Most General Property-Preserving Updates (TR)*

Davide Bresolin[1] and Ivan Lanese[2]

[1] University of Padova, Italy
Email: davide.bresolin@unipd.it
[2] Focus Team, University of Bologna/INRIA, Italy
Email: ivan.lanese@gmail.com

Abstract. Systems need to be updated to last for a long time in a dynamic environment, and to cope with changing requirements. It is important for updates to preserve the desirable properties of the system under update, while possibly enforcing new ones.
Here we consider a simple yet general update mechanism, which replaces a component of the system with a new one. The context, i.e., the rest of the system, remains unchanged. We define contexts and components as Constraint Automata interacting via either asynchronous or synchronous communication, and we express properties using Constraint Automata too. Then we build most general updates which preserve specific properties, considering both a single property and all the properties satisfied by the original system, in a given context or in all possible contexts.

1 Introduction

Update is a relevant topic [18], both for automatic updates, as in the context of adaptive systems [16] or autonomic computing [14], and for manual updates. A main reason is that one wants systems to last for a long time in a changing environment and to satisfy changing user requirements. However, a main point, namely correctness of the system after update, has received scarce attention till now, as remarked also in [13].

In this paper we consider a very simple yet general update mechanism, which replaces a part of the system with a new one. Formally, the system is seen as a context $\mathcal{C}$ containing the component to be updated $\mathcal{A}$, i.e., the system has the form $\mathcal{C}[\mathcal{A}]$. An update replaces $\mathcal{A}$ with $\mathcal{B}$, thus the system upon update has the shape $\mathcal{C}[\mathcal{B}]$. A basic question is: how to build a most general $\mathcal{B}$ such that if $\mathcal{C}[\mathcal{A}]$ satisfies a given property Φ, then also $\mathcal{C}[\mathcal{B}]$ satisfies the same property? This question is answered in Section 3. Note that $\mathcal{C}[\mathcal{B}]$ may satisfy further properties that $\mathcal{C}[\mathcal{A}]$ does not satisfy. The answer to the question above, which relates $\mathcal{A}$ and $\mathcal{B}$, depends both on the context $\mathcal{C}$ and on the property Φ. From this observation two generalizations emerge naturally. On one side, one may ask how to build a

* The authors acknowledge the support from the Italian INdAM – GNCS project 2016 *Logic, automata and games for self-adaptive systems.*

most general $\mathcal{B}$ such that for a given context $\mathcal{C}$, *all the properties* satisfied by $\mathcal{C}[\mathcal{A}]$ are also satisfied by $\mathcal{C}[\mathcal{B}]$ (Section 3). We call such an update correct for a given context w.r.t. any property. On the other side, one may ask how to build a most general $\mathcal{B}$ such that *for each context* $\mathcal{C}$ if $\mathcal{C}[\mathcal{A}]$ satisfies property Φ, then $\mathcal{C}[\mathcal{B}]$ satisfies the same property (Section 4). We say that this is a correct update (w.r.t. the property Φ) that can be applied in any context. Finally, one may combine the two generalizations asking how to build a most general $\mathcal{B}$ to ensure correctness of update *in any context and w.r.t. any property* (Section 4).

The questions above are very general, and the detailed answer depends on the choice of the model for components and contexts, of the composition operators, and of the formalism for expressing properties. We consider here components, contexts and properties represented as Constraint Automata [6, 5], which have been used in the literature, e.g., to give a formal semantics to REO connectors [3] and Rebeca actors [19]. We consider both asynchronous and synchronous composition for components and contexts. We leave the systematic exploration of the research space above to future work. We illustrate the results of our approach by means of a simple running example. All the operations on Constraint Automata were computed using the tool GOAL [20], an interactive tool for defining and manipulating automata, which we extended to deal with Constraint Automata.

2 Constraint Automata

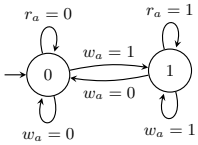
We model components, contexts and properties as Constraint Automata (CAs) [6], defined below. Throughout the paper we assume a finite set Data of *data values* which can be communicated and a finite set of states for the CAs. As in [6], the finiteness assumption is needed for the effectiveness of our constructions.

Definition 1 (Constraint Automata).
A constraint automaton $\mathcal{A}$ is a tuple $\langle Q, N, q_0, \rightarrow \rangle$ where:

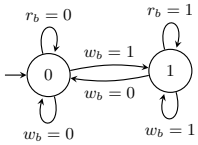
1. *Q is a finite set of* states*;*
2. *N is a finite set of* node names *representing the interface between the CA and the outside world;*
3. *$q_0 \in Q$ is the* initial state*;*
4. *$\rightarrow \subseteq Q \times \mathsf{CIO}(N) \times Q$ is the* transition relation*, where $\mathsf{CIO}(N)$ is the set of* concurrent I/O operations *$c : N \mapsto \mathsf{Data} \cup \{\bot\}$ mapping every node in N to an element of Data, or to $\bot$ if no data is written/read. We assume that c is never the constant function with value $\bot$.*

Transitions of a CA are of the form $q \xrightarrow{c} p$, where c is a concurrent I/O operation. A *run* of a CA is a finite/infinite sequence $\rho = q_0 \xrightarrow{c_0} q_1 \xrightarrow{c_1} \ldots$ such that q_0 is the initial state and, for every i, $q_i \xrightarrow{c_i} q_{i+1}$ is a transition of the CA. In this case, we say that ρ *accepts* the *trace* $w = c_0 c_1 \ldots$. The *language* of a CA $\mathcal{A}$, denoted $\mathscr{L}(\mathcal{A})$, is the set of traces accepted by $\mathcal{A}$. Since a prefix of a run is again a run, languages are closed under prefix.

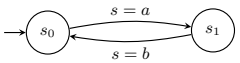
Given $c \in \mathsf{CIO}(N)$, we define $\mathsf{Nodes}(c)$ as the set of nodes through which data flow, formally $\mathsf{Nodes}(c) = \{n \in N \mid c(n) \neq \bot\}$. The *domain restriction*



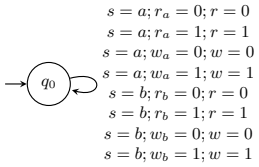
(a) The register a.



(b) The register b.



(c) The scheduler.



(d) The synchronizer.

Fig. 1. The CAs for Example 1.

of a concurrent I/O operation c on a subset of nodes $N' \subseteq N$ is written $c{\downarrow_{N'}}$. Given two disjoint sets of nodes N_1 and N_2, and two CIOs $c_1 \in \mathsf{CIO}(N_1)$ and $c_2 \in \mathsf{CIO}(N_2)$, we define their *union* as the unique CIO $c_1 \cup c_2 \in \mathsf{CIO}(N_1 \cup N_2)$ such that $(c_1 \cup c_2){\downarrow_{N_1}} = c_1$ and $(c_1 \cup c_2){\downarrow_{N_2}} = c_2$.

Example 1. We introduce here our running example.

We consider a system which allows one to read and write information from/to two one-bit registers, denoted as a and b. The system is the composition of the four components represented in Figure 1: two registers, a scheduler that determines which register is active, and a synchronizer that communicates with the registers and the scheduler and proposes to the outside world the nodes r (read) and w (write) to access the currently active register. Labels on edges represent CIOs, written as semicolon-separated sets of assignments. Each assignment $n = d$ specifies that the data value d is communicated on node n. No communication occurs on nodes that do not appear in the label.

Registers are two-state CAs communicating with the synchronizer on nodes r_a (read a) and w_a (write a) for register a, and on nodes r_b and w_b for register b. The scheduler interacts with the synchronizer on node s. Essentially, the two registers are scheduled in round-robin order a, b. The synchronizer is a one-state CA that forwards the external operations to the currently active register.

We use CAs also to describe properties, which are prefix-closed sets of (finite or infinite) traces. We represent a property as a CA Φ accepting the corresponding set of traces. We say that a CA $\mathcal{A}$ satisfies the property Φ, written $\mathcal{A} \models \Phi$, iff $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\Phi)$. CAs are as expressive as the safety linear μ-calculus [15] and, as a consequence, more expressive of the safety fragment of temporal logics like LTL and CTL.

2.1 Composition of CAs

We consider here a particular type of composition, where a component is embedded in a context. We examine two forms of synchronization between the context

and the component: *synchronous* and *asynchronous*. Formally, we assume to have two CAs: $\mathcal{A}$ (the component) and $\mathcal{C}$ (the context). We also assume two disjoint finite sets of node names U and O. Communication between the component and the context goes through U, while communication between the context and the external world goes through O.

In the asynchronous case, at every step the context communicates either with the component via nodes in U, or with the external world via nodes in O, or with both at the same time. In the synchronous case, at every step the context communicates with both the component and the external world.

The embedding of $\mathcal{A}$ in $\mathcal{C}$ is defined by means of two operations on CAs [5]: projection and (synchronous or asynchronous) join.

