A survey on compositional algorithms for verification and synthesis in supervisory control

Downloaded from: https://research.chalmers.se, 2026-04-06 02:38 UTC

Citation for the original published paper (version of record):
Malik, R., Mohajerani, S., Fabian, M. (2023). A survey on compositional algorithms for verification and synthesis in supervisory control. Discrete Event Dynamic Systems: Theory and Applications, 33(3): 279-340. http://dx.doi.org/10.1007/s10626-023-00378-8

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Discrete Event Dynamic Systems
https://doi.org/10.1007/s10626-023-00378-8

A survey on compositional algorithms for verification and synthesis in supervisory control

Robi Malik[1] · **Sahar Mohajerani**[2] · **Martin Fabian**[2]

Received: 16 March 2022 / Accepted: 3 May 2023
© The Author(s) 2023

Abstract
This survey gives an overview of the current research on compositional algorithms for verification and synthesis of modular systems modelled as interacting finite-state machines. Compositional algorithms operate by repeatedly simplifying individual components of a large system, replacing them by smaller so-called *abstractions*, while preserving critical properties. In this way, the exponential growth of the state space can be limited, making it possible to analyse much bigger state spaces than possible by standard state space exploration. This paper gives an introduction to the principles underlying compositional methods, followed by a survey of algorithmic solutions from the recent literature that use compositional methods to analyse systems automatically. The focus is on applications in supervisory control of discrete event systems, particularly on methods that verify critical properties or synthesise controllable and nonblocking supervisors.

Keywords Supervisory control theory · Compositional verification · Finite-state machines · Discrete event systems

1 Introduction

Discrete event systems are a useful modelling paradigm for developing controllers of safety-critical applications in industrial automation, automotive electronics, avionics, etc. Such systems are typically composed of several components and subsystems that interact with each other in a high degree of concurrency, which gives rise to a complexity that is difficult to handle manually; and this complexity tends to increase with each added new component.

✉ Robi Malik
robi@waikato.ac.nz

Sahar Mohajerani
mohejera@chalmers.se

Martin Fabian
fabian@chalmers.se

1 University of Waikato, Hamilton, New Zealand

2 Chalmers University of Technology, Gothenburg, Sweden

Published online: 21 August 2023

At the same time, the safety-criticality of the systems requires safe and correct functionality, as malfunctions can have disastrous effects.

To support the development of correct control systems, *supervisory control theory* (Ramadge and Wonham 1989; Cassandras and Lafortune 2008) provides a general framework for constructing reactive control functions for discrete event systems. Given a model of the *plant* to be controlled and a *specification* of the desired behaviour, methods are provided to design or to compute a *supervisor* that dynamically restricts the plant behaviour while ensuring that the specification is satisfied. The plant and specification are in general given as several interacting finite-state machines (Balemi 1992; Krook et al. 2018; Reijnen et al. 2020).

The algorithms used by this approach, like most other algorithms to check the correctness of finite-state systems, face a computational challenge known as *state-space explosion*. To confirm correctness, it is in principle necessary to explore the complete system state space, which grows exponentially with the number of components (Bérard et al. 2001), and this quickly becomes intractable for many real-world applications.

Compositional reasoning can mitigate the state-space explosion by taking advantage of the modular structure (Graf and Steffen 1990; Clarke et al. 1994). The idea is to perform local reasoning about individual system components, simplify them as much as possible, and only put them in a larger context when necessary. Local reasoning facilitates reuse of analysis results: if a part of a system is modified, all results from analysing the unmodified parts remain valid. It is also possible to reason about a system that is not yet fully defined (Andersen et al. 1994). In addition, there are computational benefits as the simplification of components often reduces their number of states.

Compositional reasoning is well suited for supervisory control applications if the system is modelled by a large number of state machines that are loosely coupled through synchronisation. *Modular* (Wonham and Ramadge 1988), *hierarchical* (Zhong and Wonham 1990), *concurrent* (Willner and Heymann 1991), and *decentralised* (Rudie and Wonham 1992) supervisory control exploit the system structure in different ways and incorporate ideas of compositional reasoning to facilitate supervisor design. More recently, fully automatic algorithms based on compositional reasoning have been proposed (Flordal and Malik 2009; Hill et al. 2010; Su et al. 2010a; Mohajerani et al. 2014), which can solve problems of industrial scale and handle much larger state spaces than previously possible.

This paper surveys the current state-of-the-art in compositional verification and synthesis for discrete event systems, focusing on algorithmic solutions that take a model of synchronised finite-state machines as input and compute an answer automatically with minimal user interaction. The two tasks of verification and synthesis are considered separately.

– The goal of *verification* is to determine whether a system satisfies a property of interest. Compositional verification typically follows an approach proposed by Graf and Steffen (1990), where system components are simplified separately before they are composed with other components. Compositional verification methods differ in how the components are simplified, which is dependent on the type of properties being verified. To verify arbitrary temporal logic properties, simplification can only be done using *bisimulation* (Baier and Katoen 2008). If only safety properties are being verified, simplification can be done using *weak bisimulation* (Milner 1989) or *language equivalence* (Ware and Malik 2008). Other properties of interest are the absence of deadlock, which is done using *failures equivalence* as simplification (Roscoe et al. 1995), and the nonblocking property, which is done using weak bisimulation (Su et al. 2010a) or *conflict equivalence*

(Malik et al. 2006; Flordal and Malik 2009). These properties and their compositional verification are explored in Section 4.
- The goal of *synthesis* is to compute the control logic needed to ensure that a system satisfies certain properties. This is a more difficult problem than verification as it cannot be separated into different classes of properties—usually several properties have to be enforced simultaneously and must be considered together. This paper focuses on the standard synthesis objective in supervisory control, namely the synthesis of *controllable and nonblocking supervisors* (Ramadge and Wonham 1989). Compositional synthesis algorithms can be based on a similar framework as that of Graf and Steffen (1990) where components are simplified and composed gradually, although additional considerations are necessary. The main difference between compositional synthesis approaches are which components are composed and how they are simplified. *Local synthesis* identifies unsafe states within subsystems and removes them before composing the subsystems with the rest of the system (Hill and Tilbury 2008; Flordal et al. 2007). *Projection* removes events from components and replaces them by *deterministic abstractions* (Feng and Wonham 2008; Schmidt and Breindl 2008). Other methods use *nondeterministic abstractions*, which can be computed by considering removed events as unobservable (Hill et al. 2010; Su et al. 2010b) or using variations of weak bisimulation (Mohajerani et al. 2014, 2017). Some of these methods ensure that the computed supervisors are *maximally permissive* while others do not. These methods and their differences are surveyed in Section 5.

The remainder of this paper is organised as follows. Section 2 introduces the principles of compositional reasoning and describes a general framework which is used as reference throughout the paper. This section is intended to be generally accessible and uses only a low level of formal notation. Section 3 brings in background on finite-state machines and temporal logic. Then Section 4 surveys compositional verification algorithms with subsections considering different classes of properties to be verified, and Section 5 surveys compositional synthesis algorithms with subsections for different types of abstraction. Finally, Section 6 adds concluding remarks.

2 The idea of compositional methods

2.1 Verification and synthesis

Many discrete event systems are modelled as finite-state machines (FSM). Research in supervisory control has led to *modular* (Wonham and Ramadge 1988) and *decentralised* (Rudie and Wonham 1992) strategies, where control functions are distributed over several components. In *concurrent* supervisory control, not only the control but also the system to be controlled consists of several components (Willner and Heymann 1991). Using the *composition* operator $\|$ to express interaction between components, a concurrent system can be described modularly as

$$G = G_1 \| G_2 \| \cdots \| G_n \ , \tag{2.1}$$

where G is the system model, and $G_1, G_2, \ldots, G_n$ are its components.

In supervisory control, the most common tasks performed on a system model are verification and synthesis. The task of *verification*, or *model checking*, is to determine whether a system such as Eq. 2.1 satisfies a *property* of interest, φ. This can be written as

$$G_1 \| G_2 \| \cdots \| G_n \models \varphi \ . \tag{2.2}$$

The property φ can be a universal property such as controllability or being nonblocking (Ramadge and Wonham 1989), or an application-specific property in the form of a permissible language or a temporal logic expression (Pnueli 1977). The result of verification is an answer of "yes" or "no", indicating whether or not the system G satisfies the property φ.

If verification gives a negative answer, most model checking algorithms also provide a *counterexample* that explains why the system fails the property that was checked. The user then examines the counterexample and adapts the model to remove it, and verifies again. This cycle is repeated until no more counterexamples are found, at which point the model is considered correct. Ramadge (1983) proposes *synthesis* as a way to automate this cycle. Given a system (2.1) and its desired property φ, synthesis computes an additional component, the so-called *supervisor* that applies control to the system and restricts it so that the property is ensured. This can be written as

$$\mathrm{synth}(G_1 \| G_2 \| \cdots \| G_n) = S\ . \tag{2.3}$$

Here, S is the computed supervisor, which is an FSM that can be composed with the system (2.1) to constrain its behaviour so that

$$G_1 \| G_2 \| \cdots \| G_n \| S \models \varphi\ . \tag{2.4}$$

The supervisor may also be represented by several FSMs, the composition of which make up the global supervisor. Other supervisor representations are also possible, e.g., Ramadge and Wonham (1989) consider a supervisor as a map that defines control decisions to enable or disable events after observing traces of the system behaviour.

While the concurrent design approach facilitates the construction of complex systems by composition of small components, there are algorithmic problems with verification and synthesis as the number of components increases. Although the model may consist of separate components, both verification and synthesis depend on the composed system. The straightforward method to perform these tasks is to first construct G according to Eq. 2.1 and then check whether $G \models \varphi$ or synthesise $\mathrm{synth}(G) = S$. This approach, referred to as *monolithic* verification or synthesis, has been implemented in various forms (Feng and Wonham 2006; Zhang and Wonham 2002; Åkesson et al. 2006).

The simplest monolithic verification and synthesis algorithms perform an *explicit* state enumeration, where every state of the composition G in Eq. 2.1 is constructed and held in memory individually. Monolithic algorithms have been reported to solve verification problems with 100 million states on standard PCs (Malik 2016), but industrially interesting system require much larger models.

The manageable number of states can be increased significantly using a *symbolic* representation of the state space, where smart data structures are used that can store sets of states without the need to list each element explicitly. The most common data structure for this purpose is the *binary decision diagram* (*BDD*) (Akers 1978; Bryant 1986). *Symbolic model checking* with BDDs has been reported to solve verification problems with 10^{20} states and beyond (Burch et al. 1992).

Unfortunately, all monolithic algorithms suffer from state-space explosion as composition can result in the multiplication of the number of states. If two FSMs G_1 and G_2 have N_1 and N_2 states, respectively, their composition $G_1 \| G_2$ can have up to $N_1 \cdot N_2$ states. If the n components in Eq. 2.1 have N states each, then their composition G can have up to N^n

states. The complexity is exponential in the number of components, and even the seemingly large number of 10^{20} states can be reached with only a small number of components.

The compositional algorithms surveyed in this paper mitigate this exponential complexity by simplifying the FSM model before passing it to monolithic analysis using explicit state enumeration or BDDs. After simplification, the input to monolithic analysis is smaller and the effective size of models that can be analysed increases beyond what can be achieved with BDDs alone (Flordal and Malik 2009; Mohajerani et al. 2014).

2.2 Compositional abstraction process

Instead of constructing and analysing the full composition G of all system components (2.1), the idea of compositional methods is to construct and analyse a smaller *abstraction* $\tilde{G}$ instead (Clarke et al. 1994). This abstraction $\tilde{G}$ is constructed in such a way that it is *equivalent* to the system G, which may be written as

$$\tilde{G} \simeq G \, . \tag{2.5}$$

The symbol $\simeq$ denotes an abstract equivalence relation, which can take different forms. One of the conditions that $\simeq$ needs to satisfy is that the result of verifying or synthesising from the abstraction $\tilde{G}$ can be used to make conclusions about the original system G. In the case of verification, this means that the abstraction $\tilde{G}$ satisfies the property φ of interest precisely when the system G does,

$$\tilde{G} \models \varphi \quad \text{if and only if} \quad G \models \varphi \, . \tag{2.6}$$

Typically, the abstraction $\tilde{G}$ is simpler than the original system G and can be analysed more easily by a monolithic verification or synthesis algorithm. For this approach to be useful in practice, the abstraction needs to be constructed without first constructing the full composition (2.1) of G.

