Static and Dynamic Property-Preserving Updates[☆]

Davide Bresolin

University of Padova, Italy
Email: davide.bresolin@unipd.it

Ivan Lanese[*]

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. The update can be performed both statically, by restarting the system, or dynamically. In both the cases, 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. In order to apply our approach also to dynamic update, we consider the *state transfer problem*, namely how to find the state in which the new component should be started to ensure that the overall behaviour is correct.

Keywords: Update, Constraint Automata, Property preservation, State transfer

1. Introduction

Update is a relevant topic [1], both for automatic updates, as in the context of adaptive systems [2] or autonomic computing [3], and for manual updates.

☆The authors acknowledge the support from the Italian INdAM – GNCS projects 2016 *Logic, automata and games for self-adaptive systems* and 2017 *Logic and automata for interval model-checking*. The authors also thank the anonymous reviewers for their useful comments and suggestions.

*Corresponding author

Update may be triggered by different needs, e.g., to fix bugs, to satisfy changing user requirements, or to match changes in the computing environment. Answering these needs is necessary since one wants systems to last for a long time in a changing scenario. However, a main point, namely correctness of the system after update, has received less attention till now, as remarked also in [4]. A real-world example highlighting the importance of this aspect is the following: what was called "one of the biggest computer errors in banking history", deducting about \$15 millions from over 100000 customers' accounts, was due to a software update [5].

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[1] to be updated $\mathcal{A}$, i.e., the system has the form $\mathcal{C}[\mathcal{A}]$. An update replaces component $\mathcal{A}$ with a new component $\mathcal{B}$, thus the system upon update has the shape $\mathcal{C}[\mathcal{B}]$. The update can be performed either *statically*, that is $\mathcal{C}[\mathcal{A}]$ is shut down, and the new system $\mathcal{C}[\mathcal{B}]$ starts in the initial state, or *dynamically*, by replacing $\mathcal{A}$ with $\mathcal{B}$ without restarting $\mathcal{C}$ (dynamic updates are also called live updates, or on-the-fly updates). In this second case, when $\mathcal{C}[\mathcal{B}]$ starts, $\mathcal{C}$ is in the same state as when $\mathcal{A}$ has been removed.

In the simpler case of static update a basic question is: how to build a most general new component $\mathcal{B}$ such that if system $\mathcal{C}[\mathcal{A}]$ satisfies a given property Φ, then also system $\mathcal{C}[\mathcal{B}]$ satisfies the same property? This question is answered in Section 3. Note that system $\mathcal{C}[\mathcal{B}]$ may satisfy further properties that system $\mathcal{C}[\mathcal{A}]$ does not satisfy. The answer to the question above, which relates components $\mathcal{A}$ and $\mathcal{B}$, depends both on the context $\mathcal{C}$ and on the property Φ. From this observation two generalisations emerge naturally. On one side, one may ask how to build a most general new component $\mathcal{B}$ such that for a given context $\mathcal{C}$, *all the properties* satisfied by system $\mathcal{C}[\mathcal{A}]$ are also satisfied by system $\mathcal{C}[\mathcal{B}]$ (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 new component $\mathcal{B}$ such that *for each context* $\mathcal{C}$ if system $\mathcal{C}[\mathcal{A}]$ satisfies property Φ, then system $\mathcal{C}[\mathcal{B}]$ satisfies the same property (Section 4). We say that such an update is correct (w.r.t. property Φ), hence it can be applied in any context. Finally, one may combine the two generalisations asking how to build a most general new component $\mathcal{B}$ to ensure correctness of update *in any context and w.r.t. any property* (Section 4).

The questions above can be rephrased also for dynamic update, by requiring that the system behaving first as $\mathcal{C}[\mathcal{A}]$, and then as $\mathcal{C}[\mathcal{B}]$, satisfies the desired properties. Here there is an additional element coming into play, namely how to select the state in which $\mathcal{B}$ starts, depending on the state of $\mathcal{A}$ just before the update. This problem is known as the *state transfer problem*, and has been first

[1]Component should be intended in a broad sense as a part of the system with a well-defined interface towards the rest of the system. In this sense, examples of components can be a procedure body, a library, or an actual component in some component model.

studied in [6]. The problem is notoriously hard and is normally dealt with by manual tedious work, as reported, e.g., in [7, 8]. We will compute (Section 5) a partial function that, for each state of $\mathcal{A}$, tells us whether the update can be safely performed or not and, in the first case, which states of $\mathcal{B}$ can be selected as starting states.

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 [9, 10], which have been used in the literature, e.g., to give a formal semantics to REO connectors [11] and Rebeca actors [12]. We use sets of prefix-closed finite and infinite traces as semantic framework. We concentrate on safety properties, thus prefix-closed sets of traces are enough. 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 main results of our approach by means of a simple running example. All the operations on Constraint Automata were computed using the tool GOAL [13][2], an interactive tool for defining and manipulating automata, which we extended to deal with Constraint Automata.

The present paper is an extended and revised version of [14], which concentrated on static update. Here we also consider dynamic update, hence the state transfer problem. We also provide more efficient constructions for update in a given context (both for a single property and for all properties) when the selected context is input-deterministic. The present paper includes full proofs of all the results, which were absent in [14]. Finally, the whole paper has been carefully revised, and extended with additional examples and a more detailed description of related work.

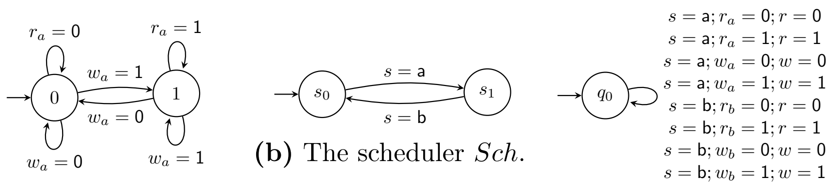
2. Constraint Automata

We model components, contexts and properties as (nondeterministic) Constraint Automata (CAs) [9], 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 [9], 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]GOAL files for the examples and intermediate results to compute them are available at `http://www.cs.unibo.it/~lanese/work/ic2018-adaptation-examples.zip`. GOAL interprets CAs as Finite State Automata, hence minimisation and determinisation produce the correct result, but other operations such as (synchronous and asynchronous) joins do not. Our extension allows one to perform joins on CAs correctly.



$r_a = 0$
$r_a = 1$
$w_a = 1$
$w_a = 0$
$w_a = 0$
$w_a = 1$
0
1
$s = \mathsf{a}$
$s = \mathsf{b}$
s_0
s_1
q_0
$s = \mathsf{a}; r_a = 0; r = 0$
$s = \mathsf{a}; r_a = 1; r = 1$
$s = \mathsf{a}; w_a = 0; w = 0$
$s = \mathsf{a}; w_a = 1; w = 1$
$s = \mathsf{b}; r_b = 0; r = 0$
$s = \mathsf{b}; r_b = 1; r = 1$
$s = \mathsf{b}; w_b = 0; w = 0$
$s = \mathsf{b}; w_b = 1; w = 1$

(a) The register R_a. **(b)** The scheduler Sch. **(c)** The synchroniser Syn.

Figure 1: The CAs for Example 1.

2. *N is a finite set of* nodes *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 $c : N \mapsto \mathsf{Data} \cup \{\perp\}$ *mapping every node in N to an element of* Data*, or to $\perp$ if no data is written/read. We require that in c there is always at least one node where data is read/written (that is, c is never the constant function with value $\perp$).*

We write transitions of a CA in the form $q \xrightarrow{c} p$, where p and q are states and c is a concurrent I/O operation (CIO). A *run* of a CA is a finite/infinite sequence $\rho = q_0 \xrightarrow{c_0} q_1 \xrightarrow{c_1} \dots$ 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* $w = c_0 c_1 \dots$. We write $q_0 \overset{w}{\Longrightarrow} q_n$ for a finite run starting in q_0, ending in q_n, and accepting word w. 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 a state $q_{\mathcal{A}}$ of a CA $\mathcal{A}$, we denote by $\mathscr{L}(q_{\mathcal{A}})$ the set of traces accepted by $\mathcal{A}$ when starting in state $q_{\mathcal{A}}$ (instead of q_0). We write w^i for the prefix of w of length i.

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

We consider a system which allows the user to read and write information from/to two one-bit registers, denoted as R_a and R_b. The system is the composition of four components, represented in Figure 1: two registers (only register R_a is shown, R_b is analogous), a scheduler Sch that determines which register is active, and a synchroniser Syn that communicates with the registers and the scheduler and proposes to the outside world the nodes r (read) and w (write) to access the active register. Labels on edges represent CIOs, written as semicolon-separated sets of assignments. Each assignment $n = \mathsf{d}$ 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. Register R_a *communicates with the synchroniser on nodes* r_a *(read* R_a*) and* w_a *(write* R_a*), while register* R_b *on nodes* r_b *and* w_b*. The scheduler interacts with the synchroniser on node* s*. Essentially, the two registers are scheduled in round-robin order* R_a, R_b*. The synchroniser is a one-state CA that forwards the external operations to the currently active register.*

As already said, 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. As a consequence the natural definition of the *satisfaction relation* between systems and properties is as follows.

Definition 2 (Satisfaction Relation)**.** *A CA* $\mathcal{A}$ *satisfies a property* Φ*, written* $\mathcal{A} \models \Phi$*, iff* $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\Phi)$.

CAs interpreted as sets of traces are as expressive as the safety linear μ-calculus [15, 16] and, as a consequence, more expressive than the safety fragment of temporal logics like LTL [17]. Alternatively, one could interpret CAs as trees, so to be more expressive than the safety fragment of the linear μ-calculus and of branching-time temporal logics like CTL. In this setting the natural notion of satisfaction would be simulation. We have chosen sets of traces over trees and trace inclusion over simulation (or other more complex behavioural relations) for simplicity and we leave the exploration of the alternatives to future work.

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 synchronisation 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 nodes 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 [10], 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 of them at the same time. In the synchronous case [18], 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 [10]: (synchronous or asynchronous) projection and (synchronous or asynchronous) join. We start by defining the two forms of join.

Definition 3 (Asynchronous Join)**.** *The* asynchronous join *of two CAs* $\mathcal{A} = \langle Q_\mathcal{A}, N_\mathcal{A}, q_0^\mathcal{A}, \to_\mathcal{A}\rangle$ *and* $\mathcal{C} = \langle Q_\mathcal{C}, N_\mathcal{C}, q_0^\mathcal{C}, \to_\mathcal{C}\rangle$ *is defined as the CA* $\mathcal{A} \bowtie_a \mathcal{C} = \langle Q_\mathcal{A} \times Q_\mathcal{C}, N_\mathcal{A} \cup N_\mathcal{C}, (q_0^\mathcal{A}, q_0^\mathcal{C}), \to_a\rangle$ *such that:*

- $(q,p) \xrightarrow{c}_a (q',p')$ *if* $\mathsf{Nodes}(c) \cap N_\mathcal{A} \neq \emptyset$*,* $\mathsf{Nodes}(c) \cap N_\mathcal{C} \neq \emptyset$*,* $q \xrightarrow{c\downarrow_{N_\mathcal{A}}}_\mathcal{A} q'$ *and* $p \xrightarrow{c\downarrow_{N_\mathcal{C}}}_\mathcal{C} p'$*;*

- $(q,p) \xrightarrow{c}_a (q',p)$ *if* $\mathsf{Nodes}(c) \cap N_\mathcal{C} = \emptyset$ *and* $q \xrightarrow{c\downarrow_{N_\mathcal{A}}}_\mathcal{A} q'$*;*
- $(q,p) \xrightarrow{c}_a (q,p')$ *if* $\mathsf{Nodes}(c) \cap N_\mathcal{A} = \emptyset$ *and* $p \xrightarrow{c\downarrow_{N_\mathcal{C}}}_\mathcal{C} p'$*.*

Definition 4 (Synchronous Join)**.** *The* synchronous join *of two CAs* $\mathcal{A} = \langle Q_\mathcal{A}, N_\mathcal{A}, q_0^\mathcal{A}, \rightarrow_\mathcal{A}\rangle$ *and* $\mathcal{C} = \langle Q_\mathcal{C}, N_\mathcal{C}, q_0^\mathcal{C}, \rightarrow_\mathcal{C}\rangle$ *is defined as the CA* $\mathcal{A} \bowtie_s \mathcal{C} = \langle Q_\mathcal{A} \times Q_\mathcal{C}, N_\mathcal{A} \cup N_\mathcal{C}, (q_0^\mathcal{A}, q_0^\mathcal{C}), \rightarrow_s\rangle$ *such that:*

- $(q,p) \xrightarrow{c}_s (q',p')$ *if* $\mathsf{Nodes}(c) \cap N_\mathcal{A} \neq \emptyset$*,* $\mathsf{Nodes}(c) \cap N_\mathcal{C} \neq \emptyset$*,* $q \xrightarrow{c\downarrow_{N_\mathcal{A}}}_\mathcal{A} q'$ *and* $p \xrightarrow{c\downarrow_{N_\mathcal{C}}}_\mathcal{C} p'$*.*

Given a CA $\mathcal{B}$ with nodes 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. In the synchronous projection at every step the CA $\mathcal{B}$ communicates with the environment via nodes in O, while in the asynchronous projection the CA $\mathcal{B}$ can take internal steps before communicating with the environment.

We first define the synchronous projection.