Definition 2 (Asynchronous Join). *The* asynchronous join *of two CAs* $\mathcal{A} = \langle Q_A, U, q_0^A, \rightarrow_A \rangle$ *and* $\mathcal{C} = \langle Q_C, U \cup O, q_0^C, \rightarrow_C \rangle$ *is defined as the CA* $\mathcal{A} \bowtie_a \mathcal{C} = \langle Q_A \times Q_C, U \cup O, (q_0^A, q_0^C), \rightarrow_a \rangle$ *such that:*

- $(q,p) \xrightarrow{c}_a (q',p')$ *if* $\mathsf{Nodes}(c) \cap U \neq \emptyset$, $q \xrightarrow{c\downarrow_U}_A q'$ *and* $p \xrightarrow{c}_C p'$*;*
- $(q,p) \xrightarrow{c}_a (q,p')$ *if* $\mathsf{Nodes}(c) \cap U = \emptyset$ *and* $p \xrightarrow{c}_C p'$*.*

Definition 3 (Synchronous Join). *The* synchronous join *of two CAs* $\mathcal{A} = \langle Q_A, U, q_0^A, \rightarrow_A \rangle$ *and* $\mathcal{C} = \langle Q_C, U \cup O, q_0^C, \rightarrow_C \rangle$ *is defined as the CA* $\mathcal{A} \bowtie_s \mathcal{C} = \langle Q_A \times Q_C, U \cup O, (q_0^A, q_0^C), \rightarrow_s \rangle$ *such that:*

- $(q,p) \xrightarrow{c}_s (q',p')$ *if* $\mathsf{Nodes}(c) \cap U \neq \emptyset$, $\mathsf{Nodes}(c) \cap O \neq \emptyset$, $q \xrightarrow{c\downarrow_U}_A q'$ *and* $p \xrightarrow{c}_C p'$*.*

Given a CA $\mathcal{B}$ with node names from a set $U \cup O$, the projection on O removes the nodes in U from the interface of $\mathcal{B}$ and hides the communications occurring at those nodes. To define the projection, we need the relation $\rightsquigarrow_O^* \subseteq Q \times Q$, which is the smallest relation such that:

- $q \rightsquigarrow_O^* q$ for each $q \in Q$;
- if $q \rightsquigarrow_O^* p$ and $p \xrightarrow{c} r$ with $\mathsf{Nodes}(c) \cap O = \emptyset$, then $q \rightsquigarrow_O^* r$.

Definition 4 (Projection). *The* projection *of a CA* $\mathcal{B} = \langle Q, U \cup O, q_0, \rightarrow \rangle$ *on* O *is defined as the CA* $\mathcal{B}\downarrow_O = \langle Q, O, q_0, \rightarrow^* \rangle$ *such that* $q \xrightarrow{c}{}^* p$ *iff there exists* $d \in \mathsf{CIO}(U \cup O)$, $r \in Q$ *such that* $d\downarrow_O = c$ *and* $q \rightsquigarrow_O^* r \xrightarrow{d} p$*.*

The *asynchronous embedding* $\mathcal{C}[\mathcal{A}]_a$ and the *synchronous embedding* $\mathcal{C}[\mathcal{A}]_s$ of the component $\mathcal{A} = \langle Q_A, U, q_0^A, \rightarrow_A \rangle$ in the context $\mathcal{C} = \langle Q_C, U \cup O, q_0^C, \rightarrow_C \rangle$ are defined as

$$\mathcal{C}[\mathcal{A}]_a = (\mathcal{A} \bowtie_a \mathcal{C})\downarrow_O \qquad \mathcal{C}[\mathcal{A}]_s = (\mathcal{A} \bowtie_s \mathcal{C})\downarrow_O$$

The above definitions hide all nodes of the component and expose only the nodes from O. We will drop the subscript a or s to refer to both kinds of embedding.

Example 2. We can now build the system outlined in Example 1 by embedding the scheduler into the context, which is obtained by embedding the two registers (in any order) into the synchronizer. All the embeddings are asynchronous.

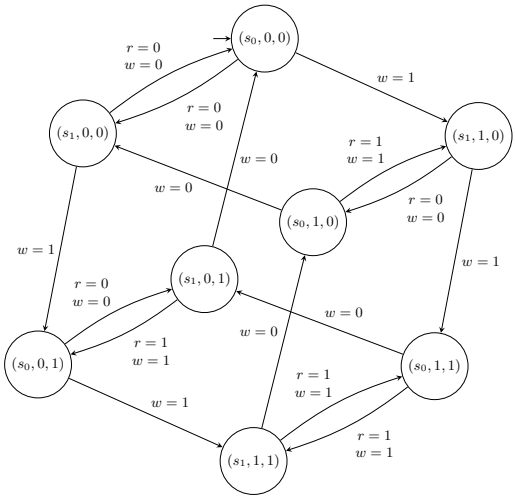


Fig. 2. Embedding of the scheduler in the context.

The states of the whole system, represented in Figure 2, are tuples (s_i, v_a, v_b) where s_i is the state of the scheduler and v_a and v_b are the values of the registers a and b, respectively.

Trace inclusion is a *congruence* w.r.t. the embedding of CAs, as formally stated by the following lemma. This result holds since we consider sets of traces which are closed under prefixes.

Lemma 1. *Given two components $\mathcal{A}$ and $\mathcal{B}$, for every context $\mathcal{C}$ we have that if $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$ then $\mathscr{L}(\mathcal{C}[\mathcal{A}]_a) \subseteq \mathscr{L}(\mathcal{C}[\mathcal{B}]_a)$ and $\mathscr{L}(\mathcal{C}[\mathcal{A}]_s) \subseteq \mathscr{L}(\mathcal{C}[\mathcal{B}]_s)$.*

Proof. Let us prove the asynchronous case (the synchronous case is analogous). Assume that $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$, and suppose towards a contradiction that $\mathscr{L}(\mathcal{C}[\mathcal{A}]_a) \not\subseteq \mathscr{L}(\mathcal{C}[\mathcal{B}]_a)$. Then there exists a trace $w \in \mathscr{L}(\mathcal{C}[\mathcal{A}]_a)$ such that $w \notin \mathscr{L}(\mathcal{C}[\mathcal{B}]_a)$. The trace is obtained from a run $\rho = (s_0, q_0) \xrightarrow{c_0} (s_1, q_1) \xrightarrow{c_1} \ldots$, which is the composition of a run ρ_C of $\mathcal{C}$ and a run ρ_A of $\mathcal{A}$. By hypothesis, there exists a run ρ_B of $\mathcal{B}$ with the same labels of ρ_A. By composing ρ_B with the above run ρ_C we get a run of $\mathcal{C}[\mathcal{B}]_a$ with label w, against our hypothesis[3].

2.2 Determinization and complementation of CAs

In the next sections, we will need to complement CAs. Unfortunately, CAs are not closed under complementation. We solve the problem following the approach in [5], reported below.

[3] This run may not be maximal, but this is not a problem for us since our sets of traces are closed under prefixes.

Given a nondeterministic CA $\mathcal{A}$, by using the standard *subset construction* for finite word automata it is possible to obtain an equivalent deterministic CA $\mathsf{Subset}(\mathcal{A})$ that, in the worst case, is exponentially larger than $\mathcal{A}$.

We can complement $\mathsf{Subset}(\mathcal{A})$ by enriching it with a set of final states $F \subseteq Q$ and a *Büchi acceptance condition*. We say that a finite run is accepting whenever the last state of the run is final, while an infinite run is accepting if the set of final states F is visited infinitely often. Formally, given a deterministic CA $\mathcal{A} = \langle Q, N, q_0, \rightarrow_A \rangle$ we can build a CA with final states $\overline{\mathcal{A}} = \langle Q_\perp, N, q_0, \rightarrow_{\overline{A}}, F \rangle$ accepting the complement language as follows:

- $Q_\perp = Q \cup \{q_\perp\}$ where $q_\perp$ is a distinguished *sink state* not included in Q;
- $F = \{q_\perp\}$ (only the sink state is final);
- $q \xrightarrow{c}_{\overline{A}} q'$ iff $q \xrightarrow{c}_A q'$;
- $q \xrightarrow{c}_{\overline{A}} q_\perp$ for all $q \in Q$ and $c \in \mathsf{CIO}(N)$ such that there is no q' such that $q \xrightarrow{c}_A q'$;
- $q_\perp \xrightarrow{c}_{\overline{A}} q_\perp$ for all $c \in \mathsf{CIO}(N)$.

In the following we will need to compute expressions of the form $\mathcal{C}[\overline{\mathcal{A}}]$. This can be done by using the construction for standard CAs, and by choosing as final states of the result the set $Q_C \times \{q_\perp\}$, where Q_C is the set of states of $\mathcal{C}$ and $q_\perp$ is the sink state of $\overline{\mathcal{A}}$.[4]

We will also use the following operations on deterministic CAs with final states. $\mathsf{Prefix}(\mathcal{A})$ is the CA obtained by removing all non-final states from $\mathcal{A}$ and taking the connected component including the initial state. Notably, $\mathscr{L}(\mathsf{Prefix}(\mathcal{A}))$ is the maximal prefix-closed language included in $\mathscr{L}(\mathcal{A})$. $\mathsf{Switch}(\mathcal{A})$ is the CA with final states obtained from $\mathcal{A}$ by selecting as final states the non-final states of $\mathcal{A}$, and vice versa.

3 Updates Correct for a Given Context

Given a system $\mathcal{C}[\mathcal{A}]$, an update replacing $\mathcal{A}$ with $\mathcal{B}$ is *correct* w.r.t. a property Φ iff whenever $\mathcal{C}[\mathcal{A}] \models \Phi$ also $\mathcal{C}[\mathcal{B}] \models \Phi$. We assume that $\mathcal{A}$ and $\mathcal{B}$ have the same interface, that is, the same set of node names. This is not restrictive since one can always add node names that are never used.