To this end, Graf and Steffen (1990) propose to construct the abstraction $\tilde{G}$ gradually by transforming the original system (2.1) in several steps, repeatedly simplifying individual components and subsystems. This process is visualised in Fig. 1. Starting with the original system G,

$$G_1 \| G_2 \| G_3 \| G_4 \| G_5 \| \cdots \| G_n \, , \tag{2.7}$$

the first step may be to simplify the first component G_1 and replace it by a smaller abstraction $\tilde{G}_1$, which is related to G_1 through the equivalence (2.5), i.e., $\tilde{G}_1 \simeq G_1$. The result of this replacement is

$$\tilde{G}_1 \| G_2 \| G_3 \| G_4 \| G_5 \| \cdots \| G_n \, . \tag{2.8}$$

The remaining components G_i may be simplified likewise and replaced by equivalent components $\tilde{G}_i \simeq G_i$ in $n-1$ steps, producing

$$\tilde{G}_1 \| \tilde{G}_2 \| \tilde{G}_3 \| \tilde{G}_4 \| \tilde{G}_5 \| \cdots \| \tilde{G}_n \, . \tag{2.9}$$

Once all components have been simplified individually, the next step is to select two or more components and replace them by their composition. For example, replacing $\tilde{G}_1$ and $\tilde{G}_2$ by $G_{12} = \tilde{G}_1 \| \tilde{G}_2$ results in

$$G_{12} \| \tilde{G}_3 \| \tilde{G}_4 \| \tilde{G}_5 \| \cdots \| \tilde{G}_n \, . \tag{2.10}$$

Next, G_{12} may be simplified and replaced by $\tilde{G}_{12} \simeq G_{12}$, resulting in

$$\tilde{G}_{12} \| \tilde{G}_3 \| \tilde{G}_4 \| \tilde{G}_5 \| \cdots \| \tilde{G}_n \, . \tag{2.11}$$

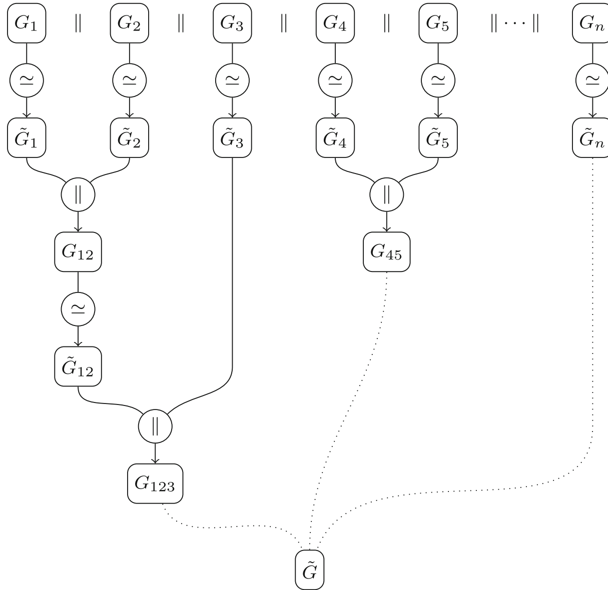


Fig. 1 Compositional abstraction process for $G_1\|\cdots\|G_n$

The next step may be to compose $\tilde{G}_{12}$ and $\tilde{G}_3$ into $G_{123} = \tilde{G}_{12}\|\tilde{G}_3$ or to compose $\tilde{G}_4$ and $\tilde{G}_5$ into $G_{45} = \tilde{G}_4\|\tilde{G}_5$, for example. The process continues until (2.7) is transformed into a single component

$$\tilde{G}, \tag{2.12}$$

called the *compositional abstraction* of the original system G. The mathematical properties of the underlying equivalence $\simeq$ ensure that the compositional abstraction $\tilde{G}$ is equivalent to the original system G, i.e., $\tilde{G} \simeq G$, so that verification or synthesis produces the same result that would be obtained from G.

By working with individual components and subsystems, the compositional abstraction is computed without ever constructing the full composition G of the system. Considering the way how the numbers of states are multiplied after composition, even a small reduction in state numbers at early stages may result in a substantial reduction of the number of states of the final compositional abstraction and the effort to analyse it afterwards.

2.3 Local events and hiding

Discrete event systems are typically synchronised based on shared *events*. Each component G_i is an FSM with its own *event set* or *alphabet* Σ_i. If the alphabets of two or more components share events, the shared events are executed in *lock-step* (Hoare 1985) when these components are synchronised. An event can only be executed by the system if all components that have the event in their alphabet are in a state where the event is enabled. If an event σ appears in the alphabet Σ_i of some component G_i, but G_i is not in a state where it can execute σ, then the composed system (2.1) cannot execute the event σ.

It is typical for events to be shared, but it is rare for an event to be in the alphabet of all components. Many modular systems are coupled loosely such that several events appear in

only a few components, and occasionally an event may appear in only one component. Events that appear in the alphabet of only one component G_i of a composition (2.1) are called *local* to G_i, and local events are of special interest when computing a compositional abstraction.

An event local to one component G_i does not affect any other component of the system (2.1). An operation of *hiding* can be used to remove such events from the alphabet of G_i. Hiding is denoted by $G_i \setminus \Upsilon$, where $\Upsilon \subseteq \Sigma_i$ is a set of events to be removed from component G_i, the result being a component with alphabet $\Sigma_i \setminus \Upsilon$.

In process algebra (Bergstra and Klop 1984), hiding is done by replacing all transitions of a hidden event by *silent* transitions labelled by the event τ, a special event that never partakes in the synchronisation with other components. In discrete event systems, it is also common to use *natural projection* (Cassandras and Lafortune 2008) where local events are erased from the language, replacing a component with an FSM that accepts the language of the original component after removal of local events. Figure 2 shows examples of these operations.

The result of hiding is not necessarily equivalent to the component before hiding, i.e., it is not guaranteed that $G_1 \simeq G_1 \setminus \Upsilon$ even if Υ only contains local events. Nevertheless, it can often be ensured that hiding an event from a component does not change the result of checking the property of interest of the composed system, as long as only local events are hidden. Then hiding becomes another possible way to perform the above transformation of (2.7) into (2.8).

This is important because local events or silent transitions often enable simplifications that are not possible otherwise. And even though local events may be rare at the start, it is possible to choose components for partial composition to expose local events. In the example of Fig. 1, there may be events that appear only in $\tilde{G}_4$ and $\tilde{G}_5$ and in no other components. Then composition of $G_{45} = \tilde{G}_4 \| \tilde{G}_5$ causes the events previously shared between $\tilde{G}_4$ and $\tilde{G}_5$ to become local to G_{45}, enabling further simplification.

2.4 An abstract framework

Clarke et al. (1989) and Graf and Steffen (1990) propose frameworks for compositional verification that can be used to explain the methods that appear in this survey. This section describes such a framework, which will be used as reference throughout.

The starting point is a set $\mathcal{P}$ of *processes*, which may be FSMs (Hopcroft et al. 2001) or Kripke structures (Baier and Katoen 2008) or other objects. There is also a set $\hat{\Sigma}$ of *symbols* that may contain events that label FSM transitions or propositions that label states of Kripke structures. Associated with each process $A \in \mathcal{P}$ is its *alphabet* $\Sigma_A \subseteq \hat{\Sigma}$. Further, there are two process operations:

– If $A, B \in \mathcal{P}$ are processes, then their *composition* $A \| B \in \mathcal{P}$ is a process with alphabet $\Sigma_{A\|B} = \Sigma_A \cup \Sigma_B$. In this paper, composition is assumed to be associative and commuta-

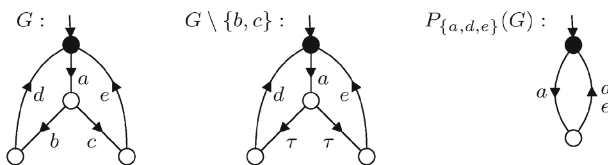


Fig. 2 Examples of hiding. The FSM G has two local events $\Upsilon = \{b, c\}$. Process-algebraic hiding $G \setminus \{b, c\}$ replaces their transitions with τ-transitions, while natural projection $P_{\{a,d,e\}}(G)$ produces a deterministic FSM without these events

tive. While this is not necessary for compositional abstraction, the assumption simplifies the exposition and is usually satisfied in supervisory control applications.

– If $A \in \mathcal{P}$ is a process and $\Upsilon \subseteq \hat{\Sigma}$ is a set of symbols, then the result of *hiding* is a process $A \setminus \Upsilon \in \mathcal{P}$ with alphabet $\Sigma_{A \setminus \Upsilon} = \Sigma_A \setminus \Upsilon$. Assumptions about hiding include that hiding an empty set of symbols or symbols that do not appear in the alphabet of a process has no effect, the order in which symbols are hidden does not change the result, and hiding of local symbols commutes with composition:

$$A \setminus \emptyset = A \qquad \text{for } A \in \mathcal{P}\ ; \tag{2.13}$$

$$A \setminus \Upsilon = A \setminus (\Sigma_A \cap \Upsilon) \qquad \text{for } A \in \mathcal{P} \text{ and } \Upsilon \subseteq \hat{\Sigma}\ ; \tag{2.14}$$

$$(A \setminus \Upsilon_1) \setminus \Upsilon_2 = A \setminus (\Upsilon_1 \cup \Upsilon_2) \qquad \text{for } A \in \mathcal{P} \text{ and } \Upsilon_1, \Upsilon_2 \subseteq \hat{\Sigma}\ ; \tag{2.15}$$

$$(A \setminus \Upsilon) \| B = (A \| B) \setminus \Upsilon \qquad \text{for } A, B \in \mathcal{P} \text{ and } \Upsilon \subseteq \hat{\Sigma} \setminus \Sigma_B\ . \tag{2.16}$$

Further, there is a notion of *equivalence* of processes, written $A \simeq B$ for $A, B \in \mathcal{P}$. The relation $\simeq$ is assumed to be an equivalence relation, i.e., it is reflexive, symmetric, and transitive. There also is a set Φ of relevant *properties* and a *satisfaction relation* $\models$, which determines for each process $A \in \mathcal{P}$ and each property $\varphi \in \Phi$ whether the process *satisfies* the property, written $A \models \varphi$. Properties may also be expressed using symbols from $\hat{\Sigma}$, in which case $\Sigma_\varphi \subseteq \hat{\Sigma}$ denotes the set of symbols used by property $\varphi \in \Phi$.

In summary, seven components characterise the compositional framework:

$$\mathbf{CF} = \langle \hat{\Sigma}, \mathcal{P}, \Phi; \simeq, \|, \setminus, \models \rangle\ , \tag{2.17}$$

where $\hat{\Sigma}$ is a set of symbols, $\mathcal{P}$ is a set of processes, Φ is a set of properties, $\simeq\ \subseteq \mathcal{P} \times \mathcal{P}$ is an equivalence relation, $\| \colon \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ is the associative and commutative operation of composition, $\setminus \colon \mathcal{P} \times 2^{\hat{\Sigma}} \to \mathcal{P}$ is the operation of hiding that satisfies (2.13)–(2.16), and $\models\ \subseteq \mathcal{P} \times \Phi$ is the property satisfaction relation.

In the framework of Graf and Steffen (1990), the goal is to minimise a collection of nondeterministic FSMs and compute an equivalent FSM that only uses events in some specified subset. In the above notation, assume processes $A_1, \ldots, A_n \in \mathcal{P}$, and a *target alphabet* $\hat{\Omega} \subseteq \hat{\Sigma}$ that contains the symbols that should be retained in the abstraction. Its complement $\hat{\Upsilon} = \hat{\Sigma} \setminus \hat{\Omega}$ contains symbols that can be hidden.

Graf and Steffen (1990) show that the compositional abstraction process can be used to compute an abstraction $\tilde{A} \in \mathcal{P}$ with alphabet $\Sigma_{\tilde{A}} = \hat{\Omega}$ such that

$$(A_1 \| \cdots \| A_n) \setminus \hat{\Upsilon} \simeq \tilde{A}\ . \tag{2.18}$$

The compositional abstraction $\tilde{A}$ is computed by repeatedly applying abstraction based on the process equivalence $\simeq$ to individual components, hiding local symbols from $\hat{\Upsilon}$, and composition until $A_1 \| \cdots \| A_n$ is transformed into $\tilde{A}$. Graf and Steffen (1990) show that this works when the process equivalence $\simeq$ is *weak bisimulation* (Milner 1989). Using weak bisimulation, they can compute an equivalent process with a minimal number of states, and therefore the approach is also known as *compositional minimisation*.

More generally, the method produces a result that satisfies Eq. 2.18 when the process equivalence $\simeq$ is a *congruence* with respect to composition and hiding, i.e.,

– For all processes $A, B, C \in \mathcal{P}$, if $A \simeq B$ then also $A \| C \simeq B \| C$.
– For all processes $A, B \in \mathcal{P}$ and all symbol sets $\Upsilon \subseteq \hat{\Sigma}$, if $A \simeq B$ then also $A \setminus \Upsilon \simeq B \setminus \Upsilon$.

To ensure that Eq. 2.18 holds, hiding must be restricted to events that are local and that do not appear in the target alphabet $\hat{\Omega}$.

Clarke et al. (1989) describe a more general framework for compositional verification that considers arbitrary properties. Their goal is to determine, for given processes $A_1, \ldots, A_n \in \mathcal{P}$ and property $\varphi \in \Phi$ whether

$$A_1 \| \cdots \| A_n \models \varphi \, . \tag{2.19}$$

The method to solve this problem is to use the compositional abstraction process to compute an abstraction $\tilde{A}$, and afterwards check using conventional methods whether $\tilde{A} \models \varphi$. For this to work, the process equivalence and hiding operation must meet additional requirements to ensure that the results of property verification are preserved, namely

- For all processes $A, B \in \mathcal{P}$ and all properties $\varphi \in \Phi$, if $A \simeq B$ and $A \models \varphi$, then also $B \models \varphi$.
- For all processes $A \in \mathcal{P}$, all properties $\varphi \in \Phi$, and all symbol sets $\Upsilon \subseteq \Sigma_A \setminus \Sigma_\varphi$, it holds that $A \models \varphi$ if and only if $A \setminus \Upsilon \models \varphi$.

The first condition requires that the process equivalence preserves the class of properties considered for verification, and this shows how process equivalences are linked to specific verification tasks. Restricting the set Φ of properties enables the use of more liberal equivalence relations that allow for more abstraction.

