Def. 23 in [7]), as condition 2 in Def. 25 trivially holds in case $\mathcal{A}ct_u$ is empty. So Thm. 32 carries over to the setting of [7] (coinciding with Thm. 28 in [7]).

Theorem 33 (Ample reduction preserves functionality and performance)**.** *Let $\mathcal{P}$ be a plant, and $\mathcal{P}'$ the reduced plant obtained from an ample reduction. Then $sup\mathcal{CN}(\mathcal{P}') \lesssim_{f,p} sup\mathcal{CN}(\mathcal{P})$.*

VI. On-the-fly reduction

Section V gives sufficient conditions for a reduction function on a single (max,+) automaton to preserve functional and performance properties. For an efficient reduction, we avoid 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. Given (max,+) timed system $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ and $\mathcal{A} = \{\mathcal{A}_i \mid 1 \leq i \leq n\}$, the ample function selects a set $ample(s)$ in each state s of the composition, such that conditions (A1) till (A5) are met. The ample set is induced by a *cluster* $\mathcal{C} \subseteq \mathcal{A}$, and computed as $ample(s) = enabled_{\mathcal{C}}(s) = enabled(s) \cap \mathcal{A}ct(\mathcal{C})$, where $\mathcal{A}ct(\mathcal{C}) = \bigcup_{\mathcal{A}_i \in \mathcal{C}} \mathcal{A}ct_i$ and $\mathcal{A}ct_i$ is the alphabet of (max,+) automaton $\mathcal{A}_i$. This cluster-inspired ample approach, based on [24], is a generalization of the traditional on-the-fly method of Peled [5] that selects the enabled activities $enabled_i(s) = enabled(s) \cap \mathcal{A}ct_i$ of one (max,+) automaton $\mathcal{A}_i$ as ample set if possible, while exploring a state $s = \langle s_1, s_2, \ldots, s_n \rangle$. If this is not possible, all enabled activities in s are selected as ample set.

A cluster that ensures that the ample conditions are met is called a *safe cluster*. To consider the local state $\pi_{\mathcal{C}}(s)$ in a cluster $\mathcal{C} \subseteq \mathcal{A}$, we define a projection π:

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

Given local state $\pi_{\mathcal{C}}(s)$, $enabled(\pi_{\mathcal{C}}(s))$ denotes the set of enabled activities in the composition of the (max,+) 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 $\mathcal{A}_{c_1} \parallel \ldots \parallel \mathcal{A}_{c_k}$ for cluster $\mathcal{C} = \{\mathcal{A}_{c_1}, ..., \mathcal{A}_{c_k}\}$, but disabled in the global composition due to a (max,+) automaton outside the cluster disabling the activity. We only consider independence of activities across (max,+) automata, and not within the same (max,+) automaton. The former can be checked locally, whereas the latter requires an exploration on the internal transition structure. We treat activities inside the same (max,+) automaton as dependent.

Definition 34 (Cluster safety)**.** *Let $\mathcal{C} \subseteq \mathcal{A}$ be any cluster, and s be a state in the composition of $\mathcal{A}$. 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 $B \notin \mathcal{A}ct(\mathcal{C})$, A and B are uncontrollable-independent;*
- ***(C2.2)*** *for any $A \in enabled(\pi_{\mathcal{C}}(s)) \cap \mathcal{A}ct_i$, $\mathcal{A}_i \in \mathcal{C}$;*
- ***(C3)*** *$enabled_{\mathcal{C}}(s) \supseteq enabled(s) \cap \mathcal{A}ct_u$;*
- ***(C4)*** *$enabled_{\mathcal{C}}(s) \supseteq enabled(s) \cap \{\omega\}$;*
- ***(C5.1)*** *if $A \in enabled_{\mathcal{C}}(s)$ and $enabled(A(s)) \cap \mathcal{A}ct_u \neq \emptyset$, then $enabled_{\mathcal{C}}(s) = enabled(s)$;*

(C5.2) *for $A, B \in enabled(s)$ if $enabled(AB(s)) \cap \mathcal{A}ct_u \neq \emptyset$, then $A \in enabled_{\mathcal{C}}(s) \Leftrightarrow B \in enabled_{\mathcal{C}}(s)$.*

Condition (C2.2) requires that each activity in $enabled(\pi_{\mathcal{C}}(s))$ does not occur outside of the cluster. Together with (C2.1), this ensures that no activity $A \notin enabled_{\mathcal{C}}(s)$, dependent on some activity in $enabled_{\mathcal{C}}(s)$, becomes enabled by executing only activities outside the cluster. Condition (C3) ensures that uncontrollable activities are always preserved. Condition (C4) ensures that ω, if enabled, is preserved. Conditions (C5.1) and (C5.2) ensure that if a path to a marked state remains after synthesis on the full composition, then also an equivalent path remains after synthesis on the reduced composition. Note that $\mathcal{C} = \mathcal{A}$ is a safe cluster in any state. We define a cluster-inspired ample reduction based on cluster safety on a (max,+) timed system.

Definition 35 (Cluster-inspired ample reduction)**.** *A cluster-inspired ample reduction ample for a (max,+) 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 $ample(s)$ for all states $s \in S$ satisfies the following condition:*

(M1) *$ample(s) = enabled_{\mathcal{C}}(s)$ where $\mathcal{C} \subseteq \mathcal{A}$ is safe in s.*

Theorem 36. *Let $\mathcal{P} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ be a plant modeled as (max,+) timed system, and $\mathcal{P}'$ be the (max,+) automaton obtained by a cluster-inspired ample reduction. Then, $sup\mathcal{CN}(\mathcal{P}') \lesssim_{f,p} sup\mathcal{CN}(\mathcal{P})$.*

In each composition state s, we compute a safe cluster starting from a candidate activity. To guarantee conditions (C5.1) and (C5.2), we check whether an activity A might enable an uncontrollable activity in $enabled(A(s))$ or $enabled(AB(s))$ for some uncontrollable-independent activity B. To avoid computing these enabled sets during the on-the-fly reduction, we introduce a new set $\mathcal{U}$ that can be generated a priori from the network of (max,+) automata. $\mathcal{U}$ contains all activities that enable (or do not disable) an uncontrollable activity in any of the automata.

$$\mathcal{U} = \bigcup_{\mathcal{A}_i \in \mathcal{A}} \{A \in \mathcal{A}ct_i \mid enabled(A(s)) \cap \mathcal{A}ct_u \neq \emptyset, s \in S_i\}.$$

The reduction is most effective if this set $\mathcal{U}$ is small, and not effective if it contains all activities. After executing some activity $A \in \mathcal{U}$, an uncontrollable activity U might still be disabled by some other (max,+) automaton, even though A enables it locally, so $\mathcal{C}$ is conservative.

A safe cluster in a state s can be computed with Alg. 1. In the algorithm, $[\mathcal{A}_i \mid B \in Act_i]$ denotes a list comprehension that creates a list of all elements $\mathcal{A}_i$ for which $B \in Act_i$. Function $first()$ picks the first element. With $A \leftarrow \perp$, we denote that no activity is assigned to A.

The algorithm first ensures that conditions (C3) and (C4) are met (lines 2-6). If $enabled(s)$ contains ω, then set $\mathcal{A}$ is returned, since all (max,+) automata have ω in their alphabet and will be added to the cluster. If not, then for each enabled uncontrollable activity, all (max,+) automata having this activity in the alphabet are added to the cluster (lines 5-6). The algorithm then checks for each activity enabled in the current cluster $\mathcal{C}$ whether condition (C2.1) or (C2.2) is violated, ensuring that also (C5.1) and (C5.2) are satisfied. The algorithm starts with *candidate* activity A. If A is enabled in the composition (line 9), we check if $A \in \mathcal{U}$, as A might then enable an uncontrollable activity and condition (C5.1) or (C5.2) might get violated. When $A \in \mathcal{U}$ and A is enabled within the current cluster, the set of all automata is returned (line 11). Otherwise, we add all (max,+) automata containing A (line 12) and add a (max,+) automaton for each dependent activity outside the current cluster (lines 13-15). This ensures that condition (C2.1) is satisfied for activity A and the cluster obtained after executing lines 9-15. If A is enabled in the composition of (max,+) automata in the cluster, but not in the full composition, then we add a (max,+) 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 16-19. After handling A, we check whether there are other unprocessed activities that are locally enabled in the new cluster (line 20-23).

Algorithm 1 Algorithm to compute a safe cluster.

```
1:  proc ComputeCluster(s, candidate)
2:    A ← candidate; C ← ∅; processed ← ∅
3:    if ω ∈ enabled(s) then
4:      return A
5:    for U ∈ enabled(s) ∩ Act_u do
6:      C ← C ∪ {A_i | U ∈ Act_i}
7:    while A ≠ ⊥ do
8:      processed ← processed ∪ {A}
9:      if A ∈ enabled(s) then
10:       if A ∈ U then
11:         return A
12:       C ← C ∪ {A_i | A ∈ Act_i}
13:       for B ∈ {D ∈ Act | R(D) ∩ R(A) ≠ ∅} do
14:         if B ∉ Act(C) then
15:           C ← C ∪ {[A_i | B ∈ Act_i].first()}
16:     if A ∉ enabled(s) ∧ A ∈ enabled(π_C(s)) then
17:       for A_i ∈ A do
18:         if A ∈ Act_i ∧ A_i ∉ C ∧ A ∉ enabled(s_i) then
19:           C ← C ∪ {A_i} ; break
20:     if processed ≠ enabled(π_C(s)) then
21:       A ← [enabled(π_C(s)) \ processed].first()
22:     else
23:       A ← ⊥
24:   return C
```

Algorithm 2 Algorithm to select a candidate activity.

```
1:  proc SelectCandidate(s, A)
2:    if ω ∈ enabled(s) then
3:      return ω
4:    else if Act_u ∩ enabled(s) ≠ ∅ then
5:      return [Act_u ∩ enabled(s)].first()
6:    else
7:      return arg min_{A∈enabled(s)} GetWeight(A, s, A)

1:  proc GetWeight(A, s, A)
2:    w ← 0
3:    for A_i ∈ A do
4:      if A ∈ Act_i then
5:        for B ∈ enabled(s_i) do
6:          if B ∈ enabled(s) then
7:            w ← w + |A| · |Act|
8:          else
9:            w ← w + 1
10:   return w
```

The algorithm continues until all locally enabled activities are processed.

From the reasoning above, it is clear that defining $ample(s) = enabled_{\mathcal{C}}(s)$ for all states s encountered in the on-the-fly reduction and safe cluster C computed in s satisfies conditions (C1) through (C5.2). This gives the following result.

Theorem 37. *Let s be a state in the composition $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ and $A \in enabled(s)$ be the candidate activity. Then,* ComputeCluster(s, A) *returns a safe cluster in s.*

In each composition state, there are typically multiple valid safe clusters. Heuristics can be used to select a cluster that likely yields a large reduction. One approach is to select a safe cluster yielding the smallest ample set starting from each candidate activity in the enabled set. This heuristic often performs well [25], since it allows to prune most enabled transitions. A disadvantage is that a safe cluster is constructed starting from each candidate. For larger enabled sets, this leads to a significant overhead.

Alg. 2 is an alternative heuristic that selects only one candidate activity. First, it checks if $enabled(s)$ contains ω or an uncontrollable activity. As they will always be in the ample set, they are a good candidate choice. Otherwise, a weight is computed for each candidate based on two aspects, and a candidate with the minimum weight is selected. The first aspect considers whether an activity is selected that does not occur in (max,+) automata that have locally enabled activities that are also in $enabled(s)$, since this implies that the ample set will increase. We add weight $|\mathcal{A}| \cdot |\mathcal{A}ct|$ as this is the maximum total sum of locally enabled activities. The second aspect considers minimizing the number of other enabled activities within the cluster, since possibly (max,+) automata need to be added where these are not enabled. For each other locally enabled activity, we increase the weight by one.

