

Scaling GR(1) Synthesis via a Compositional Framework for LTL Discrete Event Control

Hernan Gagliardi¹, Victor Braberman¹², and Sebastian Uchitel¹²³

¹ Universidad de Buenos Aires, Argentina

² CONICET, Argentina

³ Imperial College London, UK

Abstract. We present a compositional approach to controller synthesis of discrete event system controllers with linear temporal logic (LTL) goals. We exploit the modular structure of the plant to be controlled, given as a set of labelled transition systems (LTS), to mitigate state explosion that monolithic approaches to synthesis are prone to. Maximally permissive safe controllers are iteratively built for subsets of the plant LTSs by solving weaker control problems. Observational synthesis equivalence is used to reduce the size of the controlled subset of the plant by abstracting away local events. The result of synthesis is also compositional, a set of controllers that when run in parallel ensure the LTL goal. We implement synthesis in the MTSA tool for an expressive subset of LTL, GR(1), and show it computes solutions to that can be up to 1000 times larger than those that the monolithic approach can solve.

Keywords: Modular Discrete Event System Control · Compositional Controller Synthesis · LTL · GR(1)



1 Introduction

The problem of automatically constructing control rules from a specification (i.e., synthesis) has been addressed by different areas in Computer Science including Control of Discrete Event Systems (DES) [28], Reactive Synthesis [27] and Automated Planning [13]. Each has distinct perspectives on representational and computational aspects. All three use different input languages to provide compact problem descriptions for an underlying semantics that can grow exponentially. These different input languages have led to different algorithmic approaches to achieve more efficient ways to solve synthesis problems.

In this paper, we study control of DES described modularly as the parallel composition of interacting transition systems. The semantics is a monolithic automaton whose state space can grow exponentially with respect to the number of intervening components. This state explosion makes tractability of DES control problems a challenge. Modularity has enabled compositional reasoning in supervisory control. Notably, in [25] a compositional framework is described for synthesis of non-blocking supervisory controllers for safety goals. Compositional DES control for liveness goals (e.g., [6, 26, 30]) has not been studied.

We present a compositional synthesis framework for modular DES control problems with LTL goals. Our compositional approach does not compose the transition systems describing the plant and then compute a discrete event controller as does [6]. Instead, roughly, we iteratively pick a subset of plant transition systems, we compute a safe controller for the subset, and replace the subset with a minimized controlled version of the subset that abstracts events not shared with the rest of the plant. Each iteration does a *partial synthesis* of the original control problem. The *final result is a controller expressed as a set of controllers* that can be run in parallel to control the original plant, reducing memory footprint and mitigating state explosion of the composed controller.

Our approach is inspired by [25] for building modular non-blocking supervisors for finite regular languages.

Moving to ω -regular languages such as those expressed using LTL introduces various challenges. First, in contrast to supervisory control for non-blocking supervisors, maximally permissive controllers for LTL control problems are not guaranteed to exist. We compute *safe controllers* for a weaker goal, that allows maximally permissive controllers and supports deferring the construction of a *live controller* that ensures LTL in the final step. Second, although we use the synthesis observational equivalence [25] for minimization, the equivalence does not actually preserve LTL equirealizability as it introduces uncontrollable loops which require special considerations. Last, in [25] non-blocking conditions are represented modularly with marked states in each plant automaton. Thus, non-blocking can be analysed straightforwardly for subplants. In contrast, LTL is a condition that refers to the behaviour of the composed plant and must be decomposed when analysing subplants.

We assess if the compositional synthesis can perform better than a monolithic approach in which the plant composition is constructed in full. For this we implemented the approach in the MTSA tool [5] which is the only tool we are aware of that is available and *natively* supports controller synthesis for modular DES plants. MTSA supports a specific class of LTL formulae referred to as GR(1) [26]. *We show that using the same GR(1) explicit state synthesis engine, implemented in MTSA, the compositional approach solves control problems up to $1000\times$ larger than what a monolithic approach can.* In addition, we show that the compositional framework can sometimes yield more effective results when symbolic synthesis engines such as Strix [23] and Spectra [20] are used for DES controller synthesis via a translation [16].

The remainder of this paper presents background in Section 2, an overview in Section 3, novel definitions and results for compositional synthesis in Section 4, the compositional algorithm in Section 5, and an evaluation in Section 6. We conclude in Sections 7 and 8 with related work and conclusions.

2 Preliminaries

We model discrete event systems by composing Labelled Transition Systems.

Definition 1 (Labelled Transition System). A *Labelled Transition System (LTS)* is a tuple $(S, \Sigma, \rightarrow, \hat{s})$ where S is a finite set of states; Σ is a finite set of event labels; $\rightarrow \subseteq S \times \Sigma \times S$ are transitions; and $\hat{s} \in S$ is the initial state.

Definition 2 (Parallel composition). Let $M_i = (S_i, \Sigma_i, \rightarrow_{M_i}, \hat{s}_i)$ with $i \in \{1, 2\}$ be two LTSs, their parallel composition is an LTS $M_1 || M_2 = (S_1 \times S_2, \Sigma_1 \cup \Sigma_2, \rightarrow_{M_1 || M_2}, (\hat{s}_1, \hat{s}_2))$, where $\rightarrow_{M_1 || M_2}$ is the smallest relation that satisfies:

- if $s_1 \xrightarrow{l}_{M_1} s'_1$ and $l \in \Sigma_1 \setminus \Sigma_2$, then $(s_1, s_2) \xrightarrow{l}_{M_1 || M_2} (s'_1, s_2)$
- if $s_2 \xrightarrow{l}_{M_2} s'_2$ and $l \in \Sigma_2 \setminus \Sigma_1$, then $(s_1, s_2) \xrightarrow{l}_{M_1 || M_2} (s_1, s'_2)$
- if $s_1 \xrightarrow{l}_{M_1} s'_1, s_2 \xrightarrow{l}_{M_2} s'_2$ and $l \in \Sigma_1 \cap \Sigma_2$, then $(s_1, s_2) \xrightarrow{l}_{M_1 || M_2} (s'_1, s'_2)$

Parallel composition is associative and commutative. If $\mathcal{M} = \{M_1, M_2, \dots, M_n\}$ is a set of LTS, we may refer to $M_1 || M_2 || \dots || M_n$ as $\mathcal{M}$ too.

We use Σ^* for the set of all finite sequences of labels of Σ , i.e., traces. We say $l \in \Sigma$ is enabled in s if there is s' such that $(s, l, s') \in \rightarrow$. We denote $\rightarrow(s)$ as the set of enabled events in s . An LTS is deadlock-free if all states have outgoing transitions. An LTS is deterministic if there is no state that has two outgoing transitions on the same event. We use $\rightarrow$ to denote the transitive closure of transitions and assume it holds for the empty trace: $x \xrightarrow{\epsilon} x$. We say a (finite or infinite) sequence w of labels in Σ is a trace of M if there is an s' such that $\hat{s}_M \xrightarrow{w} s'$. The concatenation of two traces s and t is written as st . For $\Omega \subseteq \Sigma$, we define the natural projection $P_\Omega : \Sigma^* \rightarrow \Omega^*$ as the operation that removes from a trace $\pi \in \Sigma^*$ all events not in Ω .

We model controllers with legal LTS that must not block the DES.

Definition 3 (Legal LTS). Given LTS $M = (S_M, \Sigma, \rightarrow_M, \hat{s}_M)$ and $C = (S_C, \Sigma_C, \rightarrow_C, \hat{s}_C)$ with $\Sigma_u \in \Sigma$. We say that C is a legal LTS for M with respect to Σ_u , if for all (s_M, s_C) such that $(\hat{s}_M, \hat{s}_C) \rightarrow_{M || C} (s_M, s_C)$ the following holds: $\rightarrow_{M || C}((s_M, s_C)) \cap \Sigma_u = \rightarrow_M(s_M) \cap \Sigma_u$

We use Linear Temporal Logic to declaratively specify control requirements.

Definition 4 (Linear Temporal Logic (LTL) formula). An LTL formula is defined inductively using the standard Boolean connectives and temporal operators X (next), U (strong until) as follows: $\phi := e \mid \neg\phi \mid \phi \vee \psi \mid X\phi \mid \phi U \psi$, where e is an atomic event. As usual we introduce $\Diamond$ (eventually), and $\Box$ (always) as syntactic sugar. The semantics of an LTL formula is defined via a relation $\models$ between infinite traces and LTL formulae, noted $\pi \models \phi$.

We now define discrete event control problems.

Definition 5 (Control Problem). Given a set of deterministic LTS $\mathcal{M}$ modelling the plant to be controlled, Σ the union of the alphabets of the LTS in $\mathcal{M}$, a set of controllable events $\Sigma_c \subseteq \Sigma$ and an LTL formula φ over Σ modelling the controller goal, we define $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$ to be an Control Problem. A deterministic LTS C is a solution for $\mathcal{E}$ (i.e., a controller) if C is legal for $\mathcal{M}$ with respect to the set of uncontrolled events $\Sigma_u = \Sigma \setminus \Sigma_c$, $\mathcal{M} || C$ is deadlock-free and for every infinite trace π in $\mathcal{M} || C$ we have $\pi \models \varphi$.

$\mathcal{E}$ is realizable if a solution for $\mathcal{E}$ exists. A solution C is *maximally permissive* [15] if for any solution C' , if π is a trace of C' then it is a trace of C .

Definition 6 (Winning States). Let $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$ a control problem with $\hat{s}$ the initial state of $\mathcal{M}$. We say $s \in S_{\mathcal{M}}$ is a winning state if there is a solution C for $\mathcal{E}$ with initial state $\hat{c}$, a state $c \in S_c$ and a trace π such that $(\hat{c}, \hat{s}) \xrightarrow{\pi}_{C \parallel M} (c, s)$.

We focus on a particular form of controller goals referred to as GR(1).

Definition 7 (Generalised Reactivity(1) [26]). A GR(1) formula φ is an LTL formula restricted with this structure $\varphi = \bigwedge_{i \in n} \Box \Diamond \phi_i \implies \bigwedge_{j \in m} \Box \Diamond \gamma_j$ where ϕ_i and γ_j are boolean combination of events.

Note that realizable GR(1) control problems may not have a maximally permissive controller. An intuition is that a controller may delay achieving its goal an unbounded but finite number of times. This cannot be captured as an LTS.

Also note that for a realizable GR(1) control problem, if a plant state is losing (i.e., not winning) it will never be traversed when the plant is being controlled. Also note that stripping a plant of its losing states not only preserves realizability but also results in the smallest sub-plant that preserves realizability.

We use an observational equivalence taken from [25]. In [25] the relation to reduce the size of the plant, hiding local events $\mathcal{T}$ while preserving solutions to supervisory control problems with non-blocking goals. The equivalence does not preserve realizability of GR(1) control problems; an adaptation is required.

Definition 8 (Synthesis observation equivalence [25]). Let $M = (S, \Sigma, \rightarrow, \hat{s})$ be a LTS with $\Sigma = \Omega \dot{\cup} \Upsilon$, Σ_c is the set of controllable events and Σ_u is the set of uncontrollable events. An equivalence relation $\sim_{\Upsilon} \subseteq S \times S$ is a synthesis observational equivalence on M with respect to Υ if the following conditions hold for all $x_1, x_2 \in S$ such that $x_1 \sim_{\Upsilon} x_2$:

1. if $x_1 \xrightarrow{u} y_1$ for $u \in \Sigma_u$, then there exist $\pi_1, \pi_2 \in (\Upsilon \cap \Sigma_u)^*$ such that $x_2 \xrightarrow{\pi_1 P_{\Omega}(u) \pi_2} y_2$ and $y_1 \sim_{\Upsilon} y_2$
2. if $x_1 \xrightarrow{c} y_1$ for $c \in \Sigma_c$, then there exists a path $x_2 = x_2^0 \xrightarrow{\tau_1} \dots \xrightarrow{\tau_n} x_2^n \xrightarrow{P_{\Omega}(c)} y_2$ such that $y_1 \sim_{\Upsilon} y_2$ and $\tau_1, \dots, \tau_n \in \Upsilon$, and if $\tau_i \in \{1..n\} \in \Sigma_c$ then $x_1 \sim_{\Upsilon} x_2^i$

Definition 9 (Quotient LTS). Let $M = (S, \Sigma, \rightarrow, \hat{s})$ be an LTS, and $\sim \subseteq S \times S$ be an equivalence relation. The quotient LTS of M modulo $\sim$ is $M/\sim = (S/\sim, \Sigma, \rightarrow/\sim, [\hat{s}])$ where $S/\sim$ is the set of equivalence classes of $\sim$ and $\rightarrow/\sim = \{([s], \sigma, [s']) \mid s \xrightarrow{\sigma} s'\}$

3 Overview

In this section we first provide an overview of our approach that does not include an equivalence minimization step. We then revisit the overview explaining where some aspects of minimization require special treatment.