The second condition says that hiding preserves properties provided that the hidden symbols do not appear in the property. For properties expressed using symbols from $\hat{\Sigma}$, e.g., temporal logic formulas, symbols that appear in the property cannot be hidden. This can be avoided using a target symbol set $\hat{\Omega}$ as in the approach of Graf and Steffen (1990) described above.

Combining the above two conditions with the congruence requirements of Graf and Steffen (1990), a framework for compositional verification can be based on the following four assumptions.

(CV1) *Process equivalence is a congruence with respect to composition.* For all processes $A, B, C \in \mathcal{P}$, if $A \simeq B$ then also $A \| C \simeq B \| C$.

(CV2) *Process equivalence is a congruence with respect to hiding.* For all processes $A, B \in \mathcal{P}$ and all symbol sets $\Upsilon \subseteq \hat{\Sigma}$, if $A \simeq B$ then also $A \setminus \Upsilon \simeq B \setminus \Upsilon$.

(CV3) *Process equivalence respects properties.* For all processes $A, B \in \mathcal{P}$ and all properties $\varphi \in \Phi$, if $A \simeq B$ and $A \models \varphi$, then also $B \models \varphi$.

(CV4) *Hiding preserves properties.* If $A \in \mathcal{P}$ and $\varphi \in \Phi$ and $\Upsilon \subseteq \Sigma_A \setminus \Sigma_\varphi$, then $A \models \varphi$ if and only if $A \setminus \Upsilon \models \varphi$.

These conditions impose restrictions on the components of a compositional framework (2.17). If they are all satisfied, then the result of compositional abstraction satisfies any property from the set Φ that is satisfied by the original system. Using the results of Graf and Steffen (1990) and the axioms (2.13)–(2.16) about hiding, it can be shown that (CV1)-(CV4) are equivalent to the conditions given by Clarke et al. (1989).

Figure 3 gives an overview of the transformations used in the compositional abstraction process in the form of rewrite rules. Each of the three operations of *abstraction*, *hiding*, and *composition* rewrites a system of n composed processes

$$A_1 \| A_2 \| A_3 \| \cdots \| A_n \, , \tag{2.20}$$

where the alphabet of process A_i is Σ_i, into a new system. (It is enough to consider only the first component or the first two components as the components can be rearranged arbitrarily

Abstraction:

$$\frac{A_1 \parallel A_2 \parallel A_3 \parallel \cdots \parallel A_n}{\tilde{A}_1 \parallel A_2 \parallel A_3 \parallel \cdots \parallel A_n} \qquad \text{if } A_1 \simeq \tilde{A}_1$$

Hiding:

$$\frac{A_1 \parallel A_2 \parallel A_3 \parallel \cdots \parallel A_n}{(A_1 \setminus \Upsilon) \parallel A_2 \parallel A_3 \parallel \cdots \parallel A_n} \qquad \text{if } \Upsilon \subseteq \Sigma_1 \text{ and } \Upsilon \cap (\Sigma_2 \cup \cdots \cup \Sigma_n \cup \Sigma_\varphi) = \emptyset$$

Composition:

$$\frac{A_1 \parallel A_2 \parallel A_3 \parallel \cdots \parallel A_n}{(A_1 \parallel A_2) \parallel A_3 \parallel \cdots \parallel A_n}$$

Fig. 3 Rewrite rules of compositional abstraction

due to the assumption of associativity and commutativity of composition.) The repeated application of these transformations, in any order, results in an abstraction $\tilde{A}$ that is equivalent to the original system (2.20), provided that the framework satisfies the conditions (CV1)–(CV4). Then checking whether the original system (2.20) satisfies a property $\varphi \in \Phi$ is equivalent to checking whether the abstraction $\tilde{A}$ satisfies this property.

The following proposition states these observations formally.

Proposition 2.1 *Assume a compositional framework* $\mathbf{CF} = \langle \hat{\Sigma}, \mathcal{P}, \Phi; \simeq, \parallel, \setminus, \models \rangle$ *satisfies assumptions (CV1)-(CV4). For any processes* $A_1, \ldots, A_n \in \mathcal{P}$ *and property* $\varphi \in \Phi$*, if the repeated application of abstraction based on the equivalence* $\simeq$*, hiding of local symbols, and composition transforms* $A_1 \| \cdots \| A_n$ *into* $\tilde{A}$*, it holds that* $A_1 \| \cdots \| A_n \models \varphi$ *if and only if* $\tilde{A} \models \varphi$.

Some compositional synthesis methods allow for a component to be used more than once. Considering Fig. 1 this means that, after composing $G_{123} = \tilde{G}_{12} \| G_3$, the component G_3 is still available for composition with other components such as G_{45}. This reuse may be beneficial for computational or structural reasons. In the compositional framework, reuse can be expressed by the replacement of a process A with $A \| A$. For this replacement to be sound, it is required that $A \simeq A \| A$. While this equivalence holds for deterministic FSMs where $A = A \| A$, in general it depends on the process equivalence and composition operation used.

Section 4 below considers specific verification tasks, each of which leads to a different instance of the compositional framework.

2.5 Computational complexity and practical considerations

The goal of compositional algorithms is to improve on conventional algorithms that produce the same results, and therefore complexity and performance are important considerations. This section starts with an analysis of the computational complexity of compositional abstraction in relation to monolithic verification or synthesis and concludes with a discussion of some practical considerations for the implementation of efficient compositional algorithms.

The worst-case time complexity to verify a property or synthesise a supervisor for a composed system (2.1) is polynomial in the number of states of the composed system using

a straightforward monolithic algorithm, for the cases considered in this paper[1]. However, the number of states in the composed system is exponential in the number of components. Gohari and Wonham (2000) show that standard supervisory control problems are NP when measured by the number of components, so it is unlikely that there exists any algorithm with better than exponential time complexity to solve these problems. Thus, the only hope for compositional algorithms—and other methods to improve performance—is to produce results faster for practically relevant cases.

Assume compositional abstraction is applied to a system (2.1) and produces a result $\tilde{G}$. In the worst case, the size of $\tilde{G}$ is of the same order of magnitude as the composition of the original system, and $\tilde{G}$ is subjected to verification or synthesis by a monolithic algorithm. Thus, the worst-case time complexity of the compositional algorithm cannot be better than that of a monolithic algorithm. Fortunately, the final abstraction often is exponentially smaller than the composition of the original system, and this gives hope for substantial performance improvement.

On the other hand, compositional abstraction requires additional computation steps. Considering Fig. 1, the compositional abstraction of a system with n components involves up to $2n - 1$ hiding and abstraction steps and $n - 1$ composition steps. The input to each of these steps is a subsystem of the original system which, assuming that abstraction does not increase the size of components, has at most the size of the original system. If each abstraction operation can be performed in the same time complexity as the monolithic verification or synthesis of the final abstraction $\tilde{G}$, then the overall worst-case time complexity for compositional analysis is greater than monolithic analysis by a linear factor in the number of components. If this linear increase is compensated by an exponential reduction in size through abstraction, the performance improvement can be substantial.

In practice, performance improvements have also been reported (Flordal and Malik 2009; Mohajerani et al. 2014) when the complexity of abstraction is a polynomial of higher order than that of monolithic analysis, and even exponential-complexity abstraction methods have been used successfully in some cases (Roscoe et al. 1995; Ware and Malik 2008).

Another important practical question concerns the order in which the components are processed during compositional abstraction. Considering Fig. 1, there are many possible choices as to which components are selected and composed in the next step. While all choices lead to a correct result, the order of composition may have a significant impact on performance and, in the case of synthesis, on the size and structure of the computed supervisor.

In many cases, the components are grouped logically to reflect the structure of the system being modelled. Then it makes sense for system designers to determine what components should be composed in what order, choosing an order that reflects the system structure. In the case of synthesis, this approach can be used to construct distributed supervisors that match the physical system layout. An example of this is *Hierarchical Interface-Based Supervisory Control* (Leduc et al. 2005), where the system is structured into modules from the start, and the whole verification or synthesis process is guided by this structure.

In other cases, the logical relation between components may be nonexistent or unknown. For verification, the primary concern is to determine as quickly as possible whether a property is satisfied or not, and intermediate results are of little interest. Then it makes sense to determine the order of composition by an algorithm, so that compositional abstraction can be performed automatically without user interaction. Unfortunately, it is not straightforward

[1] If properties are specified in temporal logic, the time complexity may also be exponential in the size of the temporal logic formula.

to find an order that guarantees the best performance. A number of best-guess or heuristic approaches have been proposed for this purpose.

Su et al. (2010c) describe a *sequential abstraction procedure (SAP)* where the next component is always composed with the result of the previous abstraction step. Flordal and Malik (2009) propose a more flexible approach to support tree-like composition as suggested by Fig. 1. They evaluate various *candidate* sets of components and use heuristics to select a most promising candidate for the next step, usually by increasing local events or reducing the number of states in intermediate results. This approach is further developed with additional heuristics by Pilbrow and Malik (2015). As an alternative, Goorden et al. (2020) propose to build up a tree structure in advance using a clustering algorithm.

If the order of composition is determined by an algorithm, the time to make the decisions also affects the computational complexity. Most implementations limit the number of evaluated options to be linear or at most quadratic in the number of components or events, and use simple criteria such as numbers of states, transitions, or shared events. The experimental results in the above-mentioned papers suggest a negligible impact on computation time for evaluating heuristics, while different composition orders have a significant impact on performance.

3 Notation

This section provides notational background for the following sections. The material is compiled from a variety of sources. Supervisory control concepts are mainly based on Ramadge and Wonham (1989) and Cassandras and Lafortune (2008), and model checking concepts are primarily based on Baier and Katoen (2008).

3.1 Languages and finite-state machines

The behaviour of discrete event systems can be described using events and languages. *Events* represent incidents that cause transitions from one state to another and are taken from a global alphabet $\hat{\Sigma}$. This alphabet is assumed finite and contains two special events, namely the *silent event* τ and the *termination event* ω.

Given a set $\Sigma \subseteq \hat{\Sigma}$ of events, the set of all finite traces over Σ is denoted Σ^*, which includes the *empty trace* ε. A subset $L \subseteq \Sigma^*$ is called a *language* over Σ. The concatenation of two traces $s, t \in \hat{\Sigma}^*$ is written as st. A trace $s \in \hat{\Sigma}^*$ is called a *prefix* of $t \in \hat{\Sigma}^*$, written $s \sqsubseteq t$, if $t = su$ for some $u \in \hat{\Sigma}^*$. The *prefix-closure* of a language $L \subseteq \Sigma^*$ is the set of all the prefixes of its traces, $\mathrm{Pre}(L) = \{ s \in \Sigma^* \mid s \sqsubseteq t \text{ for some } t \in L \}$, and L is *prefix-closed* if $L = \mathrm{Pre}(L)$. For $\Omega \subseteq \hat{\Sigma}$, the *projection* $P_\Omega \colon \hat{\Sigma}^* \to \Omega^*$ is the operation that removes all events not in Ω from traces $s \in \hat{\Sigma}^*$. As a special case, the *natural projection* $P = P_{\hat{\Sigma}\setminus\{\tau\}} \colon \hat{\Sigma}^* \to (\hat{\Sigma} \setminus \{\tau\})^*$ removes all silent (τ) events from traces.

Definition 3.1 A (nondeterministic) *finite-state machine (FSM)* is a tuple $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, where $\Sigma \subseteq \hat{\Sigma} \setminus \{\tau, \omega\}$ is the FSM's *alphabet*, Q is a finite set of states, $\rightarrow \subseteq Q \times (\Sigma \cup \{\tau\}) \times Q$ is the *transition relation*, $Q^\circ \subseteq Q$ is the set of *initial states*, and $Q^\omega \subseteq Q$ is the set of *marked* or *accepting states*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and extended to traces in $(\Sigma \cup \{\tau\})^*$ by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. Furthermore, $x \xrightarrow{s}$ means that $x \xrightarrow{s} y$ for some $y \in Q$, and $x \rightarrow y$ means that $x \xrightarrow{s} y$ for

some $s \in (\Sigma \cup \{\tau\})^*$, and $x \stackrel{s}{\not\to}$ means that $x \stackrel{s}{\to}$ does not hold. These notations also apply to state sets, $X \stackrel{s}{\to}$ for $X \subseteq Q$ means that $x \stackrel{s}{\to}$ for some $x \in X$, and to FSMs, $G \stackrel{s}{\to}$ means that $Q^\circ \stackrel{s}{\to}$, etc.

An FSM G is *deterministic*, if it has at most one initial state, $|Q^\circ| \leq 1$, no silent transitions, $\to \subseteq Q \times \Sigma \times Q$, and the transition relation defines at most one successor for any given source state and event, i.e., $x \stackrel{\sigma}{\to} y_1$ and $x \stackrel{\sigma}{\to} y_2$ always implies $y_1 = y_2$.

While the completion of tasks is commonly expressed by reaching an accepting state, some abstraction relations can be expressed more conveniently when termination is represented by an event. Therefore, this paper also uses the following alternative FSM definition based on the termination event ω instead the set Q^ω of accepting states.