VII. Experimental evaluation

To test the effectiveness of the on-the-fly reduction, we use a set of models without data variables available from Supremica [26], the ASML lithography scanner model described in [27], and four variants of the Twilight system [27]. Twilight is an imaginary manufacturing system with two processing stations (conditioning, drilling) that processes balls according to a given recipe. This system is a simplification of the ASML lithography scanner using similar types of resources. The first variant (TW1) is described in [10]. Here, 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 automata that ensure that products are always moved forward in the production process. TW3 extends TW2 with a polish station, where each product undergoes a polish and drill step after the condition step but in arbitrary order. To analyze the scalability of synthesis with POR, we also used a variant of TW3, TW3-10s, with 10 processing stations. In TW4, we fix the order so that a product is always first conditioned, then drilled, and then polished.

We use two heuristics to compute ample sets; *AllCandidates* (Alg. 1) that tries all candidate activities to find the smallest ample set, and *SmartCandidate* (Alg. 2) that selects one candidate activity. All experiments were performed with a 2.40GHz Intel i5-6300U CPU processor and with 4GB Java heap space to run the algorithms.

We evaluate the achievable reductions while preserving only functional aspects, as well as preserving both functional and performance aspects. As Supremica models do not describe the resource usage, we assume that activities do not claim or release resources and are assigned the 0×0 (max,+) timing matrix. This implies that they are timeless, and that we effectively preserve only functional aspects. For the Twilight and ASML models, we have the timing matrices and also consider performance aspects.

a) Reduction preserving only functional aspects: Before applying POR, we compute set $\mathcal{U}$ to check whether a reduction is possible. For some models in our test set, $\mathcal{U}$ contains all activities and no reductions are possible: DosingTank, MachineBufferMachine, TankProcess, AutomaticCarParkGate, TransferLine, and TransferLine3. For the Twilight models and the ASML model, we disregard the performance aspects by assigning the 0×0 (max,+) timing matrix to activities. Table I shows the POR results where reductions are possible. The highest reductions are achieved in VolvoCell and RobotAssemblyCell, where all activities are controllable and plants describe local parts of the system. This leaves a lot of redundant interleaving of activities that can be exploited to obtain a smaller composition. A similar reasoning applies to TW2, TW3, and TW4. The reduction for TW1 is very small, as there is a lot of activity synchronization by the product-location and life-cycle automata. Recall condition (C2.2), requiring 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 TW1, the algorithm often needs to add product-location or life-cycle automata to the cluster to satisfy this condition (C2.2), which limits reduction possibilities. The reductions for TW2, TW3, and TW4 are much larger, since we do not explicitly model the product-location and life-cycle automata. For TW3-10, we had to compute the full state space on a server with much more heap space. Therefore, we have no running times on the same hardware, indicated by an X, and cannot compare the running times. In the ASML model, a large reduction of 86.1% is achieved, since all activities in this model are controllable and requirements are local.

As expected, *AllCandidates* yields similar or better reductions than *SmartCandidate* in terms of states and transitions remaining in the composition by selecting the smallest safe cluster in each state. However, there is a significant runtime overhead in computing the reduced composition (T_P) with *AllCandidates* due to the computations involved in Alg. 1. This overhead is much lower for *SmartCandidate*, where Alg. 1 is run only once in each state. The additional runtime induced by the computations in *SmartCandidate* is in most cases a factor of 2. We also considered the total time T_S needed to apply synthesis.

	full composition				*AllCandidates* heuristic				*SmartCandidate* heuristic			
	$\|S\|$	$\|T\|$	T_P (ms)	T_S (ms)	% of $\|S\|$	% of $\|T\|$	% of T_P	% of T_S	% of $\|S\|$	% of $\|T\|$	% of T_P	% of T_S
CircularTable	442	1357	52.5	31.6	72.9%	76.5%	155%	376%	72.9%	76.5%	84%	249%
IntertwinedProductCycles	65280	277441	6229.9	4746.6	0.0%	0.8%	216%	387%	0.0%	0.8%	131%	283%
RobotAssemblyCell	4741	21107	2071.2	660.6	93.4%	98.0%	10%	35%	90.6%	97.0%	3%	14%
VolvoCell	8871	29230	3919.0	518.4	83.9%	93.6%	169%	1287%	52.6%	73.8%	125%	974%
WeldingRobots	198	544	16.3	35.6	0.0%	7.0%	198%	194%	0.0%	5.7%	179%	180%
VelocityBalancing	372	775	22.0	40.8	9.1%	20.5%	155%	171%	9.1%	20.5%	131%	165%
TW1	1280	2171	322.0	61.0	0.5%	0.6%	1661%	9193%	0.5%	0.6%	819%	4440%
TW2	63	132	1.6	13.1	34.9%	50.0%	394%	148%	33.3%	47.0%	206%	122%
TW3	343	761	14.4	29.3	48.4%	67.0%	220%	162%	42.6%	59.0%	143%	130%
TW4	318	776	15.9	36.7	30.8%	47.8%	602%	327%	29.6%	42.9%	237%	174%
TW3-10s	1887381	10907298	X	X	**96.9%**	**98.9%**	X	X	**96.2%**	**98.6%**	X	X
ASML	2412	7662	174.9	394.6	86.1%	94.0%	82%	62%	65.5%	83.7%	64%	50%

TABLE I: Reductions achieved preserving functional aspects. $|S|$ is the number of states and $|T|$ is the number of transitions in the automata composition ($\mathcal{P}$ in Fig. 1). The running times to compute the composition T_P and the supervisor T_S are in milliseconds. The highest reductions are highlighted in bold.

For the heuristics, T_S includes time T_P that is needed to compute the reduced composition. For *AllCandidates*, the median additional synthesis runtime overhead is a factor of 3.3, and for *SmartCandidate* it is a factor 2.3. Again, in almost all cases, the POR algorithm gives some runtime overhead. Note that in practice the bottleneck in synthesis is not the runtime, but the memory required to store the composition. This means that for scalability, the most important metrics are the reductions that can be achieved in terms of the number of states and transitions.

b) Reduction preserving both functional and performance aspects: Table II shows the results of the reduction that preserves both functional and performance aspects. This reduction yields larger compositions, thus achieving less reduction, than the reduction preserving only functional aspects. This is as expected, since resource sharing between activities is also considered. The normalized (max,+) state spaces of the full ASML and full TW3-10s models could not be computed due to insufficient memory.

VIII. Related work

The application of POR techniques in the domain of supervisory control theory has been first investigated by Hellgren et al. [28]. There, POR is used to reduce the state space when checking deadlocks. A setting with only controllable actions is considered and the models adhere to a specific structure, with resource booking/unbooking and acyclic product life cycles where each resource can occur only once. Shaw [29] introduces an on-the-fly model checking approach for both controllability and nonblockingness. Because the aim is model checking rather than synthesis, the required conditions are different from the ones we use. For example, for checking controllability, a reduction might remove uncontrollable events from a plant model that are independent with controllable events. This is not valid if one wants to apply synthesis in a subsequent step.

There has been some initial work in applying POR techniques to timed systems. Bengtsson et al. [30] apply POR on timed automata for reachability analysis. These automata execute asynchronously, in their own local time scale, and synchronize their time scales on communication transitions. Yoneda et al. [31] investigated POR for timed Petri nets, for verification of similar timing relations. Theelen et al. [32] apply ideas from POR on Scenario-Aware Data Flow models, using an independence relation among actions to resolve non-deterministic choices that have no impact on the performance metrics.

Our POR technique can be used to obtain a smaller supervisory controller for the given plant. There are also other approaches to construct a reduced nonblocking supervisor. Dietrich et al. [33] impose three sufficient conditions on a restricted supervisor to preserve nonblockingness. Morgenstern and Schneider [34] propose a stronger notion of nonblockingness called *forceable nonblockingness*. Given a plant, a controller is forceable nonblocking if every (in)finite run of the controlled system visits a marked state. This means that a marked state *will* be reached, no matter how the plant behaves, whereas in the original setting, the plant only *has the possibility* to reach a marked state. A synthesis algorithm is provided that computes such a controller. Huang and Kumar [35] describe an approach to generate a reduced controller under the traditional notion of nonblockingness. Another approach is to find a smaller equivalent least-restrictive supervisor [36]. A supervisor typically has information about the enabling and disabling of events, and information to keep track of the plant evolution. The latter may contain redundancy. The technique exploits this redundancy to obtain a smaller supervisor. Compared to our approach, all approaches except the last one do not ensure a property of least-restrictiveness. Also, it is not straightforward to combine these reduction techniques with preserving other aspects, such as performance-related properties. Our POR achieves this by adding sufficient conditions to the reduction function and adapting the notion of event-dependence used.

Su et al. [17] describe a related approach to compute a maximally-permissive supervisor that optimizes makespan. It does not consider latency and throughput preservation. Parallelism is encoded using a mutual exclusion function. In our approach, we encode the specific resource usage, and thereby the mutual exclusion on resources, in the activities. The evaluation in the approach of [17] relies

	automata composition				*AllCandidates* heuristic				*SmartCandidate* heuristic			
	$\|S\|$	$\|T\|$	T_P (ms)	T_S (ms)	% of $\|S\|$	% of $\|T\|$	% of T_P	% of T_S	% of $\|S\|$	% of $\|T\|$	% of T_P	% of T_S
TW1	1280	2171	234.5	54.8	0.5%	0.6%	2334%	10205%	0.5%	0.6%	1177%	5249%
TW2	63	132	1.3	11.4	33.3%	46.2%	1084%	527%	31.7%	44.7%	502%	295%
TW3	343	761	11.0	31.2	27.1%	39.8%	604%	340%	21.9%	32.2%	334%	274%
TW4	318	776	23.9	27.8	29.6%	44.3%	368%	487%	27.4%	39.6%	200%	339%
ASML	2412	7662	195.6	317.5	**80.4%**	**91.4%**	122%	108%	**39.6%**	**67.2%**	118%	127%

	normalized (max,+) state space		*AllCandidates* heuristic				*SmartCandidate* heuristic			
	$\|C\|$	$\|\Delta\|$	$\|C\|$	% of $\|C\|$	$\|\Delta\|$	% of $\|\Delta\|$	$\|C\|$	% of $\|C\|$	$\|\Delta\|$	% of $\|\Delta\|$
TW1	2967	5277	2959	5261	0.3%	0.3%	2959	5261	0.3%	0.3%
TW2	282	564	200	332	29.1%	41.1%	206	344	27.0%	39.0%
TW3	3282	7136	2169	3848	33.9%	46.1%	2266	4175	31.0%	41.5%
TW4	11637	26147	8018	14907	31.1%	43.0%	8283	16068	28.8%	38.5%
ASML	X	X	69659	X	97702	X	1022464	X	1690309	X

TABLE II: Reductions achieved preserving both functional and performance aspects. The running times to compute the composition T_P and the supervisor T_S are in milliseconds. The highest reductions are highlighted in bold.

on the construction of a tree automaton, which grows exponentially in size in the worst case. In our (max,+) state space, redundancy in subsequences with the same timing information is encoded more efficiently.

Supervisory control of (max,+) automata is also considered in [13], [14]. Here, the conventional (max,+) automaton definition is used, where the timing aspects are coupled to the system states. As a result, synthesis is performed on a model including timing information. In our approach, we abstract from the timing information during synthesis, which benefits scalability of the approach.

IX. Conclusions

We presented a POR technique for a network of (max,+) automata specifying a plant and its requirements to obtain a smaller supervisor, while preserving controllability, nonblockingness, reduced least-restrictiveness, throughput, and latency. The reduction helps in synthesis and performance analysis of supervisory controllers, as less memory is needed to perform synthesis and to store the resulting reduced supervisor and timed state space. The technique is inspired by an existing cluster-based ample set reduction for non-timed systems. The reduced supervisor is computed by exploiting the structure of the automata and information about the (in)dependence among activities. The experimental evaluation shows that our POR technique is successful for models where a small set of states has uncontrollable activities enabled, and where automata describe local parts of the system. We obtained reductions up to 80.4% and 91.4% in the number of states and transitions. The possible reductions are highly dependent on the amount of synchronization on activities among automata and the extent to which activities use the same resources. In our models, the POR technique successfully exploits redundant interleaving related to processing stations that can perform operations in parallel and robot movements that can be executed simultaneously.