Overview without Minimization. Consider a control problem $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$ where $\mathcal{M}$ is a set of LTS. A monolithic approach to solving the problem (e.g., [6]) composes all the LTSs in $\mathcal{M}$ and then computes a controller. Our approach starts with $\mathcal{M}$ and an empty set $\mathcal{C}$ which will store *safe controllers* (we discuss them below). We iteratively extract a subset $\mathcal{M}'$ of $\mathcal{M}$ of size greater than 1, compose the LTSs in $\mathcal{M}'$, compute a safe controller for the composition (with a weaker property φ' and a larger controllable alphabet – we explain why below). and store it in $\mathcal{C}$. In addition, we reintroduce a controlled version of $\mathcal{M}'$ to $\mathcal{M}$, we refer to this LTS as an *controlled subplant* (we explain these below).

In each iteration, $\mathcal{C}$ increases by one and $\mathcal{M}$ decreases by at least one. The iteration ends if the subset $\mathcal{M}'$ (which is picked by a heuristic) is equal to $\mathcal{M}$, typically when $\mathcal{M}$'s size is 1 or 2. At this point we apply the standard monolithic approach for φ to $\mathcal{M}'$. The controller built in this last step of the algorithm can be thought of as a *live controller* in contrast to the safe controllers built before.

The *solution* to the original control problem $\mathcal{E}$ is the composition of the safe controllers in $\mathcal{C}$ with the live controller computed after the iterations.

Safe Controllers. We provide an intuition of what is referred to as safe controllers and why they are computed for a property φ' that is weaker than φ and an expanded controllable alphabet. We do so by working through an example.

Consider the control problem $\langle \mathcal{M}, \Sigma_c, \varphi \rangle$ depicted in Figure 1. A monolithic synthesis approach would first compose the three LTSs in $\mathcal{M}$ and then compute a controller. The composition (see Fig. 10 of the Appendix) has 23 states.

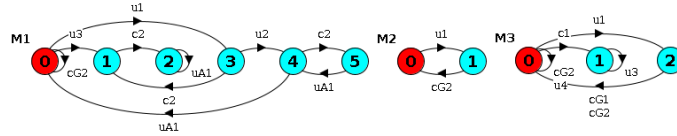


Fig. 1. Modular plant for a control problem $\mathcal{E}$ with $\mathcal{M} = \{M_1, M_2, M_3\}$, GR(1) goal $\varphi = \Box\Diamond uA1 \implies \Box\Diamond cG1 \wedge \Box\Diamond cG2$ and controllable alphabet $\Sigma_c = \{cG1, cG2, c1, c2\}$.

Consider a first iteration of our algorithm in which a heuristic selects the subset $\mathcal{M}' = \{M_1, M_2\}$ and computes its composition (see Fig. 2).

We would like to control $\mathcal{M}'$ to achieve φ . However, this is not possible as φ refers to $cG1$ that is not in the alphabet of the LTS in $\mathcal{M}'$. Thus, we weaken φ by *projecting* it to the alphabet of $\mathcal{M}'$: $\varphi' = \Box\Diamond uA1 \implies \top \wedge \Box\Diamond cG2$.

Now we would like to control $\mathcal{M}'$ to achieve the weaker φ' . Yet this is not possible because a controller cannot prevent the occurrence of $u3$ in the initial state of $\mathcal{M}'$. But event $u3$ would never occur in the initial state of the plant $\mathcal{M}$ because $M_3 \notin \mathcal{M}'$ disallows it in its initial state! Indeed, when computing a controller for $\mathcal{M}'$ we must account for the fact that certain aspects of the problem may be solvable when considering the rest of the plant. For this we

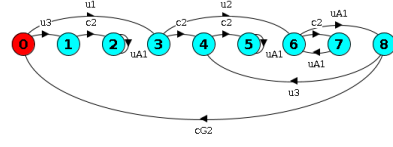


Fig. 2. Parallel composition of subplant $\mathcal{M}' = \{M_1, M_2\} \subset \mathcal{M}$, i.e., $M_1 \parallel M_2$.

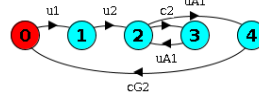


Fig. 3. Safe Controller $(M_1 \parallel M_2)_{safe}$ for subplant $M_1 \parallel M_2$, $\varphi' = \Box \Diamond uA1 \implies \top \wedge \Box \Diamond cG2$, controllable events $\Sigma_c \cup \{u3\}$, and $\mu = \emptyset$.

expand the controllable alphabet to include all uncontrollable events shared with LTS in $\mathcal{M} \setminus \mathcal{M}'$ (i.e., $u3$).

We now explain why we compute a *safe* controller for the weaker formula φ' and the extended set of controllable events: There may be controllers that can realize φ' in different ways (often called strategies). Some of these strategies may be conflicting with restrictions imposed by LTS in $\mathcal{M} \setminus \mathcal{M}'$ or parts of the goal lost in the projection of φ . Ideally, we would compute a controller that includes all such strategies, but this is impossible. Liveness goals such as GR(1) do not allow maximally permissive controllers. Instead of constructing a controller that achieves φ' we build one that avoids states in $\mathcal{M}'$ that a controller that guarantees φ' may traverse. In other words, we build a controller that stays in winning states of φ' . This is a safety property, and if there is a controller, there is a maximally permissive one.

The safe (maximally permissive) controller for $\mathcal{M}'$, φ , and $\Sigma_c \cup \{u3\}$, referred to as $(M_1 \parallel M_2)_{safe}$ is shown in Fig.3 and is added to $\mathcal{C}$.

Controlled Subplant. We now provide an intuition of what controlled subplants are. Having computed a safe controller for $M_1 \parallel M_2$ for a weaker goal than φ we must describe what remains to be controlled.

A naive approach would be to add to $\mathcal{M}$ the controlled version of $M_1 \parallel M_2$, in other words $(M_1 \parallel M_2)_{safe} \parallel (M_1 \parallel M_2)$. However, the $(M_1 \parallel M_2)_{safe}$ was built assuming it controlled $u3$ when in fact it does not. Indeed, the composition would not let the subplant $M_1 \parallel M_2$ take a $u3$ transitions from states 0 and 8.

To adequately model the behavior of the controlled subplant (see Fig.4) we must re-introduce $u3$ transitions to the safe controller. Specifically, we must add them to states 0 and 4 of the controller, these are the states the controller would be in if it were composed with the subplant and the subplant were in states 0 or 8. Added transitions are sent to a deadlock state because we know that from this point on, it is impossible to achieve φ' and thus impossible to achieve φ .

Indeed, the safe controller with the added transitions and deadlock state represent the safe controller with the assumptions it makes on when uncontrollable shared events should not happen. The deadlock state is in a sense abstracting states 1, 2, 4 and 5 of the subplant $M_1 \parallel M_2$ of Fig. 2 as bad states.

After the First Iteration. At the end of the first iteration of our simplified algorithm we have: $\mathcal{M} = \{M_3, (M_1 \parallel M_2)_{cont}\}$ and $\mathcal{C} = \{(M_1 \parallel M_2)_{safe}\}$.

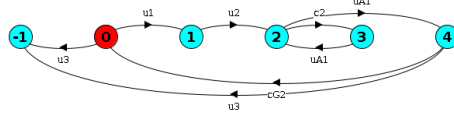


Fig. 4. Controlled Subplant $(M_1||M_2)_{cont}$ built from the safe controller $(M_1||M_2)_{safe}$ in Fig. 3 by adding $u3$ transitions from state 0 and state 4 to a deadlock state

As the subplants are required to be of size larger than 1, the only possible $\mathcal{M}' \subseteq \mathcal{M}$ is actually $\mathcal{M}$. Here termination of the iterations kicks in and the algorithm proceeds to build a live controller using a monolithic approach for: $\langle \{M_3, (M_1||M_2)_{cont}\}, \Sigma_c, \varphi \rangle$. In other words, it builds the plant $M_3||(M_1||M_2)_{cont}$ (see Fig. 11 in Appendix), computes a controller $(M_3||(M_1||M_2)_{cont})_{live}$ (see Fig. 5) and returns the solution $\mathcal{C} = \{(M_3||(M_1||M_2)_{cont})_{live}, (M_1||M_2)_{safe}\}$.

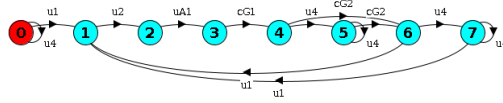


Fig. 5. Controller $(M_3||(M_1||M_2)_{cont})_{live}$ for Fig. 11.

Note that the largest LTS for which the compositional algorithm built a controller was of size 14, whilst a monolithic approach applied to $M_1||M_2||M_3$ would have had to analyse an LTS of size 22.

Introducing Minimization. The approach to synthesis presented formally in the next sections relies heavily on minimization to perform well: It minimizes the controlled subplants before adding them to $\mathcal{M}$. The minimization procedure preserves the synthesis observational equivalence defined in [24]. We apply minimization, as does [24], hiding events that are local to the controlled subplant (i.e., events that are not part of the alphabet of the remaining LTSs in $\mathcal{M}$). In Fig. 6 we depict the minimized version of $(M_1||M_2)_{cont}$ using local events $\mathcal{I} = \{u2, uA1, c2\}$. Note that minimization may introduce non-determinism that can be solved, as in [24] using relabeling.

The approach proceeds by composing $(M_1||M_2)_{cont}/\sim_{\mathcal{I}}$ and M_3 (see Fig.7) and attempts to build a controller for φ . However, such a controller does not exist as the trace $u1.uA1^\omega$ cannot be avoided. Yet such a trace is not possible in $(M_1||M_2||M_3)$ (see Fig.10 in Appendix). This mismatch can be traced back to the minimization procedure that introduces a looping transition labelled $uA1$ in state 1 in $(M_1||M_2)_{cont}/\sim_{\mathcal{I}}$ which $(M_1||M_2)_{cont}$ does not have. Note that these loops are not a problem for compositional control for finite traces such as in [24].

Having introduced uncontrollable looping transitions, they must be kept track of (we add them to a set μ) and solve control problems in future iter-

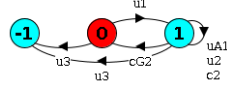


Fig. 6. $(M_1 \parallel M_2)_{cont} / \sim_{\mathcal{R}}$, a minimized version of $(M_1 \parallel M_2)_{cont}$ with local events $\mathcal{R} = \{u2, uA1, c2\}$.

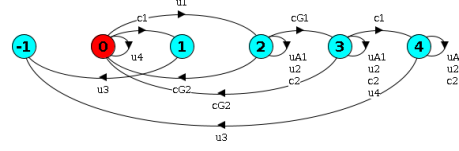


Fig. 7. Plant $(M_1 \parallel M_2)^c / \sim_{\mathcal{R}} \parallel M_3$ to be controlled assuming $\Box \Diamond (\neg u2 \wedge \neg uA1 \wedge \neg c2)$

ations that assume events in μ do not occur infinitely (i.e., $\Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$). Although not a GR(1) formula, it can be solved using GR(1) synthesis.

The live controller for $(M_1 \parallel M_2)_{cont} / \sim_{\mathcal{R}} \parallel M_3$ and property $\Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$ is depicted on the left of Fig. 8. This live controller plus the safe controller in Fig. 3 are a modular solution to $\mathcal{E}$.

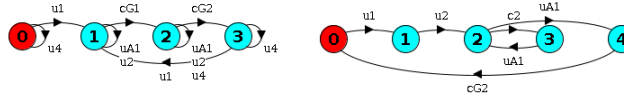


Fig. 8. Modular representation of a controller for the discrete event problem $\mathcal{M} = \{M_1, M_2, M_3\}$ to be controlled to satisfy $\varphi = \Box \Diamond uA1 \implies \Box \Diamond cG1 \wedge \Box \Diamond cG2$ with controllable alphabet $\Sigma_c = \{cG1, cG2, c1, c2, c3\}$. Controller on the left is a solution to Fig.7, controller on the right is the safe controller from Fig. 3.

Note that using minimization, the largest LTS for which synthesis was computed for has 5 states, while a monolithic approach would have computed a controller for a plant of 22 states. This exemplifies how state explosion of the composed plant might be avoided using our compositional approach.

Subplant minimization can result in a non-deterministic monitor. To remove non-determinism we use the same procedure as in [24]. Due to space restrictions we do not discuss renaming in this paper, but our implementation and experimentation includes it. We refer the reader to [24] for more information.