Definition 3.2 An *FSM with ω-transitions* is a tuple $G = \langle \Sigma, Q, \to, Q^\circ \rangle$, where $\Sigma \subseteq \hat{\Sigma} \setminus \{\tau, \omega\}$ is an alphabet, Q is a finite set of states, $\to \subseteq Q \times (\Sigma \cup \{\tau, \omega\}) \times Q$ is the transition relation, $Q^\circ \subseteq Q$ is the set of initial states, and states reached by ω do not have any outgoing transitions, i.e., if $x \stackrel{\omega}{\to} y$ then there does not exist $\sigma \in \Sigma \cup \{\tau, \omega\}$ such that $y \stackrel{\sigma}{\to}$.

The termination event ω denotes completion of a task and does not appear anywhere else but to mark such completions. States reached by ω do not have any outgoing transitions, so that the termination event, if it occurs, is always the final event of any trace. An FSM with ω-transitions can be transformed into an equivalent FSM with accepting states by defining $Q^\omega = \{\, x \in Q \mid x \stackrel{\omega}{\to} \,\}$ and deleting all ω-transitions. Conversely, an FSM with accepting states is transformed into an FSM with ω-transitions by adding a new state $\bot$ with transitions $x \stackrel{\omega}{\to} \bot$ for all accepting states $x \in Q^\omega$.

Example 3.1 The diagram to the left in Fig. 4 represents an FSM $G = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$ with alphabet $\Sigma = \{\mathsf{f}, \mathsf{r}, \mathsf{s}\}$ and state set $Q = \{\mathsf{I}, \mathsf{D}, \mathsf{W}\}$. State I is initial and accepting, $Q^\circ = Q^\omega = \{\mathsf{I}\}$. Its transitions are $\mathsf{I} \stackrel{\mathsf{s}}{\to} \mathsf{W}$, $\mathsf{W} \stackrel{\mathsf{f}}{\to} \mathsf{I}$, $\mathsf{W} \stackrel{\tau}{\to} \mathsf{D}$, and $\mathsf{D} \stackrel{\mathsf{r}}{\to} \mathsf{I}$. The diagram to the right shows an equivalent FSM with ω-transitions, where state I is designated as accepting through the transition $\mathsf{I} \stackrel{\omega}{\to} \bot$.

Some abstraction criteria are based on *always enabled events*. An event $\sigma \in \hat{\Sigma}$ is *enabled* at state x of an FSM G if $x \stackrel{\sigma}{\to}$ or $\sigma \in \hat{\Sigma} \setminus \Sigma$. If σ is enabled at all reachable states, i.e., all states x where $G \stackrel{s}{\to} x$ for some $s \in (\Sigma \cup \{\tau\})^*$, then σ is said to be *always enabled* in G. For FSMs with ω-transitions, the requirement $s \in (\Sigma \cup \{\tau\})^*$ means that σ does not need to be enabled after termination. Otherwise, if $\sigma \in \Sigma$ and there exists a reachable state x such that $x \stackrel{\sigma}{\not\to}$, then G is said to *disable* σ.

A second transition relation $\Rightarrow$ is introduced to capture the special meaning of the silent event τ. For $s \in (\Sigma \cup \{\omega\})^*$, the relation $x \stackrel{s}{\Rightarrow} y$ denotes the existence of a trace $t \in (\Sigma \cup \{\tau, \omega\})^*$ such that $x \stackrel{t}{\to} y$ and $P(t) = s$. That is, $\stackrel{s}{\to}$ denotes a path with *exactly* the events in s, while $\stackrel{s}{\Rightarrow}$ denotes a path with an arbitrary number of τ events shuffled with the events in s. The relation $\Rightarrow$ is applied to state sets and FSMs in the same way as $\to$.

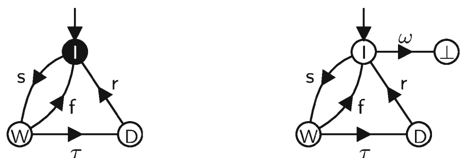


Fig. 4 An FSM $G = \langle \{\mathsf{f}, \mathsf{r}, \mathsf{s}\}, \{\mathsf{I}, \mathsf{D}, \mathsf{W}\}, \to, \{\mathsf{I}\}, \{\mathsf{I}\} \rangle$ and its equivalent representation with ω-transitions

In the context of this paper where event sets are frequently changed through hiding, it is convenient to extend the transition relation to traces over the global alphabet $\hat{\Sigma}$, with the understanding that events not in the alphabet of an FSM can always be executed without state change. Therefore, it is defined that $x \xrightarrow{\sigma} x$ for all states $x \in Q$ and all events $\sigma \in \hat{\Sigma} \setminus (\Sigma \cup \{\tau, \omega\})$. As a result,

$$x \stackrel{s}{\Rightarrow} y \quad \text{if and only if} \quad x \stackrel{P_{\Sigma \cup \{\omega\}}(s)}{\Longrightarrow} y \quad \text{for all } s \in (\hat{\Sigma} \setminus \{\tau\})^* . \tag{3.1}$$

Moreover, G is said to *enable* a trace s if $G \stackrel{s}{\Rightarrow}$. The *behaviour* or *language* of an FSM consists of all traces over the global alphabet that it enables. In supervisory control, it is common to distinguish the *prefix-closed* language $\mathcal{L}(G)$ that contains all enabled traces, and the *marked* or *accepted* language $\mathcal{M}(G)$ that contains traces that take the FSM to an accepting state.

$$\mathcal{L}(G) = \{\, s \in (\hat{\Sigma} \setminus \{\tau, \omega\})^* \mid G \stackrel{s}{\Rightarrow} \,\} ; \tag{3.2}$$

$$\mathcal{M}(G) = \{\, s \in (\hat{\Sigma} \setminus \{\tau, \omega\})^* \mid G \stackrel{s}{\Rightarrow} Q^\omega \,\} . \tag{3.3}$$

For FSMs with ω-transitions, the accepted language contains traces that can be continued by appending the termination event ω, but its traces do not include ω:

$$\mathcal{M}(G) = \{\, s \in (\hat{\Sigma} \setminus \{\tau, \omega\})^* \mid G \stackrel{s\omega}{\Rightarrow} \,\} . \tag{3.4}$$

When FSMs are brought together to interact, *lock-step* synchronisation in the style of Hoare (1985) is used.

Definition 3.3 Let $G_1 = \langle \Sigma_1, Q_1, \to_1, Q_1^\circ, Q_1^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \to_2, Q_2^\circ, Q_2^\omega \rangle$ be FSMs. The *synchronous composition* of G_1 and G_2 is

$$G_1 \| G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \to, Q_1^\circ \times Q_2^\circ, Q_1^\omega \times Q_2^\omega \rangle , \tag{3.5}$$

where

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2) \quad \text{if } \sigma \in (\Sigma_1 \cap \Sigma_2) \cup \{\omega\} \text{ and } x_1 \xrightarrow{\sigma}_1 y_1 \text{ and } x_2 \xrightarrow{\sigma}_2 y_2 ; \tag{3.6}$$

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, x_2) \quad \text{if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\} \text{ and } x_1 \xrightarrow{\sigma}_1 y_1 ; \tag{3.7}$$

$$(x_1, x_2) \xrightarrow{\sigma} (x_1, y_2) \quad \text{if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau\} \text{ and } x_2 \xrightarrow{\sigma}_2 y_2 . \tag{3.8}$$

Synchronous composition imposes synchronisation for transitions with events shared between two composed FSMs, which are either executed by both components together or not at all. Transitions with silent τ events or events that appear in only one of the composed FSMs are only executed by the FSM that contains the transition while leaving the state of the other unchanged. For the synchronous composition of FSMs with ω-transitions, the termination event ω is considered as shared and treated according to Eq. 3.6.

Given a finite set or multiset $\mathcal{G} = \{G_1, \ldots, G_n\}$ of FSMs, its synchronous composition is denoted as

$$\|(\mathcal{G}) = G_1 \| \cdots \| G_n . \tag{3.9}$$

The use of multisets is important to preserve multiple copies of the same component when working with nondeterministic FSMs, because in general $G \| G = G$ only holds for deterministic FSMs.

3.2 Supervisory control

For the purpose of supervisory control, the global event alphabet $\hat{\Sigma}$ is partitioned into two disjoint subsets, the set $\hat{\Sigma}_{\mathrm{c}}$ of *controllable* events and the set $\hat{\Sigma}_{\mathrm{u}}$ of *uncontrollable* events. A *supervisor* is a controlling agent that restricts the behaviour of a system represented as an FSM (called the *plant*). The supervisor observes the sequence of events occurring in the plant and then enables or disables certain controllable events, but it cannot disable any uncontrollable events.

Supervisory control theory (Ramadge and Wonham 1989; Cassandras and Lafortune 2008) is concerned with questions about what behaviour can be achieved by such supervisors. This is closely linked to the property of *controllability*.

Definition 3.4 (Ramadge and Wonham 1989) Let $K, L \subseteq \hat{\Sigma}^*$ be prefix-closed[2] languages, and let $\Sigma_{\mathrm{u}} \subseteq \hat{\Sigma}$. Then K is said to be Σ_{u}-*controllable* with respect to L if $K\Sigma_{\mathrm{u}} \cap L \subseteq K$.

In this definition, L represents the possible behaviour of the *plant*, and K represents a desired *specification* behaviour to be achieved by control. Controllability means that every uncontrollable event that is possible in the plant is also possible in the specification.

Definition 3.4 is parameterised with an uncontrollable event set Σ_{u} to allow for changing alphabets in compositional synthesis. This will often be the set of all uncontrollable events, $\Sigma_{\mathrm{u}} = \hat{\Sigma}_{\mathrm{u}}$, in which case K is simply called *controllable* with respect to L.

In addition to controllability, the behaviour of a supervised system is typically also required to be *nonblocking* or *nonconflicting* to avoid livelocks or deadlocks.

Definition 3.5 (Ramadge and Wonham 1989) Two languages $K, L \subseteq \hat{\Sigma}^*$ are *nonconflicting* if $\mathrm{Pre}(K) \cap \mathrm{Pre}(L) = \mathrm{Pre}(K \cap L)$.

As $\mathrm{Pre}(K \cap L) \subseteq \mathrm{Pre}(K) \cap \mathrm{Pre}(L)$ always holds, Definition 3.5 is equivalent to $\mathrm{Pre}(K) \cap \mathrm{Pre}(L) \subseteq \mathrm{Pre}(K \cap L)$. Thus, the (not necessarily prefix-closed) languages K and L being nonconflicting means that every prefix of their synchronised behaviour can be extended to a trace accepted by both K and L. In other words, every incomplete task can be completed by extending it to an accepted trace. The nonconflicting property is weaker than typical liveness properties in model checking, as it only requires that it always remains *possible* to complete tasks, not that every task is completed eventually.

When using FSMs with ω-transitions to define the languages K and L, the interpretation of a nonconflicting supervisor based on Definition 3.5 is captured by considering the termination event ω as controllable. This corresponds to *marking supervisory control* (Wonham 2013), where the supervisor decides whether or not the system has completed its tasks and is allowed to terminate. Alternatively, ω could be considered as uncontrollable for a model where only the plant decides whether the system terminates.

Ramadge and Wonham (1989) show that the specified behaviour K can be achieved by a supervisor controlling the plant if and only if K is controllable with respect to L, and K and L are nonconflicting. If K fails to be controllable or nonconflicting, then this behaviour cannot be enforced. Either the supervisor would attempt to disable some uncontrollable event although it is possible in the plant, or it would enter a livelock or deadlock situation where it is no longer possible to reach a trace accepted by the plant and specification together.

If K fails to be controllable or nonconflicting, the next best solution is to restrict K to some sublanguage $K' \subseteq K$, removing any behaviour that leads to violation of controllability

[2] The original definition of Ramadge and Wonham (1989) is not restricted to prefix-closed languages and includes aspects of the nonconflicting property described below.

or to conflict. Ramadge and Wonham (1989) show that controllability and the nonconflicting property with respect to L are closed under union of languages, and therefore a unique largest such language K' exists.

Definition 3.6 (Ramadge and Wonham 1989) Let $K, L \subseteq \hat{\Sigma}^*$. The *supremal controllable sublanguage* of K with respect to L is

$$\sup \mathcal{C}(K, L) = \bigcup \{\, K' \subseteq K \mid \mathrm{Pre}(K') \text{ is controllable with respect to } \mathrm{Pre}(L), \text{ and } K' \text{ and } L \text{ are nonconflicting} \,\}\,. \tag{3.10}$$

Ramadge and Wonham (1989) show that this supremal controllable sublanguage is again controllable and nonconflicting. It is the largest possible sub-behaviour of the specification K that can be achieved by a supervisor controlling the plant L in a nonconflicting way. It may be the empty language, in which case the supervisory control problem has no solution for the given plant and specification. Much of supervisory control theory is devoted to proving the existence of such supremal languages, and finding ways to *synthesise* or compute them.

The standard synthesis problem in supervisory control theory is to find a *maximally permissive controllable and nonblocking* supervisor, which amounts to the computation of $\sup\mathcal{C}(K, L)$. If the languages K and L are given by two FSMs, this problem is solved by a monolithic algorithm that explores the synchronous composition of the FSMs. In a compositional setting, it is more challenging to synthesise a maximally permissive supervisor. Maximal permissiveness is not essential for all applications, and it may be worth to forego it and only synthesise a *controllable and nonblocking* supervisor, which amounts to finding an element of the set $\mathcal{C}(K, L)$ of controllable and nonconflicting sublanguages of K.

For algorithmic processing, it is more convenient to work with states and transitions of an FSM instead of languages. To this end, Flordal et al. (2007) use a sub-FSM relationship to define controllability.