Acknowledgment

This research was supported by the Dutch NWO-TTW, carried out as part of the Robust Cyber-Physical Systems (RCPS) program, project number 12694, and carried out as part of the Concerto project, under the responsibility of ESI (TNO) with ASML as the carrying industrial partner.

References

[1] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] L. Ouedraogo, R. Kumar, R. Malik *et al.*, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. Aut. Sc. and Eng.*, vol. 8, no. 3, pp. 560–569, July 2011.

[3] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2010.

[4] W. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history," *Annual Reviews in Control*, 2018.

[5] 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.

[6] ——, "Ten years of partial order reduction," in *Proc. CAV*. Springer, 1998, pp. 17–28.

[7] B. van der Sanden, M. Geilen, M. Reniers *et al.*, "Partial-order reduction for performance analysis of max-plus timed systems," in *Proc. ACSD*. IEEE, 2018.

[8] H. Flordal, R. Malik, M. Fabian *et al.*, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[9] B. van der Sanden, M. Geilen, M. Reniers *et al.*, "Partial-order reduction for supervisory controller synthesis," *To appear in IEEE Trans. on Automatic Control*, 11 2021.

[10] B. van der Sanden, J. Bastos, J. Voeten *et al.*, "Compositional specification of functionality and timing of manufacturing systems," in *Proc. FDL*. IEEE, 2016, pp. 1–8.

[11] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183 – 235, 1994.

[12] S. Miremadi, Z. Fei, K. Åkesson *et al.*, "Symbolic representation and computation of timed discrete-event systems," *IEEE Trans. on Aut. Sc. and Eng.*, vol. 11, no. 1, pp. 6–19, 2014.

[13] J. Komenda, S. Lahaye, and J.-L. Boimond, "Supervisory control of (max,+) automata: A behavioral approach," *Discrete Event Dynamic Systems*, vol. 19, pp. 525–549, 2009.

[14] S. Lahaye, J. Komenda, and J.-L. Boimond, "Supervisory control of (max,+) automata: extensions towards applications," *Int. Journal of Control*, vol. 88, no. 12, pp. 2523–2537, 2015.

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

[16] F. Baccelli, G. Cohen, G. J. Olsder *et al.*, *Synchronization and linearity: an algebra for discrete event systems*. John Wiley and Sons, 1992.

[17] R. Su, J. van Schuppen, and J. Rooda, "The synthesis of time optimal supervisors by using heaps-of-pieces," *IEEE Trans. on Automatic Control*, vol. 57, no. 1, pp. 105–118, Jan 2012.

[18] S. Gaubert and J. Mairesse, "Modeling and analysis of timed petri nets using heaps of pieces," *IEEE Trans. on Automatic Control*, vol. 44, no. 4, pp. 683–697, 1999.

[19] M. Geilen and S. Stuijk, "Worst-case performance analysis of Synchronous Dataflow scenarios," in *Conf. on CODES/ISSS*, 2010, pp. 125–134.

[20] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM J. Comput*, vol. 4, no. 1, pp. 77–84, 1975.

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

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

[23] A. Mazurkiewicz, "Trace theory," in *Petri Nets: Applications and Relationships to Other Models of Concurrency*. Springer, 1987, pp. 278–324.

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

[25] J. Geldenhuys, H. Hansen, and A. Valmari, "Exploring the scope for partial order reduction," in *Proc. ATVA*. Springer, 2009, pp. 39–53.

[26] K. Åkesson, M. Fabian, H. Flordal *et al.*, "Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proc. WODES*, 2006, pp. 384–385.

[27] B. van der Sanden, "Performance analysis and optimization of supervisory controllers," Ph.D. dissertation, Eindhoven University of Technology, 2018.

[28] A. Hellgren, M. Fabian, and B. Lennartson, "Deadlock detection and controller synthesis for production systems using partial order techniques," *Proc. IEEE CCA*, vol. 2, pp. 1472–1477, 1999.

[29] A. M. Shaw, "Partial order reduction with compositional verification," Master's thesis, University of Waikato, Hamilton, New Zealand, 2014.

[30] J. Bengtsson, B. Jonsson, J. Lilius *et al.*, "Partial order reductions for timed systems," in *Proc. CONCUR*. Springer, 1998, pp. 485–500.

[31] 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.

[32] B. Theelen, M. Geilen, and J. Voeten, "Performance model checking scenario-aware dataflow," in *Proc. FORMATS*. Springer, 2011, pp. 43–59.

[33] P. Dietrich, R. Malik, W. M. Wonham *et al.*, *Synthesis and Control of Discrete Event Systems*. Springer, 2002, ch. Implementation Considerations in Supervisory Control, pp. 185–201.

[34] A. Morgenstern and K. Schneider, "Synthesizing deterministic controllers in supervisory control," in *Informatics in Control, Automation and Robotics II*. Springer, 2007, pp. 95–102.

[35] J. Huang and R. Kumar, "Directed control of discrete event systems for safety and nonblocking," *IEEE Trans. on Aut. Sc. and Eng.*, vol. 5, no. 4, pp. 620–629, Oct 2008.

[36] R. Su and W. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dynamic Systems*, vol. 14, no. 1, pp. 31–53, Jan 2004.

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

Appendix

Proofs for Section II (Modeling)

Proposition 5. *Let $\mathcal{S}$ be a normalized (max,+) state space with run $\rho = c_0 A_1 c_1 A_2 c_2 A_3 \ldots$ such that $c_i = \langle s_i, \boldsymbol{\gamma}_i \rangle$ for each i. Then, for all $n \geq 0$, $\bar{\boldsymbol{\gamma}}_n = \sum_{k=0}^{n-1} w_2(c_k, A_{k+1}, c_{k+1}) + \boldsymbol{\gamma}_n$.*

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:

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

□

Proofs for Section IV (Normalized (max,+) State Space Reduction)

Lemmas 19 and 20 show that ratio-equivalent runs have the same throughput and latency values. Intuitively, this is proven as follows. By Def. 18, for each prefix in one run we can match a (possibly) shorter prefix in the other run. The difference between the prefixes is a suffix α bounded in length by d.

For throughput preservation, observe that the maximum weight difference between the prefixes for both weights w_1 and w_2 is bounded by the maximum cumulative sum of the weights w_1 and w_2 of α. By Prop. 12, it suffices to consider the maximum cumulative weights w_1 and w_2 over all simple cycles in the state space to determine the maximum cumulative sum for α for both weights. In the limit, a bounded weight difference can be ignored. Therefore, ρ and σ have the same ratio value, and consequently also the same throughput value.

The maximum latency in a run between A_{src} and A_{snk} on resource r is determined by some source-sink occurrence pair. The prefixes of ρ and σ are matched by swapping ratio-independent activities. Given condition 3 of Def. 15, the source and sink activity can only be part of swaps where the resource availability time of r is not affected. These swaps thus do not affect the latency value for the source-sink occurrence pair, and hence ρ and σ have the same maximum latency.

Lemma 19 (Equivalent runs have the same throughput)**.** *Let $\rho, \sigma \in \mathcal{R}(\mathcal{S})$ be runs in $\mathcal{S}$. If $\rho \equiv \sigma$, then $\mathit{Ratio}(\rho) = \mathit{Ratio}(\sigma)$.*

Proof. Since $\rho \equiv \sigma$, by Def. 18, we have $\rho \succeq_n^d \sigma$ for all $n \geq 0$ and for some d. This means that there exists a $k \geq n$, and run $\hat{\rho}$ with $Act(\hat{\rho}[..k]) \equiv_{\hat{c}} Act(\rho[..k])$ such that finite prefix $Act(\hat{\rho}[..k]) = Act(\sigma[..n]) \cdot \alpha$ for some activity sequence α, and $k - n \leq d$.

The maximum difference between $w_i(\rho[..n]) \leq w_i(\hat{\rho}[..k])$ and $w_i(\sigma[..n])$ for $i \in \{1, 2\}$ and any $n \geq 0$ is bounded by the maximum cumulative w_i sum of suffix α, whose length is bounded by d. Let k_i denote the maximum cumulative

w_i sum of α. By Prop. 12, it suffices to look at the maximum cumulative w_i sum over all simple cycles in the graph to obtain k_i. By symmetry of ρ and σ, $w_i(\rho[..n]) \leq w_i(\sigma[..n]) + k_i$ for some $k_i \geq 0$. Then, it follows that $w_i(\rho[..n]) - k_i \leq w_i(\sigma[..n]) \leq w_i(\rho[..n]) + k_i$.

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:

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

□

Lemma 20 (Equivalent runs have the same latency)**.** *Let $\rho, \sigma \in \mathcal{R}(\mathcal{S})$ be two runs in $\mathcal{S}$. 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 $\rho \equiv \sigma$, $\rho \succeq_l^c \sigma$ for all $l \geq 0$ and some c. This means that there exists some $d \geq l$ and run prefixes $\rho[..d]$ and $\hat{\rho}[..d]$ with $Act(\hat{\rho}[..d]) \equiv_{\hat{c}} Act(\rho[..d])$ such that $Act(\hat{\rho}[..d]) = Act(\sigma[..l]) \cdot \alpha$ for some activity sequence α and $d - l \leq c$. Prefix $\hat{\rho}[..d]$ is obtained from $\rho[..d]$ by repeatedly swapping 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. Let $\hat{\rho}' = \hat{c} \to^* c_s \xrightarrow{AB}{}^* c(AB) \to^* c_d$ and $\hat{\rho}'' = \hat{c} \to^* c_s \xrightarrow{BA}{}^* c(BA) \to^* c_d$.

First, consider the case where both A and B are different from A_{src} and A_{snk}. Let $\bar{\gamma}_d$ be the resource availability vector of interest corresponding to configuration c_d without normalization. Since A, B are ratio-independent, their matrices commute, and therefore the value of $\bar{\gamma}_d$ remains the same:

$$\begin{aligned}
\bar{\gamma}_d &= \bigotimes_{t=1}^{d} M(A_t) \otimes \mathbf{0} \\
&= \bigotimes_{t=s+2}^{d} M(A_t) \otimes M(B) \otimes M(A) \otimes \bigotimes_{t=1}^{s-1} M(A_t) \otimes \mathbf{0} \\
&= \bigotimes_{t=s+2}^{d} M(A_t) \otimes M(A) \otimes M(B) \otimes \bigotimes_{t=1}^{s-1} M(A_t) \otimes \mathbf{0}.
\end{aligned}$$

Now assume that $A_{src} = A$ in $\hat{\rho}'$. Let $\bar{\gamma}_s$ denote the resource availability vector in configuration c_s without normalization. Since A_{src} and B are ratio-independent, resource r is not used by B. This means that $\bar{\gamma}_{s+1}$ for configuration $c_{s+1} = c_s(B)$ satisfies $(\bar{\gamma}_s)_r = (\bar{\gamma}_{s+1})_r$. This also means that the resource availability time for r is not affected by the swap. A similar reasoning can be used for the case where A_{snk} and B are swapped. Since the resource availability vectors corresponding to the start of the k-th occurrences of A_{src} and A_{snk} are the same for resource r, we conclude that $\lambda_k(\rho) = \lambda_k(\sigma)$ for any $k \geq 0$. It follows that $\lambda_{max}(\rho, A_{src}, A_{snk}, r) = \lambda_{max}(\sigma, A_{src}, A_{snk}, r)$. □

Thm. 23 captures the main result of this section, namely that an ample state-space reduction preserves performance properties. Its proof intuitively goes as follows. If state-space $\mathcal{S}$ is reduced to $\mathcal{S}'$ through an ample reduction, then $\mathcal{S} \approx_p \mathcal{S}'$, because their runs are equivalent. If a run $\rho \in \mathcal{R}(\mathcal{S})$ is not present in $\mathcal{S}'$, then we can construct an equivalent run $\rho' \in \mathcal{R}(\mathcal{S}')$ such that $\rho \equiv \rho'$. We construct ρ' in a recursive manner by modifying ρ into equivalent runs, ρ^n, with ever-longer prefixes in $\mathcal{S}'$. ρ' itself is the limit of that process and is entirely in $\mathcal{S}'$. Whenever from some configuration $c_i = \rho^n[i]$ for some i an activity $A \in enabled_{\mathcal{S}}(c_i)$ is followed next, while $A \notin enabled_{\mathcal{S}'}(c_i)$, then, because of the ample conditions, we can swap an activity that is ratio-independent with A to the front that is in $enabled_{\mathcal{S}'}(c_i)$. Such an activity can be found within $|C|$ swaps (Lemma 38 below). Due to the swapping, other activities can move further back. Again, using Lemma 38, these activities move at most $|C|$ positions back. Because the construction of ρ' is such that activities move backward or forward compared to ρ by at most $|C|$ places, one can show that for all n, $\rho \preceq_n^{|C|} \rho'$ and $\rho' \preceq_n^{|C|} \rho$. Therefore, $\rho \equiv \rho'$ and they have the same throughput and latency according to Lemmas 19 and 20.