4 Compositional Synthesis

This section defines a data structure used by the compositional synthesis algorithm, two operations over the data structure, and theorems that enunciate when these operations preserve the semantics of the data structure.

During algorithm execution three variables are maintained, $(\mathcal{M}, \mathcal{C}$ and $\mu)$ that form what we refer to as a synthesis tuple. They were discussed in Section 3.

Definition 10 (Synthesis tuple). A synthesis tuple is tuple $(\mathcal{M}, \mathcal{C}, \mu)$, where $\mathcal{M}$ is a set of deterministic LTSs modelling a modular plant, $\mathcal{C}$ is a set of LTSs modelling safe controllers, and μ is a set of events in the alphabet of $\mathcal{M}$.

When solving a control problem $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$, the algorithm starts with a synthesis tuple of the following form: $(\mathcal{M}, \emptyset, \emptyset)$. The algorithm will iteratively reduce the size of $\mathcal{M}$ while populating $\mathcal{C}$ and μ . Each intermediate step will yield a synthesis tuple that preserves the control problem to be solved. To formalize this notion of preservation we define the set of controllers that are a solution to a synthesis tuple. This set of controllers will serve as an invariant.

Definition 11 (Controllers of a Synthesis Tuple). *Let $T = (\mathcal{M}, \mathcal{C}, \mu)$ be a synthesis tuple with $\mathcal{C} = \{C_1, \dots, C_n\}$, $\Sigma_c \subseteq \Sigma$ a controllable alphabet, and φ an LTL formula. We define the controllers of a synthesis tuple as $Cont_{\Sigma_c, \varphi}(T) = \{\mathcal{C}_M || C_1 || \dots || C_n \cdot \mathcal{C}_M \mid \mathcal{C}_M \text{ is a solution to } \mathcal{E}\}$ where $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi \rangle$.*

Remark 1. Given the synthesis tuple $T = (\mathcal{M}, \emptyset, \emptyset)$ then the controllers in $Cont_{\Sigma_c, \varphi}(T)$ are the same as those for the control problem $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$.

We say that two synthesis tuples T and T' are equivalent with respect to a controllable alphabet Σ_c and an LTL formula φ if they have the same set of controllers: $T \cong_{\Sigma_c, \varphi} T'$ iff $Cont_{\Sigma_c, \varphi}(T) = Cont_{\Sigma_c, \varphi}(T')$.

In the following we introduce two equivalence preserving transformation operations over synthesis tuples. The compositional synthesis algorithm in Section 5 iteratively applies these equivalence preserving transformations.

4.1 Composition

This synthesis tuple transformation method reduces the size of $\mathcal{M}$ in a synthesis tuple by composing in parallel a subset of the LTS in $\mathcal{M}$. It is straightforward to see that parallel composition preserves equivalence of synthesis tuples.

Theorem 1 (Composition preserves controllers). *Let φ be an LTL formula, Σ_c a set of controllable events, and T, T' synthesis tuples $(\mathcal{M}, \mathcal{C}, \mu)$ and $(\mathcal{M}', \mathcal{C}, \mu)$ respectively. If $\mathcal{M}_1$ and $\mathcal{M}_2$ are a partition of $\mathcal{M}$ such as $\mathcal{M}' = \mathcal{M}_1 \cup \mathcal{M}_2$ where $\mathcal{M}_1$ is the parallel composition of the LTS in $\mathcal{M}_1$, then $T \cong_{\Sigma_c, \varphi} T'$.*

The theorem follows from the fact that parallel composition is associative.

4.2 Partial Synthesis

Partial Synthesis is the key transformation operation over synthesis tuples. Given a tuple $(\mathcal{M}, \mathcal{C}, \mu)$ it removes one LTS M from $\mathcal{M}$, computes a safe controller C for a weaker property φ' and adds the safe controller to $\mathcal{C}$. It also adds a minimized LTS modelling the controlled subplant, i.e. the result of C controlling M_i but hiding local events $\mathcal{T}$. In addition it updates μ .

We define the projection of a formula the alphabet of a subsystem.

Definition 12 (Formula projection). *A formula projection is a function $\alpha : LTL \times 2^\Sigma \rightarrow LTL$ such that for any formula φ and set $\Sigma' \subseteq \Sigma$ we have $\varphi \implies \alpha(\varphi, \Sigma')$ and $\alpha(\varphi, \Sigma')$ is a formula that only refers to events in Σ' .*

To achieve completeness, it is essential partial controllers built are maximally permissive. That is, there is a controller that is maximal in terms of traces it allows. Maximally permissive controllers in general do not exist. Therefore, we construct *safe controllers* that avoid reaching a state in the plant from which achieving a the goal is certainly impossible. That is, a controller that keeps the plant in potentially winning states but does not guarantee traces that satisfy the liveness property. Liveness is achieved in the algorithm's last step.

Definition 13 (Safe Controller). Let $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$ be a control problem with $\mathcal{M} = (S, \Sigma, \rightarrow, \hat{s})$. If $\mathcal{E}$ is realizable, we say $(\hat{S}, \Sigma, \rightarrow_S, \hat{s})$ is the safe controller for $\mathcal{E}$ where $\hat{S}$ is a superset of the set of winning states in S for $\mathcal{E}$ and $\rightarrow_S$ contains all the transitions in $\mathcal{M}$ between states $s \in \hat{S}$.

As explained in the overview, a model of the subplant controlled by the safe controller must be reintroduced into $\mathcal{M}$, however this cannot be done by simply introducing their parallel composition as the safe controller will have been built with an extended controllable alphabet. We build a *controlled subplant* LTS that models M being controlled by the safe controller C with shared uncontrollable events Ω . Recall from the overview (see Fig. 4) that we must add to the safe controller transitions to a deadlock state representing the safe controller's assumptions on when uncontrolled shared events should not happen.

Definition 14 (Controlled Subplant). Let $C = (S_C, \Sigma_C, \rightarrow_C, \hat{s}_C)$ be a safe controller for $\mathcal{E} = \langle M, \Sigma_c \cup \Omega, \varphi \rangle$ with $M = \langle S_M, \Sigma_M, \rightarrow_M, \hat{s}_M \rangle$, φ an LTL formula, and $\Omega \subseteq \Sigma_u$ the set of uncontrolled shared events. The controlled subplant M by C with respect to Ω is $\langle S_C \cup \{\perp\}, \Sigma_C, \rightarrow, \hat{s}_C \rangle$ where $\rightarrow = \rightarrow_C \cup \{(s, l, \perp) \mid l \in \Omega \wedge s \in S_C \wedge \exists s' \in S_M \cdot s \xrightarrow{l}_M s' \wedge \forall s' \in S_M \cdot (s, l, s') \notin \rightarrow_C\}$.

We now enunciate how the definitions above can be used to evolve a synthesis tuple by computing a safe controller and adding a minimized controlled subplant.

Roughly, the theorem states that the solution to a synthesis tuple remains the same if given a tuple $(\mathcal{M}, \mathcal{C}, \mu)$, (i) we take one of the LTS in the plant to be controlled ($M_1 \in \mathcal{M}$), (ii) compute a safe controller for M_1 , the projection of φ to the alphabet of M_1 , and an controllable alphabet extended with the uncontrollable events shared by M_1 and $\mathcal{M} \setminus \{M_1\}$, (iii) add the safe controller to $\mathcal{C}$, (iv) add a minimized version of the subplant M_1 controlled by the safe controller to $\mathcal{M}$ by quotienting it to remove transitions on local events, and (v) add any uncontrollable events on self loops induced by the quotient into μ . The proof of this theorem is included in the appendix.

Theorem 2 (Partial synthesis preserves controllers). Given a synthesis tuple $(\mathcal{M}, \mathcal{C}, \mu)$, an LTL formula φ , and a set Σ_c of controllable events. Let $M \subseteq \mathcal{M}$ with alphabet $\Sigma_M = \Upsilon \dot{\cup} \Omega$ where Υ has the events not shared with any other LTS in $\mathcal{M} \setminus M$ nor in φ . Let φ' be a projection of φ to Σ_M . Let M_{safe} be a safe controller for $\langle M, \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi' \rangle$. Let M_{cont} be the subplant M controlled by M_{safe} with shared events Ω .

If $M_{cont} / \sim_{\Upsilon}$ is deterministic and μ' is the set of self-loop events added by $\sim_{\Upsilon}$ reduction, then $(\mathcal{M}, \mathcal{C}, \mu) \cong_{\Sigma_c, \varphi} (\mathcal{M} \setminus \{M\} \cup \{M_{cont} / \sim_{\Upsilon}\}, \mathcal{C} \cup \{M_{safe}\}, \mu \cup \mu')$.

Note that the theorem requires $M_{cont}/\sim_{\mathcal{Y}}$ to be deterministic, which is not always the case. Non-determinism can be avoided by renaming labels in M and using distinguishers [24]. We avoid presenting these due to space restrictions.

5 Compositional Synthesis Algorithm

The algorithm has as inputs a control problem $\mathcal{E} = \langle \mathcal{M}, \varphi, \Sigma_c \rangle$ and a heuristic that returns a subset of the plant of size greater than 1.

CompSynthesis ($\langle \mathcal{M}, \varphi, \Sigma_c \rangle$, heuristic)	PartialSynthesis ($\mathcal{M}, \mathcal{C}, \text{sp}, \Sigma_c, \varphi$)
1: $\mathcal{C} \leftarrow \{\}, \mu \leftarrow \{\}$	1: $\varphi' \leftarrow \text{getProjection}(\varphi, \Sigma_{\text{sp}})$
2: while <i>true</i> do	2: $\Sigma_{\varphi'} \leftarrow \text{getAlphabet}(\varphi')$
3: $\text{sp} \leftarrow \text{getSubplant}(\mathcal{M}, \text{heuristic})$	3: $\mathcal{Y} \leftarrow \Sigma_{\text{sp}} \setminus (\Sigma_{\mathcal{M}} \cup \Sigma_{\varphi'})$
4: $\mathcal{M} \leftarrow \mathcal{M} \setminus \text{sp}$	4: $M_{\text{safe}} \leftarrow \text{getSafeController}_{\mathcal{Y}}(\langle \text{sp}, \Sigma_c, \varphi' \rangle, \mu)$
5: if $ \mathcal{M} == 0$ then	5: if $\neg \text{isRealizable}(M_{\text{safe}})$ then
6: $\text{lc} \leftarrow \text{getLiveController}(\langle \text{sp}, \Sigma_c, \varphi \rangle, \mu)$	6: return UNREALIZABLE
7: if $\neg \text{isRealizable}(\text{lc})$ then	7: end if
8: return UNREALIZABLE	8: $M_{\text{cont}} \leftarrow \text{getControlledSubplant}_{\mathcal{Y}}(M_{\text{safe}}, \text{sp})$
9: end if	9: $\mathcal{C} \leftarrow \mathcal{C} \cup M_{\text{safe}}$
10: return $\mathcal{C} \cup \{\text{lc}\}$	10: $\langle M_{\text{cont}}^{\sim}, \mu' \rangle \leftarrow \text{minimize}(M_{\text{cont}}, \mathcal{Y})$
11: end if	11: $\mu \leftarrow \mu \cup \mu', \mathcal{M} \leftarrow \mathcal{M} \cup \{M_{\text{cont}}^{\sim}\}$
12: $\text{PartialSynthesis}(\mathcal{M}, \mathcal{C}, \text{sp}, \Sigma_c, \varphi)$	
13: end while	

The synthesis tuple is initialized and a loop that strictly monotonically reduces the size of $\mathcal{M}$ starts. In the loop, the heuristic first selects and removes a subset of $\mathcal{M}$. If $\mathcal{M}$ is not empty, we project φ to the alphabet of the subplant, compute the local alphabet ($\mathcal{Y}$) used to perform minimization, compute the safe controller and the controlled subplant. If there is no solution when computing the safe controller, then the algorithm returns UNREALIZABLE. Otherwise, it adds the safe controller to $\mathcal{C}$ and computes both the quotient of the controlled subplant and the events that the quotient introduces loops for.

The loop terminates when the subplant selected has all LTS in $\mathcal{M}$. At this point we compute a live solution for the remaining control problem and return it with all previously computed partial controllers.

Procedure *getSubplant* returns a subplant of $\mathcal{M}$ using the *heuristic*. Procedures *getLiveController* and *getSafeController* solve control problems of the form $\langle \text{sp}, \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi \rangle$ according to Definitions 5 and 13. Procedure *getControlledSubplant* returns a controlled subplant according to Definition 14, and *minimize* applies minimization based for the equivalence defined in Definition 8 and also returns the events for which self-loops have been introduced.