Definition 3.7 Let $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ, Q_1^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$ be FSMs. G_1 is a *sub-FSM* of G_2, written $G_1 \subseteq G_2$, if $\Sigma_1 = \Sigma_2$, $Q_1 \subseteq Q_2$, $\rightarrow_1 \subseteq \rightarrow_2$, $Q_1^\circ \subseteq Q_2^\circ$, and $Q_1^\omega \subseteq Q_2^\omega$.

Definition 3.8 (Flordal et al. 2007) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an FSM, let $E \subseteq G$ be a sub-FSM of G, and let $\Sigma_\mathrm{u} \subseteq \hat{\Sigma}$. Then E is Σ_u-*controllable in* G if, for every event $\mu \in \Sigma_\mathrm{u} \cap \Sigma$ and all states $x, y \in Q$ such that $E \rightarrow x$ and $G \rightarrow x \xrightarrow{\mu} y$, it holds that $E \rightarrow x \xrightarrow{\mu} y$.

A sub-FSM E is controllable in G if every reachable uncontrollable transition of G also exists in E. Here, G is the plant and E the specification. More generally, if the plant and specification behaviours are represented using different state spaces, G and E can be constructed using synchronous composition.

Definitions 3.4 and 3.8 are equivalent for deterministic FSMs: if E is a sub-FSM of a deterministic FSM G, then E is controllable in G if and only if $\mathcal{L}(E)$ is controllable with respect to $\mathcal{L}(G)$. In the nondeterministic case, Definition 3.8 assumes that a supervisor can disable transitions rather than events. For example, if G contains two transitions $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ from state x with controllable event σ, then a supervisor can allow the system to enter state y_1 but not y_2. This makes sense under the assumption that the nondeterministic model is an abstraction of an originally deterministic system, so that the supervisor can distinguish transitions using knowledge about the global state.

Definition 3.9 An FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ is *nonblocking* if, for every state x such that $G \rightarrow x$ it holds that $x \rightarrow Q^\omega$. Two FSMs G and H are *nonconflicting* if $G \| H$ is nonblocking.

An FSM is nonblocking, if it is possible to reach an accepting state from every reachable state. For an FSM with ω-transitions to be nonblocking, it is required that $G \xrightarrow{s} x$ with $s \in \Sigma^*$ implies that there exists $t \in \Sigma^*$ such that $x \xrightarrow{t\omega}$, i.e., every trace that does not contain the termination event can be extended to a trace that ends with termination. Definition 3.9 extends the language-based Definition 3.5 of the nonconflicting property for deterministic FSMs: two nonblocking deterministic FSMs G and H are nonconflicting if and only if $\mathcal{M}(G)$ and $\mathcal{M}(H)$ are nonconflicting. This does not hold for nondeterministic FSMs. The use of the transition relation in Definition 3.9 ensures that the nonblocking property is preserved by hiding.

Definition 3.10 (Flordal et al. 2007) Let G be an FSM, and let $E \subseteq G$ be a sub-FSM. The *supremal controllable and nonblocking sub-FSM* of G with respect to E is

$$\sup\mathcal{C}(E, G) = \bigcup \{\, E' \subseteq E \mid E' \text{ is controllable in } G \text{ and nonblocking} \,\} . \tag{3.11}$$

Definition 3.10 redefines synthesis based on sub-FSMs. The operator $\cup$ denotes the least upper bound in the lattice of FSMs with the sub-FSM relation (Definition 3.7). It can be shown that the least upper bound of controllable and nonblocking FSMs is again controllable and nonblocking. Therefore $\sup\mathcal{C}(E, G)$ is the supremal controllable and nonblocking sub-FSM of E. It can be computed by removing transitions until a fixpoint is reached (Flordal et al. 2007).

3.3 Abstraction by state merging

This subsection describes abstraction by *state merging*, which is a foundation for several more specific abstraction procedures that appear later in the paper. The idea of state merging is to identify certain states of an FSM as equivalent and group equivalent states together to form a new FSM, called the *quotient FSM*.

Definition 3.11 Let Z be a set. A relation $\sim\, \subseteq Z \times Z$ is called an *equivalence relation* on Z if it is reflexive, symmetric, and transitive. Given an equivalence relation $\sim$ on Z, the *equivalence class* of $z \in Z$ is $[z] = \{\, z' \in Z \mid z \sim z' \,\}$, and $Z/{\sim} = \{\, [z] \mid z \in Z \,\}$ is the set of all equivalence classes modulo $\sim$.

Definition 3.12 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM with ω-transitions[3], and let $\sim\, \subseteq Q \times Q$ be an equivalence relation on its state set. The *quotient FSM* of G modulo $\sim$ is $G/{\sim} = \langle \Sigma, Q/\sim, \rightarrow/\sim, \tilde{Q}^\circ \rangle$, where $\rightarrow/{\sim} = \{\, ([x], \sigma, [y]) \mid x \xrightarrow{\sigma} y \,\}$ and $\tilde{Q}^\circ = \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\}$.

The state space of the quotient FSM consists of equivalence classes, each representing a set of equivalent states that have been merged. The quotient FSM $G/{\sim}$ contains a transition between two such sets if the original FSM G contains a transition from a state in the source set to a state in the target set.

[3] The definitions in this subsection use ω-transitions for conciseness, as this avoids extra conditions for accepting states.

The language of a quotient FSM always includes the language of the original FSM, i.e., $\mathcal{L}(G) \subseteq \mathcal{L}(G/{\sim})$ and $\mathcal{M}(G) \subseteq \mathcal{M}(G/{\sim})$ for every FSM and equivalence relation. This fact is used in model checking to verify certain properties, for example, if a quotient FSM satisfies a particular safety property, then the original FSM satisfies the same property. The converse inclusion $\mathcal{L}(G/{\sim}) \subseteq \mathcal{L}(G)$ does not hold in general, so state merging only preserves verification results in one direction (Bérard et al. 2001).

Several compositional methods use quotient FSMs modulo specific equivalence relations, to ensure that the abstraction is in some sense equivalent to the original FSM. One such relation is known as *bisimulation*.

Definition 3.13 (Milner 1989) Let $G_1 = \langle \Sigma, Q_1, \to_1, Q_1^\circ \rangle$ and $G_2 = \langle \Sigma, Q_2, \to_2, Q_2^\circ \rangle$ be two FSMs with equal event sets. A relation $\approx \subseteq Q_1 \times Q_2$ is a *bisimulation* between G_1 and G_2, if for all $x_1 \in Q_1$ and $x_2 \in Q_2$ and all $\sigma \in \Sigma \cup \{\tau, \omega\}$ such that $x_1 \approx x_2$, the following conditions hold:

- If $x_1 \xrightarrow{\sigma}_1 y_1$ for some $y_1 \in Q_1$, then there exists $y_2 \in Q_2$ such that $x_2 \xrightarrow{\sigma}_2 y_2$ and $y_1 \approx y_2$.
- If $x_2 \xrightarrow{\sigma}_2 y_2$ for some $y_2 \in Q_2$, then there exists $y_1 \in Q_1$ such that $x_1 \xrightarrow{\sigma}_1 y_1$ and $y_1 \approx y_2$.

It is important to note that Definition 3.13 does not directly define a relation $\approx$, it merely provides conditions to determine whether a given relation can be called a bisimulation. For a global relation, two states x_1 and x_2 are called *bisimilar* if there exists a bisimulation $\approx$ such that $x_1 \approx x_2$. This means that two states are bisimilar if they have exactly the same enabled events, with transitions leading to states that are again bisimilar, or in other words bisimilar states have the exact same branching structure of their future transitions. Furthermore, two FSMs are bisimilar if they have bisimilar initial states.

Definition 3.14 (Milner 1989) Two FSMs $G_1 = \langle \Sigma_1, Q_1, \to_1, Q_1^\circ \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \to_2, Q_2^\circ \rangle$ with equal event sets are called *bisimilar*, written $G_1 \approx G_2$, if there exists a bisimulation $\approx$ between G_1 and G_2, such that for every initial state $x_1^\circ \in Q_1^\circ$ there exists $x_2^\circ \in Q_2^\circ$ such that $x_1^\circ \approx x_2^\circ$, and vice versa.

Relationships between two FSMs as given here are useful to define process equivalences, while algorithmic computation typically works with a relation on the state set of a single FSM. Formally, given an FSM G, a relation is a bisimulation *on* G if it is a bisimulation between G and itself. A useful property of bisimulation is that the bisimilarity between an FSM and its quotient is guaranteed.

Proposition 3.1 *Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an FSM, and let $\approx \subseteq Q \times Q$ be a bisimulation on G. Then $G \approx (G/{\approx})$.*

A limitation of bisimulation is that it treats the silent event τ like an ordinary event and fails to exploit the fact that silent transitions do not engage in synchronous composition with other components. As an alternative, *weak bisimulation* or *observation equivalence* recognises this special meaning of the silent event. It is defined in a similar way to bisimulation.

Definition 3.15 (Milner 1989) Let $G_1 = \langle \Sigma, Q_1, \to_1, Q_1^\circ \rangle$ and $G_2 = \langle \Sigma, Q_2, \to_2, Q_2^\circ \rangle$ be two FSMs with equal event sets. A relation $\sim \subseteq Q_1 \times Q_2$ is a *weak bisimulation* between G_1 and G_2, if for all $x_1 \in Q_1$ and $x_2 \in Q_2$ and all $\sigma \in \Sigma \cup \{\tau, \omega\}$ such that $x_1 \sim x_2$, the following conditions hold:

- If $x_1 \stackrel{P(\sigma)}{\Rightarrow}_1 y_1$ for some $y_1 \in Q_1$, then there exists $y_2 \in Q_2$ such that $x_2 \stackrel{P(\sigma)}{\Rightarrow}_2 y_2$ and $y_1 \sim y_2$.
- If $x_2 \stackrel{P(\sigma)}{\Rightarrow}_2 y_2$ for some $y_2 \in Q_2$, then there exists $y_1 \in Q_1$ such that $x_1 \stackrel{P(\sigma)}{\Rightarrow}_1 y_1$ and $y_1 \sim y_2$.

G_1 and G_2 are called *weakly bisimilar*, written $G_1 \sim G_2$, if there exists a weak bisimulation $\sim$ between G_1 and G_2, such that for every initial state $x_1^\circ \in Q_1^\circ$ there exists $x_2 \in Q_2$ such that $Q_2^\circ \stackrel{\varepsilon}{\Rightarrow}_2 x_2$ and $x_1^\circ \sim x_2^\circ$, and vice versa.

The main difference between bisimulation and weak bisimulation is that the transition relation $\rightarrow$ is replaced by the extended transition relation $\Rightarrow$. That is, for two states to be weakly bisimilar, they must be able to execute the same events and reach weakly bisimilar states afterwards, while possibly executing an arbitrary number of silent transitions before and/or after the event. Additionally for weak bisimulation, states are considered as initial if they can be reached by a sequence of silent transitions from an actual initial state.

When simplifying an FSM using a quotient modulo a weak bisimulation on its state set, it is common to limit the occurrence of τ-transitions. An equivalent result to Proposition 3.1 holds for weak bisimulations on FSMs with accepting states. When using ω-transitions, care needs to be taken not to merge the state reached after ω with other states. For example, if $x \stackrel{\omega}{\rightarrow} y$ and $z \stackrel{\tau}{\rightarrow} z$ with no other transitions from z, then y and z are weakly bisimilar states, resulting in a τ-selfloop after ω in the quotient and violating the requirement that there can be no transitions after ω. The following alternative quotient construction avoids the problem by removing all τ-selfloops, which preserves weak bisimulation.

Definition 3.16 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim \subseteq Q \times Q$ be an equivalence relation. The *τ-loop free quotient* is $G/_\circ\!\sim\; = \langle \Sigma, Q/\!\sim, \rightarrow_\circ, \tilde{Q}^\circ \rangle$ where $[x] \stackrel{\sigma}{\rightarrow}_\circ [y]$ for $\sigma \in \Sigma \cup \{\omega\}$ if $x \stackrel{\sigma}{\rightarrow} y$, and $[x] \stackrel{\tau}{\rightarrow}_\circ [y]$ if $x \stackrel{\tau}{\rightarrow} y$ and $[x] \neq [y]$, and $\tilde{Q}^\circ = \{\, [x] \in Q/\!\sim \mid x \in Q^\circ \,\}$.

Proposition 3.2 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim \subseteq Q \times Q$ be a weak bisimulation on G. Then $G \sim (G/_\circ\!\sim)$.*

Following Propositions 3.1 and 3.2, an FSM can be simplified by computing an appropriate equivalence relation on its state set, and then forming a quotient. Fernandez (1990) describes an efficient $O(m \log n)$ partition refinement algorithm to compute a coarsest bisimulation relation on a given FSM, where m is the number of transitions and n is the number of states of the FSM. This algorithm can also be used for weak bisimulation, except that it must be based on the extended transition relation $\Rightarrow$. This poses a computational challenge, because to determine whether $x \stackrel{\sigma}{\Rightarrow} y$ holds, it is necessary to explore all the states reached by zero or more τ-transitions from x, which usually involves transitive closure computation. The time complexity to compute the relation $\Rightarrow$ is $O(n^3)$, which also becomes the worst-case time complexity to compute a coarsest weak bisimulation relation.

3.4 Temporal logic

Temporal logic is commonly used in model checking to express critical properties of systems (Baier and Katoen 2008). Through the language of formal logic, various aspects of behaviour can be expressed concisely. Research in model checking has shown how to determine options for verification of particular properties by analysing the structure of their temporal logic formulas (Bérard et al. 2001).