Definition 5 (Synchronous Projection)**.** *The* synchronous projection *of a CA* $\mathcal{B} = \langle Q, U \cup O, q_0, \rightarrow_\mathcal{B}\rangle$ *on* O *is defined as the CA* $\mathcal{B}\downarrow_O = \langle Q, O, q_0, \rightarrow_s\rangle$ *such that* $q \xrightarrow{c}_s p$ *iff there exist* $d \in \mathsf{CIO}(U \cup O)$ *such that* $d\downarrow_O = c$ *and* $q \xrightarrow{d}_\mathcal{B} p$*.*

To define the asynchronous 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 6 (Asynchronous Projection)**.** *The* asynchronous projection *of a CA* $\mathcal{B} = \langle Q, U \cup O, q_0, \rightarrow_\mathcal{B}\rangle$ *on* O *is defined as the CA* $\mathcal{B}\Downarrow_O = \langle Q, O, q_0, \rightarrow_a\rangle$ *such that* $q \xrightarrow{c}_a p$ *iff there exist* $d \in \mathsf{CIO}(U \cup O)$ *and* $r \in Q$ *such that* $d\downarrow_O = c$ *and* $q \rightsquigarrow_O^* r \xrightarrow{d}_\mathcal{B} p$*.*

The *asynchronous embedding* $\mathcal{C}[\mathcal{A}]_a$ and the *synchronous embedding* $\mathcal{C}[\mathcal{A}]_s$ of the component $\mathcal{A} = \langle Q_\mathcal{A}, U, q_0^\mathcal{A}, \rightarrow_\mathcal{A}\rangle$ in the context $\mathcal{C} = \langle Q_\mathcal{C}, U \cup O, q_0^\mathcal{C}, \rightarrow_\mathcal{C}\rangle$ 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 the nodes of the component $\mathcal{A}$ and expose only the nodes from the external interface O of the context $\mathcal{C}$. In the following, 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 Sch into the context, which is obtained by embedding the two registers* R_a *and* R_b *(in any order) into the synchroniser Syn. All the embeddings are asynchronous, hence the resulting system is* $Syn[R_a]_a[R_b]_a[Sch]_a$*. Out of the* 24 *possible states,* 8 *become unreachable after the join, and other* 8 *after the projection. By removing the unreachable states, we get the system represented in Figure 2. The states 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* R_a *and* R_b*, respectively.*

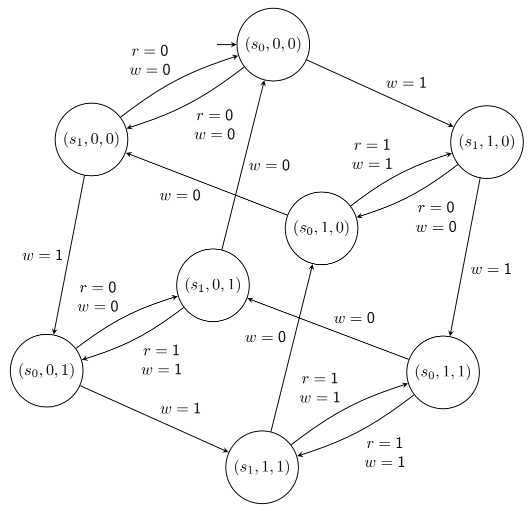


Figure 2: Embedding of the scheduler in the context.

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 $\rho_{\mathcal{C}}$ of $\mathcal{C}$ and a run $\rho_{\mathcal{A}}$ of $\mathcal{A}$. By hypothesis, there exists a run $\rho_{\mathcal{B}}$ of $\mathcal{B}$ with the same labels as $\rho_{\mathcal{A}}$. By composing $\rho_{\mathcal{B}}$ with the above run $\rho_{\mathcal{C}}$ we get a run of $\mathcal{C}[\mathcal{B}]_a$ with label w, against our hypothesis[3]. □

2.2. Determinisation and complementation of CAs

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 $\mathcal{A}$. This also shows that deterministic and nondeterministic CAs recognise the same class of languages. We can exploit this result to prove that the language of a CA $\mathcal{A}$ is determined by its finite traces.

Lemma 2. *Given a CA $\mathcal{A} = \langle Q, N, q_0, \rightarrow \rangle$ and an infinite trace w, we have that $w \in \mathscr{L}(\mathcal{A})$ if and only if for every $i \in \mathbb{N}$, $w^i \in \mathscr{L}(\mathcal{A})$.*

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

Proof. The only if direction follows from the fact that traces are closed under prefix. Let us consider the other implication. We can assume without loss of generality that $\mathcal{A}$ is deterministic. Hence, there exists a unique run ρ_i accepting w^i. Run 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 $w \in \mathscr{L}(\mathcal{A})$. $\square$

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 [10]. 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. We call the resulting CA a *CA with final states*. Formally, given a deterministic CA $\mathcal{A} = \langle Q, N, q_0, \rightarrow_{\mathcal{A}}\rangle$ we can build a CA $\overline{\mathcal{A}} = \langle Q_\perp, N, q_0, \rightarrow_{\overline{\mathcal{A}}}, F\rangle$ with final states 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);
- $q \xrightarrow{c}_{\overline{\mathcal{A}}} q'$ iff $q \xrightarrow{c}_{\mathcal{A}} q'$;
- $q \xrightarrow{c}_{\overline{\mathcal{A}}} q_\perp$ for all $q \in Q$ and $c \in \mathsf{CIO}(N)$ such that there is no q' such that $q \xrightarrow{c}_{\mathcal{A}} q'$;
- $q_\perp \xrightarrow{c}_{\overline{\mathcal{A}}} q_\perp$ for all $c \in \mathsf{CIO}(N)$.

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

We will also exploit the following operations on deterministic CAs with final states. $\mathsf{Prefix}(\mathcal{A})$ denotes the CA (without final states) obtained by removing all non-final states from the CA $\mathcal{A}$ with final states and taking the connected component including the initial state. Notably, $\mathscr{L}(\mathsf{Prefix}(\mathcal{A}))$ is the maximal prefix-closed language included in $\mathscr{L}(\mathcal{A})$. Also, $\mathsf{Switch}(\mathcal{A})$ is the CA with final states obtained from a deterministic CA $\mathcal{A}$ with final states by selecting as final states the non-final states of $\mathcal{A}$, and vice versa.

We will use CAs with final states as intermediate steps in our constructions, and recover CAs without final states by using function Prefix. Hence, when not otherwise stated, CAs do not have final states.

[4]This construction is not correct for general CAs with final states, but it is correct in this restricted case [10].

3. Updates Correct for a Given Context

We start this section with a formal definition of correct update.

Definition 7 (Correct Update)**.** *Given a system $\mathcal{C}[\mathcal{A}]$, an update replacing component $\mathcal{A}$ with a new component $\mathcal{B}$ 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 components $\mathcal{A}$ and $\mathcal{B}$ have the same interface, that is, the same set of nodes. This is not restrictive since one can always add nodes that are never used.

This section considers both the cases "all properties, given context" and "given property, given context". We show that they can be both reduced to instances of the following problem: given a context $\mathcal{C}$ and a specification $\mathcal{S}$ representing the correct behaviour of the whole system, find which are the possible new components $\mathcal{B}$ such that $\mathcal{C}[\mathcal{B}]_x \models \mathcal{S}$, with $x \in \{a, s\}$. By definition of $\models$, these are the solutions of the following language inequation:

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

Among all new components $\mathcal{B}$ satisfying the inequation we select one generating the largest language, and we call it a *most general solution* of the inequation. We denote it as $\mathsf{MGCA}_x(\mathcal{C}, \mathcal{S})$. Such a solution is unique up to language equivalence.

Lemma 3. *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 CA $\mathcal{U}$ 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 towards a 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 $\rho_\mathcal{C}$ of $\mathcal{C}$ and a run $\rho_\mathcal{U}$ of $\mathcal{U}$. By hypothesis, $\rho_\mathcal{U}$ 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 "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".

We discuss in Section 3.1 how to solve Inequation (1), we apply it to compute most general updates in Section 3.2, and we present more efficient solutions in the case of input-deterministic contexts in Section 3.3.

3.1. Solving the Inequation

Inequation (1) has been studied by the logic synthesis and controller design communities, where it is known as the "unknown component problem" [18]. The following result is part of the theory developed in [18, 19].

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

Our synchronous embedding is a particular case of the synchronous composition in [18, Chapter 2][5], thus the theory therein can be directly applied. Unfortunately, this is not the case for the asynchronous embedding. However, [19] generalises the theory in [18] to any composition operator satisfying suitable properties. We show in Appendix A that our asynchronous embedding satisfies those properties. As a consequence, the thesis follows also in the asynchronous case, now from [19, Theorem 3.1].
The literature does not provide, for our setting, a constructive way of building a most general CA satisfying the language inequation above. We propose one below. One would expect that for $x \in \{a, s\}$ a most general CA is $\overline{\mathcal{C}[\overline{\mathcal{S}}]_x}$. However, since CAs are not closed under complementation, such a CA in general cannot be built. We show that $\mathsf{MGCA}_x(\mathcal{C}, \mathcal{S}) = \mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x)))$ is the best possible 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 4. *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 5. *For every deterministic CA $\mathcal{A}$ with final states 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 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. *$\mathsf{MGCA}_x(\mathcal{C}, \mathcal{S})$ is a most general CA such that $\mathscr{L}(\mathsf{MGCA}_x(\mathcal{C}, \mathcal{S})) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]_x)}$, for every $x \in \{a, s\}$.*

[5]We remark that the definition in [18, Chapter 2] is at the level of sets of traces, while we give the definition at the level of CAs.

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

We start from the first condition. By noticing that $\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x)$ is deterministic we can apply Lemma 5 obtaining $\mathscr{L}(\mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x)))) \subseteq \overline{\mathscr{L}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x))}$. By Lemma 4 we have that $\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]_x) \subseteq \mathscr{L}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x))$, which is equivalent to $\overline{\mathscr{L}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x))} \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]_x)}$. The fact that $\mathscr{L}(\mathcal{B}) \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]_x)}$ follows by concatenating the two inclusions.

To prove that $\mathcal{B}$ is a most general CA, take any CA $\mathcal{B}'$ satisfying $\mathscr{L}(\mathcal{B}') \subseteq \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]_x)}$. 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 $\mathscr{L}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x))) = \overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]_x)}$ 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 $\overline{\mathscr{L}(\mathcal{C}[\overline{\mathcal{S}}]_x)}$, then they are also in $\mathscr{L}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x)))$. Hence, $w \in \mathscr{L}(\mathsf{Prefix}(\mathsf{Switch}(\mathsf{Subset}(\mathcal{C}[\overline{\mathcal{S}}]_x)))) = \mathscr{L}(\mathcal{B})$.

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 $\mathscr{L}(\mathcal{B})$. From Lemma 2 $w \in \mathscr{L}(\mathcal{B})$ and the thesis follows. □

3.2. Computing Most General Updates

Theorems 3 and 4 below show that $\mathsf{MGCA}_x(\mathcal{C}, \mathcal{S})$ for $x \in \{a, s\}$ is a most general update correct for a given context $\mathcal{C}$, for a suitable instantiation of $\mathcal{S}$. The former considers any property representable as a CA, the latter a given property Φ.

Theorem 3.
Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$, $\mathcal{B} = \mathsf{MGCA}_x(\mathcal{C}, \mathcal{C}[\mathcal{A}]_x)$ 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 $\mathscr{L}(\mathcal{X}) \nsubseteq \mathscr{L}(\mathcal{B})$. By Theorems 1 and 2 we have that $\mathcal{B}$ is a most general CA that satisfies Inequation (1). Hence, we have that $\mathscr{L}(\mathcal{C}[\mathcal{X}]_x) \nsubseteq \mathscr{L}(\mathcal{C}[\mathcal{A}]_x)$. 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. □

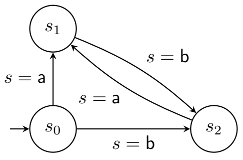


Figure 3: Most general scheduler $Sch_\forall$ of Example 3.

Example 3. *We can apply Theorem 3 to obtain a most general update for the case "given context, all properties" of the system in Example 1, where we consider the scheduler Sch as the component to be updated, and $Syn[R_a][R_b]$ as the context. The trivial application of the construction we propose would lead to a CA with more than* 30 *states. However, it is possible to minimise it (up to language equivalence), obtaining the CA $Sch_\forall$ in Figure 3, where s_0 is the initial state. The solution recognises the traces where one of the sequences* $\mathsf{abababa}\ldots$ *and* $\mathsf{bababab}\ldots$ *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 R_b, R_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 R_a and never activates R_b is not correct. 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".*

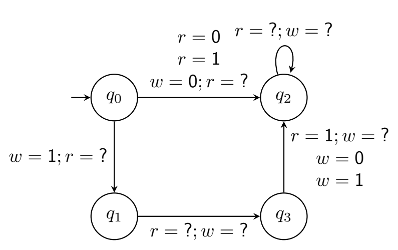
Theorem 4. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ and a property Φ such that $\mathcal{C}[\mathcal{A}]_x \models \Phi$, $\mathcal{B} = \mathsf{MGCA}_x(\mathcal{C}, \Phi)$ is a most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. Φ.*

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 $\mathcal{B} = \mathsf{MGCA}_x(\mathcal{C}, \Phi)$. $\square$

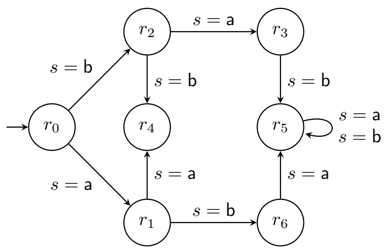
Example 4. *Consider the property P1 = "if* w=1 *is executed at the first step, then at the third step* r=0 *cannot be executed" from Example 3. It can be formalised by the CA Φ in Figure 4a. There, we use ? to denote* 0, 1 *or* $\perp$, *and we assume that at least one node in each constraint has non $\perp$ value. The system of Example 1 satisfies Φ. We want to characterise the updates that preserve Φ.*

We can apply Theorem 4 to obtain the most general scheduler Sch_Φ 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* $s = \mathsf{a}, s = \mathsf{b}, s = \mathsf{a}$ *or* $s = \mathsf{b}, s = \mathsf{a}, s = \mathsf{b}$*: in this case the value* 1 *written in the register at the first step is not changed in the second step, and it is made available in the third step.*

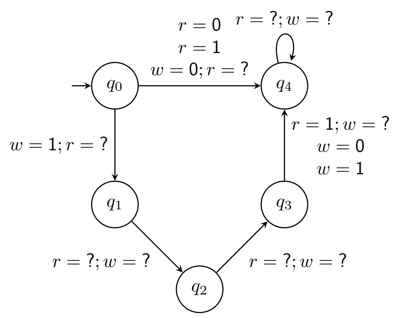


(a) CA Φ for P1.

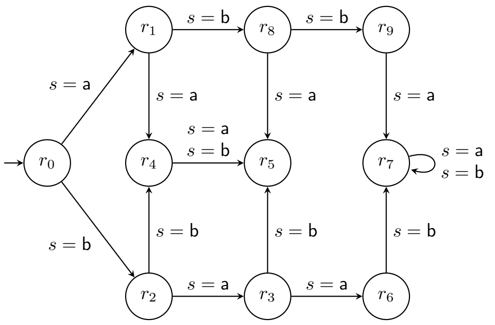


(b) Most general scheduler Sch_{Φ}.

Figure 4: CAs of Example 4.



(a) CA Ψ for P2.



(b) Most general scheduler Sch_{Ψ}.

Figure 5: CAs of Example 5.

Note that the construction used in Theorem 4 does not depend on $\mathcal{A}$. However, if $\mathcal{C}[\mathcal{A}]$ does not satisfy the property Φ then every update is trivially correct, since the premise in the definition of correct update (Definition 7) is false. If the property Φ does not hold for $\mathcal{C}[\mathcal{A}]$, then the same construction can be used to compute a most general update ensuring it, as shown by Theorem 5 below. Ensuring a new safety property after the update can be used, e.g., to fix a bug or close a security vulnerability.