Lemma 38 (adapted from [37, Lemma 8.15])**.** *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 let $\rho[i, j] = c_i B_i c_{i+1} \dots B_j c_{j+1}$ be a run fragment in $\mathcal{S}$. Let ample be an ample reduction on $\mathcal{S}$. Let $A \in ample(c_i)$ and $\{B_i, \dots, B_j\} \subseteq enabled(c_i)$. If $\{B_i, \dots, B_j\} \cap ample(c_i) = \emptyset$, then (i) $A \in enabled(c_k)$ for $i \leq k \leq j$, (ii) activities A and B_k for all $i \leq k \leq j$ are ratio-independent in c_i, and (iii) $j - i + 1 \leq |C|$.*

Proof. The proof of the first two properties (i) and (ii) follows the same structure as the proof of [37, Lemma 8.15].

To prove (iii) that the length $j-i+1$ of the fragment is at most $|C|$, let $Res' \subseteq Res$ denote the set of resources used by activities in $ample(c_i)$, i.e., $Res' = \bigcup_{A \in ample(c_i)} R(A)$. Since the normalized (max,+) state space is finite, the claim of each resource is linked to the claim of each other resource via the resource dependencies over each cycle, as described in the text after Def. 4. A fragment of length $|C|$ comprises a cycle in the state space, and on this cycle there is therefore at least one activity that uses one of the resources in Res' that is also used by some activity $A \in ample(c_i)$. By Def. 15, this activity is ratio-dependent with A in c_i. □

Theorem 23 (Ample reduction preserves throughput and latency)**.** *Let $\mathcal{S}$ be a finite normalized (max,+) state space, and $\mathcal{S}'$ the reduced (max,+) state space induced by an ample reduction. Then $\mathcal{S} \approx_p \mathcal{S}'$.*

Proof. 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 a reduced (max,+) state space induced by an ample reduction function. To conclude that $\mathcal{S} \approx_p \mathcal{S}'$, by Def. 14, we need to show that $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$ and $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$. We show that for each run $\rho \in \mathcal{R}(\mathcal{S})$, there exists a run $\rho' \in \mathcal{R}(\mathcal{S}')$ with $\rho \equiv \rho'$. By Lemmas 19 and 20, these runs have the same throughput and latency. By Def. 11 and Def. 13 it then follows that $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$ and $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$.

Let $\rho \in \mathcal{R}(\mathcal{S})$ be some run in $\mathcal{S}$. Now, we need to show that there exists an equivalent run in $\mathcal{R}(\mathcal{S}')$, i.e., a run $\rho' \in \mathcal{R}(\mathcal{S}')$ such that $\rho \succeq \rho'$ and $\rho' \succeq \rho$. Note that the reverse direction is trivial since $\mathcal{R}(\mathcal{S}') \subseteq \mathcal{R}(\mathcal{S})$. To define such a run ρ', we first define a sequence σ^n of finite prefixes of the string $Act(\rho')$ recursively as follows:

$$\sigma^0 = \varepsilon$$
$$\sigma^{n+1} = \sigma^n \cdot \alpha^n(k), \text{where } \alpha^n \text{ and } k \text{ are as described below.}$$

We define fragments of activity sequences in the same way as run fragments. Given activity sequence $\alpha = \alpha(0)\alpha(1)\ldots$, let $\alpha[..k]$ denote $\alpha(0)\ldots\alpha(k)$ and $\alpha[k..]$ denote $\alpha(k)\alpha(k+1)\ldots$. Run fragment α^n in the definition of σ^n is then such that $\sigma^n \cdot \alpha^n \equiv_{\hat{c}} Act(\rho[..n+|C|])$ and $k \in \mathbb{N}$ in that definition is the smallest number such that $\alpha^n(k) \in ample(\sigma^n(\hat{c}))$.

We prove that such α^n and k exist by induction.

- For $n = 0$, $\alpha^0 = Act(\rho[..|C|])$. By Lemma 38, there is some $k \leq |C|$ such that $\alpha^0(k) \in ample(\sigma^0(\hat{c})) = ample(\hat{c})$.
- For $n \geq 0$, using the induction hypothesis, let $\sigma^n \cdot \alpha^n \equiv_{\hat{c}} Act(\rho[..n+|C|])$. $\sigma^{n+1} = \sigma^n \cdot \alpha^n(k)$, with $\alpha^n(k) \in ample(\sigma^n(\hat{c}))$, and for all $m < k$, $\alpha^n(m) \notin ample(\sigma^n(\hat{c}))$. Hence, according to Lemma 38, all $\alpha^n(m)$ for $m < k$ are ratio-independent of $\alpha^n(k)$ and $\alpha^n \equiv_c \alpha^n(k) \cdot \alpha^n[..k-1] \cdot \alpha^n[k+1..]$ for $c = \sigma^n(\hat{c})$. Therefore, $\sigma^{n+1} \cdot \alpha^n[..k-1] \cdot \alpha^n[k+1..] \cdot Act(\rho[n+1+|C|]) = \sigma^n \cdot \alpha^n(k) \cdot \alpha^n[..k-1] \cdot \alpha^n[k+1..] \cdot Act(\rho[n+1+|C|]) \equiv_{\hat{c}} \sigma^n \cdot \alpha^n \cdot Act(\rho[n+1+|C|]) \equiv_{\hat{c}} Act(\rho[..n+|C|]) \cdot Act(\rho[n+1+|C|]) = Act(\rho[..n+1+|C|])$. This gives us an α^{n+1}, namely, $\alpha^{n+1} = \alpha^n[..k-1] \cdot \alpha^n[k+1..] \cdot Act(\rho[n+1+|C|])$, such that $\sigma^{n+1} \cdot \alpha^{n+1} \equiv_{\hat{c}} Act(\rho[..n+1+|C|])$. And, again, Lemma 38 tells us that there is some $k \leq |C|$ such that $\alpha^{n+1}(k) \in ample(\sigma^{n+1}(\hat{c}))$.

Let σ^n and α^n for each $n \geq 0$ be constructed using this procedure, then run ρ^n, the run with activity sequence $\sigma^n \cdot \alpha^n \cdot Act(\rho[n+|C|\,..])$, is equivalent to ρ. Let ρ' be the run corresponding to the infinite string $\bigsqcup_{n \geq 0} \sigma^n$ being the supremum of the sequence σ^n. To prove that $\rho \succeq \rho'$, from the definition of σ^n, it follows immediately that $\rho \succeq_n^{|C|} \rho'$ for all $n \geq 0$ and thus that $\rho \succeq \rho'$.

Conversely, we need to show that $\rho' \succeq \rho$. We show that for all $n \in \mathbb{N}$, $\rho' \succeq_n^{|C|} \rho$. Hence, we need to show that there is some β such that $\sigma^{n+|C|} \equiv_{\hat{c}} Act(\rho[..n]) \cdot \beta$. With the following procedure we construct such a β. Intuitively, β consists of all the activities that occur in $\sigma^{n+|C|}$ that do not occur in $Act(\rho[..n])$. From the construction of ρ', we know that activities from run ρ move forward by at most $|C|$ places when constructing ρ', so activities in $\sigma^{n+|C|}$ may have come from the run fragment up to $\rho[..n+2|C|]$, but not further in ρ. Therefore, we know that the activities for β occur in the run fragment $\rho[n..n+2|C|]$. β is of length $|C|$, so there are also $|C|$ activities in $\rho[n..n+2|C|]$ that we don't need for β. When, in the following, we talk about an activity of a run, and a corresponding activity in another run, this is short-hand for the following. The n-th occurrence of an activity in the former run corresponds to the n-th occurrence of the same activity in the latter run. Let c be the configuration reached at the end of $\rho[..n]$.

We maintain the invariant that $Act(\rho[n..n+2|C|]) \equiv_c \beta_k \cdot \alpha_k \cdot Act(\rho[N_k..n+2|C|])$ for some activity sequences α_k and β_k and index N_k. Let $N_0 = n$, $\alpha_0 = \beta_0 = \epsilon$. Then, the invariant trivially holds. While $N_k < n+2|C|$ we repeat the following procedure and prove the invariant is preserved.

- If $A = Act(\rho)(N_k)$ is an activity in $\rho'[n+1..n+|C|]$, then we add it to β_k: $\beta_{k+1} = \beta_k \cdot A$, $N_{k+1} = N_k + 1$, $\alpha_{k+1} = \alpha_k$. Then, $\beta_{k+1} \cdot \alpha_{k+1} \cdot \rho[N_{k+1}..n+2|C|] = \beta_k \cdot A \cdot \alpha_k \cdot Act(\rho[N_k+1..n+2|C|]) \equiv_c \beta_k \cdot \alpha_k \cdot A \cdot Act(\rho[N_k+1..n+2|C|]) = \beta_k \cdot \alpha_k \cdot Act(\rho[N_k..n+2|C|]) \equiv_c Act(\rho[n..n+2|C|])$. Note that $A \cdot \alpha_k \equiv_c \alpha_k \cdot A$, because all activities in α_k correspond to $Act(\rho)(m)$ for some $m < N_k$, i.e., in the construction of ρ', they have been reordered and hence, they are ratio-independent.
- Else, if $A = Act(\rho)(N_k)$ is not an activity in $\rho'[n+1..n+|C|]$, we add it to α_k, $\alpha_{k+1} = \alpha_k \cdot A$, $\beta_{k+1} = \beta_k$, $N_{k+1} = N_k + 1$. Then, $\beta_{k+1} \cdot \alpha_{k+1} \cdot Act(\rho[N_{k+1}..n+2|C|]) = \beta_k \cdot \alpha_k \cdot A \cdot Act(\rho[N_k+1..n+2|C|]) = \beta_k \cdot \alpha_k \cdot Act(\rho[N_k..n+2|C|]) \equiv_c Act(\rho[n..n+2|C|])$.

Eventually, $N_k = n + 2|C|$ and $Act(\rho[n..n+2|C|]) \equiv_c \beta_k \cdot \alpha_k$. Thus, $Act(\rho[..n+2|C|]) \equiv_{\hat{c}} Act(\rho[..n]) \cdot \beta_k \cdot \alpha_k$. From the construction of $\sigma^{n+|C|}$, we have that $\sigma^{n+|C|} \cdot \alpha_k \equiv_{\hat{c}} Act(\rho[..n+2|C|]) \equiv_{\hat{c}} Act(\rho[..n]) \cdot \beta_k \cdot \alpha_k$. Therefore, $\sigma^{n+|C|} \equiv_{\hat{c}} Act(\rho[..n]) \cdot \beta_k$. Thus, it follows that $\rho' \succeq \rho$.

Therefore, we have shown that for each run $\rho \in \mathcal{R}(S)$, there exists a run $\rho' \in \mathcal{R}(S')$ such that $\rho \equiv \rho'$.