The correctness and completeness argument for the algorithm is as follows. The algorithm encodes the original problem $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$ as a synthesis tuple $T = (\mathcal{M}, \emptyset, \emptyset)$. The solutions of T by definition are $\text{Cont}_{\Sigma_c, \varphi}(T)$ are exactly those of $\mathcal{E}' = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$ (see Remark 1 after Definition 11). It is straightforward to see that after each iteration of the while loop, the tuple is updated via two solution preserving operations (parallel composition and partial synthesis). The loop terminates because we assume that the heuristic will never infinitely often return subsets of size 1 when $|\mathcal{M}| > 1$ size greater than one is replaced in $\mathcal{M}$

with its composition. Thus, the set of partial controllers by the algorithm, when composed in parallel is a solution to the original problem.

6 Evaluation

The purpose of the evaluation is to assess if a compositional synthesis of controllers for modular DES plants and LTL goals can perform better than a monolithic approach in which the plant composition is constructed in full.

For this we implemented compositional synthesis within MTSA [6]. The only tool we are aware of that is available and *natively* supports controller synthesis for modular DES plants and infinite-trace goals. By this we mean means that other tools do not support describing the system to be controlled as a parallel composition of a set of DES along with some form of specifying liveness guarantees that are to be achieved. Notably, [19] and [12] do modular DES plants, but only allow specifying safety goals. MTSA tool supports GR(1) formulae. We compare MTSA with and without compositional synthesis.

We also study how reactive synthesis tools perform on translated versions of GR(1) DES control problems, both monolithically and compositionally. Specifically, we study *Spectra* [20] (a synthesis tool specialized in GR(1)) and *Strix* [23] (a compositional LTL synthesis tool, winner at SyntComp 2023). Note that we did not implement the compositional approach within *Spectra* and *Strix*, rather we had MTSA call them to measure how much time they take to solve the intermediate control problems compositional synthesis necessitates.

A replication package including tool and control problems is available in [1].

6.1 Compositional Synthesis of DES for GR(1) goals

Applying the composition synthesis algorithm to control problems where φ is a GR(1) formula involves three challenges. First and foremost, procedures *getLiveController* and *getSafeController* must solve control problems that in principle are not GR(1) (i.e., $\Box\Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$). However, these problems can actually be solved using a GR(1) synthesis algorithm due to the shape of the subplant under analysis (the only loops consisting of uncontrollable events that can be found in the subplant are in μ and are self-loops) and the fact that GR(1) formulae are closed under stuttering. The procedure is to remove in the plant all self-loop transitions labeled with some $l \in \mu \subseteq \Sigma_u$, compute a controller for φ , and add to the controller the self-loop transitions previously removed to make it legal. This procedure is correct and complete (see proof in Appendix).

Theorem 3. *Given control problems $\mathcal{E} = \langle \mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}, \Sigma_c, \varphi \rangle$ and $\mathcal{E}' = \langle \mathcal{M}', \Sigma_c, \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi \rangle$ where $\mathcal{M}' = \mathcal{M} \setminus \{M_{sub}\} \cup \{M_{cont}/\sim_{\mathcal{I}}\}$, $\Sigma_c \subseteq \Sigma$ is a set of controllable events, μ and μ' are sets of events and M_{cont} is a controlled subplant for $\langle \mathcal{M}_{sub}, \Sigma_c, \Box\Diamond(\bigwedge_{l \in \mu} \neg l) \implies \varphi \rangle$ where $\mathcal{M}_{sub} \subseteq \mathcal{M}$. If there is a solution $C = (S_c, \Sigma, \rightarrow_c, \hat{s}_c)$ for control problem $\mathcal{E}$, then C' is a solution for $\mathcal{E}'$ where $C' = (S_{c'}, \Sigma, \rightarrow_{c'}, \hat{s}_c)$ is defined as follows:*

- $S_{C'} = S_C$
- $\rightarrow_{C'} \Rightarrow \rightarrow_C \cup \{(s, l, s) \mid s \in S_c \wedge l \in \Sigma_u \cap (\mu \cup \mu') \wedge \exists (s_{\mathcal{M}'}, s_c) \in S_{\mathcal{M}'||C} \cdot (s_{\hat{\mathcal{M}}'}, \hat{s}_C) \rightarrow_{\mathcal{M}'||C} (s_{\mathcal{M}'}, s_C) \wedge (s_{\mathcal{M}'}, l, s_{\mathcal{M}'}) \in \rightarrow_{\mathcal{M}'}\}$.

Also, if there is no solution for $\mathcal{E}$ then there is no solution for $\mathcal{E}'$.

A second challenge is an efficient way to compute a safe controller, that is a controller that keeps a plant in states that are non-losing. Furthermore, that the over-approximation of winning states is sufficiently small to achieve state space reductions that make the compositional approach efficient. This can be achieved for GR(1) formulae straightforwardly as the arena for which the underlying game is solved maps back one-to-one to states of the plant. In addition, the GR(1) synthesis procedure [26] establishes precisely all winning states of the arena.

Finally, projecting a GR(1) formula to a weaker formula can be done straightforwardly. In our implementation we use the following procedure: For formulae of the form $\varphi = \bigwedge_{i \in n} \Box \Diamond \phi_i \implies \bigwedge_{j \in m} \Box \Diamond \gamma_j$ where ϕ_i and γ_j are boolean combination of events in conjunctive normal form and $\Sigma' \in \Sigma$ do the following. For every $l \in \Sigma'$, replace every occurrence of l and $\neg l$ with F in every ϕ_i and replace every occurrence of l and $\neg l$ with $\top$ in every γ_j . It is straightforward to see that this projection yields weaker formulae: $\varphi \implies \alpha(\varphi, \Sigma')$.

6.2 GR(1) DES Synthesis via Reactive Synthesis

To study the performance of GR(1) compositional synthesis for DES via reactive synthesis we implemented a translation of MTSA inputs using an approach based on [17]. Nonetheless, we developed two different translations. The first translates each LTS of the plant and also encodes parallel composition rules. We refer to this translation as *modular*. The second translates the composed plant, a single LTS. We refer to this translation as *non-modular*. Note that the translation is orthogonal to whether monolithic or compositional synthesis will be applied.

For *Strix* we use a binary-encoding for events and local states to reduce the number of atomic propositions needed. Such an encoding is unnecessary for *Spectra* as it supports variables (and performs binary-encoding internally).

For both *Strix* and *Spectra* we first evaluate how they perform using the modular and non-modular translations for various instances of the benchmark. We then take the best performing translation for each tool to evaluate how a monolithic synthesis approach compares with a compositional one.

Monolithic synthesis with *Strix* and *Spectra* is achieved by simply translating the plant of the control problem and executing the reactive synthesis tool.

To avoid modifying *Strix* and *Spectra* to do compositional synthesis, performance is approximated by running a modified version of the compositional MTSA implementation. Every time MTSA calls its internal GR(1) synthesis procedure with a subplant, we mimick the call by calling the synthesis procedure of *Strix* or *Spectra* with a translation of the subplant. The output of reactive synthesis procedure is discarded, and the result of the native GR(1) synthesis procedure is used. We compute the overall time by subtracting the time spent by

	Inst.	Solved		Largest instance solved by both Mono and Comp										State space lgst. inst.	
		Instances		n	k	State space	Comp Perf.		Reduction		Controller				
		Mono	Comp				States	Time	States	Time	LTSs	States	Red.		
AT _R	45	35	20	4	4	2 ³⁸	6244	110s	-91%	-99.63%	9	6545	-95.6%	2 ⁴¹	
AT _I _R	36	21	15	9	1	2 ⁴²	17920	24s	33%	93%	–	–	–	2 ⁴⁶	
BW	81	45	50	9	1	2 ²²	1604	3s	99.6%	96.2%	9	2499	95.6%	2 ²⁸	
CM	81	20	21	4	2	2 ³²	229993	1251s	-51%	-81.1%	7	242865	-91%	2 ³²	
RS	81	17	29	3	3	2 ²⁴	24285	230s	90%	84%	6	212664	94.3%	2 ³⁰	
TA	81	50	71	6	4	2 ⁵⁰	6560	98s	97.2%	93.4%	13	7951	83.3%	2 ⁶⁷	
TL	81	30	29	3	6	2 ²⁰	374993	231s	-6%	-49%	6	377790	-99.8%	2 ²⁰	
DP	10	4	9	5	-	2 ²⁰	185	1s	88%	96.8%	9	391	97.3%	2 ³⁶	
GR	20	8	11	8	-	2 ²⁹	378	1s	99.22%	96%	17	853	78%	2 ³⁸	
MO	4	3	4	6	-	2 ¹⁰⁰	5180	380s	59%	34%	38	14011	44%	2 ¹²⁷	
PC	10	4	10	5	-	2 ²⁸	6561	220s	94.8%	-87%	11	7152	94.3%	2 ⁵⁰	

Table 1. Experimental Results for Native DES GR(1) Controller Synthesis.

internal MTSA synthesis procedure. We refer to these approximations of native compositional implementations as *DES Strix* and *DES Spectra*.

6.3 Benchmark

We use modular DES GR(1) control problems taken from multiple sources. Control problems are parameterizable some scale in two dimensions (number of intervening components, n , and number of states per component, k) and some scale only in the number of intervening components.

For the case studies scalable in two dimensions, we use all problem families introduced in [2], modified from supervisory control problems to GR(1) control problems: Air Traffic (AT), Bidding Workflow (BW) and Travel Agency (TA), Transfer Line (TL), and Cat and Mouse (CM). We also used a Robot Search Mission (RS) taken from robotic literature [31]. All the n - k instances for these problems are known to be realizable, except for those in AT where $n \leq k$ must hold. Thus, we report separately on instances that are realizable (AT_R) and those that are not (AT_I_R). We studied for instances with $n, k \leq 9$.

The one-dimension control problems we use are the classic concurrency Dining Philosophers (DP) problem and the AI planning problem Gripper (GR) [29]. Finally, we also use a SYNTech [20] case study named Moving Obstacle (MO) originally specified as a reactive synthesis problem (i.e., a turn-based game in which each player can update all its variables in each turn). We translated MO manually into a modular discrete event problem by developing one LTS for each safety property of the specification plus some additional LTS to replicate the turn-based logic of reactive synthesis control. The maximum value for n varies depending on how large the plant for the control problems grows.

Experiments were run on a computer with an Intel i7-7700 CPU @ 3.60GHz, with 8GB of RAM, and a timeout of 30 min. In addition, we use a simple deterministic heuristic for selecting subplants that always chooses the first two LTS from the vector representation of the set $\mathcal{M}$.

6.4 Results

Monolithic vs. Compositional GR(1) DES Controller Synthesis. Table 1 shows results for all case studies. The first column has problem family names. The next three aggregate information of entire problem families: we show the total number of instances per problem family (Column Inst) and the number of solved instances by the monolithic (Mono) and compositional (Comp) methods.

These first three columns show that *for 8 out of 10 control problem families, the compositional method improves over the monolithic one* (we count AT_R and $AT_{!R}$ as the same family). In all cases the solved instances one of the methods is a superset of the instances solved by the other.

All problems that the monolithic approach could not solve were due to running out of memory. Also note that providing more RAM to the monolithic method makes little difference in dealing with the combinatorial growth of the state space resulting from increasing the size of problems using k and n . Indeed, doubling memory from 8Gb to 16Gb allows the monolithic approach to add less than 5% more instances. The two problem families in which monolithic performs better than compositional can be explained by looking at the remaining columns.

From the 5th column onwards we focus on the instance with the largest composed plant that was solved by both monolithic and compositional methods. We first report on the values n and k for the instance and its potential state space (i.e., the product of the sizes of all LTS of the plant). We then report the performance of the compositional method by showing the number of states of the largest plant which the compositional approach solves a control problem for, and the end-to-end time to compute the final controller. The next two columns report the reduction in the largest control problem to be solved with respect to the monolithic method, and the reduction in time to build the final controller compared to the monolithic approach. Negative values in this column indicate that the monolithic approach did better than the compositional.

For all but three problem families, when considering the largest instance solved by both methods, the compositional approach was able to significantly reduce the plant on which it computes a live controller. This is an indication of why for these problem families the compositional method has better performance. The exception being $AT_{!R}$ for which, despite achieving reductions, the total number of instances solved by the compositional approach was not greater. This may indicate some bias of our approach towards realizable problems.

The three instances for which we report a negative reduction ($AT_R(n = 4, k = 4)$, $CM(n = 4, k = 2)$, and $TL(n = 3, k = 6)$) show that state explosion of subplants can occur and is a likely cause for poor (or in the case of CM, not significantly better) performance of compositional synthesis.