Theorem 5. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ and a property Φ, $\mathcal{B} = \mathsf{MGCA}_x(\mathcal{C}, \Phi)$ 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 4. □

Example 5. *Consider the property P2 = "if* w=1 *is executed at the first step, then at the fourth step* r=0 *cannot be executed", represented by the CA Ψ in Figure 5a. This property does not hold in the system of Example 1. We can use Theorem 5 above to build the most general scheduler Sch_{Ψ} in Figure 5b, which accepts either computations of length at most* 3*, where the property is trivially satisfied since the fourth step is never executed; or computations where the register active at the first step is disabled at the second and third step, and*

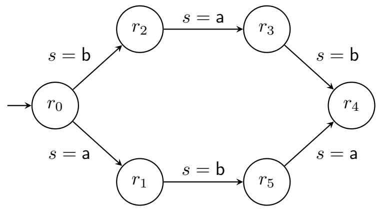


Figure 6: Most general scheduler $Sch_{\forall,\Psi}$ of Example 6.

activated again at the fourth step. In the latter case the property is satisfied since the value 1 *written at the first step is preserved and made available again at the fourth step.*

The result above ensures that the selected property Φ holds, but it does not preserve the properties that hold in $\mathcal{C}[\mathcal{A}]$. In order to enforce both the properties of $\mathcal{C}[\mathcal{A}]$ and the new property Φ one should consider as specification a CA whose language is the intersection of the language of Φ and of $\mathcal{C}[\mathcal{A}]$. Since Φ and $\mathcal{C}[\mathcal{A}]$ have the same set of nodes, the language of the synchronous join $\Phi \bowtie_s \mathcal{C}[\mathcal{A}]$ corresponds to the intersection of the language of language of Φ and of $\mathcal{C}[\mathcal{A}]$. Hence, we do not need to introduce a separate operator to compute the intersection of two CAs.

Theorem 6. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ and a property Φ, $\mathcal{B} = \mathsf{MGCA}_x(\mathcal{C}, \Phi \bowtie_s \mathcal{C}[\mathcal{A}]_x)$ is a most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ ensures that Φ and all the properties of $\mathcal{C}[\mathcal{A}]_x$ hold in $\mathcal{C}[\mathcal{B}]_x$.*

Proof. Analogous to the proof of Theorem 4. $\square$

Example 6. *Consider again the property P2 = "if* w=1 *is executed at the first step, then at the fourth step* r=0 *cannot be executed", represented by the CA Ψ in Figure 5a. We can now use Theorem 6 above to build the most general scheduler $Sch_{\forall,\Psi}$ in Figure 6, that respects both property P2 and all the properties of the original system in Figure 2. Notice that the result is the intersection (i.e., the synchronous join) of the scheduler $Sch_{\forall}$ in Figure 3 and the scheduler Sch_{Ψ} in Figure 5b, and includes only traces of length at most* 3*. This is due to the fact that any computation longer than* 3 *of the original system falsifies property P2.*

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

Theorem 7. *Given a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$, the time complexity of computing a most general new component $\mathcal{B}$ 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 (in isolation, or together with all the properties of $\mathcal{C}[\mathcal{A}]_x$) is a double exponential in the size of the specification $\mathcal{S}$ and an exponential in the* *size of $\mathcal{C}$*[6].

Proof. All the problems above can be solved by building, for a suitable specification CA $\mathcal{S}$, the CA $\mathsf{MGCA}_x(\mathcal{C}, \mathcal{S})$. Since the time complexity of building $\mathsf{MGCA}_x(\mathcal{C}, \mathcal{S})$ is a double exponential in the size of $\mathcal{S}$ and an exponential in the size of $\mathcal{C}$, the thesis follows. $\square$

The 2-EXPTIME complexity arises from a double subset construction. We show in Theorem 8 below that the problem is indeed EXPSPACE-hard at least in the case "given property, given context". It seems not easy to adapt the proof to the case "all properties, given context". Finding a lower bound for the latter case is an open problem.

Theorem 8. *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 new component $\mathcal{B}$ such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. Φ, or that makes Φ hold (in isolation) 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). The configuration of the game is a state in the graph. 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 [20]. A detailed description of the approach, including the proof of Theorem 8 and the needed background material on three-player games taken from [20], can be found in Appendix B.

3.3. Efficient Solutions for Input-deterministic Contexts

Theorem 8 shows that the considered problems are computationally hard. However, the same problems can be solved more efficiently if the considered context is input-deterministic[7].

Definition 8. *A CA $\mathcal{C} = \langle Q, U \cup O, q_0, \rightarrow \rangle$ is* input-deterministic *iff for each pair of transitions $p \xrightarrow{c} q$ and $p \xrightarrow{c'} q'$ such that $c\downarrow_U = c'\downarrow_U$ we have $c = c'$ and $q = q'$.*

Intuitively, in an input-deterministic context, given the starting state and the data flow on the interface towards the component, both the data flow on

[6]The size of a CA is the sum of the number of states and the number of transitions (which may depend on the size of set Data).

[7]Our definition of input-deterministic corresponds to the definition of deterministic for Finite State Machines, see e.g. [18].

the external interface and the next state are uniquely determined. In this case, Inequation (1) can be solved in polynomial time (a further condition is needed in the asynchronous case).

In the synchronous case, given an input-deterministic context $\mathcal{C} = \langle Q_\mathcal{C}, U \cup O, q_0^\mathcal{C}, \rightarrow_\mathcal{C}\rangle$ and a specification $\mathcal{S} = \langle Q_\mathcal{S}, O, q_0^\mathcal{S}, \rightarrow_\mathcal{S}\rangle$, we define a component $\mathsf{IDMGCA}_s(\mathcal{C}, \mathcal{S}) = \langle Q_\mathcal{C} \times Q_\mathcal{S} \cup \{q_\perp\}, U, (q_0^\mathcal{C}, q_0^\mathcal{S}), \rightarrow\rangle$ as follows:

1. $(q_1^\mathcal{C}, q_1^\mathcal{S}) \xrightarrow{u} (q_2^\mathcal{C}, q_2^\mathcal{S})$ iff there exists $o \in \mathsf{CIO}(O)$ such that $q_1^\mathcal{C} \xrightarrow{u \cup o}_\mathcal{C} q_2^\mathcal{C}$ and $q_1^\mathcal{S} \xrightarrow{o}_\mathcal{S} q_2^\mathcal{S}$;

2. $(q_1^\mathcal{C}, q_1^\mathcal{S}) \xrightarrow{u} q_\perp$ iff for each $o \in \mathsf{CIO}(O)$ there is no $q_2^\mathcal{C} \in Q_\mathcal{C}$ such that $q_1^\mathcal{C} \xrightarrow{u \cup o}_\mathcal{C} q_2^\mathcal{C}$;

3. $q_\perp \xrightarrow{u} q_\perp$ for every $u \in \mathsf{CIO}(U)$.

We can now prove the correctness of our construction.

Theorem 9. *Given an input-deterministic context $\mathcal{C}$ and a specification $\mathcal{S}$, $\mathcal{B}_s = \mathsf{IDMGCA}_s(\mathcal{C}, \mathcal{S})$ is a most general solution of $\mathscr{L}(\mathcal{C}[\mathcal{X}]_s) \subseteq \mathscr{L}(\mathcal{S})$ (Inequation (1), synchronous case).*

Proof. To prove the thesis we have to show that $\mathcal{B}_s$ satisfies $\mathscr{L}(\mathcal{C}[\mathcal{B}_s]_s) \subseteq \mathscr{L}(\mathcal{S})$, and that any other CA $\mathcal{B}'$ satisfying Inequation (1) is such that $\mathscr{L}(\mathcal{B}') \subseteq \mathscr{L}(\mathcal{B}_s)$.

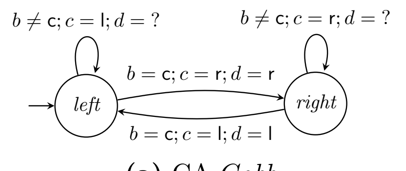
Let us consider the first implication. Take a word $w \in \mathscr{L}(\mathcal{C}[\mathcal{B}_s]_s)$ and consider a corresponding run $\rho = (p_0^\mathcal{C}, p_0^\mathcal{B}) \xrightarrow{o_0} (p_1^\mathcal{C}, p_1^\mathcal{B}) \xrightarrow{o_1} \ldots$, which is the composition of a run $\rho_\mathcal{C} = p_0^\mathcal{C} \xrightarrow{u_0 \cup o_0}_\mathcal{C} p_1^\mathcal{C} \xrightarrow{u_1 \cup o_1}_\mathcal{C} \ldots$ of $\mathcal{C}$ and a run $\rho_\mathcal{B} = p_0^\mathcal{B} \xrightarrow{u_0} p_1^\mathcal{B} \xrightarrow{u_1} \ldots$ of $\mathcal{B}_s$.

We can show by induction that, for every $i \geq 0$, $p_i^\mathcal{B} = (q_i^\mathcal{C}, q_i^\mathcal{S})$ with $q_i^\mathcal{C} = p_i^\mathcal{C}$. The base case follows from the construction of $\mathcal{B}_s$, which guarantees $p_0^\mathcal{B} = (q_0^\mathcal{C}, q_0^\mathcal{S})$. For the inductive case, suppose that $p_i^\mathcal{B} = (q_i^\mathcal{C}, q_i^\mathcal{S})$ with $q_i^\mathcal{C} = p_i^\mathcal{C}$. Consider the label u_i: since we know from $\rho_\mathcal{C}$ that $p_i^\mathcal{C} \xrightarrow{u_i \cup o_i}_\mathcal{C} p_{i+1}^\mathcal{C}$ and $q_i^\mathcal{C} = p_i^\mathcal{C}$, then we know that the transition $p_i^\mathcal{B} \xrightarrow{u_i} p_{i+1}^\mathcal{B}$ is derived from the first case of the construction of $\mathcal{B}_s$, hence $(q_i^\mathcal{C}, q_i^\mathcal{S}) \xrightarrow{u_i} (q_{i+1}^\mathcal{C}, q_{i+1}^\mathcal{S})$. Then there exists $o_i' \in \mathsf{CIO}(O)$ such that $q_i^\mathcal{C} \xrightarrow{u_i \cup o_i'}_\mathcal{C} q_{i+1}^\mathcal{C}$ and $q_i^\mathcal{S} \xrightarrow{o_i'}_\mathcal{S} q_{i+1}^\mathcal{S}$. Since $\mathcal{C}$ is input-deterministic we have that $o_i = o_i'$ and $q_{i+1}^\mathcal{C} = p_{i+1}^\mathcal{C}$, as desired.

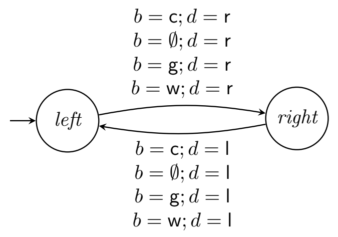
We can notice that the induction above builds a run $\rho_\mathcal{S} = q_0^\mathcal{S} \xrightarrow{o_0}_\mathcal{S} q_1^\mathcal{S} \ldots$ of the specification $\mathcal{S}$ that accepts w. Hence, $w \in \mathscr{L}(\mathcal{S})$.

Let us consider the second implication. Take a $\mathcal{B}'$ satisfying Inequation (1) and a word $w \in \mathscr{L}(\mathcal{B}')$. Suppose towards a contradiction that $w \notin \mathscr{L}(\mathcal{B}_s)$.

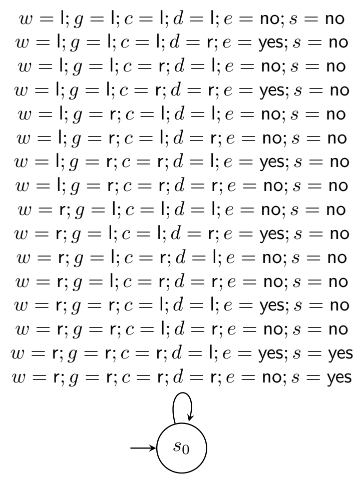
Consider the longest prefix w' of w such that there exists a run $\rho = (p_0^\mathcal{C}, p_0^{\mathcal{B}'}) \xrightarrow{u_0 \cup o_0} \ldots \xrightarrow{u_n \cup o_n} (p_{n+1}^\mathcal{C}, p_{n+1}^{\mathcal{B}'})$ of $\mathcal{C}[\mathcal{B}']_s$ with $w' = u_0 \ldots u_n$. Since $\mathcal{B}'$ is a solution of Inequation (1), we have that the word $v = o_0 o_1 \ldots o_n$ is in $\mathscr{L}(\mathcal{S})$, hence there is a run $\rho_\mathcal{S} = q_0^\mathcal{S} \xrightarrow{o_0}_\mathcal{S} \ldots \xrightarrow{o_n}_\mathcal{S} q_{n+1}^\mathcal{S}$ of $\mathcal{S}$. Hence, by using at each step the first case of the construction for $\mathcal{B}_s$ we can build a run $\rho_\mathcal{B} = (p_0^\mathcal{C}, q_0^\mathcal{S}) \xrightarrow{u_0} \ldots \xrightarrow{u_n} (p_{n+1}^\mathcal{C}, q_{n+1}^\mathcal{S})$ of $\mathcal{B}_s$ on w'. Hence $w' \in \mathscr{L}(\mathcal{B}_s)$. If $w' = w$


$b \neq \mathsf{c}; c = \mathsf{l}; d = ?$
$b \neq \mathsf{c}; c = \mathsf{r}; d = ?$
$b = \mathsf{c}; c = \mathsf{r}; d = \mathsf{r}$
left
right
$b = \mathsf{c}; c = \mathsf{l}; d = \mathsf{l}$

(a) CA $Cabb$.


$b = \mathsf{c}; d = \mathsf{r}$
$b = \emptyset; d = \mathsf{r}$
$b = \mathsf{g}; d = \mathsf{r}$
$b = \mathsf{w}; d = \mathsf{r}$
left
right
$b = \mathsf{c}; d = \mathsf{l}$
$b = \emptyset; d = \mathsf{l}$
$b = \mathsf{g}; d = \mathsf{l}$
$b = \mathsf{w}; d = \mathsf{l}$

(b) CA $Boat$.