In this section we consider both the cases "for all properties, given context" and "given property, given context". We show that they can be both reduced to specific instances of the following problem: given a context $\mathcal{C}$ and a specification $\mathcal{S}$ representing the correct behavior of the whole system, find the $\mathcal{B}$s such that $\mathcal{C}[\mathcal{B}] \models \mathcal{S}$. By definition of $\models$, these are the solutions of the following language inequation:

$$\mathscr{L}(\mathcal{C}[\mathcal{B}]) \subseteq \mathscr{L}(\mathcal{S}) \tag{1}$$

Among all such $\mathcal{B}$s we select one generating the largest language, and we call it a *most general solution* of the inequation. Such a solution is unique up to language equivalence.

[4] This construction is not correct for general CAs with final states, but it is correct in this restricted case [5].

Lemma 2. *For each context $\mathcal{C}$ and specification $\mathcal{S}$, Inequation* (1) *has a unique most general solution, up to language equivalence.*

Proof. Given a solution $\mathcal{B}$ of the inequation, thanks to congruence (Lemma 1) any CA generating the same language also is a solution. Hence, we can work up to language equivalence. Assume now that there are two most general solutions $\mathcal{B}_1$ and $\mathcal{B}_2$ which are not comparable. Consider the automaton $\mathcal{U}$ such that $\mathscr{L}(\mathcal{U}) = \mathscr{L}(\mathcal{B}_1) \cup \mathscr{L}(\mathcal{B}_2)$ built using the standard union construction for finite state automata. Let us prove that $\mathcal{C}[\mathcal{U}] \models \mathcal{S}$. Suppose by contradiction that this is not the case. Then there exists a trace $w \in \mathscr{L}(\mathcal{C}[\mathcal{U}])$ such that $w \notin \mathscr{L}(\mathcal{S})$. The trace is obtained from a run $\rho = (s_0, q_0) \xrightarrow{c_0} (s_1, q_1) \xrightarrow{c_1} \ldots$, which is the composition of a run ρ_C of $\mathcal{C}$ and a run ρ_U of $\mathcal{U}$. By hypothesis, ρ_U is either a run of $\mathcal{B}_1$ or a run of $\mathcal{B}_2$. As a consequence, ρ is either a run of $\mathcal{C}[\mathcal{B}_1]$ or of $\mathcal{C}[\mathcal{B}_2]$. Hence, the trace w belongs to $\mathscr{L}(\mathcal{S})$, against our hypothesis.

Since $\mathcal{U}$ is a solution of Inequation (1) more general than both $\mathcal{B}_1$ and $\mathcal{B}_2$ we have a contradiction. The thesis follows.

In Inequation (1), when $\mathcal{S}$ is the system before the update $\mathcal{C}[\mathcal{A}]$ we are in the setting "for all properties, given context", while when $\mathcal{S}$ is a CA representing a given property Φ we are in the setting "given property, given context".

Inequation (1) has been studied by the logic synthesis and controller design communities, where it is known as the "unknown component problem" [21]. The following result is part of the theory developed in [21]. The proof of the Theorem can be found in Appendix A.

Theorem 1. *$\mathcal{B}$ is a solution of Inequation* (1) *iff $\mathscr{L}(\mathcal{B}) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$.*

The literature does not provide, for our setting, a constructive way of building a most general CA satisfying the constraint above. We propose one below. One would expect that a most general CA is $\overline{\mathcal{C}[\overline{\mathcal{S}}]}$. However, since CAs are not closed under complementation, such a CA in general cannot be built. We show that $\mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}])))$ is the best possibile approximation which is a CA. We recall that $\mathcal{C}[\overline{\mathcal{S}}]$ can be built as described in Section 2.2. Let us start by showing two auxiliary results.

Lemma 3. *Let $\mathcal{A}$ be a CA with final states. Then $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathsf{Subset}(\mathcal{A}))$.*

Proof. Take $w \in \mathscr{L}(\mathcal{A})$. If w is a finite trace, then $w \in \mathscr{L}(\mathsf{Subset}(\mathcal{A}))$ follows from the correctness of the subset construction for finite traces. If w is an infinite trace, then there exists a run ρ of $\mathcal{A}$ that visits final states infinitely often. Hence, there is a corresponding run of $\mathsf{Subset}(\mathcal{A})$ that visits final states infinitely often, from which we can conclude that $w \in \mathscr{L}(\mathsf{Subset}(\mathcal{A}))$.

Lemma 4. *For every deterministic CA with final states $\mathcal{A}$ we have that: $\mathscr{L}(\mathsf{Prefix}(\mathsf{Switch}(\mathcal{A}))) \subseteq \overline{\mathscr{L}(\mathcal{A})}$.*

Proof. Take $w \in \mathscr{L}(\mathsf{Prefix}(\mathsf{Switch}(\mathcal{A})))$. Since Prefix and Switch preserve determinism, there exists a unique accepting run ρ of $\mathsf{Prefix}(\mathsf{Switch}(\mathcal{A}))$ over w. Since

the prefix-closure operator removes non-final states, the run ρ is made of final states only. The same run is also the unique run of $\mathsf{Switch}(\mathcal{A})$ over w. Hence ρ is the unique run in $\mathcal{A}$ over w, and it is composed by non-final states only. Thus, $w \notin \mathscr{L}(\mathcal{A})$.

We can now prove the correctness of our construction.

Theorem 2. *$\mathcal{B} = \mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}])))$ is a most general CA such that $\mathscr{L}(\mathcal{B}) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$.*

Proof. To prove the thesis we have to show that $\mathcal{B}$ satisfies $\mathscr{L}(\mathcal{B}) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$, and that any other CA $\mathcal{B}'$ satisfying $\mathscr{L}(\mathcal{B}') \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$ is such that $\mathscr{L}(\mathcal{B}') \subseteq \mathscr{L}(\mathcal{B})$.

We start by proving the first condition. By noticing that $\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}])$ is deterministic we can apply Lemma 4 obtaining $\mathscr{L}(\mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}])))) \subseteq \overline{\mathscr{L}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]))}$. By Lemma 3 we have that $\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]) \subseteq \mathscr{L}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]))$, which is equivalent to $\overline{\mathscr{L}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]))} \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$. The fact that $\mathscr{L}(\mathcal{B}) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$ follows by concatenating the two inclusions.

To prove that $\mathcal{B}$ is a most general CA, take any CA $\mathcal{B}'$ satisfying $\mathscr{L}(\mathcal{B}') \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$. Take any trace $w \in \mathscr{L}(\mathcal{B}')$. Since $\mathcal{B}'$ is a CA, its language is prefix closed and thus all prefixes of w belong to $\mathscr{L}(\mathcal{B}')$. We have two cases: either w is finite or it is infinite. When considering only finite traces, we have that $\mathscr{L}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]))) = \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$ since automata on finite traces can be complemented by applying the subset construction and then switching final and non-final states. Since w and all its prefixes are in $\overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}])}$, then they are also in $\mathscr{L}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}])))$. Hence, $w \in \mathscr{L}(\mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}])))) = \mathscr{L}(\mathcal{B})$.

Assume now that w is infinite. Since $\mathscr{L}(\mathcal{B}')$ is prefix closed, all the finite prefixes of w are in $\mathscr{L}(\mathcal{B}')$. By the argument above we have that they are also in $\mathscr{L}(\mathcal{B})$. Let w^i be the finite prefix of w of length i. Since $\mathcal{B}$ is deterministic, there exists a unique run ρ_i accepting w^i. The final state of ρ_i is an accepting state and ρ_i is a prefix of ρ_j for every $j > i$. We can build the accepting run ρ_ω for w by taking the limit of all runs ρ_i. Hence $w \in \mathscr{L}(\mathcal{B})$ and the thesis follows.

Theorems 3 and 5 below show that $\mathcal{B}$ is a most general update correct for a given context $\mathcal{C}$. The former considers a given property Φ, the latter any property representable as a CA.

Theorem 3. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ and a property Φ such that $\mathcal{C}[\mathcal{A}]_x \models \Phi$, $\mathcal{B} = \mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\Phi}]_x)))$ is a most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. Φ.*

Proof. Given that $\mathcal{C}[\mathcal{A}]_x \models \Phi$, replacing $\mathcal{A}$ with $\mathcal{X}$ is a correct update w.r.t. Φ iff $\mathcal{C}[\mathcal{X}]_x \models \Phi$. This amounts to say that $\mathscr{L}(\mathcal{C}[\mathcal{X}]_x) \subseteq \mathscr{L}(\Phi)$. By Theorem 1 we have that any correct update $\mathcal{X}$ must be such that $\mathscr{L}(\mathcal{X}) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\Phi}]_x)}$ and by Theorem 2 we have that a most general CA respecting the condition is $\mathcal{B} = \mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\Phi}]_x)))$.

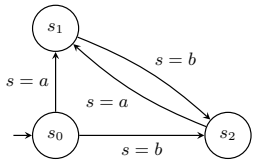


Fig. 3. Most general scheduler of Example 3.

Note that the above characterization does not depend on $\mathcal{A}$. However, if $\mathcal{C}[\mathcal{A}]$ does not satisfy the property Φ then every update is correct. Indeed the construction works for any property Φ, which may or may not hold for $\mathcal{C}[\mathcal{A}]$. Thus the approach can also be applied to ensure that new safety properties will hold after the update, e.g., to fix a bug or close a security vulnerability.