□

Proofs for Section V ((max,+) Automaton Reduction)

Lemma 26. *Given are a (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$ and a 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 uncontrollable-(in)dependent in s, then they are ratio-(in)dependent in c.*

Proof. Assume that A and B are uncontrollable-independent in s. To prove that A and B are ratio-independent in c, we show that the three conditions stated in Def. 15 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. The fact that the same resource availability vector $\boldsymbol{\gamma}'$ is reached follows directly from the observation that the matrices of A and B commute and the use of vector normalization as described in Def. 4.
- 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 directly from the vector normalization and the fact that a constant can be taken out of a vector, i.e. $\| \boldsymbol{\gamma} - c \| = \| \boldsymbol{\gamma} \| - c$.
- Condition 3 requires that A and B have no resources in common, which follows directly from the definition of uncontrollable-independence.

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

The proof for Theorem 32 needs an extra definition and lemma.

Definition 39. *Given reduction function $ample_{\mathcal{A}}$ on a (max,+) automaton $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$, 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 40. *Let $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$ be a (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 the ample reduction. corresponding to $ample_{\mathcal{A}}$ on $\mathcal{S}$, as defined by Def. 39, yielding state space $\hat{\mathcal{S}}$, then $\hat{\mathcal{S}} = \mathcal{S}'$.*

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

$$\begin{array}{ccc} \mathcal{A} & \xrightarrow{\quad ample_{\mathcal{A}} \quad} & \mathcal{A}' \\ \downarrow & & \downarrow \\ \mathcal{S} & \xrightarrow[\quad ample_{\mathcal{S}} \quad]{} & \hat{\mathcal{S}} = \mathcal{S}' \end{array}$$

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 $\hat{\mathcal{S}} = \mathcal{S}'$. □

Theorem 32. *Let $\mathcal{A}$ be a (max,+) automaton with state space $\mathcal{S}$. Let $\mathcal{S}'$ be the state space from a (max,+) automaton obtained from $\mathcal{A}$ through an ample reduction[4]. Then $\mathcal{S} \approx_p \mathcal{S}'$.*

Proof. To prove that $\mathcal{S} \approx_p \mathcal{S}'$ we need to show that $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$ and $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$.

Let $ample_{\mathcal{S}}$ be the corresponding ample reduction on $\mathcal{S}$ that yields state space $\hat{\mathcal{S}}$. By Lemma 40, $\hat{\mathcal{S}} = \mathcal{S}'$. Since $ample_{\mathcal{A}}$ satisfies condition (A1), it directly follows that $ample_{\mathcal{S}}$ satisfies (R1). By Lemma 26, if activities are uncontrollable-(in)dependent in s, then they are also ratio-(in)dependent in any c with $c = \langle s, \gamma\rangle$. Combined with the definition

[4]In the context of [7], there is no separation between controllable and uncontrollable activities and the marking of states is ignored. Def. 31 coincides with Def. 26 in [7] when all activities are considered to be controllable and all states not marked. Then, conditions (A3), (A5.1), and (A5.2) trivially hold as $\mathcal{A}ct_u$ is empty, and (A4) holds as ω is never enabled. Similarly, the notion of uncontrollable-independent activities (Def. 25) coincides with resource-independent activities (Def. 23 in [7]), as condition 2 in Def. 25 trivially holds in case $\mathcal{A}ct_u$ is empty. So Thm. 32 carries over to the setting of [7] (coinciding with Thm. 28 in [7]).

of $ample_{\mathcal{S}}$, it follows that $ample_{\mathcal{S}}$ satisfies condition (R2). By Thm. 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 $\mathcal{S} \approx_p \mathcal{S}'$. □

In the remainder of this section, we prove that $sup\mathcal{CN}(\mathcal{P}') \lesssim_{f,p} sup\mathcal{CN}(\mathcal{P})$ given a plant $\mathcal{P}$ and reduced plant $\mathcal{P}'$ obtained from an ample reduction on $\mathcal{P}$ (Theorem 33) by proving the four conditions of Def. 30. The proof intuitively goes as follows.

1. Nonblockingness: Synthesis guarantees that $sup\mathcal{CN}(\mathcal{P}')$ is nonblocking, for any P'.

2. Controllability: Condition (A3) guarantees that uncontrollable actions are preserved in an ample reduction.

3. Reduced least-restrictiveness: We need to show that $sup\mathcal{CN}(\mathcal{P}') \preceq sup\mathcal{CN}(\mathcal{P})$ (Lemma 46 below) and that for each path p to a marked state in $sup\mathcal{CN}(\mathcal{P})$, $sup\mathcal{CN}(\mathcal{P}')$ has a path p' to a marked state in $sup\mathcal{CN}(\mathcal{P}')$ that is equivalent to p (Lemma 48).

First, we show that $sup\mathcal{CN}(\mathcal{P}') \preceq sup\mathcal{CN}(\mathcal{P})$. By induction on the iterations of the synthesis algorithm given below, Alg. 3, we can show that states of $\mathcal{P}'$ are bad in iteration k of synthesis on $\mathcal{P}'$ if and only if those states are bad in iteration k of synthesis on $\mathcal{P}$ (Lemma 47). Consequently, also $sup\mathcal{CN}(\mathcal{P}') \preceq sup\mathcal{CN}(\mathcal{P})$.

Second, we show that a path p' exists in $sup\mathcal{CN}(\mathcal{P}')$ equivalent to p. As $sup\mathcal{CN}(\mathcal{P}) \preceq \mathcal{P}$, by our assumption below Def. 8, p is also a path in $\mathcal{P}$. By structural induction, we can find an equivalent path p' in $\mathcal{P}'$ (Lemma 44 below). In each step, we reorder uncontrollable-independent activities in p to extend the prefix of p', in the end obtaining path p' (using Lemma 42 below). To prove that p' is also a path in $sup\mathcal{CN}(\mathcal{P}')$, we assume towards a contradiction that path p' is not preserved in $sup\mathcal{CN}(\mathcal{P}')$. Assuming that p' is not in $sup\mathcal{CN}(\mathcal{P}')$, there must be a state on p' that turns bad during synthesis. Path p' is also a path in $\mathcal{P}$, as $\mathcal{P}' \preceq \mathcal{P}$. The earlier observation that states of $\mathcal{P}'$ are bad in iteration k of synthesis on $\mathcal{P}'$ if and only if those states are bad in iteration k of synthesis on $\mathcal{P}$ implies that p' is also not in $sup\mathcal{CN}(\mathcal{P})$. Path p' is obtained from p by swapping uncontrollable-independent activities. By induction on the number of swaps to obtain a path from an equivalent one, it can be shown that synthesis preserves equivalent paths (Lemma 41 below). The key observation is that a swap of two uncontrollable-independent activities preserves bad states (via condition 2 of Def. 25). Given that p' is not in $sup\mathcal{CN}(\mathcal{P})$, and p and p' being equivalent, this implies that also p is not in $sup\mathcal{CN}(\mathcal{P})$, which contradicts our initial assumption. Therefore, p' must be present in $sup\mathcal{CN}(\mathcal{P}')$, showing reduced least-restrictiveness of $sup\mathcal{CN}(\mathcal{P}')$.

4. Performance: Let $\mathcal{S}$ and $\mathcal{S}'$ denote the state spaces of respectively $sup\mathcal{CN}(\mathcal{P})$ and $sup\mathcal{CN}(\mathcal{P}')$.

Throughput (Lemma 49) By Prop. 12, the worst-case throughput in $\mathcal{S}$ is exhibited by a run ρ with $Act(\rho) = \alpha_1 \cdot (\alpha_2)^\omega$ (for some $\alpha_1, \alpha_2 \in Act^*$) such that α_2 corresponds to a cycle in the state space, i.e., $\alpha_1(\hat{c}) = \alpha_1 \cdot \alpha_2(\hat{c})$ with $\hat{c}$ the initial configuration of $\mathcal{S}$ and $\mathcal{S}'$. This run corresponds to a path p in $sup\mathcal{CN}(\mathcal{P})$ with $Act(p) = \alpha_1 \cdot \alpha_2 \cdot \alpha_3$, where α_2 corresponds to a cycle in $sup\mathcal{CN}(\mathcal{P})$ and where α_3 represents a path from that cycle to a marked state. We can extend this path to a path $\bar{p}$ with $Act(\bar{p}) = \alpha_1 \cdot \alpha_2^N \cdot \alpha_3$ for arbitrary $N > 0$, where we traverse cycle α_2 N times. By reduced least-restrictiveness of $sup\mathcal{CN}(\mathcal{P}')$, for each such path there exists an equivalent path $\bar{p}'$ in $sup\mathcal{CN}(\mathcal{P}')$. If $N \geq |\mathcal{S}'|$, $\bar{p}'$ is such that there exist β_1, β_2, β_3 with $\beta_1 \cdot \beta_2 \cdot \beta_3 \equiv_{\hat{s},u} \alpha_1 \cdot \alpha_2^N \cdot \alpha_3$, where $\hat{s}$ is the initial state in $\mathcal{P}$ and $\mathcal{P}'$, $|\beta_1| \geq |\alpha_1|$, $|\beta_3| \geq |\alpha_3|$, $|\beta_2| = k \cdot |\alpha_2|$ for some $k > 0$. Then, for all $m > 0$, $\alpha_1 \cdot \alpha_2^{N+(m-1)\cdot k} \cdot \alpha_3 \equiv_{\hat{s},u} \beta_1 \cdot \beta_2^m \cdot \beta_3$. As uncontrollable-independence in the automaton implies ratio-independence in the state space, $\alpha_1 \cdot \alpha_2^{N+(m-1)\cdot k} \cdot \alpha_3 \equiv_{\hat{c}} \beta_1 \cdot \beta_2^m \cdot \beta_3$. It follows that $\mathcal{S}'$ has a run ρ' with $Act(\rho') = \beta_1 \cdot (\beta_2)^\omega$ that repeats a cycle that is equivalent to α_2 in run ρ. Hence, ρ and ρ' have the same ratio values, i.e., $\mathsf{Ratio}(\rho') = \mathsf{Ratio}(\rho)$.

Latency (Lemma 50) The maximum latency between source activity A_{src} and sink activity A_{snk} for resource r is determined by some source-sink occurrence pair in some run ρ in $\mathcal{S}$. Let prefix $\rho[..m]$, for some m, with $Act(\rho[..m]) = \alpha_1 \cdot A_{src} \cdot \alpha_2 \cdot A_{snk}$ contain this occurrence pair. $\rho[..m]$ corresponds to a path p in $sup\mathcal{CN}(\mathcal{P})$ with $Act(p) = Act(\rho[..m])$. We can extend p to path $\bar{p}$ with $Act(\bar{p}) = Act(p) \cdot \alpha_3$ such that $\bar{p}$ leads to a marked state. By reduced least-restrictiveness of $sup\mathcal{CN}(\mathcal{P}')$, for each such path, there exists an equivalent path $\bar{p}'$ with $Act(\bar{p}') \equiv_{\hat{s},u} Act(\bar{p})$, where $\hat{s}$ is the initial state in $\mathcal{P}'$ and $\mathcal{P}$. Path $\bar{p}'$ can be obtained from $\bar{p}$ by swapping uncontrollable-independent activities such that $Act(\bar{p}') = \beta_1 \cdot A_{src} \cdot \beta_2 \cdot A_{snk} \cdot \beta_3$. As uncontrollable-independence in the automaton implies ratio-independence in the state space, we obtain a run ρ' in $\mathcal{S}'$ such that $Act(\rho'[..m']) = Act(\bar{p}') \equiv_{\hat{c}} Act(\rho[..m]) \cdot \alpha_3$. Note that path $\bar{p}'$ can always be extended to an infinite run, because it ends in a marked state having an ω-self loop. Activities A_{src} and A_{snk} can only be part of a swap where the resource availability time of r is not affected (as explained in the proof of Lem. 20). Hence, the latency of this occurrence is identical to the one in ρ. Given the assumption that the latency value of the considered source-sink occurrence pair in ρ is the maximum value in $\mathcal{S}$ and since $\mathcal{S}'$ does not introduce new latency values, the latency value of ρ' must be the same worst-case value as the latency value of ρ.

The remainder of this section provides a detailed proof for Theorem 33.