The basic building blocks of temporal logic formulas are *atomic propositions*, which are primitive statements known to be *true* or *false* in a given state. For example, a proposition may express that the door of an elevator is open or that the temperature of a reactor exceeds a certain threshold. The propositions are combined using logical *operators* or *connectives* to build up more complex formulas. Temporal logic includes well-known propositional connectives such as $\neg$ ("not"), $\wedge$ ("and"), $\vee$ ("or"), and $\Rightarrow$ ("implies"), the literal true, and a set of *temporal connectives* and *path quantifiers*. The most common temporal connectives are:

- $G\varphi$ ("globally φ"). For a temporal logic formula φ, the expression $G\varphi$ means that φ is true in the current state and in all future states.
- $F\varphi$ ("finally φ"). The expression $F\varphi$ means that φ is true in the current or some future state.
- $X\varphi$ ("next φ"). The expression $X\varphi$ means that φ will be true in the next state, i.e., after execution of a single transition.
- $\varphi U\psi$ ("φ until ψ"). For temporal logic formulas φ and ψ, the expression $\varphi U\psi$ means that ψ is true in the current or some future state, and φ is true in all states starting from the current state up to, but not necessarily including, the first state where ψ holds.

The temporal connectives capture the sequencing of actions in time and are best interpreted over an infinite sequence of states representing a possible execution of a state machine. For example, if p is an atomic proposition, then an infinite path $x_0 \to x_1 \to \cdots$ satisfies $G\,p$ if p is true in every state x_i on the path.

In addition to the temporal connectives, *path quantifiers* capture the idea that the future can evolve in different ways:

- $A\varphi$ ("all φ"). The expression $A\varphi$ means that the temporal logic formula φ is true for every possible future behaviour.
- $E\varphi$ ("exists φ"). The expression $E\varphi$ means that the temporal logic formula φ is true for some possible future behaviour.

The path quantifiers are best interpreted for a given state of a state machine. For example, $A\varphi$ is true in a state if every path starting from that state satisfies φ.

The *Computation Tree Logic CTL** (Emerson and Halpern 1986) is built from the above connectives. Its formulas can take the forms

$$\mathit{true} \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid X\varphi \mid \varphi U\psi \mid E\varphi\,, \tag{3.12}$$

where p is an atomic proposition and φ and ψ are CTL* formulas. Some of the above-mentioned connectives do not appear in (3.12) because they can be expressed using the others: $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \Rightarrow \psi \equiv \neg\varphi \vee \psi$, $F\varphi \equiv \mathit{true}U\varphi$, $G\varphi \equiv \neg F\neg\varphi$, and $A\varphi \equiv \neg E\neg\varphi$.

Example 3.2 Various properties can be expressed in CTL* by combining the propositional and temporal connectives and path quantifiers. For example, if req and grant are atomic propositions, then $AG\,\neg(\mathsf{req} \wedge \mathsf{grant})$ means that all the states on all the paths starting from the current state satisfy $\neg(\mathsf{req} \wedge \mathsf{grant})$, i.e., req and grant will never be true simultaneously in any reachable state. As another example, $AG\,(\mathsf{req} \Rightarrow AF\,\mathsf{grant})$ means that all paths starting from any reachable state where req is true also contain a state where grant holds, i.e., visiting a req state is guaranteed to be followed by visiting a grant state.

A formal definition of the semantics of CTL* is typically based on *Kripke structures* (Clarke et al. 1986), which are FSMs whose states are labelled with propositions.

Definition 3.17 A *Kripke structure* is a tuple $K = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, L\rangle$ where $\langle \Sigma, Q, \rightarrow, Q^\circ\rangle$ is an FSM, Π is a finite set of *propositions*, and $L\colon Q \rightarrow 2^\Pi$ is the *state labelling function*.

The state labelling function defines at what states the propositions are true or false: proposition $p \in \Pi$ is true in state $x \in Q$ if $p \in L(x)$. This can be viewed as a generalisation of FSMs with several sets of accepting states, one for each proposition. The availability of propositions makes it possible to express behaviour without reference to events, and indeed Kripke structures are often defined without an event alphabet Σ. Definition 3.17 retains the events, which is more convenient for supervisory control applications.

Given a Kripke structure $K = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, L\rangle$ and a CTL* formula φ that uses propositions from Π, the *semantics* of CTL* defines whether or not the Kripke structure K satisfies the formula φ. A precise definition (Baier and Katoen 2008) distinguishes *path formulas* and *state formulas* and proceeds in several steps. Path formulas such as $\varphi U \psi$ are defined to be true or false for a given sequence of states as explained above, and state formulas are defined to be true or false in a given state. The simplest state formulas are the atomic propositions, whose truth value is determined by the state labelling function. The path quantifiers are applied to path formulas to produce state formulas, for example, $A\varphi$ is true in a state if φ is true on every infinite[4] path starting from that state. Finally, a Kripke structure K satisfies a state formula φ, written $K \models \varphi$, if φ is true in every initial state of K.

The CTL* model checking problem is to determine for a given Kripke structure K and state formula φ whether $K \models \varphi$ holds. For compositional verification, the problem becomes to determine whether

$$K_1 \| \cdots \| K_n \models \varphi \tag{3.13}$$

for Kripke structures $K_1, \ldots, K_n$, which requires a definition of the composition of Kripke structures.

Definition 3.18 Let $K_1 = \langle \Sigma_1, \Pi_1, Q_1, \rightarrow_1, Q_1^\circ, L_1\rangle$ and $K_2 = \langle \Sigma_2, \Pi_2, Q_2, \rightarrow_2, Q_2^\circ, L_2\rangle$ be Kripke structures. Their *synchronous composition* is

$$K_1 \| K_2 = \langle \Sigma_1 \cup \Sigma_2, \Pi_1 \cup \Pi_2, Q_1 \times Q_2, \rightarrow, Q_1^\circ \times Q_2^\circ, L\rangle \,, \tag{3.14}$$

where $\rightarrow$ is as defined in Definition 3.3 and the labelling function L is such that $p \in L((x_1, x_2))$ if and only if one of the following conditions holds:

- $p \in \Pi_1 \cap \Pi_2$ and $p \in L_1(x_1)$ and $p \in L_2(x_2)$;
- $p \in \Pi_1 \setminus \Pi_2$ and $p \in L_1(x_1)$;
- $p \in \Pi_2 \setminus \Pi_1$ and $p \in L_2(x_2)$.

According to this definition, a synchronised state is labelled by a proposition shared between two composed Kripke structures if both state components are labelled by that proposition, whereas the labelling of states with propositions that appear in only one of the composed Kripke structures only depends on the Kripke structure containing it.

For Kripke structures without events, their synchronous composition can be be defined based on propositions (Clarke et al. 1999). In this case, a transition $(x_1, x_2) \rightarrow (y_1, y_2)$ exists in a synchronous composition $K_1 \| K_2$, if $x_1 \rightarrow_1 y_1$ and $x_2 \rightarrow_2 y_2$ and $L_1(x_1) \cap \Pi_2 =$

[4] The restriction to infinite paths is necessary for path formulas such as $F\varphi$, but it causes problems for Kripke structures with deadlock states, i.e., states without outgoing transitions. To allow for deadlocks, it is possible to consider *maximal* paths instead, which are either infinite or finite in such a way that the last state is a deadlock state.

$L_2(x_2) \cap \Pi_1$ and $L_1(y_1) \cap \Pi_2 = L_2(y_2) \cap \Pi_1$, i.e., the source and target states evaluate shared propositions in the same way. Both the definitions of synchronous composition of Kripke structures based on events or propositions are associative and commutative, and can be used in the compositional framework.

CTL* model checking is usually based on state labels, but in supervisory control it is more common to reason about the sequencing of events. Baier and Katoen (2008) propose a way to transform an FSM with event set Σ into a Kripke structure with proposition set Σ. In this transformation, a state y is labelled with $\sigma \in \Sigma$ if and only if it is entered by a transition $x \xrightarrow{\sigma} y$. That is, the proposition σ is true precisely when the event σ has occurred on the previous transition; initially or after a silent τ transition all propositions are false. At most one such proposition is true in each state—states entered by transitions with different events are replicated. This transformation does not commute with synchronous composition and therefore cannot be used compositionally. It is only applied to the final result of compositional abstraction. Fortunately, it is rarely needed for supervisory control applications. The following example shows how controllability (Definition 3.4) can be expressed more directly in CTL*.

Example 3.3 Consider a system composed of deterministic plant and specification FSMs $G = G_1 \| \cdots \| G_n$ and $E = E_1 \| \cdots \| E_m$. To express Σ_u-controllability, two propositions g_μ and e_μ are defined for each uncontrollable event $\mu \in \Sigma_\mathrm{u}$, where g_μ means that μ is enabled by G and e_μ means that μ is enabled by E. This meaning is achieved by converting each plant component $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ \rangle$ to a Kripke structure $G_i' = \langle \Sigma_i, \Pi_i, Q_i, \rightarrow_i, Q_i^\circ, L_i \rangle$ with $\Pi_i = \{\, \mathsf{g}_\mu \mid \mu \in \Sigma_i \cap \Sigma_\mathrm{u} \,\}$ and $L_i(x) = \{\, \mathsf{g}_\mu \in \Pi_i \mid x \xrightarrow{\mu}_i \,\}$. Similarly, specification components E_j are converted to E_j' with $\Pi_j = \{\, \mathsf{e}_\mu \mid \mu \in \Sigma_j \cap \Sigma_\mathrm{u} \,\}$ and $L_j(x) = \{\, \mathsf{e}_\mu \in \Pi_j \mid x \xrightarrow{\mu}_j \,\}$. Then the Σ_u-controllability of $\mathcal{L}(E)$ with respect to $\mathcal{L}(G)$ is expressed by the CTL* formula

$$AG \bigwedge_{\mu \in \Sigma_\mathrm{u}} (\mathsf{g}_\mu \Rightarrow \mathsf{e}_\mu)\,. \tag{3.15}$$

This formula states that, in every reachable state of the synchronous composition of G and E, every uncontrollable event μ enabled by all plant components G_i is also enabled by all specification components E_j.

4 Compositional verification

Compositional verification can be carried out by repeatedly hiding symbols, simplifying components, and composing subsystems as described by the compositional framework in Section 2. Depending on the type of property to be verified, simplification is based on a different process equivalence $\simeq$. Table 1 gives an overview of the process equivalences and associated properties, which are explained in the following subsections.

For each of these equivalence relations, there is a normalisation algorithm that transforms a given FSM into a unique equivalent normal form. This normal form constitutes a possible abstraction that can replace the original FSM in compositional verification, although it is not always guaranteed to be smaller. For example, the minimal deterministic FSM that accepts the same language as a given nondeterministic FSM may have exponentially more states (Hopcroft et al. 2001). In these cases, there are alternative methods to simplify FSMs while ensuring a reduction of the number of states, which are also explained in the following subsections.

Table 1 Process equivalences used for compositional verification

Process equivalence		Preserved	Normalisation	Guaranteed	Explained
Name	Symbol	properties	complexity	reduction	in
Bisimulation	$\approx$	CTL*	$O(\|\rightarrow\| \cdot \log \|Q\|)$	yes	Section 4.1
Language equivalence	$\simeq_{\mathcal{L}}$	safety	$O(\|\rightarrow\| \cdot 2^{\|Q\|})$	no	Section 4.2
Failures equivalence	$\simeq_{\mathcal{F}}$	absence of deadlock	$O(\|\rightarrow\| \cdot 2^{\|Q\|})$	no	Section 4.3
Conflict equivalence	$\simeq_{\text{conf}}$	nonblocking	$O(\|\Sigma\| \cdot 2^{5\|Q\|})$	no	Section 4.4
Weak bisimulation	$\sim$	safety, nonblocking	$O(\|Q\|^3)$	yes	Section 4.4.1

4.1 CTL* model checking

The compositional CTL* model checking problem is to determine whether a composed system satisfies a CTL* formula,

$$G_1 \| \cdots \| G_n \models \varphi \; . \tag{4.1}$$

The components $G_1, \ldots, G_n$ may be Kripke structures or FSMs. For Kripke structures, the property φ is expressed using their atomic propositions, and for FSMs, the property uses their events as atomic propositions.

Baier and Katoen (2008) show that bisimulation (Definition 3.14) is the coarsest equivalence of FSMs that preserves all CTL* properties. Therefore, bisimulation can be used as a process equivalence to define a framework according to Section 2.4 for the compositional verification of (4.1). The components of this framework are

$$\mathbf{CF} = \langle \hat{\Sigma}, \mathcal{P}, \Phi; \approx, \|, \setminus, \models \rangle \; , \tag{4.2}$$

where:

- The set $\hat{\Sigma}$ of symbols is a global event or proposition alphabet.
- The set $\mathcal{P}$ of processes is either the set of Kripke structures or the set of nondeterministic FSMs with any event alphabet $\Sigma \subseteq \hat{\Sigma}$.
- The set Φ of properties is the set of CTL* formulas φ expressed either using the atomic propositions of the Kripke structures or using events in $\hat{\Sigma}$.
- Process equivalence is bisimulation. For FSMs, $A \approx B$ is defined according to Definition 3.14. For Kripke structures, the definition is strengthened to ensure that only states with the same labels are bisimilar. That is, a relation $\approx \subseteq Q_1 \times Q_2$ between the state sets of two Kripke structures is a bisimulation if it satisfies the conditions of Definition 3.13 and additionally $x_1 \approx x_2$ implies $L(x_1) = L(x_2)$. This condition can be limited to propositions used in properties.
- Composition $\|$ is either lock-step synchronous composition of nondeterministic FSMs (Definition 3.3) or its extended version for Kripke structures (Definition 3.18).
- Hiding $G \setminus \Upsilon$ is the standard replacement of events in Υ by the silent event τ. For FSMs, this is restricted to events not used in properties.
- Property satisfaction $G \models \varphi$ is given by the CTL* semantics in Section 3.4.