Theorem 4. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ and a property Φ, $\mathcal{B} = \mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\Phi}]_x)))$ is a most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ ensures that Φ holds in $\mathcal{C}[\mathcal{B}]_x$.*

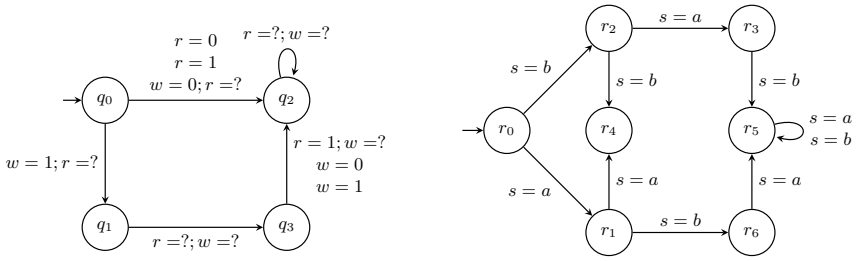
Proof. Analogous to the proof of Theorem 3.

Theorem 5.
Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$, $\mathcal{B} = \mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{C}[\mathcal{A}]_x}]_x)))$ is a most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. any property.

Proof. By Theorem 2 we have that $\mathscr{L}(\mathcal{B}) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{C}[\mathcal{A}]_x}]_x)}$. Hence, by Theorem 1 we also have that $\mathcal{B}$ satisfies Inequation (1), that is, that $\mathscr{L}(\mathcal{C}[\mathcal{B}]_x) \subseteq \mathscr{L}(\mathcal{C}[\mathcal{A}]_x)$. Take any property Φ. By definition $\mathcal{C}[\mathcal{A}]_x$ satisfies Φ iff $\mathscr{L}(\mathcal{C}[\mathcal{A}]_x) \subseteq \mathscr{L}(\Phi)$. By transitivity $\mathscr{L}(\mathcal{C}[\mathcal{B}]_x) \subseteq \mathscr{L}(\Phi)$, that is replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. any property.

To prove that $\mathcal{B}$ is a most general update, assume that there is a correct update $\mathcal{X}$ such that $\mathcal{B}$ is not more general than $\mathcal{X}$, that is $\mathscr{L}(\mathcal{X}) \not\subseteq \mathscr{L}(\mathcal{B})$. By Theorems 1 and 2 we have that $\mathcal{B}$ is a most general CA that satisfies Inequation (1). Hence, we have that $\mathscr{L}(\mathcal{C}[\mathcal{X}]_x) \not\subseteq \mathscr{L}(\mathcal{C}[\mathcal{A}]_x)$. Consider $\mathcal{C}[\mathcal{A}]_x$ as property. By definition we have that $\mathcal{C}[\mathcal{A}]_x \models \mathcal{C}[\mathcal{A}]_x$ but $\mathcal{C}[\mathcal{X}]_x \not\models \mathcal{C}[\mathcal{A}]_x$, against the hypothesis that replacing $\mathcal{A}$ with $\mathcal{X}$ is a correct update for any property.

Example 3. We can apply Theorem 5 to obtain a most general update for the case "given context, all properties" of the system in Example 1. By minimizing the result (up to language equivalence) we obtain the CA in Figure 3, where s_0 is the initial state. The solution recognizes the traces where one of the sequences $abababa\ldots$ and $bababab\ldots$ is communicated on node s. This implies that, e.g., replacing the original scheduler with a new one activating the registers in round-robin order b, a is a correct update. This matches the intuition, since the two registers are identical and swapping when they are accessible has no visible effect. Instead, using a scheduler that, e.g., always activates a and never activates b is



(a) CA Φ for P1. **(b)** Most general scheduler.

Fig. 4. CAs of Example 4.

not. A property falsified by this incorrect update is, for instance, P1 = "if *w=1* is executed at the first step, then at the third step *r=0* cannot be executed".

Example 4. Consider the property P1 above. It can be formalized by the CA Φ in Figure 4a. There, we use ? to denote 0, 1 or $\perp$, and we assume that at least one node in each constraint has non $\perp$ value. The system of Example 1 satisfies Φ. We want to characterize the updates that preserve Φ.

We can apply Theorem 3 to obtain the most general scheduler depicted in Figure 4b. Notice that it accepts the following computations:

- any computation of length at most 2: in this case the third step is never reached, and the property is trivially satisfied;
- any computation that starts with $s = a, s = b, s = a$ or $s = b, s = a, s = b$: in this case the value 1 written in the register at the first step is not changed in the second step, and made available in the third step.

We now move to the study of the complexity of our construction.

Theorem 6. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ finding a most general $\mathcal{B}$ such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update for a given property Φ or for any property, or an update that makes Φ hold is in 2-EXPTIME.*

Proof. All problems can be solved by building, for a suitable specification CA $\mathcal{S}$, the CA $\mathcal{B} = \mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}])))$. Since building $\mathcal{B}$ requires at most 2-EXPTIME, the thesis follows.

The 2-EXPTIME complexity arises from a double subset construction.

Theorem 7. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ and a property Φ such that $\mathcal{C}[\mathcal{A}]_x \models \Phi$, finding a most general $\mathcal{B}$ such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. Φ, or that makes Φ hold is EXPSPACE-hard.*

The lower bound is proved by reducing a suitable three-player game to Inequation (1). The game is played on a finite-state graph, with the first player (the component) and the third player (the specification) in a coalition against the second player (the context). At every round of the game, given the current

state, the successor state is determined by the choice of moves of the players. A suitable safety condition establishes who wins the game. The reduction shows that a winning strategy for Player 1 corresponds to a correct update of the system, if $\mathcal{C}[\mathcal{A}]_x \models \Phi$, and to an update that makes Φ hold otherwise. The problem of finding a winning strategy in this game is EXPSPACE-complete [9].

Theorem 7 deals with the case "given property, given context". It seems not easy to adapt the reduction to the case "all properties, given context". Finding a lower bound for the latter case is an open problem.

The rest of the section is devoted to the proof of Theorem 7, preceded by the needed background material on three-player games taken from [9].

Given an alphabet A_i of actions for each Player i ($i = 1, 2, 3$), a *three-player game* is a tuple $\mathcal{G} = \langle Q, q_0, \delta \rangle$ where:

- Q is a finite set of *states* with a distinguished *initial state* q_0;
- $\delta : Q \times A_1 \times A_2 \times A_3 \mapsto Q$ is a total and deterministic *transition function* that, given a current state q, and actions $a_1 \in A_1$, $a_2 \in A_2$, $a_3 \in A_3$ of the players, returns the unique successor state $q' = \delta(q, a_1, a_2, a_3)$.

Observations. For $i = 1, 2, 3$, a set $O_i \subseteq 2^Q$ of *observations* (for player i) is a partition of Q. Let $\mathrm{obs}_i : Q \mapsto O_i$ be the function that assigns to each state $q \in Q$ the (unique) observation for player i that contains q, i.e., such that $q \in \mathrm{obs}_i(q)$. The functions obs_i are extended to sequences $\rho = q_0 \dots q_n$ of states in the natural way. We say that player i is *blind* if $O_i = \{Q\}$, that is, player i has only one observation. Player i has *perfect information* if $O_i = \{\{q\} \mid q \in Q\}$, that is player i can distinguish each state.

Strategies. For $i = 1, 2, 3$, let Σ_i be the set of *strategies* $\sigma_i : O_i^+ \mapsto A_i$ of player i that, given a sequence of past observations, return the next action for player i. If a player i is blind, its strategies can be represented by infinite words on A_i.

Outcome. Given strategies $\sigma_i \in \Sigma_i$ (with $i = 1, 2, 3$), the *outcome play* from a state q_0 is the infinite sequence $\rho = q_0 q_1 \dots$ such that for all $j \geq 0$, we have $q_{j+1} = \delta(q_j, a_j^1, a_j^2, a_j^3)$ where $a_j^i = \sigma_i(q_0 \dots q_j)$, for $i = 1, 2, 3$.

Safety Objectives. Given a set $T \subseteq Q$ of *safe states*, the *safety objective* requires that the outcome only visits states in T.

Decision problem. Given a game $\mathcal{G} = \langle Q, q_0, \delta \rangle$ and a safety objective $T \subseteq Q$ the three-player decision problem is to decide if there exists a strategy σ_1 for Player 1 such that for each strategy σ_2 for Player 2, there exists a strategy σ_3 for Player 3 such that the outcome of the game satisfies the safety objective.

In order to derive a lower bound on the complexity of our approach, we show how we can reduce to Inequation (1) any three-player game where Player 1 is blind and Player 3 has perfect information. This problem is known to be EXPSPACE-complete [9]. Take a game $\mathcal{G} = \langle Q, q_0, \delta \rangle$ with alphabets A_1, A_2, A_3, observations O_2 for Player 2 and set of safe states $T \subseteq Q$. We build a context $\mathcal{C}_\mathcal{G}$ and a specification $\mathcal{S}_\mathcal{G}$ as follows:

- the context receives the moves of Player 1 from the component through a node act_1 in U;

– the context forwards the moves of Player 1 to the specification, chooses the move of Player 2 and receives the observations from the specification, respectively through nodes fwd_1, act_2, obs_2 in O;
– the specification receives the moves of Player 1 and 2, chooses the move of Player 3, computes the next state of the game following δ and sends the corresponding observation to the context.