$w = \mathsf{l}; g = \mathsf{l}; c = \mathsf{l}; d = \mathsf{l}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{l}; g = \mathsf{l}; c = \mathsf{l}; d = \mathsf{r}; e = \mathsf{yes}; s = \mathsf{no}$
$w = \mathsf{l}; g = \mathsf{l}; c = \mathsf{r}; d = \mathsf{l}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{l}; g = \mathsf{l}; c = \mathsf{r}; d = \mathsf{r}; e = \mathsf{yes}; s = \mathsf{no}$
$w = \mathsf{l}; g = \mathsf{r}; c = \mathsf{l}; d = \mathsf{l}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{l}; g = \mathsf{r}; c = \mathsf{l}; d = \mathsf{r}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{l}; g = \mathsf{r}; c = \mathsf{r}; d = \mathsf{l}; e = \mathsf{yes}; s = \mathsf{no}$
$w = \mathsf{l}; g = \mathsf{r}; c = \mathsf{r}; d = \mathsf{r}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{r}; g = \mathsf{l}; c = \mathsf{l}; d = \mathsf{l}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{r}; g = \mathsf{l}; c = \mathsf{l}; d = \mathsf{r}; e = \mathsf{yes}; s = \mathsf{no}$
$w = \mathsf{r}; g = \mathsf{l}; c = \mathsf{r}; d = \mathsf{l}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{r}; g = \mathsf{l}; c = \mathsf{r}; d = \mathsf{r}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{r}; g = \mathsf{r}; c = \mathsf{l}; d = \mathsf{l}; e = \mathsf{yes}; s = \mathsf{no}$
$w = \mathsf{r}; g = \mathsf{r}; c = \mathsf{l}; d = \mathsf{r}; e = \mathsf{no}; s = \mathsf{no}$
$w = \mathsf{r}; g = \mathsf{r}; c = \mathsf{r}; d = \mathsf{l}; e = \mathsf{yes}; s = \mathsf{yes}$
$w = \mathsf{r}; g = \mathsf{r}; c = \mathsf{r}; d = \mathsf{r}; e = \mathsf{no}; s = \mathsf{yes}$
s_0

(c) CA Mon.

Figure 7: CAs for the wolf, goat and cabbage problem of Example 7.

we have a contradiction since by assumption $w \notin \mathscr{L}(\mathcal{B}_s)$. Otherwise, since w' is the longest prefix, we have that for each $o \in \mathsf{ClO}(O)$ there is no transition $(p^{\mathcal{C}}_{m+1}, p^{\mathcal{B}'}_{m+1}) \xrightarrow{u_{m+1} \cup o} (p^{\mathcal{C}}_{m+2}, p^{\mathcal{B}'}_{m+2})$ in $\mathcal{C}[\mathcal{B}']_s$. Hence, by the construction of $\mathcal{B}_s$ we have that $(p^{\mathcal{C}}_{m+1}, q^{\mathcal{S}}_{m+1}) \xrightarrow{u_{m+1}} q_\perp$. Since $q_\perp$ is a sink state accepting all possible words, we have that $w \in \mathscr{L}(\mathcal{B}_s)$, against the hypothesis. □

Example 7 (Wolf, goat and cabbage problem)**.** *Consider the well-known wolf, goat and cabbage problem, in which a farmer must transport a wolf, a goat and a cabbage from one side of a river to the other, using a boat which can only hold one item at the time beyond the farmer. Hence, every time the farmer crosses the river, at least two items must be left unattended on the banks. If left unattended on the same bank, the wolf would eat the goat, and the goat would eat the cabbage.*

We can model this problem as an embedding of CAs, and exploit Theorem 9 to obtain the set of all sequences of crossings for the farmer that keep all the three items intact. The wolf, the goat and the cabbage are modelled each by a CA with two states, representing the fact that the item is either on the left bank or on the right bank of the river. We name the CAs $Wolf$, $Goat$ and $Cabb$, respectively. Each such CA has an interface composed by three nodes. Two nodes, b (for boat) and d (for destination), are the same for all CAs, while one is different: w for $Wolf$, g for $Goat$ and c for the $Cabb$. Let us describe in detail the behaviour of the CA $Cabb$, represented in Figure 7a (the others are analogous). $Cabb$ communicates its position at the next step by sending either l *(left) or* r *(right) on the node c. Furthermore, it synchronises with $Boat$ through the node b, that communicates the item carried on the boat (*w *for the* *wolf,* g *for the goat,* c *for the cabbage, and* $\emptyset$ *for empty boat), and node d, that communicates the position,* l *or* r*, of the boat at the next step.*

The CA Boat for the boat is shown in Figure 7b and is similar to the CA of an item: it consists of two states (left and right), and synchronises with the CAs $Wolf$*,* $Goat$ *and* $Cabb$ *through nodes b and d. The model includes also a "monitor"* Mon *(Figure 7c), that checks the positions of the three items and of the boat, and sends* yes *on node e (for eat) when the wolf is left unattended with the goat or the goat is left unattended with the cabbage, and* no *otherwise. Similarly, the monitor sends* yes *on node s (for success) when all items are on the right side of the river, and* no *otherwise. The context* $\mathcal{C}$ *is then defined as the synchronous join of all the above CAs, followed by the synchronous projection on b, e and s, formally* $(Mon \bowtie_s Boat \bowtie_s Wolf \bowtie_s Goat \bowtie_s Cabb)\downarrow_{\{b,e,s\}}$*. It communicates with the external world through nodes e and s, and with the component through node b. Thus, the component represents the farmer and decides which item is carried on the boat at each step.*

Given a state of the context and an input w*,* g*,* c*, or* $\emptyset$ *on node b from the component, the next state and the values to be communicated on nodes e and s are uniquely determined, and thus we have that* $\mathcal{C}$ *is input-deterministic. The safe strategies for the farmer are exactly those that respect the property P3 = "*yes *is never communicated on node e". By applying Theorem 9 to* $\mathcal{C}$ *and to the property P3 we can obtain the most general update in Figure 8. The states are labelled by the position of the items: the river is represented by a vertical line* $|$*, and the initial letter of each item (b represents the boat) is on its left or on its right according to the bank it is on. Indeed, the result provides the correct solutions to the game, namely the farmer must carry the goat first, come back with the boat empty, then transport either the wolf or the cabbage, come back with the goat, leave the goat to carry the missing item, and finally go back to fetch the goat. The most general update also features additional transitions, all leading to state* $q_\bot$*, which represent actions refused by the context (carrying an item which is not on the same side as the boat) and hence lead to a deadlock. Note that our approach cannot be used to build a most general CA satisfying the property P4 = "eventually* yes *is communicated on node s", since P4 is not a safety property.*

The asynchronous case is more complex. Indeed, because of the projection, even if the context is input-deterministic, the embedding of the component in the context may not be deterministic, and this would make the approach not sound. To avoid this issue we add the constraint that at each step the context synchronises with the component (but may not synchronise with the external world).

Given an input-deterministic context $\mathcal{C} = \langle Q_\mathcal{C}, U \cup O, q_0^\mathcal{C}, \rightarrow_\mathcal{C} \rangle$, such that for each transition $q_1^\mathcal{C} \xrightarrow{u \cup o}_\mathcal{C} q_2^\mathcal{C}$ the CIO $u \in \mathsf{CIO}(U)$ is not the constant function with value $\bot$, and a specification $\mathcal{S} = \langle Q_\mathcal{S}, O, q_0^\mathcal{S}, \rightarrow_\mathcal{S} \rangle$, we define $\mathsf{IDMGCA}_a(\mathcal{C}, \mathcal{S}) = \langle Q_\mathcal{C} \times Q_\mathcal{S} \cup \{q_\bot\}, U, (q_0^\mathcal{C}, q_0^\mathcal{S}), \rightarrow \rangle$ as follows:

1. $(q_1^\mathcal{C}, q_1^\mathcal{S}) \xrightarrow{u} (q_2^\mathcal{C}, q_2^\mathcal{S})$ iff there exists $o \in \mathsf{CIO}(O)$ such that $q_1^\mathcal{C} \xrightarrow{u \cup o}_\mathcal{C} q_2^\mathcal{C}$ and

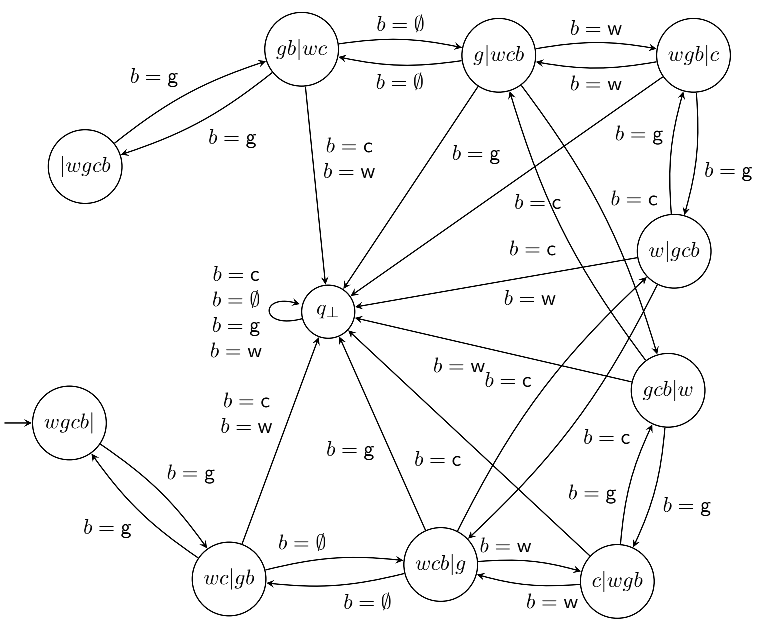


Figure 8: Most general update for the wolf, goat and cabbage problem of Example 7.

$q_1^{\mathcal{S}} \xrightarrow{o}_{\mathcal{S}} q_2^{\mathcal{S}}$;

2. $(q_1^{\mathcal{C}}, q_1^{\mathcal{S}}) \xrightarrow{u} (q_2^{\mathcal{C}}, q_1^{\mathcal{S}})$ iff $q_1^{\mathcal{C}} \xrightarrow{u}_{\mathcal{C}} q_2^{\mathcal{C}}$;

3. $(q_1^{\mathcal{C}}, q_1^{\mathcal{S}}) \xrightarrow{u} q_\perp$ iff none of the conditions above applies;

4. $q_\perp \xrightarrow{u} q_\perp$ for every $u \in \mathsf{CIO}(U)$.

We can now prove the correctness of our construction.

Theorem 10.
Given an input-deterministic context $\mathcal{C} = \langle Q_{\mathcal{C}}, U \cup O, q_0^{\mathcal{C}}, \rightarrow_{\mathcal{C}} \rangle$, such that for each transition $q_1^{\mathcal{C}} \xrightarrow{u \cup o}_{\mathcal{C}} q_2^{\mathcal{C}}$ the CIO u is not the constant function with value $\perp$, and a specification $\mathcal{S}$, $\mathcal{B}_a = \mathsf{IDMGCA}_a(\mathcal{C}, \mathcal{S})$ is a most general solution of $\mathscr{L}(\mathcal{C}[\mathcal{X}]_a) \subseteq \mathscr{L}(\mathcal{S})$ (Inequation (1), asynchronous case).

Proof. To prove the thesis we have to show that $\mathcal{B}_a$ satisfies $\mathscr{L}(\mathcal{C}[\mathcal{B}_a]_a) \subseteq \mathscr{L}(\mathcal{S})$, and that any other CA $\mathcal{B}'$ satisfying Inequation (1) is such that $\mathscr{L}(\mathcal{B}') \subseteq \mathscr{L}(\mathcal{B}_a)$.

Let us consider the first implication. Take a word $w \in \mathscr{L}(\mathcal{C}[\mathcal{B}_a]_a)$ and consider a corresponding run ρ'. In order to study ρ', we consider generalised runs where in transitions $p \xrightarrow{c} q$ CIO c may be the constant function with value $\perp$, and in this case $p = q$. There are a generalised run $\rho = (p_0^{\mathcal{C}}, p_0^{\mathcal{B}}) \xrightarrow{o_0} (p_1^{\mathcal{C}}, p_1^{\mathcal{B}}) \xrightarrow{o_1} \ldots$ of $\mathcal{C}[\mathcal{B}_a]_a$ and runs $\rho_{\mathcal{C}} = p_0^{\mathcal{C}} \xrightarrow{u_0 \cup o_0}_{\mathcal{C}} p_1^{\mathcal{C}} \xrightarrow{u_1 \cup o_1}_{\mathcal{C}} \ldots$ of $\mathcal{C}$ and $\rho_{\mathcal{B}} = p_0^{\mathcal{B}} \xrightarrow{u_0} p_1^{\mathcal{B}} \xrightarrow{u_1} \ldots$ of $\mathcal{B}_a$ such that ρ' is the run obtained from the generalised run ρ by dropping the steps where o_i is the constant function with value $\perp$.

We can show by induction that, for every $i \geq 0$, $p_i^{\mathcal{B}} = (q_i^{\mathcal{C}}, q_i^{\mathcal{S}})$ with $q_i^{\mathcal{C}} = p_i^{\mathcal{C}}$. The base case follows from the construction of $\mathcal{B}_a$, which guarantees $p_0^{\mathcal{B}} = (q_0^{\mathcal{C}}, q_0^{\mathcal{S}})$. For the inductive case, suppose that $p_i^{\mathcal{B}} = (q_i^{\mathcal{C}}, q_i^{\mathcal{S}})$ with $q_i^{\mathcal{C}} = p_i^{\mathcal{C}}$. Consider the label u_i: since we know from $\rho_{\mathcal{C}}$ that $p_i^{\mathcal{C}} \xrightarrow{u_i \cup o_i}_{\mathcal{C}} p_{i+1}^{\mathcal{C}}$ and $q_i^{\mathcal{C}} = p_i^{\mathcal{C}}$, then we know that the transition $p_i^{\mathcal{B}} \xrightarrow{u_i} p_{i+1}^{\mathcal{B}}$ is derived from case 2 or 1 of the construction of $\mathcal{B}_a$, depending on whether o_i is empty or not:

1. o_i is not empty, hence $(q_i^{\mathcal{C}}, q_i^{\mathcal{S}}) \xrightarrow{u_i} (q_{i+1}^{\mathcal{C}}, q_{i+1}^{\mathcal{S}})$. Then there exists $o'_i \in \mathsf{CIO}(O)$ such that $q_i^{\mathcal{C}} \xrightarrow{u_i \cup o'_i}_{\mathcal{C}} q_{i+1}^{\mathcal{C}}$ and $q_i^{\mathcal{S}} \xrightarrow{o'_i}_{\mathcal{S}} q_{i+1}^{\mathcal{S}}$. Since $\mathcal{C}$ is input-deterministic we have that $o_i = o'_i$ and $q_{i+1}^{\mathcal{C}} = p_{i+1}^{\mathcal{C}}$, as desired.

2. o_i is empty, hence $(q_i^{\mathcal{C}}, q_i^{\mathcal{S}}) \xrightarrow{u_i} (q_{i+1}^{\mathcal{C}}, q_i^{\mathcal{S}})$. Then $q_i^{\mathcal{C}} \xrightarrow{u_i}_{\mathcal{C}} q_{i+1}^{\mathcal{C}}$. Since $\mathcal{C}$ is input-deterministic we have that o_i is the constant function with value $\perp$ and $q_{i+1}^{\mathcal{C}} = p_{i+1}^{\mathcal{C}}$, as desired.

We can notice that the induction above builds a generalised run $\rho_{\mathcal{S}} = q_0^{\mathcal{S}} \xrightarrow{o_0}_{\mathcal{S}} q_1^{\mathcal{S}} \dots$ of the specification $\mathcal{S}$ that accepts w. Hence, $w \in \mathscr{L}(\mathcal{S})$.