Lemma 41 (Synthesis preserves all equivalent paths if one representative path is preserved)**.** *Let $p = \hat{s} \xrightarrow{\sigma}{}^* s$ be a path in plant $\mathcal{P}$ and let $p' = \hat{s} \xrightarrow{\tau}{}^* s$ with $\sigma \equiv_{\hat{s},u} \tau$ be a path obtained from p in steps $p = p^0 \to p^1 \to \ldots \to p^n = p'$ by repeatedly swapping uncontrollable-independent activities. If p is a path in $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$, then also p' is a path in $\mathcal{A}_{sup}$.*

Proof. We prove the construction for a single step. The proof for a sequence of steps follows then by repeated application for each of the consecutive steps. Consider paths $p = \hat{s} \rightarrow^* s_i \xrightarrow{AB}{}^* \bar{s} \rightarrow^* s$ and $p' = \hat{s} \rightarrow^* s_i \xrightarrow{BA}{}^* \bar{s} \rightarrow^* s$ in plant $\mathcal{P}$, where A and B be uncontrollable-independent activities in state s_i. Assume that after synthesis on $\mathcal{P}$, p is still a path in $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$. Now, we need to show that also p' is a path in $\mathcal{A}_{sup}$.

Since p is a path in $\mathcal{A}_{sup}$, we know that for each state s_j on path p it holds that $s_j \in N^k$ and $s_j \notin B^k$ for each $k \geq 0$ in Algorithm 3, where N^k is the set of nonblocking states in iteration k, and B^k is the set of bad states in iteration k that are blocking or uncontrollable.

Towards a contradiction, assume that p' is not a path in $\mathcal{A}_{sup}$. This means that there is a state s_k on path p' that is added to set B^k in some iteration $k \geq 0$ of Algorithm 3. If state s_k is also on path p, then we have an immediate contradiction, because this would imply that path p would not have been present in $\mathcal{A}_{sup}$. So, state s_k is a state that cannot be on path p, meaning that $s_k = B(s_i)$. As p is a path in $\mathcal{A}_{sup}$, we know that all states on p, including state $\bar{s}$ are nonblocking, and therefore $\bar{s} \in N^k$. State s_k is therefore not added to B^k in line 10 of Algorithm 3, and must be added in line 12. In this case, there must be an outgoing uncontrollable activity U from state s_k leading to a bad state $s' \in B^k_j$ for some $j \geq 0$. This leads us to the required contradiction, as the second condition of Definition 25 states that no uncontrollable activity can be enabled in $s_k = B(s_i)$ since A and B are uncontrollable-independent activities in state s_i. □

Lemma 42 (adapted from [37, Lemma 8.15])**.** *Let $p[i,j] = s_i B_i s_{i+1} \dots B_j s_j$ be a path fragment in $\mathcal{A}$. If $ample(s_i)$ satisfies condition (A2), (A5.1) and (A5.2) and $\{B_i, \dots, B_j\} \cap ample(s_i) = \emptyset$, then all activities $A \in ample(s_i)$ are uncontrollable-independent of $\{B_i, \dots, B_j\}$. In addition, we have $A \in enabled(s_k)$ for $i < k \leq j$.*

Proof. Proof follows the same structure as the proof of [37, Lemma 8.15]. □

Definition 43 (Path equivalence)**.** *Let p and p' be two paths in $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$. We define $p \succeq p'$ iff there exists a $d \in \mathbb{N}$ such that for all $n \geq 0$ it holds that $p \succeq^d_n p'$. We define $p \succeq^d_n p'$ iff there exists some $k \geq n$, and path $\hat{\rho}$ in $\mathcal{A}$ with path prefix $\hat{p}[..k]$ with $Act(\hat{p}[..k]) \equiv_{\hat{s},u} Act(p[..k])$ such that $Act(\hat{p}[..k]) = Act(p'[..n]) \cdot \tau$ for some activity sequence τ, and $k - n \leq d$. Paths p and p' are equivalent, denoted $p \equiv p'$, iff $p \succeq p'$ and $p' \succeq p$.*

Lemma 44 (Ample reduction preserves a path to each marked state)**.** *Let $\mathcal{A} = \langle S, \hat{s}, S^m, \mathcal{A}ct, reward, M, T\rangle$ be an automaton that has been extended with ω-self-loops. Let $\mathcal{A}_r = \langle S_r, \hat{s}_r, S^m_r, \mathcal{A}ct_r, reward_r, M_r, T_r\rangle$ be the reduced automaton induced by ample reduction ample. Let $s \in S_r$ and let $p = s \xrightarrow{\alpha}{}^*_{\mathcal{A}} s_m$ be a path in $\mathcal{A}$ with $\alpha \in \mathcal{A}ct^*$ and $s_m \in S^m$. Then, in $\mathcal{A}_r$, there exists a path $p_r = s \xrightarrow{\beta}{}^*_r s_m$ with $\beta \in \mathcal{A}ct^*$ and $\alpha \equiv_{s,u} \beta$.*

Proof. Let $p = s \xrightarrow{\alpha}{}^* s_m$ be some path in $\mathcal{A}$. Assume w.l.o.g. that the last activity of α is ω. We define path p_r (and hence β) in $\mathcal{A}_r$ with $\beta^n = Act(p_r[..n])$ for all $n \leq |p|$ recursively as follows:

$$\beta^0 = \varepsilon$$
$$\beta^{n+1} = \beta^n \cdot \gamma(k), \text{where } \gamma \text{ and } k \text{ are as described below.}$$

γ is such that $\beta^n \cdot \gamma \equiv_{s,u} \alpha$, k is the smallest number such that $\gamma(k) \in ample(\beta^n(s))$.
We prove that such γ and k exist by induction.

- For $n = 0$, $\gamma = \alpha$. By Lemma 42, and the fact that ω, the last activity of α, is uncontrollable dependent with any activity, there is some $k < |\alpha|$, such that $\gamma(k) \in ample(\beta^0(s)) = ample(s)$.
- For $n > 0$, using the induction hypothesis, let $\beta^n \cdot \gamma \equiv_{s,u} \alpha$. $\beta^{n+1} = \beta^n \cdot \gamma(k)$ with $\gamma(k) \in ample(\beta^n(s))$, and for all $m < k$, $\gamma(m) \notin ample(\beta^n(s))$. Hence, according to Lemma 42, all $\gamma(m)$ for $m < k$ are uncontrollable independent of $\gamma(k)$ and $\gamma \equiv_{s',u} \gamma(k) \cdot \gamma[..k-1] \cdot \gamma[k+1..]$ for $s' = \beta^n(s)$. $\beta^{n+1} \cdot \gamma[..k-1] \cdot \gamma[k+1..] = \beta^n \cdot \gamma(k) \cdot \gamma[..k-1] \cdot \gamma[k+1..] \equiv_{s,u} \beta^n \cdot \gamma \equiv_{s,u} \alpha$.

With $\beta = \beta^{|\alpha|}$ as defined above, the corresponding path p_r in $\mathcal{A}_r$ is such that $p_r = s \xrightarrow{\beta}{}^*_r s_m$ with $\beta \in \mathcal{A}ct^*$ and $\alpha \equiv_{s,u} \beta$. □

Lemma 45. *Let $\mathcal{A}$ be an automaton that has been extended with ω-self-loops, and let $\mathcal{A}'$ be the reduced automaton induced by reduction function ample. Let $\mathcal{A}$ be nonblocking. If ample satisfies conditions (A1), (A2), (A3), and (A4) then $\mathcal{A}'$ is nonblocking.*

Proof. We show that $\mathcal{A}'$ is nonblocking by showing that $\mathcal{A}'$ is ω-reachable. Let $s \in S'$ be any reachable state in $\mathcal{A}'$, such that $\hat{s} \xrightarrow{\alpha_1}{}^*_{\mathcal{A}'} s$ for some $\alpha_1 \in \mathcal{A}ct^*$. Then, also $\hat{s} \xrightarrow{\alpha_1}{}^*_{\mathcal{A}} s$ since $\mathcal{A}' \preceq \mathcal{A}$. Since $\mathcal{A}$ is nonblocking, it is also ω-reachable. Therefore, there exists a path $p = s \xrightarrow{\alpha_2}{}^*_{\mathcal{A}} s_m$ for some $\alpha_2 \in \mathcal{A}ct^*$ and $s_m \in S$ such that $\omega \in enabled(s_m)$. By Lemma 44, there exists a path $p_r = s \xrightarrow{\beta}{}^*_{\mathcal{A}'} s_m$ in $\mathcal{A}'$ for some $\beta \in \mathcal{A}ct^*$ with $\alpha_2 \equiv_{\hat{s},u} \beta$. □

To prove reduced least-restrictiveness of the reduced supervisor, we need a definition of the synthesis procedure, given in Alg. 3. We also need two extra lemmas.

Algorithm 3 Synthesis algorithm SUPCN, adapted from [2]

1: **proc** SUPCN($\mathcal{P} = \langle S, \hat{s}, S^m, \mathcal{A}ct, T\rangle$)
2: $k \leftarrow 0, X^k \leftarrow S$
3: **repeat**
4: $i \leftarrow 0, N_0^k \leftarrow S^m \cap X^k$
5: **repeat** ▷ compute the nonblocking states
6: $N_{i+1}^k \leftarrow N_i^k \cup \{s \in X^k \mid s \rightarrow s', s' \in N_i^k\}$
7: $i \leftarrow i + 1$
8: **until** $N_i^k = N_{i-1}^k$
9: $N^k \leftarrow N_i^k$
10: $j \leftarrow 0, B_0^k \leftarrow X^k \setminus N^k$
11: **repeat** ▷ compute the bad states
12: $B_{j+1}^k \leftarrow B_j^k \cup \{s \in X^k \mid s \xrightarrow{U} s', s' \in B_j^k, U \in \mathcal{A}ct_u\}$
13: $j \leftarrow j + 1$
14: **until** $B_j^k = B_{j-1}^k$
15: $B^k \leftarrow B_j^k$
16: $X^{k+1} \leftarrow X^k \setminus B^k$
17: $k \leftarrow k + 1$
18: **until** $X^k = X^{k-1}$
19: **if** $\hat{s} \in X^k$ **then**
20: **return** $\langle X^k, \hat{s}, S^m \cap X^k, \mathcal{A}ct, T \cap (X^k \times \mathcal{A}ct \times X^k)\rangle$
21: **else**
22: **return** $\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\rangle$

Lemma 46. *Let $\mathcal{P}$ be a plant and $\mathcal{P}'$ the reduced plant obtained from an ample reduction. Then, $sup\mathcal{CN}(\mathcal{P}') \preceq sup\mathcal{CN}(\mathcal{P})$.*

Proof. By induction on the iterations of the synthesis algorithm in Alg. 3, we can show that states of $\mathcal{P}'$ are bad in iteration k of synthesis on $\mathcal{P}'$ if and only if those states are bad in iteration k of synthesis on $\mathcal{P}$ (Lemma 47). From the fact that $\mathcal{P}'$ is obtained from $\mathcal{P}$ through a reduction, it follows that $\mathcal{P}' \preceq \mathcal{P}$. Consequently, since $\mathcal{P}' \preceq \mathcal{P}$, also $sup\mathcal{CN}(\mathcal{P}') \preceq sup\mathcal{CN}(\mathcal{P})$. □

Lemma 47. *Let $\mathcal{P}$ be a plant and $\mathcal{P}'$ the reduced plant obtained from an ample reduction. Let Alg. 3 be the synthesis algorithm used for synthesis on $\mathcal{P}$ and $\mathcal{P}'$. States of $\mathcal{P}'$ are bad in iteration k of synthesis on $\mathcal{P}'$ if and only if those states are bad in iteration k of synthesis on $\mathcal{P}$.*

Proof. We show that synthesis on reduced plant $\mathcal{P}'$ marks states as bad in an iteration of the loop of line 3 of Algorithm 3 if and only if synthesis on plant $\mathcal{P}$ marks those states bad in the same iteration. To distinguish synthesis on $\mathcal{P}$ from synthesis on $\mathcal{P}'$, let B, N, X, and $\bar{B}$, $\bar{N}$, $\bar{X}$ denote the sets of bad, nonblocking, and initially present states in Algorithm 3 for $\mathcal{P}$ and $\mathcal{P}'$ respectively. We show by induction on k that $\bar{B}_j^k = B_j^k \cap S'$ and $\bar{N}^k = N^k \cap S'$ for iterations $k, j \geq 0$ of Algorithm 3 on plants $\mathcal{P}'$ and $\mathcal{P}$.