Formally, the context is a two-state CA $\mathcal{C}_\mathcal{G} = \langle \{r_0, r_1\}, \{act_1, fwd_1, act_2, obs_2\}, r_0, \rightarrow_C \rangle$ such that:
– $r_0 \xrightarrow{act_1=a_1;fwd_1=a_1;act_2=a_2} r_1$ for every $a_1 \in A_1$ and $a_2 \in A_2$;
– $r_1 \xrightarrow{act_1=*;obs_2=o_2} r_0$ for some "dummy move" $* \notin A_1$ and every $o_2 \in O_2$.

The definition of the specification $\mathcal{S}_\mathcal{G}$ is more involved. The language of $\mathcal{S}_\mathcal{G}$ includes all traces that describe a winning play for Player 1 and all traces that do not describe a play of the game (to prevent the context from cheating). To model the alternation between moves and observations, the set of states of $\mathcal{S}_\mathcal{G}$ includes two copies of the safe states T of the game and a special sink state *WIN* to generate traces that do not correspond to a play. Formally $\mathcal{S}_\mathcal{G} = \langle T \cup T' \cup \{WIN\}, \{fwd_1, act_2, obs_2\}, q_0, \rightarrow_S \rangle$ where:
– T is the set of safe states of $\mathcal{G}$ and $T' = \{q' \mid q \in T\}$;
– $q_i \xrightarrow{fwd_1=a_1;act_2=a_2} q'_j$ iff there exists $a_3 \in A_3$ such that $\delta(q_i, a_1, a_2, a_3) = q_j$;
– $q'_j \xrightarrow{obs_2=o_2} q_j$ if $O_2(q_j) = o_2$;
– $q'_j \xrightarrow{obs_2=o_2} WIN$ if $O_2(q_j) \neq o_2$;
– $WIN \xrightarrow{c} WIN$ for every $c \in \mathsf{CIO}(O)$.

Lemma 5. *Let $\mathcal{B}$ be a CA that is a most general solution of $\mathscr{L}(\mathcal{C}_\mathcal{G}[\mathcal{B}]_s) \subseteq \mathscr{L}(\mathcal{S}_\mathcal{G})$. Then Player 1 has a winning strategy $w = u_0 u_1 \ldots$ on the game $\mathcal{G}$ iff $w_* = u_0 * u_1 * \ldots$ belongs to $\mathscr{L}(\mathcal{B})$.*

Proof. Let us start by proving that if w is a winning strategy then $w_* \in \mathscr{L}(\mathcal{B})$. Suppose towards a contradiction that $w = u_0 u_1 \ldots$ is a winning strategy for Player 1 but that $w_* \notin \mathscr{L}(\mathcal{B})$. Since the language of $\mathcal{B}$ is prefix closed, we can find a finite prefix w'_* of w_* such that $w'_* \notin \mathscr{L}(\mathcal{B})$. Let $\mathcal{Y}$ be a CA such that $\mathscr{L}(\mathcal{Y})$ is the set of prefixes of w'_*. Take any computation $r_0 \xrightarrow{act_1=u_0;fwd_1=u_0;act_2=v_0} r_1 \xrightarrow{act_1=*;obs_2=o_1} r_1 \ldots$ of the context synchronizing with w'_*. We build a matching computation of the specification as follows. Since w is a winning strategy for Player 1, at any step $i \geq 0$ there exists z_i such that $\delta(q_i, u_i, v_i, z_i) = q_{i+1}$ with $q_{i+1} \in T$. Hence, $q_i \xrightarrow{fwd_1=u_i;act_2=v_i} q'_{i+1}$ is a transition of $\mathcal{S}_\mathcal{G}$. Consider now the next transition $r_1 \xrightarrow{act_1=*;obs_2=o_i} r_0$ of the context. If $O_2(q_{i+1}) \neq o_i$ then the specification goes to *WIN* and the trace generated by $\mathcal{C}_\mathcal{G}[\mathcal{Y}]_s$ belongs to $\mathscr{L}(\mathcal{S}_\mathcal{G})$. Otherwise, $q'_{i+1} \xrightarrow{obs_2=o_i} q_{i+1}$ is a transition of $\mathcal{S}_\mathcal{G}$. Thus, given that the computation of the context is arbitrary, we know that $\mathcal{Y}$ is a solution of Inequation (1), against the hypothesis that $\mathcal{B}$ was a most general one.

To prove that if $w_* \in \mathscr{L}(\mathcal{B})$ then w is a winning strategy for Player 1, let us assume that $w_* = u_0 * u_1 * \ldots \in \mathscr{L}(\mathcal{B})$. Consider a strategy $\sigma_2 : O_2^+ \mapsto A_2$ of

Player 2. We build a winning strategy σ_3 for Player 3. At each step simulate the game with a pair of transitions of $\mathcal{C}_\mathcal{G}[\mathcal{B}]_s$ as follows. At step $i \geq 0$, $\mathcal{B}$ performs the actions $act_1 = u_i$ followed by $act_1 = *$. The context performs the action $act_1 = u_i; fwd_1 = u_i; act_2 = v_i$, such that $\sigma_2(o_0 \dots o_{i-1}) = v_i$. Since $\mathcal{B}$ is a solution of Inequation (1), we can find a matching transition $q_i \xrightarrow{fwd_1=u_i;act_2=v_i} q'_{i+1}$ of $\mathcal{S}_\mathcal{G}$. For the second transition of the context we can choose the correct observation $r_1 \xrightarrow{act_1=*;obs_2=o_i} r_0$ with $o_i = O_2(q_{i+1})$. This last step is matched by the specification with a transition $q'_{i+1} \xrightarrow{obs_2=o_i} q_{i+1}$ where $q_{i+1} \neq \mathit{WIN}$. By definition of $\mathcal{S}_\mathcal{G}$, for every transition $q_i \xrightarrow{fwd_1=u_i;act_2=v_i} q_{i+1}$ we can find an action $z_i \in A_3$ such that $\delta(q_i, u_i, v_i, z_i) = q_{i+1}$ with $q_{i+1} \in T$. Hence the play is winning for Player 1, and given that we chose an arbitrary strategy σ_2 for Player 2, we have proved that the strategy w is winning for Player 1.

Thanks to Lemma 5 above, the problem of deciding whether there is a winning strategy in any three-player game with a blind Player 1 can be reduced to checking whether a most general solution of Inequation (1) for the given $\mathcal{C}_\mathcal{G}$ and $\mathcal{S}_\mathcal{G}$ contains at least one infinite word. In the case of correct update w.r.t. a property Φ, one takes a CA $\mathcal{A}$ that immediately deadlocks (hence $\mathcal{C}_\mathcal{G}[\mathcal{A}]$ satisfies any safety property) and computes a most general $\mathcal{B}$ such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. $\mathcal{S}_\mathcal{G}$. In the case of update that makes a property Φ hold, one takes a CA $\mathcal{A}$ generating every possible behavior (hence $\mathcal{C}_\mathcal{G}[\mathcal{A}]$ does not satisfy the property $\mathcal{S}_\mathcal{G}$) and computes a most general $\mathcal{B}$ such that replacing $\mathcal{A}$ with $\mathcal{B}$ makes $\mathcal{S}_\mathcal{G}$ hold. In both the cases Player 1 has a winning strategy if $\mathcal{B}$ contains at least an infinite word. Given that finding an infinite word is linear in the size of $\mathcal{B}$, and since solving the game is EXPSPACE-hard we know that finding $\mathcal{B}$ is at least as hard.

4 Updates Correct for all Contexts

In this section we study both the cases "given property, for all contexts" and "all properties, for all contexts". Similarly to the previous section, we can assume w.l.o.g. that $\mathcal{A}$ and $\mathcal{B}$ have the same interface U, and that all the contexts we consider have U as internal interface.

Let us start from the case of a given property. The property defines a minimum set of node names O required for the external interface of the context. For some properties, it turns out that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update iff the traces of $\mathcal{B}$ are included in the traces of $\mathcal{A}$. However, this is not the case for all the properties. For instance, all the updates are correct w.r.t. the properties $\mathbf{tt} = \mathsf{CIO}(O)^* \cup \mathsf{CIO}(O)^\omega$ or $\mathbf{ff} = \emptyset$. Indeed, in the asynchronous case these are the only possibilities.

Theorem 8. *Let Φ be a property and $\mathcal{A}$ a CA. For the asynchronous embedding, the most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. Φ in all the contexts is* $\mathbf{tt}$ *if Φ is either* $\mathbf{tt}$ *or* $\mathbf{ff}$*, $\mathcal{A}$ otherwise.*

Proof. If the property is $\mathbf{tt}$ then all the updates are trivially correct since $\mathcal{C}[\mathcal{B}]_a \models \mathbf{tt}$. If the property is $\mathbf{ff}$ then all the updates are trivially correct since $\mathcal{C}[\mathcal{A}]_a \not\models \mathbf{ff}$. Hence, the most general update $\mathbf{tt}$ is also correct.

Assume now that Φ is neither $\mathbf{tt}$ nor $\mathbf{ff}$. $\mathcal{A}$ is a correct update by definition. We need to show that all correct uppdates are contained in $\mathcal{A}$.