Let us consider the second implication. Take a $\mathcal{B}'$ satisfying Inequation (1) and a word $w \in \mathscr{L}(\mathcal{B}')$. Suppose towards a contradiction that $w \notin \mathscr{L}(\mathcal{B}_a)$.

Consider the longest prefix w' of w such that there exists a run $\rho = (p_0^{\mathcal{C}}, p_0^{\mathcal{B}'}) \xrightarrow{u_0 \cup o_0} \dots \xrightarrow{u_n \cup o_n} (p_{n+1}^{\mathcal{C}}, p_{n+1}^{\mathcal{B}'})$ of $\mathcal{C}[\mathcal{B}']_a$ with $w' = u_0 \dots u_n$. Since $\mathcal{B}'$ is a solution of Inequation (1), the word $v = \mathsf{filter}(o_0 o_1 \dots o_n)$ is in $\mathscr{L}(\mathcal{S})$, where function filter just drops all CIOs which are the constant function with value $\perp$. Hence, there is a generalised run $\rho_{\mathcal{S}} = q_0^{\mathcal{S}} \xrightarrow{o'_0}_{\mathcal{S}} \dots \xrightarrow{o'_m}_{\mathcal{S}} q_{m+1}^{\mathcal{S}}$ of $\mathcal{S}$ such that $\mathsf{filter}(o'_0 \dots o'_m) = o_0 \dots o_n$. By using at each step case 1 or 2 of the construction for $\mathcal{B}_a$ we can build a run $\rho_{\mathcal{B}} = (p_0^{\mathcal{C}}, q_0^{\mathcal{S}}) \xrightarrow{u_0} \dots \xrightarrow{u_n} (p_{n+1}^{\mathcal{C}}, q_{n+1}^{\mathcal{S}})$ of $\mathcal{B}_a$ on w'. Thus, $w' \in \mathscr{L}(\mathcal{B}_a)$. If $w' = w$ we have a contradiction since by assumption $w \notin \mathscr{L}(\mathcal{B}_a)$. Otherwise, since w' is the longest prefix, we have that for each $o \in \mathsf{CIO}(O)$ there is no transition $(p_{m+1}^{\mathcal{C}}, p_{m+1}^{\mathcal{B}'}) \xrightarrow{u_{m+1} \cup o} (p_{m+2}^{\mathcal{C}}, p_{m+2}^{\mathcal{B}'})$ in $\mathcal{C}[\mathcal{B}']_s$. Hence, by the construction of $\mathcal{B}_a$ we have that $(p_{m+1}^{\mathcal{C}}, q_{m+1}^{\mathcal{S}}) \xrightarrow{u_{m+1}} q_{\perp}$. Since $q_{\perp}$ is a sink state accepting all possible words, we have that $w \in \mathscr{L}(\mathcal{B}_a)$, against the hypothesis. □

Let us now discuss the complexity of the construction.

Theorem 11. *Take a system $\mathcal{C}[\mathcal{A}]_x$ with $x \in \{a, s\}$ such that $\mathcal{C}$ is input-deterministic and, if $x = a$ then for each transition $q_1^{\mathcal{C}} \xrightarrow{u \cup o}_{\mathcal{C}} q_2^{\mathcal{C}}$ the CIO u is not the constant function with value $\perp$. Finding a most general new component $\mathcal{B}$ 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 (in isolation, or together with all the properties of $\mathcal{C}[\mathcal{A}]_x$) has complexity $\mathcal{O}(|\mathcal{C}||\mathcal{S}|)$ where $\mathcal{C}$ and $\mathcal{S}$ are, respectively, the size of the context $\mathcal{C}$ and of the specification $\mathcal{S}$.*

Proof. By definition of the constructions used to solve the problems under consideration. □

4. Updates Correct for all Contexts

In this section we study both the cases "given property, all contexts" and "all properties, all contexts". Similarly to the previous section, we can assume without loss of generality that components $\mathcal{A}$ 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 nodes O that the external interface of the context should provide. For some properties, replacing component $\mathcal{A}$ with a new component $\mathcal{B}$ 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 12. *Let Φ be a property and $\mathcal{A}$ a CA. For the asynchronous embedding, a 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 updates are less general than $\mathcal{A}$.

Assume towards a contradiction that $\mathscr{L}(\mathcal{B}) \nsubseteq \mathscr{L}(\mathcal{A})$ 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 without loss of generality 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 recognises u as follows. The context has $n+2$ states $s_0, \dots, s_{n+1}$ to recognise 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. $\square$

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. We capture this idea with an *observation-point language*, that we will use to build a most general update. Intuitively, the most general update must behave like $\mathcal{A}$ at observation points, while it can do anything at other points. Indeed, if the most general update does not behave like $\mathcal{A}$ at observation points one can build a context distinguishing them. The observation-point language contains all the traces in $\mathsf{CIO}(U)^*$ (since the component $\mathcal{A}$ communicates on the interface U) satisfying the constraint that the length of the trace corresponds to an observation point of the formula Φ.

Definition 9. *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 $\overline{\Phi}$ with final states has one final state $q_\perp^{\mathcal{R}}$ which is a sink. The CA $\mathcal{R}$ with final states accepting $\mathcal{R}(\Phi)$ is obtained from $\overline{\Phi}$ by removing the self loops in the sink state $q_\perp^{\mathcal{R}}$ 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.

Algorithm 1 Computation of the most general CA $\mathsf{MGU}(\mathcal{A}, \Phi)$

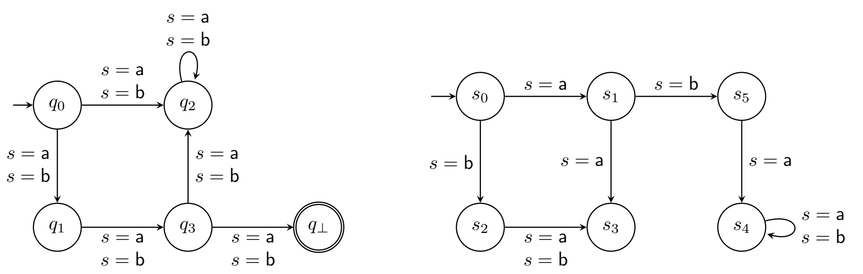
1. **Determinise** $\mathcal{R}$ using the subset construction, obtaining $\mathsf{Subset}(\mathcal{R})$.
2. **Transform** $\mathsf{Subset}(\mathcal{R})$ **into a CA without final states** by dropping the distinction between final and non-final states.
3. **Complete** $\mathcal{A}$ by adding a sink state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

Theorem 13. *Let Φ be a property and $\mathcal{A}$ a CA. For the synchronous embedding, a 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 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 $\rho = (q_0, p_0) \xrightarrow{w_0 \cup u_0} \dots \xrightarrow{w_{j-1} \cup u_{j-1}} (q_j, p_j) \xrightarrow{w_j \cup u_j} (q_{j+1}, p_{j+1})$ of $\mathcal{C}[\mathsf{MGU}(\mathcal{A}, \Phi)]$ such that the trace $u = u_0 \dots u_j$ 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



(a) CA with final states for $\mathcal{R}(\Phi)$. **(b)** Most general scheduler $\mathsf{MGU}(Sch, \Phi)$.

Figure 9: CAs of Example 8.

the sink state $q_\perp^{\mathcal{R}}$ and that the unique run of $\mathsf{Subset}(\mathcal{R})$ over u ends in a state $Q_{\mathcal{R}}$ such that $q_\perp^{\mathcal{R}} \in Q_{\mathcal{R}}$. Hence, the last state of ρ, (q_{j+1}, p_{j+1}), is equal to $(q_\perp^{\mathcal{A}}, Q_{\mathcal{R}})$. Since all states $(q_\perp^{\mathcal{A}}, Q_{\mathcal{R}})$ with $q_\perp^{\mathcal{R}} \in Q_{\mathcal{R}}$ are removed at step 5 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 4 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 an observation state $(q_\perp^{\mathcal{A}}, Q_{\mathcal{R}})$ with $q_\perp^{\mathcal{R}} \in Q_{\mathcal{R}}$, 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 recognises 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 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 8. *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 9a, and then the most general scheduler $\mathsf{MGU}(Sch, \Phi)$ in Figure 9b, 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 $r = \mathsf{a}, r = \mathsf{b}, r = \mathsf{a}$ that behave as the original scheduler Sch for the first 3 steps, and traces of length less than 3 that behave differently from Sch. This* *corresponds to the intuition that the property can only reject traces at step* 3.

Let us now discuss the complexity of the construction above.

Theorem 14. *Let Φ be a property and $\mathcal{A}$ a CA. For the synchronous embedding, the time complexity of computing a most general CA such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update w.r.t. Φ and for all contexts is a double exponential in the size of Φ and polynomial in the size of $\mathcal{A}$.*

Proof. Directly by inspection of the construction of $\mathsf{MGU}(\mathcal{A}, \Phi)$ in Algorithm 1 and of $\mathcal{R}$ just before it, by noticing that the two exponentials are due to the subset constructions. However, one can note that the subset construction for $\mathcal{R}$ is performed on a CA that, from the point of view of minimisation, is equivalent to an automaton with a single letter in its alphabet. In this case, the subset construction can be performed with time complexity $\mathcal{O}(e^{\sqrt{n \log n}})$ [21]. $\square$

We now characterise 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 correspond to sets of traces, the new component $\mathcal{B}$ should have at most the traces of $\mathcal{A}$, that is $\mathcal{B}$ should be a refinement of $\mathcal{A}$. Indeed, since refinement is a congruence, this condition is necessary and sufficient, for both the synchronous and asynchronous embedding.

Theorem 15. *Let $\mathcal{A}$ be a CA. A most general update such that replacing $\mathcal{A}$ with $\mathcal{B}$ is correct for all properties and all contexts is $\mathcal{A}$.*

Proof. $\mathcal{A}$ is a correct update thanks to Lemma 1. We need to show that each correct update is less general than $\mathcal{A}$. 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 12 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 $\mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi)) = \mathscr{L}(\mathcal{A})$. Thanks to Theorem 13, $\mathscr{L}(\mathsf{MGU}(\mathcal{A}, \Phi)) = \mathscr{L}(\mathcal{A})$ is a most general correct update. $\square$

Example 9. *By the above results, the update that replaces the original scheduler Sch in Figure 1a with the most general scheduler $Sch_\forall$ in Figure 3 is not correct w.r.t. all contexts and all 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 $Sch_\forall$ with the original one Sch, is correct for any property and for any context.

5. From Static to Dynamic Update

Previous sections concentrate on static update, where the system is shut down before update, and computation restarts from the initial state after update. We consider here dynamic update, where the component $\mathcal{A}$ is replaced by $\mathcal{B}$ at runtime. In particular, when $\mathcal{B}$ starts, the context $\mathcal{C}$ is not necessarily in the initial state, but it is in the same state it was in when $\mathcal{A}$ was removed. In this setting, starting $\mathcal{B}$ in the initial state may be wrong. This raises the issue of defining how state transfer [6] is performed: if the system is updated when component $\mathcal{A}$ was in state q, how to find a state q' of $\mathcal{B}$ such that starting $\mathcal{B}$ in q' after the update produces a correct behaviour.

Definition 10 (Dynamically Correct Update)**.** *Let $\mathcal{C} = \langle Q_\mathcal{C}, U \cup O, q_0^\mathcal{C}, \rightarrow_\mathcal{C}\rangle$ be a context and $\mathcal{A} = \langle Q_\mathcal{A}, U, q_0^\mathcal{A}, \rightarrow_\mathcal{A}\rangle$ and $\mathcal{B} = \langle Q_\mathcal{B}, U, q_0^\mathcal{B}, \rightarrow_\mathcal{B}\rangle$ two components. In system $\mathcal{C}[\mathcal{A}]$, an update replacing component $\mathcal{A}$ with $\mathcal{B}$ is* dynamically correct *w.r.t. a property Φ and a partial function $f : Q_\mathcal{A} \mapsto Q_\mathcal{B}$ iff whenever $\mathcal{C}[\mathcal{A}] \models \Phi$ then $w_\mathcal{A} w_\mathcal{B} \in \mathscr{L}(\Phi)$ for each trace $w_\mathcal{A}$ obtained from a run $(q_0^\mathcal{C}, q_0^\mathcal{A}) \overset{w_\mathcal{A}}{\Longrightarrow} (q_\mathcal{C}, q_\mathcal{A})$ of $\mathcal{C}[\mathcal{A}]$ and each trace $w_\mathcal{B}$ obtained from a run of $\mathcal{C}[\mathcal{B}]$ starting in state $(q_\mathcal{C}, f(q_\mathcal{A}))$.*

We define below transfer functions: if a transfer function is used as f in the definition above and replacing $\mathcal{A}$ with $\mathcal{B}$ is a (statically) correct update, then the update is also dynamically correct.

Definition 11 (Transfer Function)**.** *Given two CAs $\mathcal{A} = \langle Q_\mathcal{A}, U, q_0^\mathcal{A}, \rightarrow_\mathcal{A}\rangle$ and $\mathcal{B} = \langle Q_\mathcal{B}, U, q_0^\mathcal{B}, \rightarrow_\mathcal{B}\rangle$ such that $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$, a* transfer function *$f : Q_\mathcal{A} \mapsto Q_\mathcal{B}$ is a partial function satisfying the following property: for every state $q_\mathcal{A} \in Q_\mathcal{A}$, if $f(q_\mathcal{A}) = q_\mathcal{B}$ then for every trace u such that $q_0^\mathcal{A} \overset{u}{\Longrightarrow} q_\mathcal{A}$ and $w \in \mathscr{L}(q_\mathcal{B})$ we have that $uw \in \mathscr{L}(\mathcal{B})$.*

Theorem 16. *Let $\mathcal{A}$ and $\mathcal{B}$ be two components such that $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$ and f a transfer function. If replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update (in a given context or in any context) w.r.t. a property Φ, then replacing $\mathcal{A}$ with $\mathcal{B}$ is dynamically correct w.r.t. Φ and f.*

Proof. We need to show that $w_\mathcal{A} w_\mathcal{B} \in \mathscr{L}(\Phi)$ for each trace $w_\mathcal{A}$ obtained from a run $(q_0^\mathcal{C}, q_0^\mathcal{A}) \overset{w_\mathcal{A}}{\Longrightarrow} (q_\mathcal{C}, q_\mathcal{A})$ of $\mathcal{C}[\mathcal{A}]$ and each trace $w_\mathcal{B}$ obtained from a run of $\mathcal{C}[\mathcal{B}]$ starting in state $(q_\mathcal{C}, f(q_\mathcal{A}))$. The run $(q_0^\mathcal{C}, q_0^\mathcal{A}) \overset{w_\mathcal{A}}{\Longrightarrow} (q_\mathcal{C}, q_\mathcal{A})$ of $\mathcal{C}[\mathcal{A}]$ is obtained by composing a run $q_0^\mathcal{C} \overset{w_\mathcal{C}^1}{\Longrightarrow} q_\mathcal{C}$ of $\mathcal{C}$ and a run $q_0^\mathcal{A} \overset{u}{\Longrightarrow} q_\mathcal{A}$ of $\mathcal{A}$. The trace $w_\mathcal{B}$ is obtained by composing a run $\rho_\mathcal{C}$ of $\mathcal{C}$ (starting in state $q_\mathcal{C}$) accepting word $w_\mathcal{C}^2$ and a run $\rho_\mathcal{B}$ of $\mathcal{B}$ (starting in state $f(q_\mathcal{A})$) accepting word w. By definition of transfer function, $uw \in \mathscr{L}(\mathcal{B})$. Since replacing $\mathcal{A}$ with $\mathcal{B}$ is a (statically) correct update then $w_\mathcal{A} w_\mathcal{B} \in \mathscr{L}(\Phi)$. □

If $\mathcal{B}$ is a most general update, then using a transfer function is not only a sufficient condition for ensuring that the update is dynamically correct, but also a necessary condition.