- **Base case:** We need to show that $\bar{B}_j^0 = B_j^0 \cap S'$, for all $j \geq 0$, and that $\bar{N}^0 = N^0 \cap S'$.
  We first show that $\bar{N}^0 = N^0 \cap S'$. It is clear that $\bar{N}^0 \subseteq N^0 \cap S'$. The fact that also $\bar{N}^0 \supseteq N^0 \cap S'$ follows from a similar reasoning as in the proof of Lemma 45, using the properties of an ample reduction. We then prove $\bar{B}_j^0 = B_j^0 \cap S'$, for all $j \geq 0$, distinguishing two cases for j.
  - $j = 0$: The desired equality $\bar{B}_0^0 = B_0^0 \cap S'$ follows directly from line 10 of the algorithm and the fact that $\bar{N}^0 = N^0 \cap S'$.
  - $j > 0$: First, assume $s' \in \bar{B}_j^0$, for some $j > 0$. We need to show that $s' \in B_j^0 \cap S'$. From lines 10-15 of Algorithm 3, we know that there is a path $p' = s' \xrightarrow{\alpha}{}^* s''$ with $s'' \in \bar{B}_0^0$ and $\alpha \in \mathcal{A}ct_U{}^*$, i.e., there is a path p' of uncontrollable actions from s' to some s'' in $\bar{B}_0^0$. In the first iteration of the algorithm, it is clear that $S' = \bar{X}^0 \subseteq S = X^0$. Hence, p' also exists in $\mathcal{P}$. Since $\bar{B}_0^0 = B_0^0 \cap S'$, $s'' \in B_0^0$, i.e., p' ends in B_0^0. Therefore, it follows that $s' \in B_j^0$ and hence that $s' \in B_j^0 \cap S'$.
    Second, assume $s' \in B_j^0 \cap S'$, for some $j > 0$. We need to show that $s' \in \bar{B}_j^0$. Again from lines 10-15 of Algorithm 3, we know that there is a path $p = s' \xrightarrow{\alpha}{}^* s''$ with $s'' \in B_0^0$ and $\alpha \in \mathcal{A}ct_U{}^*$, i.e., there is a path p of uncontrollable actions from s' to some s'' in B_0^0. By conditions (A5.1) and (A5.2), an ample reduction does not remove any states with an enabled uncontrollable action. Hence, because synthesis has not yet removed any states from plant $\mathcal{P}'$, all states on path p are also elements of S'. Therefore, p also exists in $\mathcal{P}'$. $\bar{B}_0^0 = B_0^0 \cap S'$ then implies that $s' \in \bar{B}_j^0$.
- **Induction step:** Assume $\bar{B}_j^n = B_j^n \cap S'$ and $\bar{N}^n = N^n \cap S'$ for all $n, j \geq 0$ with $n < k$ for some $k > 0$. We need to show that $\bar{B}_j^k = B_j^k \cap S'$ for all $j \geq 0$ and $\bar{N}^k = N^k \cap S'$.

As in the base case, we first show $\bar{N}^k = N^k \cap S'$. Let $s' \in \bar{N}^k$. We need to show that $s' \in N^k \cap S'$. Because of the induction hypothesis, any path present in $\bar{X}^k$ is also still present in X^k. Since $\bar{X}^k \cap S^m = X^k \cap S^m \cap S'$, $s' \in N^k \cap S'$. Next, let $s' \in N^k \cap S'$; that is, there is a path in X^k from s' to a marked state. We need to show that $s' \in \bar{N}^k$. Reasoning again as in the proof of Lemma 45, in combination with the induction hypothesis, it follows that there must be a (possibly different) path from s' to the same marked state in $\bar{X}^k$. Hence, the desired $s' \in \bar{N}^k$ follows.

Second, we have to show that $\bar{B}_j^k = B_j^k \cap S'$. The reasoning to show this follows the base case, using in addition the fact that up to iteration k exactly the same states in S' have been removed from S' by synthesis on plant $\mathcal{P}'$ as the states from S' that have been removed from S by synthesis up to iteration k on plant $\mathcal{P}$.

□

Lemma 48 (Existence of an equivalent path in the reduced supervisor)**.** *Let $\mathcal{P} = \langle S, \hat{s}, S^m, \mathcal{A}ct, T\rangle$ be a plant, and let $\mathcal{P}' = \langle S', \hat{s}, S'^m, \mathcal{A}ct, T'\rangle$ be the reduced plant after applying an ample reduction on $\mathcal{P}$. Let $sup\mathcal{CN}(\mathcal{P}) = \langle S_{sup}, \hat{s}, S^m_{sup}, \mathcal{A}ct, T_{sup}\rangle$ be the supervisor after applying synthesis on $\mathcal{P}$, and $sup\mathcal{CN}(\mathcal{P}') = \langle S'_{sup}, \hat{s}, S'^m_{sup}, \mathcal{A}ct, T'_{sup}\rangle$ be the reduced supervisor after applying synthesis on $\mathcal{P}'$. Let $p = \hat{s} \xrightarrow{\alpha}{}^* s^m$ be a path in $sup\mathcal{CN}(\mathcal{P})$ to marked state $s^m \in S^m$ with $\alpha \in \mathcal{A}ct^*$. Then in $sup\mathcal{CN}(\mathcal{P}')$ there exists a path $p' = \hat{s} \xrightarrow{\beta}{}^* s^m$ with $\beta \in \mathcal{A}ct^*$ and $\alpha \equiv_{\hat{s},u} \beta$.*

Proof. Let $p = \hat{s} \xrightarrow{\alpha}{}^* s$ be any path in supervisor $sup\mathcal{CN}(\mathcal{P})$. As $sup\mathcal{CN}(\mathcal{P}) \preceq \mathcal{P}$, by our assumption below Def. 8, p is also a path in $\mathcal{P}$. By Lemma 44, there exists at least one equivalent path $p' = \hat{s} \xrightarrow{\beta}{}^* s$ in $\mathcal{P}'$ with $\alpha \equiv_{\hat{s},u} \beta$. To prove that p' is also a path in $sup\mathcal{CN}(\mathcal{P}')$, we assume towards a contradiction that path p' is not preserved in $sup\mathcal{CN}(\mathcal{P}')$. Assuming that p' is not in $sup\mathcal{CN}(\mathcal{P}')$, there must be a state on p' that turns bad during synthesis. Path p' is also a path in $\mathcal{P}$, as $\mathcal{P}' \preceq \mathcal{P}$. By Lemma 47, we have that states of $\mathcal{P}'$ are bad in iteration k of synthesis on $\mathcal{P}'$ if and only if those states are bad in iteration k of synthesis on $\mathcal{P}$. This implies that p' is also not in $sup\mathcal{CN}(\mathcal{P})$. Path p' is obtained from p by swapping uncontrollable-independent activities. By Lemma 41, synthesis preserves equivalent paths. Given that p' is not in $sup\mathcal{CN}(\mathcal{P})$, and p and p' being equivalent, this implies that also p is not in $sup\mathcal{CN}(\mathcal{P})$, which contradicts our initial assumption. Therefore, p' must be present in $sup\mathcal{CN}(\mathcal{P}')$. □

Lemma 49 (Ample reduction preserves throughput)**.** *Let $\mathcal{P}$ be a plant, and $\mathcal{P}'$ the reduced plant obtained from an ample reduction. Let $\mathcal{S}'$ and $\mathcal{S}$ be the state spaces of respectively $\mathcal{A}'_{sup} = sup\mathcal{CN}(\mathcal{P}')$ and $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$. Then $\mathcal{S}$ and $\mathcal{S}'$ have the same worst-case throughput value $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$.*

Proof. By Prop. 12, the worst-case throughput in $\mathcal{S}$ is exhibited by a run ρ with $Act(\rho) = \alpha_1 \cdot (\alpha_2)^\omega$ (for some $\alpha_1, \alpha_2 \in Act^*$) such that α_2 corresponds to a cycle in the state space, i.e., $\alpha_1(\hat{c}) = \alpha_1 \cdot \alpha_2(\hat{c})$ with $\hat{c}$ the initial configuration of $\mathcal{S}$ and $\mathcal{S}'$. This run corresponds to a path p in $sup\mathcal{CN}(\mathcal{P})$ with $Act(p) = \alpha_1 \cdot \alpha_2 \cdot \alpha_3$, where α_2 corresponds to a cycle in $sup\mathcal{CN}(\mathcal{P})$ and where α_3 represents a path from that cycle to a marked state. We can extend this path to a path $\bar{p}$ with $Act(\bar{p}) = \alpha_1 \cdot \alpha_2^N \cdot \alpha_3$ for arbitrary $N > 0$, where we traverse cycle α_2 N times. By reduced least-restrictiveness of $sup\mathcal{CN}(\mathcal{P}')$, for each such path there exists an equivalent path $\bar{p}'$ in $sup\mathcal{CN}(\mathcal{P}')$. If $N \geq |\mathcal{S}'|$, $\bar{p}'$ is such that there exist $\beta_1, \beta_2, \beta_3$ with $\beta_1 \cdot \beta_2 \cdot \beta_3 \equiv_{\hat{s},u} \alpha_1 \cdot \alpha_2^N \cdot \alpha_3$, where $\hat{s}$ is the initial state in $\mathcal{P}$ and $\mathcal{P}'$, $|\beta_1| \geq |\alpha_1|$, $|\beta_3| \geq |\alpha_3|$, $|\beta_2| = k \cdot |\alpha_2|$ for some $k > 0$. Then, for all $m > 0$, $\alpha_1 \cdot \alpha_2^{N+(m-1)\cdot k} \cdot \alpha_3 \equiv_{\hat{s},u} \beta_1 \cdot \beta_2^m \cdot \beta_3$. As uncontrollable-independence in the automaton implies ratio-independence in the state space (Lemma 26), $\alpha_1 \cdot \alpha_2^{N+(m-1)\cdot k} \cdot \alpha_3 \equiv_{\hat{c}} \beta_1 \cdot \beta_2^m \cdot \beta_3$. It follows that $\mathcal{S}'$ has a run ρ' with $Act(\rho') = \beta_1 \cdot (\beta_2)^\omega$ that repeats a cycle that is equivalent to α_2 in run ρ. Hence, ρ and ρ' have the same ratio values, i.e., $\mathsf{Ratio}(\rho') = \mathsf{Ratio}(\rho)$.

Now consider a run ρ' in $\mathcal{S}'$ exhibiting the worst-case throughput in $\mathcal{S}'$. As $\mathcal{S}'$ is included in $\mathcal{S}$, run ρ' is also present in $\mathcal{S}$. Therefore, $\mathcal{S}$ and $\mathcal{S}'$ have the same worst-case throughput value $\tau_{min}(\mathcal{S}) = \tau_{min}(\mathcal{S}')$. □

Lemma 50 (Ample reduction preserves latency)**.** *Let $\mathcal{P}$ be a plant, and $\mathcal{P}'$ the reduced plant obtained from an ample reduction. Let $\mathcal{S}'$ and $\mathcal{S}$ be the state spaces of respectively $\mathcal{A}'_{sup} = sup\mathcal{CN}(\mathcal{P}')$ and $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$. Then $\mathcal{S}$ and $\mathcal{S}'$ have the same maximum latency $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$.*

Proof. Let A_{src} and A_{snk} be the source and sink activity and let r be the resource on which we want to compute the start-to-start latency. Then by Def. 13,

$$\begin{aligned}
\lambda_{max}(\mathcal{S}) &= \sup_{\rho \in \mathcal{R}(\mathcal{S})} \lambda_{max}(\rho, A_{src}, A_{snk}, r) = \sup_{\rho \in \mathcal{R}(\mathcal{S})} \sup_{k \geq 1} \lambda_k(\rho) \textbf{ where} \\
\lambda_k(\rho) &= \lambda(\rho, i, j, r), \\
i &= \mathsf{getOccurence}(\rho, A_{src}, k), \text{ and} \\
j &= \mathsf{getOccurence}(\rho, A_{snk}, k).
\end{aligned}$$

The maximum latency between source activity A_{src} and sink activity A_{snk} for resource r is determined by some source-sink occurrence pair in some run ρ in $\mathcal{S}$, with some occurrences $i = \mathsf{getOccurence}(\rho, A_{src}, k)$ and $j = \mathsf{getOccurence}(\rho, A_{snk}, k)$.