Assume towards a contradiction that $\mathscr{L}(\mathcal{B}) \not\subseteq \mathscr{L}(\mathcal{A})$ is a correct update. By hypothesis we can find a trace $u \in \mathscr{L}(\mathcal{B})$ such that $u \notin \mathscr{L}(\mathcal{A})$. We can assume w.l.o.g. that $u = u_0 \dots u_n$ is finite. Moreover, since Φ is not $\mathbf{tt}$, we can find a trace $o = o_0 o_1 \dots$ such that $o \notin \mathscr{L}(\Phi)$. We build a context $\mathcal{C}$ that recognizes u as follows. The context has $n+2$ states $s_0, \dots, s_{n+1}$ to recognize u, with s_0 initial state. Transitions are of the form $s_i \xrightarrow{u_i} s_{i+1}$ for $0 \leq i \leq n$. State s_{n+1} is a sink that can do any concurrent I/O operation $c \in \mathsf{CIO}(O)$. We have that $\mathcal{C}[\mathcal{B}]_a$ produces all the traces from the alphabet $\mathsf{CIO}(O)$, including the trace o, while $\mathcal{C}[\mathcal{A}]_a$ produces only the empty trace. Since $o \notin \mathscr{L}(\Phi)$, while the empty trace is in Φ, we have proved that replacing $\mathcal{A}$ with $\mathcal{B}$ is not a correct update against our hypothesis.

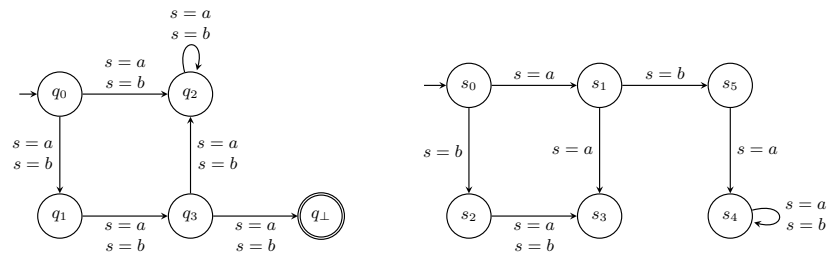
In the synchronous case the context and the component progress in lock-step. Given a property Φ, there are steps i in the computation on which Φ does not pose any restriction: if a trace z of length $i-1$ is in $\mathscr{L}(\Phi)$, then all the traces of length i having z as prefix are also in $\mathscr{L}(\Phi)$. Conversely, there are steps where Φ observes the system and whether a trace of length i is in $\mathscr{L}(\Phi)$ or not depends on the last action of the system. The *observation-point language* contains all the traces whose length identifies an observation point. Since the component $\mathcal{A}$ communicates on the internal interface U, the observation-point language is defined on the alphabet $\mathsf{CIO}(U)$.

Definition 5. *Let Φ be a property. The* observation-point language *of Φ is:*

$$\mathcal{R}(\Phi) = \{u \in \mathsf{CIO}(U)^* \mid \exists z \cdot c' \in \mathsf{CIO}(O)^*. z \in \mathscr{L}(\Phi) \wedge z \cdot c' \notin \mathscr{L}(\Phi) \wedge |u| = |z \cdot c'|\}$$

To compute a CA with final states accepting $\mathcal{R}(\Phi)$ one takes the complement $\overline{\Phi}$ of Φ. The CA $\overline{\Phi}$ has one final state $q^R_\perp$ which is a sink. The CA with final states $\mathcal{R}$ accepting $\mathcal{R}(\Phi)$ is obtained from $\overline{\Phi}$ by removing the self loops in the sink state $q^R_\perp$ and by replacing every transition of $\overline{\Phi}$ with a transition between the same pair of states for every label $c \in \mathsf{CIO}(U)$. Then, to build a most general CA $\mathsf{MGU}(\mathcal{A}, \Phi)$ such that replacing $\mathcal{A}$ with $\mathsf{MGU}(\mathcal{A}, \Phi)$ is a correct update w.r.t. Φ for all contexts, one can proceed as follows.

1. **Determinize** $\mathcal{R}$ using the subset construction.
2. **Complete** $\mathcal{A}$ by adding a sink state $q^A_\perp$ and obtaining $\mathcal{A}_\perp$.
3. **Compute the product of $\mathcal{A}_\perp$ with** $\mathsf{Subset}(\mathcal{R})$ using the synchronous join operator to obtain $\mathcal{A}_\perp \bowtie_s \mathsf{Subset}(\mathcal{R})$.
4. **Remove observation states**, that is all states $(q^A_\perp, Q_R)$ such that $q^R_\perp \in Q_R$, and take the connected component including the initial state.
5. **Transform the result into a CA without final states** by dropping the distinction between final and non-final states.



(a) CA with final states for $\mathcal{R}(\varphi)$. **(b)** Most general scheduler.

Fig. 5. CAs of Example 5.

$\mathsf{MGU}(\mathcal{A}, \Phi)$ can be computed in time which is a double exponential in the size of Φ and polynomial in the size of $\mathcal{A}$, where the two exponentials are due to the subset constructions.

Theorem 9. *Let Φ be a property and $\mathcal{A}$ a CA. For the synchronous embedding, the most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. Φ and for all contexts is* $\mathsf{MGU}(\mathcal{A}, \Phi)$.

Proof. We show that (i) $\mathsf{MGU}(\mathcal{A}, \Phi)$ is a correct update and (ii) every correct update $\mathcal{B}$ is such that $\mathscr{L}(\mathcal{B}) \subseteq \mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi))$.

To show (i) we prove that replacing $\mathcal{A}$ with $\mathsf{MGU}(\mathcal{A}, \Phi)$ is a correct update. To do this we prove the contrapositive implication, namely that for every context $\mathcal{C}$ such that $\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)] \not\models \Phi$ we have that $\mathcal{C}[\mathcal{A}] \not\models \Phi$. Thus, consider a context $\mathcal{C}$ such that $\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)] \not\models \Phi$ and let w be a trace in $\mathscr{L}(\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)])$ that does not satisfy Φ. Since Φ is prefix closed, there exists a minimal prefix w^j of w such that $w^j \notin \mathscr{L}(\Phi)$ and $w^{j-1} \in \mathscr{L}(\Phi)$. Since $\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)]$ is prefix closed, $w^j \in \mathscr{L}(\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)])$ and thus we can find a run $\rho = (q_0, p_0) \xrightarrow{w_0 \cup u_0} \ldots \xrightarrow{w_{j-1} \cup u_{j-1}} (q_j, p_j) \xrightarrow{w_j \cup u_j} (q_{j+1}, p_{j+1})$ of $\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)]$ such that the trace $u = u_0 \ldots u_j$ belongs to $\mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi))$. Two cases may arise. If $u \in \mathscr{L}(\mathcal{A})$ then $w^j \in \mathscr{L}(\mathcal{C}[\mathcal{A}])$ and we have proved that $\mathcal{C}[\mathcal{A}] \not\models \Phi$.

Otherwise, since $w^{j-1} \in \mathscr{L}(\Phi)$, $w^j \notin \mathscr{L}(\Phi)$ and $|w^j| = |u|$ then we have that $u \in \mathcal{R}(\Phi)$. This implies that there exists a run of $\mathcal{R}$ over u that ends in the sink state $q^R_\perp$ and that the unique run of $\mathsf{Subset}(\mathcal{R})$ over u ends in a state Q_R such that $q^R_\perp \in Q_R$. Hence, the last state of ρ (q_{j+1}, p_{j+1}) is equal to $(q^A_\perp, Q_R)$. Since all states $(q^A_\perp, Q_R)$ with $q^R_\perp \in Q_R$ are removed at step 4 of the construction of $\mathsf{MGU}(\mathcal{A}, \Phi)$, we have that ρ is not a valid run of $\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)]$ and a contradiction is found.

To show (ii), assume that $\mathcal{B}$ is a correct update but $\mathscr{L}(\mathcal{B}) \not\subseteq \mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi))$. This means that there exists $w \in \mathscr{L}(\mathcal{B})$ such that $w \notin \mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi))$. Since $\mathsf{MGU}(\mathcal{A}, \Phi)$ is prefix closed, we can find a minimal prefix w^j of w such that $w^{j-1} \in \mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi))$ but $w^j \notin \mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi))$ (notice that the empty prefix belongs to $\mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi))$). Since for every accepting run of $\mathcal{A}$ we can build a corresponding accepting run of $\mathsf{MGU}(\mathcal{A}, \Phi)$, we have that $w^j \notin \mathscr{L}(\mathcal{A})$. Notice

that the product CA $\mathcal{A}_\perp \bowtie_s \mathsf{Subset}(\mathcal{R})$ built at step 3 of the construction of $\mathsf{MGU}(\mathcal{A}, \Phi)$ is complete. Hence, there must exist a run of $\mathcal{A}_\perp \bowtie_s \mathsf{Subset}(\mathcal{R})$ over w^j that ends in a refusal state $(q_\perp^A, Q_R)$ with $q_\perp^R \in Q_R$, and thus we have that $w^j \in \mathcal{R}(\Phi)$. The definition of $\mathcal{R}(\Phi)$ implies that we can find a trace $z \cdot c' \in \mathsf{CIO}(O)^*$ such that $|w^j| = |z \cdot c'|$, $z \in \mathscr{L}(\Phi)$ and $z \cdot c' \notin \mathscr{L}(\Phi)$. We build a context $\mathcal{C}$ that recognizes w^j as follows. The context has $j+2$ states $s_0, \ldots, s_{j+1}$, with s_0 initial state. Transitions are of the form $s_i \xrightarrow{w_i \cup z_i} s_{i+1}$ for $0 \leq i \leq j-1$ and $s_j \xrightarrow{w_j \cup c'} s_{j+1}$. State s_{j+1} is a sink with no outgoing transitions. $\mathcal{C}[\mathcal{B}]_s$ produces the trace $z \cdot c'$, while $\mathcal{C}[\mathcal{A}]_s$ produces only prefixes of z (since $w^j \notin \mathscr{L}(\mathcal{A})$). Since Φ is a closed under prefix we have that if $z \in \mathscr{L}(\Phi)$ then all prefixes of z belong to $\mathscr{L}(\Phi)$. Since $z \cdot c' \notin \mathscr{L}(\Phi)$, while all prefixes of z belong to $\mathscr{L}(\Phi)$, we have proved that replacing $\mathcal{A}$ with $\mathcal{B}$ is not a correct update.