Algorithm 2 Computation of the largest simulation on $Q_{\mathcal{B}}$, approximating $\leq$.

```
function simulation (CA $\mathcal{B} = \langle Q_{\mathcal{B}}, U, q_0^{\mathcal{B}}, \rightarrow \rangle$)
{
  $\preceq$ = $\{(p,q) | p,q \in Q_{\mathcal{B}}\}$
  do
  {
    $\preceq'$ = $\preceq$
    for each $(p,q) \in \preceq$
    {
      if $\exists$ $p'$, $a$. $p \xrightarrow{a} p'$ and $q \not\xrightarrow{a}$ then
      {
        $\preceq$ = $\preceq \setminus \{(p,q)\}$
      }
      if $\exists$ $p'$, $a$, $q'$. $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $(p',q') \notin \preceq$ then
      {
        $\preceq$ = $\preceq \setminus \{(p,q)\}$
      }
    }
  }
  while($\preceq' \neq \preceq$)
  return($\preceq$)
}
```

Theorem 17. *Let $\mathcal{A}$ and $\mathcal{B}$ be two components such that $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$. If $\mathcal{B}$ is a most general update such that replacing $\mathcal{A}$ with $\mathcal{B}$ is a correct update (in a given context or in any context) w.r.t. a property Φ, then replacing $\mathcal{A}$ with $\mathcal{B}$ is dynamically correct w.r.t. Φ and f only if f is a transfer function.*

Proof. We show that if f is not a transfer function then the update is not dynamically correct. Since f is not a transfer function we can find a run $q_0^{\mathcal{A}} \overset{u}{\Longrightarrow} q_{\mathcal{A}}$ of $\mathcal{A}$ and a run $\rho_{\mathcal{B}}$ of $\mathcal{B}$ (starting in state $f(q_{\mathcal{A}})$) accepting word w such that $uw \notin \mathscr{L}(\mathcal{B})$. Since $\mathcal{B}$ is a most general update, then there exists a computation of $\mathcal{C}$ on uw producing a trace which is not in $\mathscr{L}(\Phi)$, hence the update is not dynamically correct. □

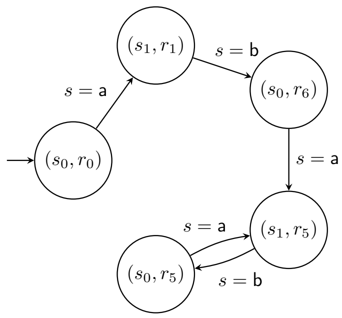
We show below how, given two components $\mathcal{A}$ and $\mathcal{B}$, to build a specific transfer function $tf_{\mathcal{A}\mapsto\mathcal{B}}$, and we discuss its properties. In the following we may drop the subscript $_{\mathcal{A}\mapsto\mathcal{B}}$ when the involved components are understood.

Definition 12. *Given a CA $\mathcal{B} = \langle Q_{\mathcal{B}}, U, q_0^{\mathcal{B}}, \rightarrow_{\mathcal{B}} \rangle$, we define a partial order relation $\leq$ on the states of $\mathcal{B}$ as follows: $q \leq q'$ iff $\mathscr{L}(q) \subseteq \mathscr{L}(q')$.*

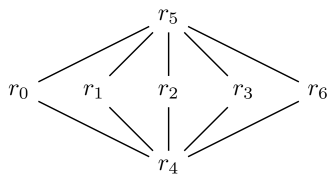
Algorithm 2 builds a relation $\preceq$ that is the largest simulation on $Q_{\mathcal{B}}$ [22]. It is known (see, e.g., [22]) that simulation implies trace inclusion: for every $p \preceq q$ we have that $\mathscr{L}(p) \subseteq \mathscr{L}(q)$. This proves that the relation $\preceq$ returned by Algorithm 2 is a subset of $\leq$ on $\mathcal{B}$.

Lemma 6. *The relation $\preceq$ returned by Algorithm 2 is a subset of $\leq$ on $\mathcal{B}$.*

Proof. Trivial, since simulation implies trace inclusion. □



(a) CA for $Sch \bowtie_s Sch_\Phi$.



(b) Lattice of relation $\preceq$.

Figure 10: Pictures for Example 10.

In order to define the transfer function $tf_{\mathcal{A}\mapsto\mathcal{B}}$ we take the synchronous join $\mathcal{A} \bowtie_s \mathcal{B}$ of the two CAs and we restrict it to reachable states. Then, for each $q_\mathcal{A} \in Q_\mathcal{A}$, we consider the set $S(q_\mathcal{A}) = \{q_\mathcal{B} \in Q_\mathcal{B} \mid (q_\mathcal{A}, q_\mathcal{B}) \text{ is reachable}\}$, ordered according to the simulation relation $\preceq$. We define $tf(q_\mathcal{A}) = q_\mathcal{B}$, where $q_\mathcal{B}$ is a greatest lower bound of $S(q_\mathcal{A})$ in $Q_\mathcal{B}$, namely a maximal element of $Q_\mathcal{B}$ such that $q_\mathcal{B} \preceq q'_\mathcal{B}$ for every $q'_\mathcal{B} \in S(q_\mathcal{A})$. Note that $q_\mathcal{B}$ is not necessarily an element of $S(q_\mathcal{A})$. However, if it belongs to $S(q_\mathcal{A})$, then it is its minimum (and it is unique). If no such element exists, then $tf(q_\mathcal{A}) = \bot$.

Theorem 18. *Given two components $\mathcal{A}$ and $\mathcal{B}$, $tf_{\mathcal{A}\mapsto\mathcal{B}}$ is a transfer function.*

Proof. To prove that tf is a transfer function, we have to prove that for every state $q_\mathcal{A} \in Q_\mathcal{A}$, if $tf(q_\mathcal{A}) = q_\mathcal{B}$ then for every trace u such that $q_0^\mathcal{A} \overset{u}{\Longrightarrow} q_\mathcal{A}$ and $w \in \mathscr{L}(q_\mathcal{B})$ we have that $uw \in \mathscr{L}(\mathcal{B})$.

Take one u such that $q_0^\mathcal{A} \overset{u}{\Longrightarrow} q_\mathcal{A}$ and one $w \in \mathscr{L}(q_\mathcal{B})$. Then we have that there is a run of $\mathcal{A} \bowtie_s \mathcal{B}$ such that $(q_0^\mathcal{A}, q_0^\mathcal{B}) \overset{u}{\Longrightarrow} (q_\mathcal{A}, q'_\mathcal{B})$. From the definition of tf, we have that $q_\mathcal{B} \preceq q'_\mathcal{B}$ and thus, by Lemma 6 that $w \in \mathscr{L}(q'_\mathcal{B})$. Hence, $uw \in \mathscr{L}(\mathcal{B})$. □

We now discuss the complexity of the computation of transfer function tf.

Theorem 19. *Given two CAs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$, a transfer function $tf_{\mathcal{A}\mapsto\mathcal{B}}$ can be built in time $\mathcal{O}(|\mathcal{A}||\mathcal{B}| + |\mathcal{B}|^3)$, where $|\mathcal{A}|$ is the size of $\mathcal{A}$ and $|\mathcal{B}|$ the size of $\mathcal{B}$.*

Proof. Algorithm 2 has complexity $\mathcal{O}(|\mathcal{B}|^3)$. The join of $\mathcal{A}$ and $\mathcal{B}$ can be computed in time $\mathcal{O}(|\mathcal{A}||\mathcal{B}|)$, and restricting it to reachable states can also be done in $\mathcal{O}(|\mathcal{A}||\mathcal{B}|)$. The computation of tf requires an additional $\mathcal{O}(|\mathcal{A}||\mathcal{B}|)$, hence the total complexity is $\mathcal{O}(|\mathcal{A}||\mathcal{B}| + |\mathcal{B}|^3)$ □

Example 10. *Consider the update that replaces the scheduler Sch of the system in Example 1 with the most general scheduler Sch_Φ satisfying property P1 = "if* w=1 *is executed at the first step, then at the third step* r=0 *cannot be executed", represented in Figure 4b.*

To obtain the transfer function tf *that makes such an update dynamically correct, we first compute both the join* $Sch \bowtie_s Sch_\Phi$ *shown in Figure 10a and, using Algorithm 2, the relation* $\preceq$ *on the states of* Sch_Φ *represented in Figure 10b.*

Then, the sets $S(\cdot)$ *for the states of* Sch *are*

$$S(s_0) = \{r_0, r_5, r_6\}$$
$$S(s_1) = \{r_1, r_5\}$$

Since the greatest lower bound of $S(s_0)$ *in* Q_{Sch_Φ} *is* r_4 *(*r_0 *and* r_6 *are uncomparable), while the greatest lower bound of* $S(s_1)$ *in* Q_{Sch_Φ} *is* r_1*, the transfer function* tf *is*

$$tf(s_0) = r_4$$
$$tf(s_1) = r_1$$

This means that the best possible ways to dynamically replace Sch *with* Sch_Φ *found by our approach are to stop* Sch *in state* s_0 *and start* Sch_Φ *in state* r_4*, or to stop* Sch *in state* s_1 *and start* Sch_Φ *in state* r_1*. Also stopping* Sch *in state* s_1 *and starting* Sch_Φ *in state* r_4 *would be correct, but would unnecessarily reduce the possible behaviours. One could imagine that, since* s_0 *is the initial state, stopping* Sch *in state* s_0 *and starting* Sch_Φ *in its initial state* r_0 *should also be correct, but this is not the case. Indeed,* Sch *could be in* s_0 *not only at the beginning, but also later in the computation, and in this second case starting* Sch_Φ *in state* r_0 *may not be correct.*

The example above shows that, beyond the transfer function tf, there are other transfer functions (and, since they are transfer functions, they produce dynamically correct updates). For instance, also the function that assigns $\perp$ to every state of Sch is a transfer function. We show below that if the new component $\mathcal{B}$ is deterministic then the transfer function tf is optimal.

Lemma 7. *If* $\mathcal{B}$ *is a deterministic CA, then the relation* $\preceq$ *returned by Algorithm 2 is precisely* $\leq$ *on* $\mathcal{B}$*.*

Proof. Trivial, since for deterministic automata simulation coincides with trace inclusion [22]. □

Theorem 20. *If* $\mathcal{A}$ *and* $\mathcal{B}$ *are two components with* $\mathcal{B}$ *deterministic, then* $tf_{\mathcal{A}\mapsto\mathcal{B}}$ *is a transfer function with maximum domain.*

Proof. To prove that tf has maximum domain, suppose towards a contradiction that there exists a transfer function tf' and a $q_\mathcal{A} \in Q_\mathcal{A}$ such that $tf(q_\mathcal{A}) = \perp$ and $tf'(q_\mathcal{A}) = q_\mathcal{B} \neq \perp$. From the definition of tf, $q_\mathcal{B}$ cannot be a greatest lower bound of $S(q_\mathcal{A})$ in $Q_\mathcal{B}$. As a consequence, there exists $q'_\mathcal{B} \in S(q_\mathcal{A})$ such that $q_\mathcal{B} \not\preceq q'_\mathcal{B}$. By Lemma 7, since $\mathcal{B}$ is deterministic we have that the relation $\preceq$

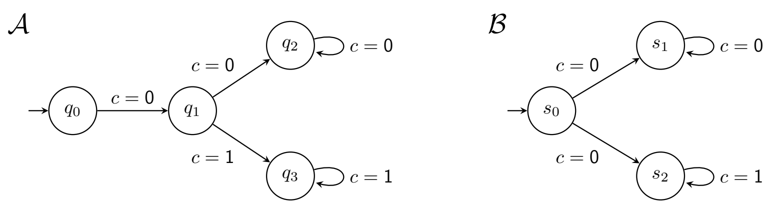


Figure 11: An example where $tf_{\mathcal{A}\mapsto\mathcal{B}}$ is not the optimal transfer function.

computed by Algorithm 2 is the relation $\leq$ from Definition 12. By definition of $\leq$, $q_{\mathcal{B}} \not\leq q'_{\mathcal{B}}$ implies $\mathscr{L}(q_{\mathcal{B}}) \not\subseteq \mathscr{L}(q'_{\mathcal{B}})$, and thus there exists a trace $w \in \mathscr{L}(q_{\mathcal{B}})$ such that $w \notin \mathscr{L}(q'_{\mathcal{B}})$. Since $q'_{\mathcal{B}} \in S(q_{\mathcal{A}})$, then the state $(q_{\mathcal{A}}, q'_{\mathcal{B}})$ is reachable and there exists a trace u such that $(q_0^{\mathcal{A}}, q_0^{\mathcal{B}}) \overset{u}{\Longrightarrow} (q_{\mathcal{A}}, q'_{\mathcal{B}})$. By definition of transfer function, $uw \in \mathscr{L}(\mathcal{B})$. Since $\mathcal{B}$ is deterministic, we have that the unique run of $\mathcal{B}$ on u is such that $q_0^{\mathcal{B}} \overset{u}{\Longrightarrow} q'_{\mathcal{B}}$. From $uw \in \mathscr{L}(\mathcal{B})$ we can conclude that $w \in \mathscr{L}(q'_{\mathcal{B}})$, and a contradiction is found. □

We can be even more precise, by extending the partial order $\leq$ pointwise to functions.