Notice that treating smaller (sub)plants does not necessarily mean a reduction in time. Instance $PC(n = 5)$ shows that the compositional method is good at reducing size, but a price is paid in time (mostly spent minimizing subplants).

The next three columns, show the size of the controller computed by the compositional method. We report in column LTSs how many LTSs is the controller composed of. The States column reports the sum of the states of each of these

LTS, other words, a measure of the memory required to store the controller on the computer that will enact the controller. We also show the reduction in memory, measured in states, with respect to the (monolithic) controller computed by the monolithic method. *Note that there are significant reductions in the size of the compositional controller for all case studies except for the three in which there are intermediate state explosions of subplants.*

The final column shows the theoretical size of the biggest instance solved by either of the methods. Subtracting column 7 (“state space”) we obtain how much bigger was the biggest problem solved by one approach than the biggest of the other: For 4 out of 10 problem families, the compositional approach was able to solve instances at least 3 orders of magnitude ($\times 2^{10}$) bigger than the monolithic approach. For 3 out 10, the increase in size that the compositional algorithm was able to manage was of at least 2^6 larger. On the other hand for the 3 case studies where monolithic fared better, the largest instances it was able to solve were only 2^6 times bigger.

The final column reports on the state space of the largest instance solved, be it the compositional or the monolithic method. For 4 out of 10 case studies, the compositional approach was able to solve instances at least 3 orders of magnitude ($\times 2^{10}$) bigger than the monolithic approach. For 3 out 10, the increase in size that the compositional algorithm was able to manage was of at least 2^6 larger. On the other hand for the 3 case studies where monolithic fared better, the largest instances it was able to solve were only 2^6 times bigger.

The survival plot on the top left of Figure 9 provides a graphical summary of the comparative performance of monolithic vs. compositional GR(1) DES controller synthesis implemented in MTSA. Overall compositional can solve more (and significantly larger) subjects but takes more time than monolithic.

We now discuss possible causes for variable performance of compositional synthesis across different problem families. We hypothesise that compositional synthesis has opportunities for increased performance when intermediate subsystems can be reduced due to minimization with respect to local events, and when reduction is achieved by reducing reachable states when the subsystem is controlled via partial synthesis. The main threat to performance is, we believe, having intermediate subsystem state explosion as a result of composing LTS that are very loosely synchronized (worst case being that the composition yields the cartesian product of states). Below we analyze two problem families where compositional synthesis performance was worse than monolithic (AT TL) and one in which it was better (DP).

For AT, there are two classes of LTS affected by parameters n and k : The number of airplane LTS is determined by n and the number of height monitors (that check that at a particular height, there is never more than one airplane) is determined by k . The height monitors interact very little with each other, while each one interacts frequently with all airplanes. Additionally, there is always just one ramp and one response monitor. The response monitor interacts with each airplane on a specific label that is local to that airplane.

The simple heuristic used composes incrementally height monitors inducing, early on, an intermediate state explosion as their composition yields a state space that is almost the size of the cartesian product of height monitors' states. In addition, there are no local shared events that can lead to intermediate reductions. We speculate that a heuristic that starts with the response monitor and adds one airplane at a time will perform better because local shared variables provide opportunities for reduction. Indeed, initial explorations seem to indicate this.

In instances of TL, there are LTSs representing the machines connected by buffers on a transfer line and an LTS at the end, the Testing Unit (TU), that takes a work piece from the last buffer and accepts it or rejects and returns it to the first buffer for reprocessing.

The naive heuristic we used selects Machines and Buffers in order from the beginning, thus the TU machine is included in the composition towards the end. These intermediate subsystems have permissive controllers because the projected formula (that only talks about events in the TU) is trivial, so they cannot be reduced by partial synthesis. We speculate that a heuristic that starts with TU and adds consecutive machines and buffers from the end towards the beginning of the line will perform better. Initial exploration indicate this is the case.

In contrast, consider instances of DP, where the compositional approach performs better. The heuristic chooses to add LTSs following the ring architecture defined by the philosophers sitting round the table. Thus, as one more adjacent LTS is added to the subsystem, some new local shared events (between a particular philosopher and a particular fork) can be minimized. Also, since this problem has a formula that refers to events in all philosophers, there are opportunities of minimize by partial synthesis solving control problem for each subsystem.

Monolithic vs Composition GR(1) synthesis via Reactive Synthesis.

Strix performed consistently better when using the non-modular translation than when using the modular one. This provides some evidence that *the compositional approach Strix has built in (where smaller problems are solved by selecting subsets of formulae) is not able to exploit the modular nature of the DES control problem.* Results for *Spectra* were opposite. The tool performed consistently better when applied to modular translations of the control problems instead of non-modular ones. Thus, for the remaining experiments we used modular translations for *Spectra* and non-modular ones for *Strix*.

Top right and bottom left survival plots of Fig. 9 show comparative performance of compositional vs monolithic *DES Strix* and *DES Spectra*. *DES Strix* solved the same number of cases with both approaches, but the monolithic method took over 50% more time. Conversely, bottom left survival plot shows that in *DES Spectra* compositional and monolithic solve the same number of cases, but the monolithic approach is slightly over 20% faster.

A closer look at *DES Spectra* results reveal that the monolithic tends to perform better on problems with shorter and more centralized formulas (i.e., share alphabet with less LTS). By splitting problem families based on whether monolithic or compositional *DES Spectra* does better we can see (bottom right of Fig. 9) that for one set of problem families compositional cumulative time grows

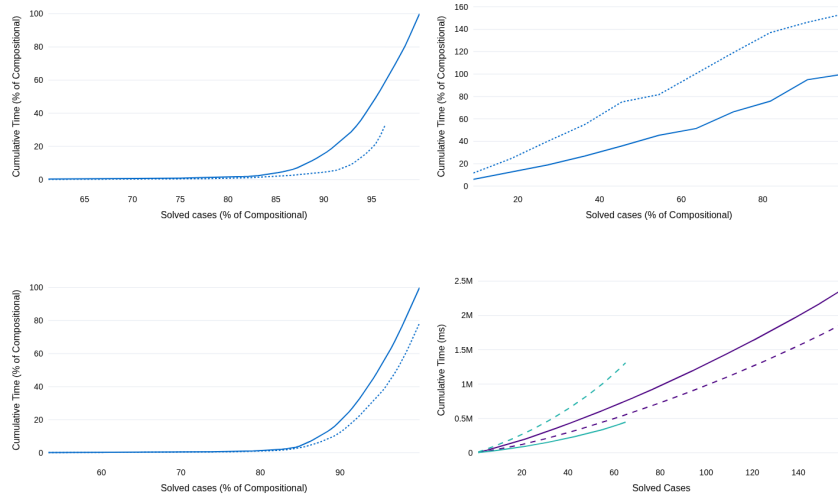


Fig. 9. Survival plots for Compositional (solid) and Monolithic (dashed) for *DES MTSA* -top left-, *DES Strix* -top right-, and *DES Spectra* -bottom left-) relative to compositional performance. Bottom right is for *DES Spectra* divided into problem families for which compositional performs better (purple) and worse (cyan) than monolithic.

much less than that of monolithic, while for the other, compositional cumulative time is higher but grows at a closer rate to that of monolithic.

Summary. The evaluation presented above contributes evidence that compositional synthesis can achieve improvements in scale compared to monolithic DES control, both for native GR(1) DES (explicit state) synthesis, and can achieve improvements in time for DES control via reactive (symbolic state) synthesis.

For the case of native GR(1) DES synthesis, we observe that performance can degrade if there is intermediate state explosion. Exploring heuristics for choosing subplants that may mitigate this problem. In [25] various heuristics are reported to make significant improvements for supervisory control for finite traces. These may also achieve better performance for GR(1) goals.

7 Related Work

Out of the three classic areas that focus on automatically constructing control rules from a specification, Discrete Event Control [28], Reactive Synthesis [27] and Automated Planning [13], this paper is more related to the first two.

Discrete Event Control was first formulated for *-regular languages [28] and then extended to ω -regular languages. In [30], Thistle addresses control problems described using a deterministic finite automaton (DFA) equipped with two acceptance conditions for ω -languages, one representing a specification and the

other representing an assumption. Supervisors are required to generate infinite words that belong to the specification ω -language under the suposition that the environment generates finite traces of the DFA and infinite traces of the assumption ω -language. The supervisor is also required to not block the environment from generating some trace in the assumption ω -language. In [17], this is referred to as a *non-conflicting* requirement. In [30], Thistle presents a supervisor synthesis algorithm that is polynomial in the number of states of the automaton and exponential in the acceptance condition size.

The control problem we study in this paper can be presented as an instance of the general supervisory control problem for ω -regular languages where the ω -regular language for the specification is given as an LTL formula. Whereas the $*$ -regular language for assumptions is given by a composition of LTS (i.e., the plant) and the ω -regular language for assumptions is the entire set of infinite traces of the plant.

Our work is related to reactive synthesis where the control problem is solely expressed by means of a logical formulae, LTL is commonly used. The system-to-be is a reactive module that shares state variables with the environment. Assumed environment behaviour is also described by means of logical formulae. The underlying interaction model between the controller and the environment differs from that of supervisory control. Reactive synthesis assumes a turn based interaction where each part can change in their turn all the variables it controls. In supervisory control, there are no turns and the environment may win all races.

The relation between supervisory control and reactive synthesis has received much attention in the last years [7]. Notably, [17] shows that the non-conflicting/non-blocking requirement of supervisory control requires solving games, obliging games, of higher complexity than those classically used for reactive synthesis. The control problem we study does not suffer from this as the non-conflicting requirement is trivially satisfied.

The class of GR(1) LTL formulae have been studied extensively as controllers for them can be computed polynomially [26] as opposed to the double exponential complexity of general LTL [27]. GR(1) has been studied extensively as a specification language for reactive systems (e.g. [21, 22]) and various variants have been proposed (e.g., [16]).

GR(1) control of DES as defined in this paper was originally proposed in [6], a monolithic synthesis algorithm is defined. Later, on-the-fly composition construction has been proposed [2, 4].

The reactive synthesis community has explored compositionality as a way to scale synthesis by identifying and performing independent synthesis tasks [8–10]. For instance, in [8] LTL synthesis is solved compositionally by first computing winning strategies for each conjunct that appears in the large formula using an incomplete heuristic. The decomposition algorithm of [9] works for LTL and decomposes the formulae into subspecifications that do not share output variables. A dependency graph used for finding the decomposition-critical propositions that, in turn, are key to soundness and completeness of the approach. In [10] decomposition means partitioning system’s input and output variables and then

applying incremental synthesis of dominant strategies. Dependency graphs must be constructed to identify components and synthesis order that would ensure completeness. On the other hand, our technique works compositionally in terms of environment’s safety assumptions (involving input, output and synchronization events) which are described as synchronizing LTS. Goal formula decomposition is a consequence of projection of the formula onto intermediate subsystems. Grouping and ordering of compositions does not affect neither soundness nor completeness (although it may affect performance).

Traditional compositional verification [3, 14] mitigates the state-space explosion by leveraging modular structure of system-under-verification. In that vein, compositional synthesis for plants expressed as finite-state systems has been studied by the community of supervisory control for non-blocking goals [11, 18]. There, synthesis is addressed by incrementally composing (and reducing) plant’s component. Also, resulting controllers are modular: they are made up by automata that interact by mean of shared labels. Our solution is inspired in such works but it addresses infinite trace goals for which there is no guarantee for the existence of maximal solutions.

8 Conclusions and Future Work

The modular representation of discrete event systems can be exploited by composition synthesis to mitigate the state explosion problem that monolithic synthesis suffers. In this paper we present how compositional synthesis can be applied to LTL. We implement compositional synthesis for GR(1) goals and show that with a very basic heuristic for selecting subplants the compositional synthesis already achieves significant gains with respect to a monolithic method, both in a native DES synthesis tool and using reactive synthesis tools via translations.

Future work must look into avoiding intermediate state explosion by means of smarter heuristics that impact the order in which a plant LTS are composed. An implementation of the compositional framework for LTL requires defining a projection function (which can simply be an existential elimination of literals) and a method for computing safe controllers. The latter can be trivialized by the conservative decision of not removing any states at all, to ensure that no winning states are lost. However, this is unlikely to provide good performance results. Thus, the challenge with full LTL is to find an effective way of removing a significant portion of subplant losing states.

Acknowledgements

This work was partially supported by projects ANPCYT PICT 2019-1442, 2021-4862, UBACYT 2023-0134, and CONICET PIP 1220220100470CO.