Example 5. Consider the property P1 represented by the CA Φ back in Figure 4a. By the above procedure we can first obtain the CA with final states for $\mathcal{R}(\Phi)$ in Figure 5a, and then the most general scheduler $\mathsf{MGU}(\mathcal{A}, \Phi)$ in Figure 5b, which makes the update correct in the synchronous case for every context and for the property Φ. We are left with two kinds of traces: traces with prefix $r = a, r = b, r = a$ that behave as $\mathcal{A}$ for the first 3 steps, and traces of length less than 3 that behave differently w.r.t. $\mathcal{A}$. This corresponds to the intuition that the property can only reject traces at step 3.

We now characterize the updates correct w.r.t. all the properties and all contexts. In this case there are strong requirements on the updates. To be correct for all contexts, the update needs to be correct for the context that reports every communication to the outside world. Since properties are sets of traces, the new component $\mathcal{B}$ should have at most the traces of $\mathcal{A}$. Indeed, this condition is necessary and sufficient, for both the synchronous and asynchronous embedding.

Theorem 10. *Let $\mathcal{A}$ be a CA. Any $\mathcal{B}$ such that $\mathscr{L}(\mathcal{B}) = \mathscr{L}(\mathcal{A})$ is a most general update such that replacing $\mathcal{A}$ with $\mathcal{B}$ is correct for all properties and for all contexts.*

Proof. Any $\mathcal{B}$ such that $\mathscr{L}(\mathcal{B}) = \mathscr{L}(\mathcal{A})$ is a correct update thanks to Lemma 1. We need to show that each correct update is contained in $\mathcal{A}$. We distinguish the cases of asynchronous composition and of synchronous composition. In the first case, take a property Φ which is neither equivalent to $\mathbf{tt}$ nor to $\mathbf{ff}$. The update is correct w.r.t. property Φ, thus thanks to Theorem 8 we have $\mathscr{L}(\mathcal{B}) \subseteq \mathscr{L}(\mathcal{A})$.

With asynchronous composition, note that the property $\Phi = \{(n = a)^i | i \in \mathbb{N}\}$ has $\mathcal{R}(\Phi) = \mathsf{CIO}(U)^*$. Hence the construction of $\mathsf{MGU}(\mathcal{A}, \Phi)$ builds a CA such that $\mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi)) = \mathscr{L}(\mathcal{A})$. According to Theorem 9, $\mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi)) = \mathscr{L}(\mathcal{A})$ is the most general correct update.

Example 6. By the above results, the update that replaces the original scheduler in Figure 1a with the most general scheduler in Figure 3 is not correct w.r.t. all contexts and all the properties, since its language is larger than the language of

the original scheduler. For instance, this update is not correct for the context that simply reports the actions of the scheduler to the outside world.

Instead, the inverse update, that replaces the most general scheduler with the original one, is correct for any property and for any context.

5 Conclusion and Related Work

We studied the problem of finding out whether an update replacing a component $\mathcal{A}$ with a component $\mathcal{B}$ in a given context $\mathcal{C}$ is correct w.r.t. a safety property Φ. We also characterized the updates correct in any context (for a given property), for any property (in a given context), and for any property in any context. In all the cases, we considered both synchronous and asynchronous composition.

While many approaches tackle system update [18], the problem of ensuring correctness of a system upon update has received scarce attention till now.

Approaches based on behavioral congruences, such as [8], allow one to prove the correctness of updates when a component is replaced by a syntactically different, but semantically equivalent one. Our approach is more general, allowing one to replace a component with a semantically different one.

Some approaches, such as [12], focus exclusively on type safety, that rules out obviously wrong behaviors, but is insufficient for establishing that given properties are preserved. In [13], instead, a program transformation to combine a program and an update into a new program presenting all the behaviors corresponding to applying the update at any allowed point is presented. The key advantage of our approach is that we can deal with many updates at once by comparing them with the most general one. In [23], a modular model checking approach to verify adaptive programs is proposed. They decompose the model checking problem following the temporal evolution of the system, while we decompose the verification problem following the structure of the system.

A line of work [2, 10] uses choreographic descriptions to obtain correctness of the updates by construction. However, this kind of approach can only deal with a few fixed properties such as deadlock freedom, race freedom and orphan-message freedom. Another related approach is presented in [11], where behavioral types are used to ensure that running sessions are not interrupted, and that provided services are preserved. Our approach is much more flexible than the two last approaches since it considers any property expressible as a CA.

The work in [17] categorizes different kinds of reconfigurations in the context of Reo connectors. Our updates correct for any property (in a given context) are called contractive in [17], and a property for which an update is correct (in a given context) is called an invariant for the update. However, in [17], nothing is said about the requirements that an update must satisfy to be contractive or to have a given invariant: these problems have been solved by the present paper.

The work in [21] is related to ours from the technical point of view. In particular, it provides us the framework to solve Inequation (1). However, [21] does not provide a construction for building an actual automaton in our case, namely, for CAs with both finite and infinite traces. Also, [21] has a different aim, since

it does not consider update at all. It highlights, however, a connection between update and another challenging problem: the automatic synthesis of systems from logical specifications. Polynomial algorithms for restricted classes of specifications have been identified [1, 7]. These results could be exploited both to make our approach more efficient and to extend it to properties that go beyond safety, like liveness and deadlock freedom. Another problem related to ours is supervisory control of discrete event systems (see, e.g., [4]). The main difference is in the composition mechanism, which features a feedback control loop and introduces latency, while this does not happen in our case.

The problem of characterizing correct updates can also be studied in other settings, and indeed we plan to consider some of them in future work. For instance, as hinted at above, one may consider more complex properties. Also, one may consider more complex automata, like timed automata or general CAs where the set of data values can be infinite. Finally, we want to apply our technique to more abstract models, starting from the ones based on CAs, such as REO [3] and Rebeca [19].

References

1. Alur, R., La Torre, S.: Deterministic generators and games for LTL fragments. ACM Trans. Comput. Logic 5(1), 1–25 (2004)
2. Anderson, G., Rathke, J.: Dynamic software update for message passing programs. In: APLAS. LNCS, vol. 7705, pp. 207–222. Springer (2012)
3. Arbab, F.: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science 14(3), 329–366 (2004)
4. Aziz, A., et al.: Supervisory control of finite state machines. In: CAV. LNCS, vol. 939, pp. 279–292. Springer (1995)
5. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by Constraint Automata. Sci. Comput. Program. 61(2), 75–113 (2006)
6. Baier, C., et al.: Formal verification for components and connectors. In: FMCO. LNCS, vol. 5751, pp. 82–101 (2008)
7. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. of Computer and System Sciences 78(3), 911–938 (2012)
8. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A behavioural congruence for web services. In: FSEN. LNCS, vol. 4767, pp. 240–256. Springer (2007)
9. Chatterjee, K., Doyen, L.: Games with a weak adversary. In: ICALP, LNCS, vol. 8573, pp. 110–121. Springer (2014)
10. Dalla Preda, M., et al.: Dynamic choreographies: Safe runtime updates of distributed applications. In: COORDINATION. LNCS, vol. 9037, pp. 67–82 (2015)
11. Di Giusto, C., Pérez, J.A.: Disciplined structured communications with consistent runtime adaptation. In: SAC. pp. 1913–1918. ACM (2013)
12. Duggan, D.: Type-based hot swapping of running modules. Acta Inf. 41(4-5), 181–220 (2005)
13. Hayden, C.M., et al.: Specifying and verifying the correctness of dynamic software updates. In: VSTTE. LNCS, vol. 7152, pp. 278–293. Springer (2012)
14. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing - degrees, models, and applications. ACM Comput. Surv. 40(3) (2008)

15. Lange, M.: Weak automata for the linear time μ-calculus. In: VMCAI. LNCS, vol. 3385, pp. 267–281. Springer (2005)
16. Leite, L.A.F., et al.: A systematic literature review of service choreography adaptation. Service Oriented Computing and Applications 7(3), 199–216 (2013)
17. Oliveira, N., Barbosa, L.S.: On the reconfiguration of software connectors. In: SAC. pp. 1885–1892. ACM (2013)
18. Seifzadeh, H., Abolhassani, H., Moshkenani, M.S.: A survey of dynamic software updating. J. of Software: Evolution and Process 25(5), 535–568 (2013)
19. Sirjani, M., et al.: Compositional semantics of an actor-based language using constraint automata. In: COORDINATION. LNCS, vol. 4038, pp. 281–297 (2006)
20. Tsai, M., Tsay, Y., Hwang, Y.: GOAL for games, omega-automata, and logics. In: CAV. LNCS, vol. 8044, pp. 883–889. Springer (2013)
21. Villa, T., et al.: The Unknown Component Problem - Theory and Application. Springer (2012)
22. Villa, T., et al.: Component-based design by solving language equations. Proceedings of the IEEE 103(11), 2152–2167 (2015)
23. Zhang, J., Goldsby, H., Cheng, B.H.C.: Modular verification of dynamically adaptive systems. In: AOSD. pp. 161–172. ACM (2009)

A Proof of Theorem 1

Our synchronous embedding is a particular case of the synchronous composition in [21, Chapter 2], thus the theory can be directly applied. Unfortunately, this is not the case for the asynchronous embedding. However, [22] generalizes the theory in [21] to any composition operator satisfying suitable properties. We show below that our asynchronous embedding satisfies those properties. As a consequence, the thesis follows from [22, Theorem 3.1] also in the asynchronous case.