Let prefix $\rho[..m]$, for some m, with $Act(\rho[..m]) = \alpha_1 \cdot A_{src} \cdot \alpha_2 \cdot A_{snk}$ contain this occurrence pair. $\rho[..m]$ corresponds to a path p in $sup\mathcal{CN}(\mathcal{P})$ with $Act(p) = Act(\rho[..m])$. We can extend p to path $\bar{p}$ with $Act(\bar{p}) = Act(p) \cdot \alpha_3$ such that $\bar{p}$ leads to a marked state. By reduced least-restrictiveness of $sup\mathcal{CN}(\mathcal{P}')$, for each such path, there exists an equivalent path $\bar{p}'$ with $Act(\bar{p}') \equiv_{\hat{s},u} Act(\bar{p})$, where $\hat{s}$ is the initial state in $\mathcal{P}'$ and $\mathcal{P}$. Path $\bar{p}'$ can be obtained from $\bar{p}$ by swapping uncontrollable-independent activities such that $Act(\bar{p}') = \beta_1 \cdot A_{src} \cdot \beta_2 \cdot A_{snk} \cdot \beta_3$. As uncontrollable-independence in the automaton implies ratio-independence in the state space, we obtain a run ρ' in $\mathcal{S}'$ such that $Act(\rho'[..m']) = Act(\bar{p}') \equiv_{\hat{c}} Act(\rho[..m]) \cdot \alpha_3$. Note that path $\bar{p}'$ can always be extended to an infinite run, because it ends in a marked state having an ω-self loop. Activities A_{src} and A_{snk} can only be part of a swap where the resource availability time of r is not affected (as explained in the proof of Lem. 20). Hence, the latency of this occurrence is identical to the one in ρ. Given the assumption that the latency value of the considered source-sink occurrence pair in ρ is the maximum value in $\mathcal{S}$ and since $\mathcal{S}'$ does not introduce new latency values, the latency value of ρ' must be the same worst-case value as the latency value of ρ and $\lambda_{max}(\mathcal{S}) = \lambda_{max}(\mathcal{S}')$. □

Theorem 33 (Ample reduction preserves functionality and performance)**.** *Let $\mathcal{P}$ be a plant, and $\mathcal{P}'$ the reduced plant obtained from an ample reduction. Then $sup\mathcal{CN}(\mathcal{P}') \lesssim_{f,p} sup\mathcal{CN}(\mathcal{P})$.*

Proof. Let $\mathcal{A}'_{sup} = sup\mathcal{CN}(\mathcal{P}')$ and $\mathcal{A}_{sup} = sup\mathcal{CN}(\mathcal{P})$. To show that $\mathcal{A}'_{sup} \lesssim_{f,p} \mathcal{A}_{sup}$, we need to prove the four conditions as stated in Def. 30.

1. Nonblockingness: We need to show that if $\mathcal{A}_{sup}$ is nonblocking then $\mathcal{A}'_{sup}$ is nonblocking. The synthesis algorithm always guarantees that the resulting supervisor $\mathcal{A}'_{sup} = sup\mathcal{CN}(\mathcal{P}')$ is nonblocking, independent of whether $\mathcal{A}_{sup}$ is nonblocking.

2. Controllability: We need to show that if $\mathcal{A}_{sup}$ is controllable with respect to $\mathcal{P}$ then $\mathcal{A}'_{sup}$ is controllable with respect to $\mathcal{P}$. Consider any string $\sigma \in \mathcal{A}ct^*$ and state $s \in S_{\mathcal{A}'_{sup}}$ such that $\hat{s} \xrightarrow{\sigma}{}^*_{\mathcal{A}'_{sup}} s$. Let $enabled_{\mathcal{A}}(s)$ denote the enabled set in state s in automaton $\mathcal{A}$.

First we prove that $\mathcal{P}'$ is controllable with respect to $\mathcal{P}$. Recall that $\mathcal{P}' \preceq \mathcal{P}$. Let $\alpha \in \mathcal{A}ct^*$ and $U \in \mathcal{A}ct_u$, and let $s, s' \in S_{\mathcal{P}}$ such that $\hat{s} \xrightarrow{\alpha}{}^*_{\mathcal{P}'} s$ and $\hat{s} \xrightarrow{\alpha}{}^*_{\mathcal{P}} s \xrightarrow{U}_{\mathcal{P}} s'$. Since $s \in S_{\mathcal{P}'}$, and condition (A3) holds in s, we have that $U \in ample(s)$. By Def. 31, then also $\hat{s} \xrightarrow{\alpha}{}^*_{\mathcal{P}'} s \xrightarrow{U}_{\mathcal{P}'} s'$, proving that $\mathcal{P}'$ is controllable with respect to $\mathcal{P}$.

As $\mathcal{P}'$ is controllable with respect to $\mathcal{P}$, $enabled_{\mathcal{P}'}(s) \cap \mathcal{A}ct_u = enabled_{\mathcal{P}}(s) \cap \mathcal{A}ct_u$. Synthesis guarantees that $\mathcal{A}'_{sup}$ is controllable with respect to $\mathcal{P}'$, which means that $enabled_{\mathcal{A}'_{sup}}(s) \cap \mathcal{A}ct_u = enabled_{\mathcal{P}'}(s) \cap \mathcal{A}ct_u$. Therefore, $enabled_{\mathcal{A}'_{sup}}(s) \cap \mathcal{A}ct_u = enabled_{\mathcal{P}}(s) \cap \mathcal{A}ct_u$, and $\mathcal{A}'_{sup}$ is controllable with respect to $\mathcal{P}$.

3. Reduced least-restrictiveness: We need to show that $\mathcal{A}'_{sup}$ is reduced least-restrictive w.r.t. $\mathcal{A}_{sup}$. Supervisor $\mathcal{A}'_{sup}$ is reduced least-restrictive with respect to supervisor $\mathcal{A}_{sup}$, if $\mathcal{A}'_{sup} \preceq \mathcal{A}_{sup}$ and for each path $\hat{s} \xrightarrow{\sigma}{}^*_{\mathcal{A}_{sup}} s_m$ in $\mathcal{A}_{sup}$ with $\sigma \in \mathcal{A}ct^*$ and $s_m \in S^m$, there exists a path $\hat{s} \xrightarrow{\tau}{}^*_{\mathcal{A}'_{sup}} s_m$ in $\mathcal{A}'_{sup}$ with $\sigma' \in \mathcal{A}ct^*$ and $\sigma \equiv \tau$. This follows by Lemmas 46 and 48.

4. Performance: Let $\mathcal{S}'$ and $\mathcal{S}$ be the corresponding normalized (max,+) state spaces of $\mathcal{A}'_{sup}$ and $\mathcal{A}_{sup}$. We need to show that $\mathcal{S}' \approx_p \mathcal{S}$. This follows directly by Lemmas 49 and 50. □

Proofs for Section VI (On-the-fly reduction)

Theorem 36. *Let $\mathcal{P} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ be a plant modeled as (max,+) timed system, and $\mathcal{P}'$ be the (max,+) automaton obtained by a cluster-inspired ample reduction. Then, $sup\mathcal{CN}(\mathcal{P}') \lesssim_{f,p} sup\mathcal{CN}(\mathcal{P})$.*

Proof. Let *ample* be a cluster-inspired ample reduction. We show that *ample* satisfies the conditions of Def. 31. The result then immediately follows from Theorem 33.

Consider any state s in the composition of $\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$. Let $ample(s) = enabled_{\mathcal{C}}(s)$, where $\mathcal{C}$ is any safe cluster in s, satisfying condition (M1). Conditions (A1), (A3), (A4), (A5.1), and (A5.2) follow directly from (C1), (C3), (C4), (C5.1), and (C5.2).

We prove (A2) by contraposition, for the case that $enabled(s) \neq \emptyset$. If $enabled(s) = \emptyset$, (A2) is trivially satisfied. Assume that (A2) does not hold. This means, that there exists a finite path fragment $p = s \xrightarrow{A_1} s_1 \xrightarrow{A_2} s_2 \xrightarrow{A_3} \dots s_{n-1} \xrightarrow{A_n}$, where $A_1 \dots A_{n-1}$ are uncontrollable-independent with $ample(s) = enabled_{\mathcal{C}}(s)$, and A_n is uncontrollable-dependent with some activity in $enabled_{\mathcal{C}}(s)$.

Since A_n is uncontrollable-dependent with some activity in $enabled_{\mathcal{C}}(s)$, by (C2.1), $A_n \in \mathcal{A}ct(\mathcal{C})$. Moreover, we have $A_n \notin enabled_{\mathcal{C}}(s)$. Since activities $A_1, \dots, A_{n-1}$ are uncontrollable-independent with $ample(s)$, $A_1, \dots, A_{n-1} \notin \mathcal{A}ct(\mathcal{C})$, meaning that they do not affect the state of $\mathcal{C}$, and therefore $\pi_{\mathcal{C}}(s)$ does not change in the first $n-1$ steps. As $A_n \in enabled(s_{n-1})$, $A_n \in enabled(\pi_{\mathcal{C}}(s))$. Since $A_n \notin enabled_{\mathcal{C}}(s)$, A_n becomes enabled in $\pi_{\mathcal{C}}(s)$ by executing one of the activities in set $A_1, \dots, A_{n-1}$. Since $A_1, \dots, A_{n-1}$ 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 (C2.2). □

Theorem 37. *Let s be a state in the composition $\mathcal{M} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ and $A \in enabled(s)$ be the candidate activity. Then,* ComputeCluster(s, A) *returns a safe cluster in s.*

Proof. Let $\mathcal{C}_1$ denote the cluster after executing lines 5-6. Set $enabled(\pi_{\mathcal{C}_1}(s))$ contains all uncontrollable enabled activities from $enabled(s) \cap \mathcal{A}ct_u$ and the special activity ω if it is enabled in state s. This means that conditions (C3) and (C4) are satisfied for $\mathcal{C}_1$ and any superset of $\mathcal{C}_1$.

Now, consider activity A and let $\mathcal{C}_k$ denote the value of $\mathcal{C}$ at line 8, and $\mathcal{C}_{k+1}$ the new cluster after executing lines 8-23. 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 (max,+) automata $\mathcal{A}_i$ with $A \in \mathcal{A}ct_i$ (added at line 12), which means that condition (C2.2) is satisfied for A. Since there are no (max,+) automata outside $\mathcal{A}_i$ with A, condition (C2.2) holds also for any superset of $\mathcal{C}_{k+1}$. After executing lines 13-15, there is no uncontrollable-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 uncontrollable-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 17-19, 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 (max,+) automaton $\mathcal{A}_i \notin \mathcal{C}_k$ where $A \notin enabled(\pi_i(s))$. Let $\mathcal{C}_{k+1} = \mathcal{C}_k \cup \{\mathcal{A}_i\}$, obtained at line 19. 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 8-19 for activity A, conditions (C2.1) and (C2.2) hold for A and keep holding for any extension to the cluster.

In line 10, we check whether $A \in \mathcal{U}$. If this is the case, then possibly an uncontrollable event becomes enabled in state $A(s)$, and the set of all (max,+) automata is returned. This guarantees that condition (C5.1) is never violated. Furthermore, it guarantees that condition (C5.2) is never violated, since it guarantees that no uncontrollable activity becomes enabled after executing A and subsequently some other uncontrollable-independent activity from s.

Upon termination of the algorithm, we obtain some cluster $\mathcal{C}_l$ where conditions (C1) till (C5.2) hold for each activity in $enabled(\pi_{\mathcal{C}_l}(s))$. From this, it follows that $\mathcal{C}_l$ is safe in s. □