References

1. Replication package: tool and control problems.
<https://zenodo.org/records/13913744> (2025)

2. Ciolek, D.A., Braberman, V.A., D'Ippolito, N.R., Uchitel, S., Sardiña, S.: Compositional supervisory control via reactive synthesis and automated planning. *IEEE Transactions on Automatic Control* **65**(8), 3502–3516 (2020). <https://doi.org/10.1109/TAC.2019.2948270>
3. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. In: Sethi, R. (ed.) *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Albuquerque, New Mexico, USA, January 19–22, 1992. pp. 342–354. ACM Press (1992). <https://doi.org/10.1145/143165.143235>, <https://doi.org/10.1145/143165.143235>
4. Delgado, T., Sorondo, M.S., Braberman, V.A., Uchitel, S.: Exploration policies for on-the-fly controller synthesis: A reinforcement learning approach. In: Koenig, S., Stern, R., Vallati, M. (eds.) *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling*, Prague, Czech Republic, July 8–13, 2023. pp. 569–577. AAAI Press (2023). <https://doi.org/10.1609/ICAPS.V33I1.27238>, <https://doi.org/10.1609/icaps.v33i1.27238>
5. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: Mtsa: The modal transition system analyser. In: *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. pp. 475–476 (2008). <https://doi.org/10.1109/ASE.2008.78>
6. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: Mtsa: The modal transition system analyser. In: *Proc. of the Int. Conf. on Automated Software Engineering (ASE)* (2008)
7. Ehlers, Tripakis: *Supervisory Control and Reactive Synthesis: A Comparative Introduction* (2017)
8. Filiot, E., Jin, N., Raskin, J.: Compositional algorithms for LTL synthesis. In: Bouajjani, A., Chin, W. (eds.) *Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21–24, 2010. Proceedings. Lecture Notes in Computer Science*, vol. 6252, pp. 112–127. Springer (2010). https://doi.org/10.1007/978-3-642-15643-4_10, https://doi.org/10.1007/978-3-642-15643-4_10
9. Finkbeiner, B., Geier, G., Passing, N.: Specification decomposition for reactive synthesis. *Innov. Syst. Softw. Eng.* **19**(4), 339–357 (2023). <https://doi.org/10.1007/S11334-022-00462-6>, <https://doi.org/10.1007/s11334-022-00462-6>
10. Finkbeiner, B., Passing, N.: Dependency-based compositional synthesis. In: Hung, D.V., Sokolsky, O. (eds.) *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19–23, 2020, Proceedings. Lecture Notes in Computer Science*, vol. 12302, pp. 447–463. Springer (2020). https://doi.org/10.1007/978-3-030-59152-6_25, https://doi.org/10.1007/978-3-030-59152-6_25
11. Flordal, H., Malik, R., Fabian, M., Åkesson, K.: Compositional synthesis of maximally permissive supervisors using supervision equivalence. *Discret. Event Dyn. Syst.* **17**(4), 475–504 (2007). <https://doi.org/10.1007/S10626-007-0018-Z>, <https://doi.org/10.1007/s10626-007-0018-z>
12. Fokink, W.J., Goorden, M.A., Hendriks, D., van Beek, D.A., Hofkamp, A.T., Reijnen, F.F.H., Etman, L.F.P., Moormann, L., van de Mortel-Fronczak, J.M., Reniers, M.A., Rooda, J.E., van der Sanden, L.J., Schiffelers, R.R.H., Thuijsman, S.B., Verbakel, J.J., Vogel, J.A.: Eclipse escet™: The eclipse supervisory control engineering toolkit. In: Sankaranarayanan, S., Sharygina, N. (eds.) *Tools and Al-*

- gorithms for the Construction and Analysis of Systems. pp. 44–52. Springer Nature Switzerland, Cham (2023)
13. Ghallab, M., Nau, D., Traverso, P.: *Automated Planning: Theory and Practice*. Elsevier, San Francisco, CA (2004)
 14. Graf, S., Steffen, B.: Compositional minimization of finite state systems. In: Clarke, E.M., Kurshan, R.P. (eds.) *Computer Aided Verification*, 2nd International Workshop, CAV '90, New Brunswick, NJ, USA, June 18–21, 1990, Proceedings. Lecture Notes in Computer Science, vol. 531, pp. 186–196. Springer (1990). <https://doi.org/10.1007/BFB0023732>, <https://doi.org/10.1007/BFB0023732>
 15. Huang, J., Kumar, R.: Directed control of discrete event systems for safety and nonblocking. *IEEE Trans Autom. Sci. Eng.* **5**(4), 620–629 (2008). <https://doi.org/10.1109/TASE.2008.923820>, <https://doi.org/10.1109/TASE.2008.923820>
 16. Majumdar, R., Piterman, N., Schmuck, A.K.: Environmentally-friendly GR(1) Synthesis (Feb 2019). <https://doi.org/10.48550/arXiv.1902.05629>, <http://arxiv.org/abs/1902.05629>, arXiv:1902.05629 [cs]
 17. Majumdar, R., Schmuck, A.K.: Supervisory Controller Synthesis for Non-terminating Processes is an Obliging Game (Aug 2021). <https://doi.org/10.48550/arXiv.2007.01773>, <http://arxiv.org/abs/2007.01773>, arXiv:2007.01773 [cs]
 18. Malik, R., Mohajerani, S., Fabian, M.: A survey on compositional algorithms for verification and synthesis in supervisory control. *Discret. Event Dyn. Syst.* **33**(3), 279–340 (2023). <https://doi.org/10.1007/S10626-023-00378-8>, <https://doi.org/10.1007/s10626-023-00378-8>
 19. Malik, R., Åkesson, K., Flordal, H., Fabian, M.: Supremica—an efficient tool for large-scale discrete event systems. *IFAC-PapersOnLine* **50**(1), 5794–5799 (2017). <https://doi.org/https://doi.org/10.1016/j.ifacol.2017.08.427>, <https://www.sciencedirect.com/science/article/pii/S2405896317307772>, 20th IFAC World Congress
 20. Maoz, S., Ringert, J.O.: Spectra: a specification language for reactive systems. *Software and Systems Modeling* **20**, 1553–1586 (2021). <https://doi.org/10.1007/s10270-021-00868-z>
 21. Menghi, C., Tsigkanos, C., Askarpour, M., Pelliccione, P., Vázquez, G., Calinescu, R., García, S.: Mission specification patterns for mobile robots: Providing support for quantitative properties. *IEEE Trans. Software Eng.* **49**(4), 2741–2760 (2023). <https://doi.org/10.1109/TSE.2022.3230059>, <https://doi.org/10.1109/TSE.2022.3230059>
 22. Menghi, C., Tsigkanos, C., Pelliccione, P., Ghezzi, C., Berger, T.: Specification Patterns for Robotic Missions (Jan 2019). <https://doi.org/10.48550/arXiv.1901.02077>, <http://arxiv.org/abs/1901.02077>, arXiv:1901.02077 [cs]
 23. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit Reactive Synthesis Strikes Back! In: Chockler, H., Weissenbacher, G. (eds.) *Computer Aided Verification*. pp. 578–586. Lecture Notes in Computer Science, Springer International Publishing, Cham (2018). <https://doi.org/10.1007/978-3-319-96145-3>
 24. Mohajerani, S., Malik, R., Fabian, M.: An algorithm for weak synthesis observation equivalence for compositional supervisor synthesis **45**(29), 239–244 (2012). <https://doi.org/10.3182/20121003-3-MX-4033.00040>, <https://www.sciencedirect.com/science/article/pii/S1474667015401776>
 25. Mohajerani, S., Malik, R., Fabian, M.: A Framework for Compositional Synthesis of Modular Nonblocking Supervisors. *IEEE Transactions on Automatic*

- Control **59**(1), 150–162 (Jan 2014). <https://doi.org/10.1109/TAC.2013.2283109>, <http://ieeexplore.ieee.org/document/6606831/>
26. Piterman, N., Pnueli, A., Sa’ar, Y.: Synthesis of Reactive(1) Designs. In: Proc. of the 7th Int. Conf. on Verification, Model Checking and Abstract Interpretation. vol. 3855. Springer-Verlag (2006)
 27. Pnueli, A., Rosner, R.: A framework for the synthesis of reactive modules. In: Vogt, F.H. (ed.) CONCURRENCY 88. pp. 4–17. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
 28. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete event systems **25**, 206–230 (1987). <https://doi.org/10.1137/0325013>
 29. Silver, T., Dan, S., Srinivas, K., Tenenbaum, J.B., Kaelbling, L.P., Katz, M.: Generalized planning in PDDL domains with pretrained large language models. CoRR **abs/2305.11014** (2023). <https://doi.org/10.48550/ARXIV.2305.11014>, <https://doi.org/10.48550/arXiv.2305.11014>
 30. Thistle, J.G.: On control of systems modelled as deterministic Rabin automata. Discrete Event Dynamic Systems: Theory and Applications **5**(4), 357–381 (Sep 1995). <https://doi.org/10.1007/BF01439153>, <http://link.springer.com/10.1007/BF01439153>
 31. Zudaire, S., Nahabedian, L., Uchitel, S.: Assured mission adaptation of UAVs **16**(3), 1–27 (2021). <https://doi.org/10.1145/3513091>, <http://arxiv.org/abs/2107.10173>

9 Appendix

9.1 Overview

We show two compositions referred to in the Overview section. The first is the parallel composition of the original modular plant $(M_1 \parallel M_2 \parallel M_3)$. The second is the composition of the plant where a subplant has been controlled and minimized $(M_3 \parallel (M_1 \parallel M_2)_{cont})$. Note the difference in size.

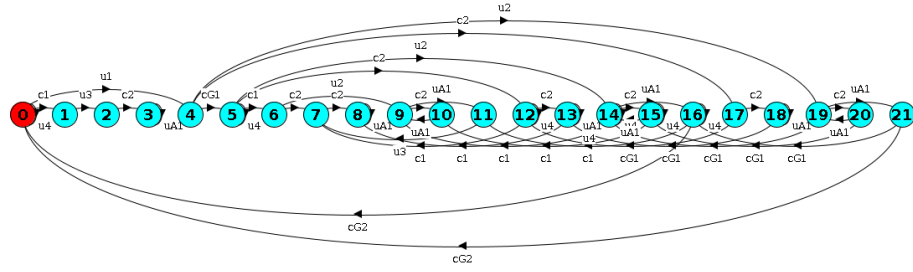


Fig. 10. Composed plant $M_1 \parallel M_2 \parallel M_3$ that a monolithic synthesis approach would have to build to solve the control problem $\mathcal{E}$.

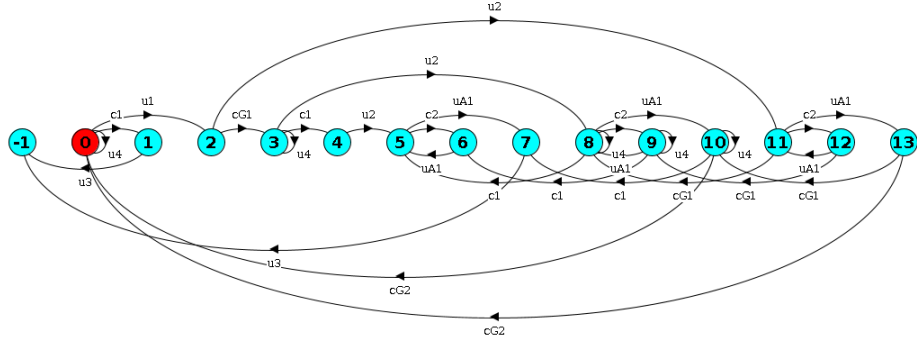


Fig. 11. Plant $M_3 \parallel (M_1 \parallel M_2)_{cont}$. It is nearly half the size of $M_1 \parallel M_2 \parallel M_3$

9.2 Theorem 3

This Lemma will be used in the proof of Theorem 3.

Lemma 1. *Given a LTS M and $M' = M / \sim_{\mathcal{R}}$, if there is a trace $\pi \in tr(M')$ and $s \in S_{M'}$ such that $\pi \in \mathcal{Y}^*$ and $s \xrightarrow{\pi}_{M'} s$ then $\forall l \in \pi$ we have $s \xrightarrow{l}_{M'} s$*

Proof. We prove by contradiction. Suppose a trace $\pi \in tr(M')$ and $s \in S_{M'}$ such that $\pi \in \mathcal{Y}^*$, $s \xrightarrow{\pi}_{M'} s$, and $\exists l \in \pi$ such that $(s, l, s) \notin \rightarrow_{M'}$. Without loss of generality can assume that $\pi = l\pi'$ and $\exists s' \in S_{M'} \cdot s \neq s'$ such that $s \xrightarrow{l}_{M'} s'$, $s' \xrightarrow{\pi'}_{M'} s$. Because $s \neq s'$ and M' is minimized respect to $\sim_{\mathcal{R}}$ we have that s and s' are not equivalent. However, by Def. 8 we have that s and s' are, reaching a contradiction: State s can simulate s' by first going through l , and state s' can simulate s by going through π' . $\square$

The following theorem proves that a control problem with a formula of the form $\Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi$ where φ is GR(1) can be solved as a GR(1) control problem. We use notation $\mathcal{M} \setminus T$ where $\mathcal{M}$ is an LTS and T is a set of transitions over $\mathcal{M}$ to denote the result of removing each transition in T from $\mathcal{M}$.