Definition 13. *Given two functions $f, g : Q_{\mathcal{A}} \mapsto Q_{\mathcal{B}}$, we have $f \leq g$ iff for every $q_{\mathcal{A}} \in Q_{\mathcal{A}}$:*

- *either $f(q_{\mathcal{A}}) = \bot$ or*
- *$g(q_{\mathcal{A}}) \neq \bot$ and $f(q_{\mathcal{A}}) \leq g(q_{\mathcal{A}})$.*

Theorem 21. *If $\mathcal{A}$ and $\mathcal{B}$ are two components with $\mathcal{B}$ deterministic, then $tf_{\mathcal{A}\mapsto\mathcal{B}}$ is a maximal transfer function w.r.t. $\leq$.*

Proof. Since $\mathcal{B}$ is deterministic, the relation $\preceq$ coincides with $\leq$. To prove that tf is maximal, suppose towards a contradiction that there exists a transfer function tf' such that $tf < tf'$. Hence, there exists a state $q_{\mathcal{A}} \in Q_{\mathcal{A}}$ such that $tf'(q_{\mathcal{A}}) = q_{\mathcal{B}}$ and $tf(q_{\mathcal{A}}) < q_{\mathcal{B}}$. From the definition of tf, $q_{\mathcal{B}}$ cannot be a greatest lower bound of $S(q_{\mathcal{A}})$ in $Q_{\mathcal{B}}$. As a consequence, there exists $q'_{\mathcal{B}} \in S(q_{\mathcal{A}})$ such that $q_{\mathcal{B}} \not\leq q'_{\mathcal{B}}$. By definition of $\leq$, $\mathscr{L}(q_{\mathcal{B}}) \not\subseteq \mathscr{L}(q'_{\mathcal{B}})$, and thus there exists a trace $w \in \mathscr{L}(q_{\mathcal{B}})$ such that $w \notin \mathscr{L}(q'_{\mathcal{B}})$. Since $q'_{\mathcal{B}} \in S(q_{\mathcal{A}})$, then the state $(q_{\mathcal{A}}, q'_{\mathcal{B}})$ is reachable and there exists a trace u such that $(q_0^{\mathcal{A}}, q_0^{\mathcal{B}}) \overset{u}{\Longrightarrow} (q_{\mathcal{A}}, q'_{\mathcal{B}})$. By definition of transfer function, $uw \in \mathscr{L}(\mathcal{B})$. Since $\mathcal{B}$ is deterministic, we have that the unique run of $\mathcal{B}$ on u is such that $q_0^{\mathcal{B}} \overset{u}{\Longrightarrow} q'_{\mathcal{B}}$. From $uw \in \mathscr{L}(\mathcal{B})$ we can conclude that $w \in \mathscr{L}(q'_{\mathcal{B}})$, and a contradiction is found. □

If $\mathcal{B}$ is not deterministic, then the transfer function tf may not be optimal, as shown by the example below.

Example 11. *Consider the CAs in Figure 11. By building the join $\mathcal{A} \bowtie_s \mathcal{B}$ we obtain that*

$$S(q_0) = \{s_0\} \qquad S(q_1) = \{s_1, s_2\}$$
$$S(q_2) = \{s_1\} \qquad S(q_3) = \{s_2\}$$

Since the relation $\preceq$ on the states of $\mathcal{B}$ is such that $s_1 \preceq s_0$ and s_2 is uncomparable with both s_0 and s_1, the transfer function tf is

$$tf(q_0) = s_0 \qquad tf(q_1) = \perp$$
$$tf(q_2) = s_1 \qquad tf(q_3) = s_2$$

Notice that tf is undefined on q_1. However, any function f that behaves as tf on q_0, q_2 and q_3 and maps q_1 to either s_1 or s_2 is still a transfer function and has a larger domain than tf.

The fact that optimality of tf requires that $\mathcal{B}$ is deterministic is not a problem in our setting, since all the constructions that we have discussed in the paper produce most general updates which are deterministic. Furthermore, if an update $\mathcal{B}$ is not deterministic, one can determinise $\mathcal{B}$ obtaining $\mathcal{B}'$ and then apply the construction to $\mathcal{B}'$. In order to transfer the result back to $\mathcal{B}$, if $tf_{\mathcal{A}\mapsto\mathcal{B}'}(q_{\mathcal{A}}) = q'_{\mathcal{B}}$, one should remember that $q'_{\mathcal{B}}$ corresponds to a set of states of $\mathcal{B}$. Selecting any of these states would produce a transfer function, since their language is a subset of $\mathscr{L}(q'_{\mathcal{B}})$, however the resulting transfer function may not be maximal (but has maximum domain).

Example 12. *Take the update $\mathcal{B}$ in Figure 11. We showed in Example 11 that applying our approach on $\mathcal{B}$ would produce a transfer function that does not have maximal domain. We can then determinise $\mathcal{B}$ obtaining $\mathcal{B}' = \mathsf{Subset}(\mathcal{B})$, represented in Figure 12. Notice that $\mathcal{B}'$ is equal to $\mathcal{A}$ up to renaming of states. Thus, by applying our approach to $\mathcal{B}'$ we obtain*

$$tf(q_0) = \{s_0\} \qquad tf(q_1) = \{s_1, s_2\}$$
$$tf(q_2) = \{s_1\} \qquad tf(q_3) = \{s_2\}$$

We can now transfer the result back to $\mathcal{B}$, obtaining that both functions tf' and tf'' below are transfer functions:

$$tf'(q_0) = s_0 \qquad tf'(q_1) = s_1$$
$$tf'(q_2) = s_1 \qquad tf'(q_3) = s_2$$

$$tf''(q_0) = s_0 \qquad tf''(q_1) = s_2$$
$$tf''(q_2) = s_1 \qquad tf''(q_3) = s_2$$

In all the discussion above, we assumed that $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$. This restriction is satisfied in all the updates we consider, but for the ones enforcing a new property (Theorems 5 and 6). If we drop this restriction, then it may happen

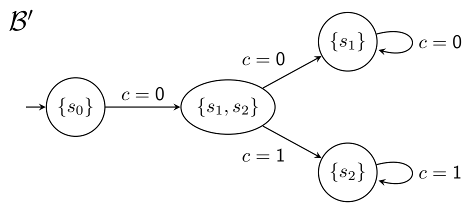


Figure 12: Determinisation of $\mathcal{B}$ from Figure 11.

that the trace performed by $\mathcal{C}[\mathcal{A}]$ before the update does not satisfy Φ. Since our properties are prefix closed, then from that point onward there will be no hope of getting a dynamically correct update. Furthermore, if we restrict the attention only to a specific state $q_{\mathcal{A}}$ of $\mathcal{A}$, it may happen that there exist runs of $\mathcal{C}[\mathcal{A}]$ leading to $q_{\mathcal{A}}$ accepting a word that does not satisfy Φ, and in this case there is always the possibility that the update is not dynamically correct. If instead all the runs of $\mathcal{C}[\mathcal{A}]$ leading to $q_{\mathcal{A}}$ accept words that satisfy Φ, then we can use the approach above to compute $tf(q_{\mathcal{A}})$.

We have discussed above how to build a transfer function. We do not consider in this paper the issue of how to actually apply an update at runtime, and how to correctly perform the state transfer specified by transfer function tf. These issues need to be discussed at a much lower level of abstraction than the one we consider here. An approach to this problem in the concrete context of C programs can be found in [23].

6. Related Work

While many approaches tackle system update [1], the problem of ensuring correctness of a system upon update has received less attention till now. We discuss below relevant classes of related work.

Refinement. Approaches based on behavioural congruences, such as [24], allow one to prove the correctness of updates when a component is replaced by a syntactically different, but semantically equivalent one. Similarly, approaches based on refinement, such as [25] in the context of Reo, allow one either to add new behaviours, while preserving the old ones, or to remove old behaviours without adding new ones (this depends on the direction of refinement). This last case is in our setting an update correct in any context and for any property, which permits only to reduce the allowed behaviours. We also consider different notions of update, selecting the properties to preserve or the allowed contexts, which cannot be matched by the behavioural congruences or refinement approaches.

The works in [26, 27] present refinement and extension processes for state machines, where formally-validated refinement patterns are applied to guarantee property preservation. While their semantic model is close to ours, their approach is not limited to safety properties. On the other side, their approach is not fully automatic: refinement and extension patterns must be provided and formally validated by the user before the update can be applied.

We remark that studying refinement for components in isolation, as in [28, 29], is not enough to ensure that the updated system refines the original one: one needs to check that refinement is a congruence.

Approaches focusing on specific properties. Some approaches, such as [30, 31], focus exclusively on type safety, that rules out obviously wrong behaviours, but it is insufficient for establishing that given properties are preserved.

A line of work [32, 33, 34, 35] 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 progress. Another related approach is presented in [36], where behavioural types are used to ensure that running sessions are not interrupted, and that provided services are preserved. Our approach is more flexible than the ones above since it considers any property expressible as a CA, while the approaches above deal with a few fixed properties such as type safety and deadlock freedom, but do not ensure any form of preservation of behaviour. We remark however that, since deadlock freedom is not a safety property, we cannot deal with it in our setting.

Model checking. In [37], a modular model checking approach to verify adaptive programs is proposed. Requirements are formalised using an extension of LTL. 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.

Model checking is also used in [38] in the context of Reo connectors. Both reconfiguration and computation are mapped into graph transformations, and properties formalised in CTL are checked using the GROOVE model checker. The focus here is more on structural properties of the Reo connector and on the interplay between different reconfiguration rules, while we concentrate on the behaviour of the system upon update. In [39] graph transformations are used also to study the state transfer problem in the context of Reo connectors. The focus here is to compute the resulting state after reconfiguration thanks to a compositional semantic model, yet there are no results on how to ensure such a resulting state is correct, apart for requiring it to be reachable from the initial state and not a deadlock state.

We remark that the update resulting from one of their atomic reconfigurations can be mimicked using our notion of asynchronous composition.

Invariant preservation. The work in [40] categorises different kinds of reconfigurations in the context of Reo connectors. The updates that are correct for any property (in a given context) are called contractive in [40], and a property for which an update is correct (in a given context) is called an invariant for the update. However, in [40], 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 [41] considers a problem similar to ours in the concrete setting of aspect-oriented programming. They use a dedicated semantic framework (Common Aspect Semantic Base) and look for the classes of properties preserved in any context by relevant classes of updates (observers, aborters, . . .). Properties are expressed in LTL and CTL*, but, differently from us, they consider atomic predicates on both states and events, hence the two approaches are not directly comparable.

Dynamic update. The problem of dynamic update, and the related state transfer problem, has been first analysed in [6]. Here they assume no knowledge about the expected behaviour of the program, and they just require that the system after the update reaches, possibly after some spurious steps, a state which would have been reachable by executing the updated system since the very beginning. Our focus is different, since we do not consider reachability of states, but properties satisfied by the observations performed on the interface with the external world. However, the idea of allowing some spurious transitions upon update before the updated system starts behaving as expected could be applied also to generalise our setting, and we plan to consider it in future work.

In [4], a program transformation combines a program and an update (a patch in their terminology) into a new program presenting all the behaviours corresponding to applying the update at any allowed point. This allows one to apply off-the-shelf analysers to check properties of the system after the update. They consider a single program at the time, as in our "given context" cases, and concentrate on backward-compatible specifications, which correspond to our updates correct for a given property, and on post-update specifications, which correspond to properties that the update enforces (as we do in Theorem 5). They also mention updates which are observationally equivalent, corresponding to our "all properties, given context" case. The approach is at the concrete level of the operational semantics of an imperative language, and underlies a tool working on C programs. The main difference w.r.t. our approach is that we build updates satisfying desired properties, while they check properties on updates provided by the user.

In [42], a framework for dynamic update in the coordination modelling language Paradigm is presented. The system starts from an initial architecture and then migrates towards a new architecture through a number of adaptation steps that are executed on-the-fly. A model checker is used to verify the correctness of the migration with respect to μ-calculus properties. The initial architecture may not respect all desired properties, hence correctness of the update involves establishing that the final architecture is eventually reached and that it respects the desired properties. Thus, their approach is close to our update that enforces new properties, yet they only verify the update and do not synthesise a most general one as we do.

Synthesis. The works in [18, 19] are related to ours from the technical point of view. In particular, [18] provides us the general framework to solve Inequation (1), while [19] gives us the properties that a composition operator should respect to fit in the general framework. However, they do not provide a construction for building an actual automaton in our case, namely, for CAs with both finite and infinite traces. Also, they have a different aim, since they do not consider update at all. They highlight, 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 [43, 44]. Even though they do not consider update, 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., [45]). 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.

7. Conclusion

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 characterised 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. In order to support dynamic update, we also studied how to map the state of the original component $\mathcal{A}$ into the state of the new component $\mathcal{B}$ so that each execution where $\mathcal{A}$ is replaced by $\mathcal{B}$ (initialised according to the state of $\mathcal{A}$) without restarting the context satisfies the desired properties. We studied the problems above in the setting of constraint automata.

The same problems can also be studied in other settings, and indeed we plan to consider some of them in future work. For instance one may consider more complex properties, as hinted at above, or more complex automata, like timed automata or general CAs where the set of data values can be infinite. We can also consider other forms of composition, e.g. via bounded or unbounded buffers or lossy channels, and other correctness criteria for updates, such as simulation or testing pre-orders. Finally, we want to apply our technique to more concrete models, starting from the ones based on CAs, such as REO [11] and Rebeca [12].

References

[1] H. Seifzadeh, H. Abolhassani, M. S. Moshkenani, A survey of dynamic software updating, J. of Software: Evolution and Process 25 (5) (2013) 535–568.

[2] L. A. F. Leite, G. A. Oliva, G. M. Nogueira, M. A. Gerosa, F. Kon, D. S. Milojicic, A systematic literature review of service choreography adaptation, Service Oriented Computing and Applications 7 (3) (2013) 199–216.

[3] M. C. Huebscher, J. A. McCann, A survey of autonomic computing - degrees, models, and applications, ACM Comput. Surv. 40 (3) (2008) 7:1–7:28.

[4] C. M. Hayden, S. Magill, M. Hicks, N. Foster, J. S. Foster, Specifying and verifying the correctness of dynamic software updates, in: VSTTE, Vol. 7152 of LNCS, Springer, 2012, pp. 278–293.

[5] S. Hansell, Glitch makes teller machines take twice what they give, The New York Times, `http://www.nytimes.com/1994/02/18/business/glitch-makes-teller-machines-take-twice-what-they-give.html`.

[6] D. Gupta, P. Jalote, G. Barua, A formal framework for on-line software version change, IEEE Trans. Software Eng. 22 (2) (1996) 120–131.

[7] A. Baumann, G. Heiser, J. Appavoo, D. D. Silva, O. Krieger, R. W. Wisniewski, J. Kerr, Providing dynamic update in an operating system, in: USENIX Annual Technical Conference, USENIX, 2005, pp. 279–291.

[8] H. Chen, R. Chen, F. Zhang, B. Zang, P. Yew, Live updating operating systems using virtualization, in: VEE, ACM, 2006, pp. 35–44.

[9] C. Baier, T. Blechmann, J. Klein, S. Klüppelholz, Formal verification for components and connectors, in: FMCO, Vol. 5751 of LNCS, Springer, 2008, pp. 82–101.

[10] C. Baier, M. Sirjani, F. Arbab, J. Rutten, Modeling component connectors in Reo by Constraint Automata, Sci. Comput. Program. 61 (2) (2006) 75–113.