It can be shown that this choice satisfies conditions (CV1)–(CV4). Bisimulation is known (Milner 1989) to be a congruence with respect to synchronous composition (CV1) and hiding (CV2). Baier and Katoen (2008) show that bisimulation of FSMs and Kripke structures

preserves CTL* properties (CV3). The preservation of CTL* under hiding (CV4) is immediate for Kripke structures as the truth values of CTL* formulas only depend on atomic propositions that are not affected by hiding. For FSMs, where properties use event labels as atomic propositions, it is important that property events are not hidden so that the relevant propositions can be added correctly to the final result of compositional abstraction.

Accordingly, compositional reasoning using hiding and bisimulation can be used to verify arbitrary CTL* properties. This is the most general verification method and works for all properties that can be expressed in CTL*—a large set that includes all the properties considered in the remainder of this paper.

On the other hand, bisimulation is a fine equivalence of nondeterministic state machines: for two states to be bisimilar, they must have identical branching structures including τ transitions. This fact greatly limits the amount of state space reduction. More simplification can be achieved by restricting the set Φ of properties that can be checked. One possibility is to remove the "next" connective X, which is only needed when the precise number of transitions leading from one state to another is relevant. Clarke et al. (1989) and Baier and Katoen (2008) propose relaxations of bisimulation that preserve CTL* without the connective X. For supervisory control, the classes of properties can be limited further.

4.2 Safety properties

Safety properties are described informally as properties that require a system never to do anything "bad", or equivalently that the system must always remain within a "safe" subset of the state space (Bérard et al. 2001). In temporal logic, this is expressed using the "globally" connective, for example in CTL*

$$AG\,\varphi \tag{4.3}$$

means that φ must be true at all times. Here, φ is a purely propositional formula (i.e., φ contains no temporal connectives or path quantifiers) that describes the safe states. In supervisory control, the most common safety property is *controllability* (Ramadge and Wonham 1989), which can be expressed in the form (4.3) as shown in Example 3.3.

Safety properties of FSMs can also be characterised as *language inclusion*. If the system behaviour is given by an FSM G with language $\mathcal{L}(G) \subseteq \Sigma^*$, a safety property can be specified as a prefix-closed language representing the maximally permitted behaviour. This language can be specified through a property FSM E, in which case the system G is said to satisfy the property E if

$$\mathcal{L}(G) \subseteq \mathcal{L}(E)\ . \tag{4.4}$$

As termination is not a concern with safety properties, it is enough to use the prefix-closed languages $\mathcal{L}(G)$ and $\mathcal{L}(E)$, i.e., marked states are not relevant here.

Example 4.1 Figure 5 shows an example of a property FSM E. Here, a two-element buffer is specified, where event put indicates that an item is placed in the buffer and get indicates that an item is removed. The property E specifies that the buffer should not overflow by disallowing the event put when there are two items in the buffer. By verifying (4.4), a language inclusion check determines whether a system G avoids buffer overflow.

For compositional verification of such a property, it is again assumed that the system G is given as the synchronous composition of n components (2.1) synchronised through events in the global alphabet $\hat{\Sigma}$. The property FSM E is assumed to be defined over some subset $\Omega \subseteq \hat{\Sigma}$ of the global alphabet. Then the goal of safety property verification is to determine

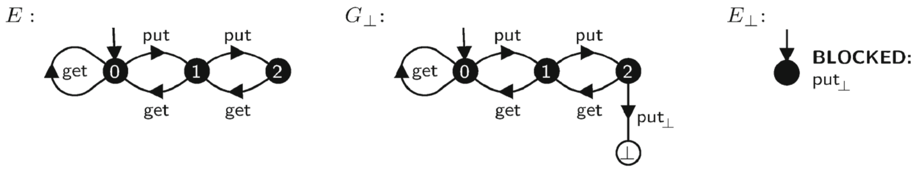


Fig. 5 Properties for overflow prevention in a two-element buffer. Property E describes that put cannot occur more than twice without get. Equivalently, $G_\perp$ distinguishes put and put$_\perp$, with property $E_\perp$ specifying that the overflow case put$_\perp$ should not occur

whether

$$P_\Omega(\mathcal{L}(G_1\|G_2\|\cdots\|G_n)) \subseteq \mathcal{L}(E)\ . \tag{4.5}$$

These observations suggest to define a framework for the compositional verification of safety properties according to Section 2.4. Its components are

$$\mathbf{CF} = \langle \hat{\Sigma}, \mathcal{P}, \Phi; \simeq, \|, \setminus, \models \rangle\ ,$$

where $\hat{\Sigma}$ is the global system alphabet, the set $\mathcal{P}$ of processes is the set of nondeterministic FSMs $\langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ with $\Sigma \subseteq \hat{\Sigma}$, and the set Φ of properties is the set of prefix-closed languages or the set of FSMs over the set Ω of property events. Composition $\|$ is standard lock-step synchronisation (Definition 3.3), and property satisfaction is defined by (4.4), or more precisely $G \models E$ if and only if $P_\Omega(\mathcal{L}(G)) \subseteq \mathcal{L}(E)$.

It remains to determine an appropriate process equivalence and hiding operation. It is clear from (4.5) that it is sufficient if the language of components is preserved by abstraction. As the equivalence relation must also be a congruence with respect to synchronous composition (CV1), the coarsest feasible equivalence for safety properties is *language equivalence*,

$$G_1 \simeq_\mathcal{L} G_2 \qquad \text{if and only if} \qquad \mathcal{L}(G_1) = \mathcal{L}(G_2)\ . \tag{4.6}$$

If (CV1) did not need to be satisfied, it would be enough to use $P_\Omega(\mathcal{L}(G_1)) = P_\Omega(\mathcal{L}(G_2))$ instead of $\mathcal{L}(G_1) = \mathcal{L}(G_2)$ in (4.6).

Hiding $G \setminus \Upsilon$ must be restricted to non-property events, i.e., $\Upsilon \cap \Omega = \emptyset$, and also needs to preserve languages. This is ensured by the standard replacement of hidden events in Υ with the silent event τ, and by natural projection. It is clear that language equivalence, hiding, and natural projection are congruences with respect to synchronous composition, and conditions (CV1)–(CV4) are satisfied.

This framework has been implemented in several variations to verify safety properties compositionally. Aziz et al. (1994) compute abstractions based on bisimulation (Definition 3.14). As bisimulation implies language equivalence (4.6), this is a valid realisation of the above framework. While bisimulation allows for efficient computation of abstractions, it is much stronger than language equivalence and the reduction in state numbers by abstraction is limited.

Cheung and Kramer (1999) verify safety properties compositionally using weak bisimulation (Definition 3.15) as abstraction. This works because weak bisimulation also implies language equivalence. While the time complexity to compute abstractions is higher, the potential for state reduction is greatly increased, particularly for large systems where many events are hidden.

Yet, weak bisimulation remains a stronger equivalence than needed for safety properties. Flordal and Malik (2009) transform controllability verification problems into equivalent nonblocking verification problems, and then perform a compositional nonblocking check

to verify this safety property. This makes it possible to use conflict-preserving abstraction rules (described in more detail in Section 4.4 below) to achieve simplification beyond weak bisimulation.

As yet another approach, Ware and Malik (2008) use language equivalence (4.6) directly as their abstraction. After hiding local events, they use the *subset construction* algorithm (Hopcroft et al. 2001) to compute a language-equivalent deterministic FSM, which is then minimised to obtain the deterministic FSM with the fewest states possible that accepts the same language as the FSM before abstraction. This approach uses the most general equivalence possible for safety properties and therefore has the strongest potential for state space reduction. However, the subset construction algorithm has exponential time complexity, $O(2^n)$. And unlike bisimulation and weak bisimulation, which guarantee that the result of abstraction is smaller or equal to the FSM before abstraction, the smallest language-equivalent deterministic FSM may be exponentially larger than the original.

To avoid the exponential complexity, Ware and Malik (2008) limit the number of states that can be constructed during abstraction. If the number of states exceeds a set limit while computing an abstraction, that computation is aborted, and another set of FSMs is composed instead. If no more abstractions are feasible, compositional minimisation stops early and defers to monolithic verification. The experiments of Ware and Malik (2008) suggest that this pragmatic approach works because the exponential worst-case of subset construction is rare in practice.

One concern when verifying safety properties with all these methods is the potentially large number of property events in Ω. Events in Ω cannot be removed and must be retained in all abstraction steps, reducing the effectiveness of compositional abstraction. To mitigate the problem, Cheung and Kramer (1999) express safety properties using a *trap* or *dump* state as shown in Fig. 5. Here, the language inclusion property E is transformed into a component $G_\perp$, such that checking whether event put can occur in state 2 becomes equivalent to checking whether the trap state $\perp$ can be reached when the system G is composed with $G_\perp$. This can be expressed using a property φ with only one symbol, such as $AG\,\neg\perp$ in CTL*, allowing all local events to be hidden within the system G and the transformed property $G_\perp$.

Ware and Malik (2008) extend this approach for language-based specifications. This is also shown in Fig. 5. The critical transition with event put to the dump state has been relabelled using a new event $\mathsf{put}_\perp$. At the same time, a parallel transition labelled with this event $\mathsf{put}_\perp$ is added to all transitions labelled put that appear in the system model G outside of $G_\perp$. As a result, the combined system $G \| G_\perp$ can execute $\mathsf{put}_\perp$ when put is possible in G but not allowed by the property E. Therefore, the system G satisfies the property E if and only if $\mathsf{put}_\perp$ is never enabled in $G \| G_\perp$. This can be verified with a different language inclusion check, checking whether

$$\mathcal{L}(G \| G_\perp) \subseteq \mathcal{L}(E_\perp)\ , \tag{4.7}$$

where $E_\perp = \langle \{\mathsf{put}_\perp\}, \{0\}, \emptyset, \{0\} \rangle$ is a property that always disables $\mathsf{put}_\perp$, represented as a one-state FSM without any transitions and with $\mathsf{put}_\perp$ in the alphabet.

The above approach to verify language inclusion can be adapted to verify controllability. Recall that a specification E is controllable with respect to plant G if all *uncontrollable* events enabled in G are also enabled in E. This can be considered as a relaxed language inclusion check, where only uncontrollable events can cause the property to fail. In practice there often are several uncontrollable events distributed over a composed specification $E = E_1 \| \cdots \| E_m$. In this case, the controllability check can be done separately for each uncontrollable event μ, and each time the above transformation can be used to replace the relevant specification

components by plants and then check whether an event $\mu_\perp$ is enabled (Ware and Malik 2008).

While the compositional approach works for safety properties, it is not the only way to verify them. Safety properties have a modularity property according to which a composed system G satisfies a safety property if some subsystem of G satisfies the same property (Bérard et al. 2001). This principle makes it possible to form abstractions by selecting groups of components, which is often simpler than compositional abstraction. Clever selection strategies based on counterexamples often make it possible to verify controllability and language inclusion by considering only a small part of the system (Åkesson et al. 2002; Brandin et al. 2004). Yet, there are cases where a property check can only be completed by considering a large subsystem, and in such cases compositional abstraction can help by simplifying that subsystem (Ware 2007).

4.3 Absence of deadlock

A *deadlock* is a situation where two or more processes are stuck in a state waiting for each other indefinitely (Tanenbaum 1992). Deadlocks have been studied extensively by Hoare (1985) in the language of *Communicating Sequential Processes (CSP)*, and have been used as liveness criteria in supervisory control with FSMs and Petri nets (Li 1997; Li et al. 2008). In an FSM, a *deadlock state* is a state without any outgoing transition, and an FSM that contains a reachable deadlock state is said to have a deadlock. Otherwise, if there are no reachable deadlock states, the FSM is *deadlock-free*. This absence of deadlocks can be expressed in CTL* as

$$AG\,EX\,true\,. \tag{4.8}$$

$EX\,true$ describes a state with at least one next state, so this formula means that every reachable state has at least one successor state.

A compositional framework for the absence of deadlocks can be defined with a single property. The set of properties is simply $\Phi = \{\mathrm{dlf}\}$, where $G \models \mathrm{dlf}$ means that an FSM G is deadlock-free. As this property does not depend on any events, all local events can be hidden during compositional verification. Thus, there is no target event set Ω as was needed in the previous sections.

The main question for compositional verification of the absence of deadlock is about the process equivalence. Unlike safety properties, the language of a nondeterministic FSM is not enough to capture its potential for deadlock. Instead, the more refined *failures* model (Hoare 1985) is used.