Theorem 3 Given a control problem $\mathcal{E} = \langle \mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}, \Sigma_c, \varphi \rangle$ and $\mathcal{E}' = \langle \mathcal{M}', \Sigma_c, \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi \rangle$ where $\mathcal{M}' = \mathcal{M} \setminus \{M_{sub}\} \cup \{M_{cont} / \sim_{\mathcal{R}}\}$, $\Sigma_c \subseteq \Sigma$ is a set of controllable events, μ and μ' are sets of events and M_{cont} is a controlled subplant for $\langle \mathcal{M}_{sub}, \Sigma_c, \Box\Diamond(\bigwedge_{l \in \mu} \neg l) \implies \varphi \rangle$ where $\mathcal{M}_{sub} \subseteq \mathcal{M}$. If there is a solution $C = (S_c, \Sigma, \rightarrow_c, \hat{s}_c)$ for control problem $\mathcal{E}$, then C' is a solution for $\mathcal{E}'$ where $C' = (S_{c'}, \Sigma, \rightarrow_{c'}, \hat{s}_c)$ is defined as follows:

- $S_{c'} = S_c$
- $\rightarrow_{C'} = \rightarrow_C \cup \{(s, l, s) \mid s \in S_c \wedge l \in \Sigma_u \cap (\mu \cup \mu') \wedge \exists (s_{\mathcal{M}'}, s_c) \in S_{\mathcal{M}'||C} \cdot (s_{\hat{\mathcal{M}}'}, \hat{s}_c) \rightarrow_{\mathcal{M}'||C} (s_{\mathcal{M}'}, s_c) \wedge (s_{\mathcal{M}'}, l, s_{\mathcal{M}'}) \in \rightarrow_{\mathcal{M}'}\}.$

Also, if there is no solution for $\mathcal{E}$ then there is no solution for $\mathcal{E}'$.

Proof. We prove first that if there exists such solution C for $\mathcal{E}$, then C' is a solution for $\mathcal{E}'$:

- C' is legal for $\mathcal{M}'$ with respect to $\Sigma_u = \Sigma \setminus \Sigma_c$
The proof is by contradiction. Suppose that C' is not legal for $\mathcal{M}'$ with respect to Σ_u . Therefore exists $(s_{C'}, s_{\mathcal{M}'}) \in S_{C'||\mathcal{M}'}$ and $l \in \Sigma_u$ such that $l \in \rightarrow_{\mathcal{M}'}(s_{\mathcal{M}'})$ and $l \notin \rightarrow_{C'}(s_{C'})$. In addition, there is also a trace $\pi \in tr(C') \cdot s_{\hat{C}'} \xrightarrow{\pi} s_{C'}$. Trace π may not be in $tr(C)$, however we can build a trace $\pi' \in tr(C)$ that reaches the same state by removing any self loops labeled with $l \in \mu \cup \mu'$. This leads us to a contradiction because we assumed that C was legal for $\mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}$ with respect to Σ_u .
- $C'||\mathcal{M}'$ is deadlock-free
Proven by contradiction. Suppose that $C'||\mathcal{M}'$ is not deadlock-free, so there is a deadlock state $s_d \in S_{C'||\mathcal{M}'}$ and a trace $\pi \in tr(C'||\mathcal{M}')$ such that $(s_{\hat{C}'}, s_{\hat{\mathcal{M}}'}) \rightarrow_{C'||\mathcal{M}'} s_d$. There are two options:

- $\pi \in tr(C)$
As we know that $C||\mathcal{M}'$ is deadlock-free, this trivially leads us to a contradiction coming from assumption that $C'||\mathcal{M}'$ is not deadlock free.
 - $\pi \notin tr(C)$
If $\pi \notin tr(C)$ there exists a prefix $\pi'l$ of π such that $\pi' \in tr(C) \wedge \pi'l \notin tr(C)$, and there is a state $\bar{s}_C \in S_C$ such that $\hat{s}_C \rightarrow_C \bar{s}_C$. By definition of $\rightarrow_{C'}$, l must be a self-loop transition and does not lead to a deadlock state, so this contradicts also initial assumption that $C'||\mathcal{M}'$ is not deadlock-free.
- $\forall \pi \in C'||\mathcal{M}', \pi \models \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi$
It will be proven by contradiction. Suppose that $\exists \bar{\pi} \in tr(C'||\mathcal{M}') \cdot \bar{\pi} \not\models \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi$. There are two options:
- $\bar{\pi} \in tr(C)$
As we assume that $\bar{\pi} \not\models \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi$, it holds that $\bar{\pi} \not\models \varphi$. Also we know that $\bar{\pi} \in tr(C)$, so it holds that $\bar{\pi} \models \varphi$ because C is solution for $\mathcal{E}$. This result lead us to a contradiction which comes from assuming that exists $\bar{\pi}$.
 - $\bar{\pi} \notin tr(C)$
We assume that $\bar{\pi} \not\models \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi$, so it holds that $\bar{\pi} \models \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \wedge \bar{\pi} \not\models \varphi$. We can build a trace $\bar{\pi}' \subseteq \bar{\pi}$ such that $\bar{\pi}' \in tr(C)$ if we remove occurrences of events $l \in \mu \cup \mu'$ of $\bar{\pi}$. As the formula is stutter-free and there is no formula event present in $\mu \cup \mu'$, we conclude that $\bar{\pi}' \models \varphi$ reaching a contradiction because $\bar{\pi}' \in tr(C)$ and C is a solution for $\mathcal{E}$.
- Therefore, it is proven that $\forall \pi \in C'||\mathcal{M}', \pi \models \Box\Diamond(\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi$.

What remains is to prove that if there is no solution C for $\mathcal{E}$, then there is none for $\mathcal{E}'$ either.

It will be proven by counter reciprocal. I will prove that, if exists a solution C' for $\mathcal{E}'$, then C is a solution for $\mathcal{E}$.

- C is legal for $\mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}$ with respect to $\Sigma_u = \Sigma \setminus \Sigma_c$.
The proof is by contradiction. Suppose that C is not legal for $\mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}$ with respect to Σ_u . Therefore exists $(s_C, s_{\mathcal{M}'}) \in S_{C||\mathcal{M}'}$ and $l \in \Sigma_u$ such that $l \in \rightarrow_{\mathcal{M}'}(s_{\mathcal{M}'})$ and $l \notin \rightarrow_C(s_C)$. In addition, there is also a trace $\pi \in tr(C) \cdot \hat{s}_C \xrightarrow{\pi}_{C||\mathcal{M}'} s_C$. As we know that $tr(C) \subseteq tr(C')$, we conclude that we can reach the same state leading us to a contradiction because we start assuming that C' is legal for $\mathcal{M}'$ with respect to Σ_u .
- $C||\mathcal{M}'$ is deadlock-free
Proven by contradiction. Suppose that $C||\mathcal{M}'$ is not deadlock-free, so there is a deadlock state $s_d \in S_{C||\mathcal{M}'}$ and a trace $\pi \in tr(C||\mathcal{M}')$ such that $(\hat{s}_C, s_{\mathcal{M}'}) \rightarrow_{C||\mathcal{M}'} s_d$. As we know that $tr(C) \subseteq tr(C')$, then $\pi \in tr(C'||\mathcal{M}')$. On the other hand, we know that $C'||\mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}$ is deadlock-free and this trivially lead us to a contradiction coming from assumption that $C||\mathcal{M}'$ is not deadlock-free.

– $\forall \pi \in C \mid \mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}, \pi \models \varphi$

It will be proven by contradiction. Suppose that $\exists \bar{\pi} \in tr(C \mid \mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\}) \cdot \bar{\pi} \not\models \varphi$. As $\bar{\pi} \in tr(\mathcal{M}' \setminus \{(s, l, s) \mid l \in \mu \cup \mu'\})$, it holds trivially that $\bar{\pi} \models \Box \Diamond (\bigwedge_{l \in (\mu \cup \mu')} \neg l)$. As we assume that $\bar{\pi} \not\models \varphi$, then it holds that $\bar{\pi} \not\models \Box \Diamond (\bigwedge_{l \in (\mu \cup \mu')} \neg l) \implies \varphi$. On the other hand, we know that $\bar{\pi} \in tr(C')$ (because $tr(C) \subseteq tr(C')$). This lead us to a contradiction because C' is a solution for $\mathcal{E}'$, the contradiction comes from assuming that exists $\bar{\pi}$.

□

9.3 Theorem 3

Lemmas 2 and 3 will be used in the proof of Theorem 2.

Lemma 2. *Given a control problem $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$, $\mathcal{M} = \{M_1, \dots, M_n\}$ and M_1^{safe} is a Safe Controller for $\mathcal{E}' = \langle M_1, \Sigma_c \cup (\Sigma \setminus \Sigma_{M_1}), \varphi' \rangle$ where φ' is a projected formula for Σ_{M_1} . If C_M is a solution for $\mathcal{E}$, then C_M is a solution for $\mathcal{E}'$.*

Proof.

- C_M is legal for M_1 with respect to $\Sigma'_u = \Sigma \setminus (\Sigma_c \cup (\Sigma \setminus \Sigma_{M_1}))$
 The proof is by contradiction. Suppose that C_M is not legal for M_1 with respect to $\Sigma'_u = \Sigma \setminus (\Sigma_c \cup (\Sigma \setminus \Sigma_{M_1}))$. Therefore exists $(s_{C_M}, s_{M_1}) \in S_{C_M||M_1}$ and $l \in \Sigma'_u$ such that $l \in \rightarrow_{M_1}(s_{M_1})$ and $l \notin \rightarrow_{C_M}(s_{C_M})$. In addition, there is also a trace $\pi \in tr(C_M||M_1)$ such that $(\hat{s}_C, \hat{s}_{M_1}) \xrightarrow{\pi}_{C_M||M_1} (s_{C_M}, s_{M_1})$.
 As we know that $\pi \in tr(C_M)$, we can assume that $\pi \in tr(\mathcal{M})$ and $(\hat{s}_C, \hat{s}_{M_1}) \xrightarrow{\pi}_{C_M||\mathcal{M}} (s_{C_M}, s_{M_1})$ where $l \in \rightarrow_{\mathcal{M}}(s_{M_1})$ and $l \notin \rightarrow_{C_M}(s_{C_M})$. This lead us to a contradiction because $\Sigma'_u \subseteq \Sigma_u$ and C_M is a solution for $\mathcal{E}$, the contradiction comes from assuming that C_M is not legal for M_1 with respect to $\Sigma'_u = \Sigma \setminus (\Sigma_c \cup (\Sigma \setminus \Sigma_{M_1}))$.
- $C_M||M_1$ is deadlock-free
 The proof is by contradiction. Suppose that $C_M||M_1$ is not deadlock-free, so there is a deadlock state $s_d \in S_{C_M||M_1}$ and a trace $\pi \in tr(C_M||M_1)$ such that $(\hat{s}_{C_M}, \hat{s}_{M_1}) \xrightarrow{\pi}_{C_M||M_1} s_d$. As we know that $tr(C_M||M_1) \subseteq tr(C_M||\mathcal{M})$, then $\pi \in tr(C_M||\mathcal{M})$ and this trivially lead us to a contradiction because C_M is a solution for $\mathcal{E}$, the contradiction comes from assuming that $C_M||M_1$ is not deadlock-free.
- $\forall \pi \in C_M||M_1, \pi \models \varphi'$
 The proof is by contradiction. Suppose that $\exists \bar{\pi} \in tr(C_M||M_1) \cdot \bar{\pi} \not\models \varphi'$. As we know that $tr(C_M||M_1) \subseteq tr(C_M||\mathcal{M})$, then $\bar{\pi} \in tr(C_M||\mathcal{M})$. This lead us to a contradiction because $\varphi \rightarrow \varphi'$, so as C_M is a solution for $\mathcal{E}$, then C_M should satisfy φ' . The contradiction comes from assuming that $\exists \bar{\pi} \in tr(C_M||M_1) \cdot \bar{\pi} \not\models \varphi'$.