[11] F. Arbab, Reo: a channel-based coordination model for component composition, Mathematical Structures in Computer Science 14 (3) (2004) 329–366.

[12] M. Sirjani, M. M. Jaghoori, C. Baier, F. Arbab, Compositional semantics of an actor-based language using constraint automata, in: COORDINATION, Vol. 4038 of LNCS, Springer, 2006, pp. 281–297.

[13] M. Tsai, Y. Tsay, Y. Hwang, GOAL for games, omega-automata, and logics, in: CAV, Vol. 8044 of LNCS, Springer, 2013, pp. 883–889.

[14] D. Bresolin, I. Lanese, Most general property-preserving updates, in: LATA, Vol. 10168 of LNCS, Springer, 2017, pp. 367–379.

[15] M. Lange, Weak automata for the linear time μ-calculus, in: VMCAI, Vol. 3385 of LNCS, Springer, 2005, pp. 267–281.

[16] N. Kobayashi, C. L. Ong, Complexity of model checking recursion schemes for fragments of the modal mu-calculus, Logical Methods in Computer Science 7 (4) (2011) 9:1 – 9:24.

[17] E. A. Emerson, Temporal and modal logic, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), MIT Press, 1990, pp. 995–1072.

[18] T. Villa, N. Yevtushenko, R. K. Brayton, A. Mishchenko, A. Petrenko, A. Sangiovanni-Vincentelli, The Unknown Component Problem - Theory and Application, Springer, 2012.

[19] T. Villa, A. Petrenko, N. Yevtushenko, A. Mishchenko, R. K. Brayton, Component-based design by solving language equations, Proceedings of the IEEE 103 (11) (2015) 2152–2167.

[20] K. Chatterjee, L. Doyen, Games with a weak adversary, in: ICALP, Vol. 8573 of LNCS, Springer, 2014, pp. 110–121.

[21] M. Chrobak, Finite automata and unary languages, Theor. Comput. Sci. 47 (3) (1986) 149–158.

[22] C. Baier, J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.

[23] C. Giuffrida, C. Iorgulescu, A. Kuijsten, A. S. Tanenbaum, Back to the future: Fault-tolerant live update with time-traveling state transfer, in: LISA, USENIX Association, 2013, pp. 89–104.

[24] F. Bonchi, A. Brogi, S. Corfini, F. Gadducci, A behavioural congruence for web services, in: FSEN, Vol. 4767 of LNCS, Springer, 2007, pp. 240–256.

[25] C. Krause, H. Giese, E. P. de Vink, Compositional and behavior-preserving reconfiguration of component connectors in Reo, J. Vis. Lang. Comput. 24 (3) (2013) 153–168.

[26] L. Harbird, A. Galloway, R. F. Paige, Towards a model-based refinement process for contractual state machines, in: ISORCW, IEEE, 2010, pp. 108–115.

[27] C. Prehofer, Property preservation for extension patterns of state transition diagrams, in: Integrated Formal Methods, Vol. 9681 of LNCS, Springer, 2016, pp. 260–274.

[28] M. Schrefl, M. Stumptner, Behavior-consistent specialization of object life cycles, ACM Trans. Softw. Eng. Methodol. 11 (1) (2002) 92–148.

[29] S. Schneider, H. Treharne, H. Wehrheim, The behavioural semantics of Event-B refinement, Formal Asp. Comput. 26 (2) (2014) 251–280.

[30] D. Duggan, Type-based hot swapping of running modules, Acta Inf. 41 (4-5) (2005) 181–220.

[31] G. P. Stoyle, M. W. Hicks, G. M. Bierman, P. Sewell, I. Neamtiu, Mutatis mutandis: safe and predictable dynamic software updating, in: POPL, ACM, 2005, pp. 183–194.

[32] G. Anderson, J. Rathke, Dynamic software update for message passing programs, in: APLAS, Vol. 7705 of LNCS, Springer, 2012, pp. 207–222.

[33] M. Dalla Preda, M. Gabbrielli, S. Giallorenzo, I. Lanese, J. Mauro, Dynamic choreographies: Theory and implementation, Logical Methods in Computer Science 13 (2).

[34] M. Dalla Preda, S. Giallorenzo, I. Lanese, J. Mauro, M. Gabbrielli, AIOCJ: A choreographic framework for safe adaptive distributed applications, in: SLE, Vol. 8706 of LNCS, Springer, 2014, pp. 161–170.

[35] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, Self-adaptive multiparty sessions, Service Oriented Computing and Applications 9 (3-4) (2015) 249–268.

[36] C. Di Giusto, J. A. Pérez, Disciplined structured communications with consistent runtime adaptation, in: SAC, ACM, 2013, pp. 1913–1918.

[37] J. Zhang, H. Goldsby, B. H. C. Cheng, Modular verification of dynamically adaptive systems, in: AOSD, ACM, 2009, pp. 161–172.

[38] C. Krause, Z. Maraikar, A. Lazovik, F. Arbab, Modeling dynamic reconfigurations in Reo using high-level replacement systems, Sci. Comput. Program. 76 (1) (2011) 23–36.

[39] C. Krause, Distributed port automata, ECEASST 41.

[40] N. Oliveira, L. S. Barbosa, On the reconfiguration of software connectors, in: SAC, ACM, 2013, pp. 1885–1892.

[41] S. Djoko-Djoko, R. Douence, P. Fradet, Aspects preserving properties, in: PEPM, ACM, 2008, pp. 135–145.

[42] S. Andova, L. Groenewegen, E. de Vink, Dynamic adaptation with distributed control in Paradigm, Science of Computer Programming 94 (2014) 333 – 361.

[43] R. Alur, S. La Torre, Deterministic generators and games for LTL fragments, ACM Trans. Comput. Logic 5 (1) (2004) 1–25.

[44] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive(1) designs, J. of Computer and System Sciences 78 (3) (2012) 911–938.

[45] A. Aziz, F. Balarin, R. K. Brayton, M. D. DiBenedetto, A. Saldanha, Supervisory control of finite state machines, in: CAV, Vol. 939 of LNCS, Springer, 1995, pp. 279–292.

Appendix A. Proof of Theorem 1, asynchronous case

To apply the theory in [19] to our asynchronous embedding, we show below that its language $\mathscr{L}(\mathcal{C}[\mathcal{A}]_a)$ can be rewritten as

$$\mathscr{L}(\mathcal{C}[\mathcal{A}]_a) = \big(\mathscr{L}(\mathcal{A})_{\top Y} \cap \mathscr{L}(\mathcal{C})_{\top X}\big)_{\perp X \circ Y} \tag{A.1}$$

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 nodes 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 $_{\perp N}$ for $_{\perp \mathsf{CIO}(N)}$. 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 $\mathscr{L}_{\perp N_2} = \{r(w) \mid w \in \mathscr{L}\}$. 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 8. *Given a context $\mathcal{C}$ with nodes $O \cup U$ and a component $\mathcal{A}$ with nodes U, we have that $\mathscr{L}(\mathcal{C}[\mathcal{A}]_a) = \big(\mathscr{L}(\mathcal{A})_{\top O} \cap \mathscr{L}(\mathcal{C})_{\top \emptyset}\big)_{\perp O}$.*

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 $_{\perp O}$ 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 $\rho = (q_0, p_0) \xrightarrow{w_0} (q_1, p_1) \xrightarrow{w_1} \ldots$ be a run accepting w. Since in our setting the set of nodes U of the component $\mathcal{A}$ is a subset of the set of nodes $O \cup U$ of the context $\mathcal{C}$, we have that the component can advance only when it can communicate to $\mathcal{C}$. Hence, 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 $w \in \mathscr{L}(\mathcal{C})$. Moreover, 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 $\rho_{\mathcal{C}}$ 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 $\rho_{\mathcal{A}}$ of $\mathcal{A}$ accepting $r(w)$. By synchronising $\rho_{\mathcal{C}}$ and $\rho_{\mathcal{A}}$ we can build a run of $\mathcal{A} \bowtie_a \mathcal{C}$ accepting w as desired[8]. □

Lemma 9. *The language restriction and language expansion operators, $\perp$ and $\top$, satisfy the properties H1, H2, and H3.*

Proof.

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

H2 We assume that $\mathscr{L}_1 = (\mathscr{L}_{1\perp X})_{\top Y}$ (the other case is symmetric). Assume $w \in (\mathscr{L}_1 \cap \mathscr{L}_2)_{\perp X}$. 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 $w \in \mathscr{L}_{1\perp X}$ and $w \in \mathscr{L}_{2\perp X}$. Hence, $w \in (\mathscr{L}_{1\perp X}) \cap (\mathscr{L}_{2\perp X})$.

Assume now that $w \in (\mathscr{L}_{1\perp X}) \cap (\mathscr{L}_{2\perp X})$. Then $w \in \mathscr{L}_{1\perp X}$ and $w \in \mathscr{L}_{2\perp X}$. From the latter we know that there exists $w_2 \in \mathscr{L}_2$ such that $r(w_2) = w$. 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.

□

Appendix B. Proof of Theorem 8

We first recall some key notions on three-player games, taken from [20].

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 $q' = \delta(q, a_1, a_2, a_3)$.

[8]Note that if we allow as CIO the constant function with value $\perp$ then it is not guaranteed that we can synchronise $\rho_{\mathcal{C}}$ and $\rho_{\mathcal{A}}$.

Observations. For $i = 1, 2, 3$, a set $O_i \subseteq 2^Q$ of *observations* (for player i) is a partition of Q. Let $\text{obs}_i : Q \mapsto O_i$ be the function that assigns to each state $q \in Q$ the (unique) observation for player i that contains q, i.e., such that $q \in \text{obs}_i(q)$. The functions obs_i are extended to sequences $\rho = q_0 \dots q_n$ of states in a pointwise 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 as 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 prove the 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 [20]. 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 moves 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 moves 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 $\mathcal{C}_\mathcal{G} = \langle \{r_0, r_1\}, \{act_1, fwd_1, act_2, obs_2\}, r_0, \rightarrow_\mathcal{C} \rangle$ such that:

- $r_0 \xrightarrow{act_1=a_1;fwd_1=a_1;act_2=a_2}_\mathcal{C} r_1$ for every $a_1 \in A_1$ and $a_2 \in A_2$;
- $r_1 \xrightarrow{act_1=*;obs_2=o_2}_\mathcal{C} r_0$ for some "dummy move" $* \notin A_1$ and every $o_2 \in O_2$.

Notice that every transition of $\mathcal{C}_\mathcal{G}$ communicates with both the component and the environment. This forces the context and the component to progress in lock-step also under asynchronous composition, and thus we have that $\mathcal{C}_\mathcal{G}[\mathcal{B}]_s = \mathcal{C}_\mathcal{G}[\mathcal{B}]_a$ for every possible component $\mathcal{B}$.

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 $\mathcal{S}_\mathcal{G} = \langle T \cup T' \cup \{\mathit{WIN}\}, \{\mathit{fwd}_1, \mathit{act}_2, \mathit{obs}_2\}, q_0, \rightarrow_\mathcal{S}\rangle$ where:

- T is the set of safe states of $\mathcal{G}$ and $T' = \{q' \mid q \in T\}$;
- $q_i \xrightarrow{\mathit{fwd}_1=a_1;\mathit{act}_2=a_2}_\mathcal{S} q'_j$ iff there exists $a_3 \in A_3$ such that $\delta(q_i, a_1, a_2, a_3) = q_j$;
- $q'_j \xrightarrow{\mathit{obs}_2=o_2}_\mathcal{S} q_j$ iff $O_2(q_j) = o_2$;
- $q'_j \xrightarrow{\mathit{obs}_2=o_2}_\mathcal{S} \mathit{WIN}$ iff $O_2(q_j) \neq o_2$;
- $\mathit{WIN} \xrightarrow{c}_\mathcal{S} \mathit{WIN}$ for every $c \in \mathsf{CIO}(O)$.

Lemma 10. *Let $\mathcal{B}$ be a CA that is a most general solution of $\mathscr{L}(\mathcal{C}_\mathcal{G}[\mathcal{B}]) \subseteq \mathscr{L}(\mathcal{S}_\mathcal{G})$. Then Player 1 has a winning strategy $w = u_0u_1\ldots$ 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 $w = u_0u_1\ldots$ 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 $r_0 \xrightarrow{\mathit{act}_1=u_0;\mathit{fwd}_1=u_0;\mathit{act}_2=v_0}_\mathcal{C} r_1 \xrightarrow{\mathit{act}_1=*;\mathit{obs}_2=o_1}_\mathcal{C} r_0 \ldots$ of the context synchronising 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, $q_i \xrightarrow{\mathit{fwd}_1=u_i;\mathit{act}_2=v_i}_\mathcal{S} q'_{i+1}$ is a transition of $\mathcal{S}_\mathcal{G}$. Consider now the next transition $r_1 \xrightarrow{\mathit{act}_1=*;\mathit{obs}_2=o_i}_\mathcal{C} r_0$ of the context. If $O_2(q_{i+1}) \neq o_i$ then the specification goes to WIN and the trace generated by $\mathcal{C}_\mathcal{G}[\mathcal{Y}]$ belongs to $\mathscr{L}(\mathcal{S}_\mathcal{G})$. Otherwise, $q'_{i+1} \xrightarrow{\mathit{obs}_2=o_i}_\mathcal{S} q_{i+1}$ 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 $\mathcal{C}_\mathcal{G}[\mathcal{B}]$ as follows. At step $i \geq 0$, $\mathcal{B}$ performs the actions $\mathit{act}_1 = u_i$ followed by $\mathit{act}_1 = *$. The context performs the action $\mathit{act}_1 = u_i; \mathit{fwd}_1 = u_i; \mathit{act}_2 = v_i$, such that $\sigma_2(o_0 \ldots o_{i-1}) = v_i$. Since $\mathcal{B}$ is a solution of Inequation (1), we can find a matching transition $q_i \xrightarrow{\mathit{fwd}_1=u_i;\mathit{act}_2=v_i}_\mathcal{S} q'_{i+1}$ of $\mathcal{S}_\mathcal{G}$. For the second transition of the context we can choose the correct observation $r_1 \xrightarrow{\mathit{act}_1=*;\mathit{obs}_2=o_i}_\mathcal{C} r_0$ with $o_i = O_2(q_{i+1})$. This last step is matched by the specification with a transition $q'_{i+1} \xrightarrow{\mathit{obs}_2=o_i}_\mathcal{S} q_{i+1}$ where $q_{i+1} \neq \mathit{WIN}$. By definition of $\mathcal{S}_\mathcal{G}$, for every transition $q_i \xrightarrow{\mathit{fwd}_1=u_i;\mathit{act}_2=v_i}_\mathcal{S} q_{i+1}$ 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 10 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 (in isolation), one takes a CA $\mathcal{A}$ generating every possible behaviour (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 iff $\mathcal{B}$ 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.