To apply the theory in [22] to our asynchronous embedding, we show below that its language $\mathscr{L}(\mathcal{C}[\mathcal{A}]_a)$ can be rewritten as

$$\mathscr{L}(\mathcal{C}[\mathcal{A}]_a) = \left(\mathscr{L}(\mathcal{A})_{\top Y} \cap \mathscr{L}(\mathcal{C})_{\top X}\right)_{\perp X \circ Y} \tag{2}$$

where X and Y are alphabets, and $\perp$, $\top$, and $\circ$ are respectively a language restriction operator, a language expansion operator, and an alphabet composition operator satisfying the following properties:

H1 given two disjoint alphabets X and Y and a language $\mathscr{L}$ over X, $(\mathscr{L}_{\top Y})_{\perp X} = \mathscr{L}$,
H2 given two disjoint alphabets X and Y and two languages $\mathscr{L}_1$ and $\mathscr{L}_2$ over $Y \circ X$, $(\mathscr{L}_1 \cap \mathscr{L}_2)_{\perp X} = (\mathscr{L}_{1\perp X}) \cap (\mathscr{L}_{2\perp X})$ provided that $\mathscr{L}_1 = (\mathscr{L}_{1\perp X})_{\top Y}$ or $\mathscr{L}_2 = (\mathscr{L}_{2\perp X})_{\top Y}$,
H3 given two disjoint alphabets X and Y and a language $\mathscr{L}$ over $Y \circ X$, $\mathscr{L}_{\perp X} = \emptyset \Leftrightarrow \mathscr{L} = \emptyset$.

Intuitively, H1 states that $\perp$ is the right inverse of $\top$, H2 concerns distributivity of $\perp$ over $\cap$, and H3 states that language restriction produces the empty language iff the starting language is empty too.

In our setting X and Y are of the form $\mathsf{CIO}(N_i)$ for some given sets of node names N_1 and N_2. Hence, we define $\mathsf{CIO}(N_1) \circ \mathsf{CIO}(N_2) = \mathsf{CIO}(N_1 \cup N_2)$. Also, as a shortcut, we write $\top_N$ for $\top_{\mathsf{CIO}(N)}$ and $\bot_N$ for $\bot_{\mathsf{CIO}(N)}$. Given a language $\mathscr{L}$ over $\mathsf{CIO}(N_1 \cup N_2)$, consider the function $f : \mathsf{CIO}(N_1 \cup N_2) \mapsto \mathsf{CIO}(N_2)^*$ defined as

$$f(c) = \begin{cases} c{\downarrow_{N_2}} & \text{if } \mathsf{Nodes}(c) \cap N_2 \neq \emptyset \\ \varepsilon & \text{otherwise} \end{cases}$$

We define r as the homomorphic extension of f to finite and infinite sequences of symbols in $\mathsf{CIO}(N_1 \cup N_2)$. The *restriction of $\mathscr{L}$ to $\mathsf{CIO}(N_2)$* is the language $\mathscr{L}_{\bot_{N_2}} = \{r(w) \mid w \in \mathscr{L}\}$. Similarly, given a language $\mathscr{L}$ over $\mathsf{CIO}(N_2)$, the *expansion of $\mathscr{L}$ to $\mathsf{CIO}(N_1)$* is the language $\mathscr{L}_{\top_{N_1}} = \{w \in \mathsf{CIO}(N_1 \cup N_2)^* \cup \mathsf{CIO}(N_1 \cup N_2)^\omega \mid r(w) \in \mathscr{L}\}$.

Lemma 6. *Given a context $\mathcal{C}$ with node names $O \cup U$ and a component $\mathcal{A}$ with node names U, we have that $\mathscr{L}(\mathcal{C}[\mathcal{A}]_a) = \big(\mathscr{L}(\mathcal{A})_{\top O} \cap \mathscr{L}(\mathcal{C})_{\top \emptyset}\big)_{\bot O}$.*

Proof. We start the proof by observing that $\mathscr{L}(\mathcal{C})_{\top\emptyset} = \mathscr{L}(\mathcal{C})$. Moreover, we recall that $\mathcal{C}[\mathcal{A}]_a = (\mathcal{A} \bowtie_a \mathcal{C}){\downarrow_O}$. Since the language projection operator $\bot O$ corresponds to the projection operator $\downarrow_O$ on CAs, we only need to show that $\mathscr{L}(\mathcal{A} \bowtie_a \mathcal{C}) = \mathscr{L}(\mathcal{A})_{\top O} \cap \mathscr{L}(\mathcal{C})$.

Assume $w \in \mathscr{L}(\mathcal{A} \bowtie_a \mathcal{C})$, and let $\rho = (q_0, p_0) \xrightarrow{w_0} (q_1, p_1) \xrightarrow{w_1} \dots$ be a run accepting w. By definition of $\bowtie_a$, we have that $p_0 \xrightarrow{w_0} p_1 \xrightarrow{w_1} \dots$ is an accepting run of $\mathcal{C}$. Hence $w \in \mathscr{L}(\mathcal{C})$. By definition of $\bowtie_a$, for every $i < |w|$ we have that either $\mathsf{Nodes}(w_i) \cap U = \emptyset$ and $q_i = q_{i+1}$ or $\mathsf{Nodes}(w_i) \cap U \neq \emptyset$ and $q_i \xrightarrow{w_i{\downarrow_U}} q_{i+1}$. This implies that we can build a run of $\mathcal{A}$ that accepts the word $r(w)$. Since $r(w) \in \mathscr{L}(\mathcal{A})$ we have that $w \in \mathscr{L}(\mathcal{A})_{\top O}$ and thus also $w \in \mathscr{L}(\mathcal{A})_{\top O} \cap \mathscr{L}(\mathcal{C})$.

Assume now that $w \in \mathscr{L}(\mathcal{A})_{\top O} \cap \mathscr{L}(\mathcal{C})$. Then $w \in \mathscr{L}(\mathcal{C})$ and we can find a run ρ_C of $\mathcal{C}$ accepting w. Since $w \in \mathscr{L}(\mathcal{A})_{\top O}$ we have that $r(w) \in \mathscr{L}(\mathcal{A})$ and thus we can find a run ρ_A of $\mathcal{A}$ accepting $r(w)$. By synchronizing ρ_C and ρ_A we can build a run of $\mathcal{A} \bowtie_a \mathcal{C}$ accepting w as desired[5].

Lemma 7. *The language restriction and language expansion operators, $\bot$ and $\top$, satisfy the properties H1, H2, and H3.*

Proof.

H1 By definition of $\top$ and $\bot$.

H2 We assume that $\mathscr{L}_1 = (\mathscr{L}_{1\bot X})_{\top Y}$ (the other case is symmetric). Assume $w \in (\mathscr{L}_1 \cap \mathscr{L}_2)_{\bot X}$. Then there exists $w' \in \mathscr{L}_1 \cap \mathscr{L}_2$ such that $w = r(w')$. Since $w' \in \mathscr{L}_1$ and $w' \in \mathscr{L}_2$ then $w \in \mathscr{L}_{1\bot X}$ and $w \in \mathscr{L}_{2\bot X}$. Hence, $w \in (\mathscr{L}_{1\bot X}) \cap (\mathscr{L}_{2\bot X})$.
Assume now that $w \in (\mathscr{L}_{1\bot X}) \cap (\mathscr{L}_{2\bot X})$. Then $w \in \mathscr{L}_{1\bot X}$ and $w \in \mathscr{L}_{2\bot X}$. From the latter we know that there exists $w_2 \in \mathscr{L}_2$ such that $r(w_2) = w$.

[5] Note that if we allow as CIO the constant function with value $\bot$ then it is not guaranteed that we can synchronize ρ_C and ρ_A.

Consider now the set $(\mathscr{L}_{1\perp X})_{\top Y} = \{w' \mid r(w') \in \mathscr{L}_{1\perp X}\}$: since $w \in \mathscr{L}_{1\perp X}$ we have that $w_2 \in (\mathscr{L}_{1\perp X})_{\top Y}$, which by hypothesis coincides with $\mathscr{L}_1$. Hence, $w_2 \in \mathscr{L}_1 \cap \mathscr{L}_2$ and $w \in (\mathscr{L}_1 \cap \mathscr{L}_2)_{\perp X}$.

H3 For the $\Rightarrow$ direction, assume towards a contradiction that $\mathscr{L}_{\perp X} = \emptyset$ but $\mathscr{L} \neq \emptyset$. Take a word $w \in \mathscr{L}$: then the word $r(w) \in \mathscr{L}_{\perp X}$. The $\Leftarrow$ direction is trivial.