□

Lemma 3. *Given a control problem $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \varphi \rangle$, $\mathcal{M} = \{M_1, \dots, M_n\}$ and M_1^{safe} is a Safe Controller for $\mathcal{E}' = \langle M_1, \Sigma_c \cup (\Sigma \setminus \Sigma_{M_1}), \varphi' \rangle$ where φ' is a projected formula for Σ_{M_1} . If C_M is a solution for $\mathcal{E}$, then $P_{\Sigma_{M_1}}(tr(C_M)) \subseteq tr(M_1^{safe})$.*

Proof. The proof is by contradiction. Suppose that $\exists \pi \in tr(C_M)$ such that $P_{\Sigma_{M_1}}(\pi) \notin tr(M_1^{safe})$. As C_M is a solution for $\mathcal{E}$, we know that $\pi \models \varphi$. Also, as $\varphi \rightarrow \varphi'$, it holds that $\pi \models \varphi'$. Assuming that $P_{\Sigma_{M_1}}(\pi) \notin tr(M_1^{safe})$, by definition

of safe controller we know that $\exists l \in \pi, s \in S_{M_1}$ such that $s \xrightarrow{l} s_e$ where s_e is not a winning state (i.e., there is no controller that includes s_e). However, by Lemma 2 we know that C_M is a solution for $\mathcal{E}'$ and $\pi \in tr(C_M)$. This lead us to a contradiction because $P_{\Sigma_{M_1}}(tr(C_M))$ goes through winning states of M_1 , so it must be contained in $tr(M_1^{safe})$. The contradiction comes from assuming that $\exists \pi \in tr(C_M)$ such that $P_{\Sigma_{M_1}}(\pi) \notin tr(M_1^{safe})$. $\square$

The following theorem proves the equirealizability of the most important step that compositional synthesis uses.

Theorem 2

Given a synthesis tuple $(\mathcal{M}, \mathcal{C}, \mu)$, an LTL formula φ , and a set Σ_c of controllable events.

Let $M \subseteq \mathcal{M}$ with alphabet $\Sigma_M = \mathcal{T} \dot{\cup} \Omega$ where $\mathcal{T}$ has the events not shared with any other LTS in $\mathcal{M} \setminus M$ nor in φ . Let φ' be a projection of φ to Σ_M .

Let M_{safe} be a safe controller for $\langle M, \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi \rangle$. Let M_{cont} be the subplant M controlled by M_{safe} with shared events Ω .

If $M_{cont}/\sim_{\mathcal{T}}$ is deterministic and μ' is the set of self-looping events induced by $\sim_{\mathcal{T}}$ reduction, then $(\mathcal{M}, \mathcal{C}, \mu) \cong_{\Sigma_c, \varphi} (\mathcal{M} \setminus \{M\} \cup \{M_{cont}/\sim_{\mathcal{T}}\}, \mathcal{C} \cup \{M_{safe}\}, \mu \cup \mu')$.

Proof. We will use the notation $\mathcal{M}' = \mathcal{M} \setminus \{M\} \cup \{M_{cont}/\sim_{\mathcal{T}}\}$, $T = (\mathcal{M}, \mathcal{C}, \mu)$, $T' = (\mathcal{M} \setminus \{M\} \cup \{M_{cont}/\sim_{\mathcal{T}}\}, \mathcal{C} \cup \{M_{safe}\}, \mu \cup \mu')$ as syntactic replacement.

$\rightarrow$) Given $S \in Cont_{\Sigma_c, \varphi}(T)$, we prove that $S \in Cont_{\Sigma_c, \varphi}(T')$.

This will be proven by existence of $\hat{C}_M$ such that is a solution for $\mathcal{E}' = \langle \mathcal{M}', \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi \rangle$ and $\hat{C}_M || M_{safe} || C = S$.

Since $S = C || C_M$ by definition where C_M is a solution for $\mathcal{E} = \langle M, \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi \rangle$. We demonstrate $\hat{C}_M$ exists using $\hat{C}_M = C_M$ and proving that:

1. C_M is a solution for $\mathcal{E}'$.

- (a) (*formula satisfiability*) We prove that $\forall \pi \in (C_M || \mathcal{M}') . \pi \models \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi$ by contradiction. Suppose C_M is not a solution for $\mathcal{E}'$ and there is a trace $\pi \in tr(C_M || \mathcal{M}')$ such that $\pi \not\models \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi$. Also, it holds that $\pi \in C_M$.

Since C_M is a solution for $\mathcal{E}$, then $\forall \pi \in tr(C_M || \mathcal{M}) . \pi \models \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$.

Note that any trace π that $\pi \models \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$, it holds that $\pi \models \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi$ because you are strengthening the assumption. Then, $\forall \pi \in tr(C_M) . \pi \models \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi$ which contradicts the original assumption.

Then, $\forall \pi \in (C_M || \mathcal{M}') . \pi \models \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi$.

- (b) C_M is legal for $\mathcal{M}'$ with respect to $\Sigma_u \cup (\mu \cup \mu')$.

Proven by contradiction.

Suppose C_M is not legal for $\mathcal{M}'$ with respect to $\Sigma_u \cup (\mu \cup \mu')$. Then,

$\exists \pi \in tr(C_M)$ such that $\exists l \in (\Sigma_u \cup (\mu \cup \mu')) \wedge \pi l \in tr(\mathcal{M}') \wedge \pi l \notin tr(C_M)$.
I separate into two cases: $l \in \Sigma_u \cup \mu$ or $l \in \mu'$

i. $l \in \Sigma_u \cup \mu$

This is not possible since C_M is legal for $\mathcal{M}$ with respect to $\Sigma_u \cup \mu$ and $tr(\mathcal{M}) \subseteq tr(\mathcal{M}')$.

ii. $l \in \mu'$

By Lemma 3, if $\pi l \notin P_{M_{safe}}(tr(C_M))$, then $\pi l \notin tr(M_{safe})$.

Then, by construction, $\pi \notin tr(M_1)$.

Then, because $tr(\mathcal{M}) \subseteq tr(\mathcal{M}')$, $\pi l \notin tr(\mathcal{M}')$.

Which enter in contradiction with original assumption. Then, C_M is legal for $\mathcal{M}'$ with respect to $\Sigma_u \cup (\mu \cup \mu')$.

(c) $(C_M || \mathcal{M}')$ is deadlock-free.

Proven by contradiction.

We suppose that $C_M || \mathcal{M}'$ has a deadlock state $(c, \mathcal{M}')$ reachable from initial state by a trace π . Since $C_M || \mathcal{M}$ is deadlock free, there exists a label $l \in \Sigma_{\mathcal{M}}$ which is enabled in (c, m) , where $(c, m) \in S_{\mathcal{M}}$ is an state reachable by trace π . Since $tr(\mathcal{M}) \subseteq tr(\mathcal{M}')$, it is not possible that (c, m) has an enabled event which is disabled in $(c, \mathcal{M}')$.

Then, $(C_M || \mathcal{M}')$ is deadlock-free.

Then, C_M is a valid solution for $\mathcal{E}'$.

2. $C_M || M_{safe} || C = S$

This is correct iff $C_M || M_{cont} = C_M$ and it is correct because $tr(C_M |_{\Sigma_{M_{cont}}}) \subseteq tr(M_{cont})$, which is true by Lemma 3.

$\leftarrow$) Given $S \in Cont_{\Sigma_u, \varphi}(T')$, We prove that $S \in Cont_{\Sigma_u, \varphi}(T)$ by existence of C_M such that is a solution for $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi \rangle$ and $C_M || C = S$. Since $S = \hat{C}_M || C || M_{safe}$ where $\hat{C}_M$ is a solution for $\mathcal{E}' = \langle \mathcal{M}', \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi \rangle$. We demonstrate C_M exists using $C_M = \hat{C}_M || M_{safe}$ and prove that:

1. $\hat{C}_M || M_{safe}$ is a solution for $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi \rangle$.

(a) (*formula satisfiability*) We will prove that $\forall \pi \in tr(\hat{C}_M || M_{safe} || \mathcal{M})$.
 $\pi \models \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$. Since $\hat{C}_M$ is a solution for $\mathcal{E}'$, then $\pi \models \Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi$. We must guarantee that $\pi \models \Box \Diamond \bigwedge_{l \in \mu'} \neg l$. As μ' is a set of self-loop labels induced by minimization, we know that there are not present as self-loop in $\mathcal{M} || M_{safe}$ because there are prior to last minimization. Therefore we can conclude that $\pi \models \Box \Diamond \bigwedge_{l \in \mu'} \neg l$. Then, $\pi \models (\Box \Diamond \bigwedge_{l \in \mu \cup \mu'} \neg l \implies \varphi) \wedge \pi \models (\Box \Diamond \bigwedge_{l \in \mu'} \neg l)$ implies $\pi \models \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$.

Finally, we can say that $\forall \pi \in tr(\hat{C}_M || M_{cont} || \mathcal{M})$. $\pi \models \Box \Diamond \bigwedge_{l \in \mu} \neg l \implies \varphi$.

(b) $\hat{C}_M || M_{safe}$ is legal for $\mathcal{M}$ with respect to $\Sigma_u \cup \mu$.

We prove this by contradiction. Then, we suppose that exists an state

$(\hat{c}, m_{safe}, m) \in S_{\hat{C}_M || M_{safe} || \mathcal{M}}$ and an event $l \in \Sigma_u \cup \mu$ that is enabled for m in over $\mathcal{M}$ transitions but not in $\hat{c}$ or m_{safe} .

We know there is a trace $\pi \in tr(\hat{C}_M || M_{safe})$ that reaches mentioned state.

Then, $\pi \in tr(\hat{C}_M) \wedge \pi \in tr(M_{safe})$.

Also, as $tr(\mathcal{M}) \subseteq tr(\mathcal{M}')$, $\pi \in tr(\mathcal{M}')$

There are two options that could explain state $(\hat{c}, m_{safe}, m)$ blocks a l event.

- We suppose that $\hat{c}$ has not enabled l event, but it is impossible because $\hat{C}$ is legal for $\mathcal{M}'$ with respect to $\Sigma_u \cup \mu \cup \mu'$.
- We suppose that m_{safe} has not enabled l event. Also, we know $\pi \in tr(M)$ and reaches an state where l is enabled in M . There is a same reasoning for M_{safe} because it is (by construction) safe and it does not block any uncontrollable event neither an event which is local for respect another component in $\mathcal{M}$.

Then, $\hat{C}_M || M_{safe}$ is legal for $\mathcal{M}$ with respect to $\Sigma_u \cup \mu$

(c) $\hat{C}_M || M_{safe} || \mathcal{M}$ is deadlock-free.

Prove by contradiction. Suppose that $\hat{C}_M || M_{safe} || \mathcal{M}$ has a deadlock state $(c_m, m_{safe}, m) \in S_{\hat{C}_M || M_{safe} || \mathcal{M}}$ reachable from initial state by a trace π .

Then, $\pi \in tr(\hat{C}_M) \wedge \pi \in tr(M_{cont}) \wedge \pi \in tr(\mathcal{M})$.

Also, as $tr(\mathcal{M}) \subseteq tr(\mathcal{M}')$, $\pi \in tr(\mathcal{M}')$

Then, there is an state $(c_m, m') \in S_{\hat{C}_M || \mathcal{M}'}$ which has at least an outgoing enabled event l since $\hat{C}_M$ is solution for $\mathcal{E}$, $\hat{C}_M || \mathcal{M}'$ and it is deadlock-free.

We prove that l is also enabled in $(c_m, m_{safe}, m) \in S_{\hat{C}_M || M_{safe} || \mathcal{M}}$

Suppose it is not. Then it is disabled by m_{safe} or m (because c_m has enabled l event).

- Suppose m_{safe} has not enabled l . This is impossible because (by construction) M_{safe} has *safe* states and by Lemma 3 we conclude that an event enabled in c_m has to be enabled in m_{safe} .
- Suppose m has not enabled l . This is impossible because $tr(\mathcal{M}) \subseteq tr(\mathcal{M}')$

Then, exists $l \in \Sigma_{\hat{C}_M || \mathcal{M}'}$ such that l is enabled in $(c_m, m_{safe}, m) \in S_{\hat{C}_M || M_{safe} || \mathcal{M}}$.

Then, $\hat{C}_M || M_{safe} || \mathcal{M}$ is deadlock-free.

Then, $\hat{C}_M || M_{safe}$ is a solution for $\mathcal{E} = \langle \mathcal{M}, \Sigma_c, \square \diamond \bigwedge_{l \in \mu} \neg l \implies \varphi \rangle$.

2. $C_M || C = S$

As we assume that $C_M = \hat{C}_M || M_{safe}$ and we know that $S = \hat{C}_M || M_{safe} || C$, it trivially holds that $C_M || C = S$ using those definitions.

□