Definition 4.1 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM. The set of *stable failures* or simply *failures* of G is

$$\begin{aligned}\mathcal{F}(G) = \{\, (s, F) \in \Sigma^* \times 2^\Sigma \mid {} & \text{there exists } x \in Q \text{ such that } G \stackrel{s}{\Rightarrow} x \stackrel{\tau}{\nrightarrow} \\ & \text{and there does not exist } \sigma \in F \text{ such that } x \stackrel{\sigma}{\rightarrow} \}\,.\end{aligned} \tag{4.9}$$

The set of failures contains pairs consisting of a trace $s \in \mathcal{L}(G)$ and a so-called *refusal* F. The trace s takes G to a state x without outgoing τ-transitions, where it will block when synchronised with another component that can only execute events in F. This implies that refusals are closed under set inclusion, i.e., if $(s, F) \in \mathcal{F}(G)$ and $F' \subseteq F$ then also $(s, F') \in \mathcal{F}(G)$. Therefore, the failures model can also be defined using maximal refusals.

It is clear that an FSM G with alphabet Σ has a deadlock state if and only if $(s, \Sigma) \in \mathcal{F}(G)$ for some $s \in \Sigma^*$. It is also known (Hoare 1985) that the failures of a process resulting from

synchronous composition or hiding can be obtained from the failures of its constituents:

$$\mathcal{F}(G_1 \| G_2) = \{\, (s, F_1 \cup F_2) \mid (s, F_1) \in \mathcal{F}(G_1) \text{ and } (s, F_2) \in \mathcal{F}(G_2) \,\} ; \qquad (4.10)$$

$$\mathcal{F}(G \setminus \Upsilon) = \{\, (P_{\Sigma\setminus\Upsilon}(s), F) \mid (s, F) \in \mathcal{F}(G) \text{ and } F \cap \Upsilon = \emptyset \,\} . \qquad (4.11)$$

Definition 4.2 Let G and H be two FSMs. G and H are said to be *failures equivalent*, written $G \simeq_{\mathcal{F}} H$, if $\mathcal{F}(G) = \mathcal{F}(H)$.

It follows from the above observations that failures equivalence is a congruence with respect to synchronous composition and hiding that preserves the existence or absence of deadlocks. A compositional framework

$$\mathbf{CF} = \langle \hat{\Sigma}, \mathcal{P}, \{\mathrm{dlf}\}; \simeq_{\mathcal{F}}, \|, \setminus, \models \rangle \qquad (4.12)$$

satisfies conditions (CV1)–(CV4). This means that the absence of deadlocks in a synchronous composition of nondeterministic FSMs can be verified compositionally using standard synchronous composition and process-algebraic hiding of local events, and any abstraction that preserves the failures of an FSM.

For practical implementation of abstractions using the failures model, Roscoe et al. (1995) use *generalised labelled transitions systems* instead of FSMs.

Definition 4.3 A *generalised labelled transition systems (GLTS)* is a structure $G = \langle \Sigma, Q, \rightarrow, Q^\circ, \mathrm{minaccs} \rangle$ where $\langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is an FSM and minaccs is a map

$$\mathrm{minaccs}\colon Q \rightarrow 2^{2^{\Sigma}} . \qquad (4.13)$$

The function minaccs assigns to each state a set of *minimal acceptances*. Given an FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$, a set $M \subseteq \Sigma$ is an *acceptance* of state $x \in Q$, written $M \in \mathrm{accs}(x)$, if either $x \stackrel{\tau}{\not\rightarrow}$ and M consists of the events enabled at x, i.e., $M = \{\, \sigma \in \Sigma \mid x \stackrel{\sigma}{\rightarrow} \,\}$, or there exists a state $y \in Q \setminus \{x\}$ such that $x \stackrel{\varepsilon}{\Rightarrow} y \stackrel{\tau}{\not\rightarrow}$ and $M \in \mathrm{accs}(y)$. An acceptance $M \in \mathrm{accs}(x)$ is a *minimal acceptance* of x, written $M \in \mathrm{minaccs}(x)$, if there is no strict subset of M that also is an acceptance of x. Acceptance sets of reachable states are the complements of refusals, i.e., if $M \in \mathrm{accs}(x)$ then there exists $s \in \mathcal{L}(G)$ such that $G \stackrel{s}{\Rightarrow} x$ and $(s, \Sigma \setminus M) \in \mathcal{F}(G)$. Noting that the subsets of refusals are again refusals, it follows that the failures $\mathcal{F}(G)$ of an FSM can be reconstructed from its language and the minimal acceptances of its reachable states.

To check whether a composed system (2.1) has a deadlock, Roscoe et al. (1995) first convert each component FSM to a GLTS, and then perform compositional abstraction with the GLTS. Composition and hiding are defined for GLTS based on (4.10) and (4.11). Further, the following GLTS abstractions preserve failures equivalence.

Normalisation. Roscoe (1994) describes an operation of *normalisation* that transforms a GLTS into a failures equivalent form by converting the FSM structure of the GLTS into an equivalent deterministic FSM using the *subset construction* algorithm (Hopcroft et al. 2001), and assigning to each subset state the minimal acceptances of its constituent states. Normalisation removes all τ-transitions and often reduces the number of states, but the latter cannot be guaranteed. As with subset construction, the worst case for the number of states in the abstraction is exponential in the number of states in the original GLTS.

Diamond elimination. Roscoe et al. (1995) describe the abstraction of *diamond elimination*, which also eliminates τ-transitions but avoids subset construction. States are

assigned additional outgoing transitions and minimal acceptances from all other states reachable by sequences of τ-transitions, and afterwards all τ-transitions are removed. The resultant GLTS may be nondeterministic due to branching non-τ transitions. Diamond elimination is guaranteed to result in at most the same number of states as the original GLTS.

Bisimulation. As bisimulation (Definition 3.14) preserves all CTL* properties, it also preserves the existence of deadlocks, and it preserves failures equivalence of GLTS provided that only states with the same minimal acceptances are considered bisimilar. A minimal bisimilar GLTS can be computed with the same algorithm as for FSMs (Fernandez 1990).

The FDR model checker (Roscoe et al. 1995) uses these ideas to verify the absence of deadlocks compositionally. After composition and hiding, the τ-transitions of GLTS components are removed by normalisation or diamond elimination, and the result is minimised based on bisimulation. The original system (2.1) is determined to have a deadlock if and only if the final compositional abstraction has an empty minimal acceptance.

4.4 The nonblocking property

The nonblocking property is commonly used in supervisory control to express liveness. An FSM is *nonblocking* if it can always reach some state that belongs to the designated set of *marked* or *accepting* states (Definition 3.9). By requiring the reachability of a marked state, the nonblocking property does not only rule out most deadlocks but also most *livelocks*. A deadlock occurs when a system gets stuck in a state with no transitions enabled, and a livelock occurs when a system continues to execute without ever completing its task (Tanenbaum 1992).

In CTL*, the nonblocking property is expressed as

$$AG\,EF\,marked \tag{4.14}$$

where "$marked$" is a proposition that is true precisely when the system is in an accepting state. The use of "exists finally" (EF) means that there always exists a path to an accepting state, but it is not guaranteed that such a path will be taken. Therefore, this property is weaker than the liveness properties commonly used in computing, where it is required that termination occurs eventually. While a nonblocking system retains the possibility to achieve termination, it may also cycle indefinitely without terminating. Also, the nonblocking property allows termination in an accepting state, thus not completely ruling out deadlock.

The nonblocking property is popular in supervisory control, where a supervisor is a safety device that prevents the system from entering unsafe states. This includes the prevention of states from where termination is impossible, but it does not enforce termination. For finite-state systems, termination is guaranteed under an additional assumption of *strong fairness* (Arnold 1994). If it is assumed that every transition that gets enabled indefinitely often will occur eventually, then a finite-state nonblocking system is guaranteed to reach an accepting state eventually.

A compositional framework to verify the nonblocking property can be defined in a similar way as the framework for deadlocks in Section 4.3. There is just one property, $\Phi = \{\text{nbl}\}$, where $G \models \text{nbl}$ means that an FSM G is nonblocking. The set of property events is $\Sigma_{\text{nbl}} = \{\omega\}$, so all local events can be hidden except for the termination event ω.

Once again, the main question for compositional verification is the search for an appropriate process equivalence. Malik et al. (2006) introduce *conflict equivalence*, which describes the required condition in an abstract and general way.

Definition 4.4 (Malik et al. 2006) Let G and H be two FSMs. G and H are said to be *conflict equivalent*, written $G \simeq_{\text{conf}} H$, if for every FSM T it holds that $G \| T$ is nonblocking if and only if $H \| T$ is nonblocking.

The idea of conflict equivalence is derived from process-algebraic *testing theory* (De Nicola and Hennessy 1984), which defines equivalences relating processes based on the results of *tests*. Two processes are considered as equivalent if the results of all tests are equal. In Definition 4.4, the FSMs G and H are considered as processes under test, T is a test, and the test result is the observation whether or not the test is nonblocking in composition with the process under test. Alternatively, T can be viewed as the unknown remainder of the system (2.1) or the part not subject to abstraction, $T = G_2 \| \cdots \| G_n$. Then Definition 4.4 ensures that the nonblocking property is preserved in every possible context.

Malik et al. (2006) show that conflict equivalence is a congruence with respect to synchronous composition and hiding while also preserving the nonblocking property, and it is the coarsest process equivalence with these properties. It is also clear that the nonblocking property is preserved by process-algebraic hiding. Therefore, a compositional framework

$$\mathbf{CF} = \langle \hat{\Sigma}, \mathcal{P}, \{\text{nbl}\}; \simeq_{\text{conf}}, \|, \setminus, \models \rangle$$

satisfies conditions (CV1)–(CV4).

While conflict equivalence is the coarsest possible equivalence for use in compositional verification of the nonblocking property, it is not immediately clear how to compute abstractions that preserve this relation. Unlike bisimulation or language equivalence, conflict equivalence does not come with readily accessible minimisation algorithms. Because of the popularity of the nonblocking property in supervisory control, much research has been devoted to finding means to simplify FSMs while preserving the nonblocking property of the global system. The following subsections describe some of these methods.

4.4.1 Weak bisimulation and variants

As the nonblocking property can be expressed in temporal logic, it is clear from results about model checking (Baier and Katoen 2008) that bisimulation (Definition 3.14) preserves the nonblocking property and also conflict equivalence. Moreover, since the nonblocking property does not imply strong liveness—its CTL* formula (4.14) only uses the EF connective—the property is insensitive to the presence of cycles of τ-transitions. Therefore, weak bisimulation (Definition 3.15) also preserves conflict equivalence. Weak bisimulation has been used frequently in compositional nonblocking verification, and accounts for a large part of state space reduction (Flordal and Malik 2009).

As explained in Section 3.3, the bisimulation algorithm (Fernandez 1990) can be used to compute a weak bisimulation $\sim$ on an FSM G, and then an abstracted FSM is obtained as an FSM quotient. However, the computation depends on the extended transition relation $\Rightarrow$, which can be much larger than the explicit transition relation $\rightarrow$, increasing the time complexity to $O(n^3)$.

If the extended transition relation $\Rightarrow$ is computed, it can replace the original transition relation $\rightarrow$. Unfortunately this approach, also known as *saturation*, can cause a substantial increase in the number of transitions. Eloranta (1991) proposes to minimise the number of

transitions while preserving weak bisimulation instead, removing explicit transitions that are implied by the relation $\Rightarrow$.

Example 4.2 FSMs G and H in Fig. 6 are weakly bisimilar. For every two states x and y, the relations $x \stackrel{\varepsilon}{\Rightarrow} y$ and $x \stackrel{a}{\Rightarrow} y$ and $x \stackrel{\omega}{\Rightarrow} y$ are equivalent between G and H. The FSM G is saturated in the sense that it contains an explicit transition $x \stackrel{a}{\to} y$ whenever $x \stackrel{a}{\Rightarrow} y$ holds, and all states with $x \stackrel{\omega}{\Rightarrow}$ are accepting states. The FSM H has minimal sets of transitions and accepting states while achieving the same relation $\Rightarrow$.

Both saturation and minimisation of the transition relation are useful during compositional minimisation. Saturation makes it easier to explore transitions and determine whether $x \stackrel{\sigma}{\Rightarrow} y$ holds for given states, but increases the memory needed to store transitions. Minimisation reduces the memory requirements and facilitates transition-based abstractions such as those considered in Section 4.4.2 below, but makes it more difficult to compute a weak bisimulation relation or other abstractions that depend on the extended transition relation $\Rightarrow$.

Su et al. (2010c) propose a modification to weak bisimulation that makes it possible to remove more states. Their abstraction, called *weak observation equivalence*, can be described as a relation similar to weak bisimulation.

Definition 4.5 Let $G_1 = \langle \Sigma_1, Q_1, \to_1, Q_1^\circ \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \to_2, Q_2^\circ \rangle$ be two FSMs with equal event sets $\Sigma_1 = \Sigma_2$. A relation $\sim_w \subseteq Q_1 \times Q_2$ is a *weak observation equivalence* between G_1 and G_2, if for all $x_1 \in Q_1$ and $x_2 \in Q_2$ and all $\sigma \in \Sigma_1 \cup \{\omega\}$ such that $x_1 \sim_w x_2$, the following conditions hold:

- If $x_1 \stackrel{\sigma}{\Rightarrow}_1 y_1$ for some $y_1 \in Q_1$, then there exists $y_2 \in Q_2$ such that $x_2 \stackrel{\sigma}{\Rightarrow}_2 y_2$ and $y_1 \sim_w y_2$.
- If $x_2 \stackrel{\sigma}{\Rightarrow}_2 y_2$ for some $y_2 \in Q_2$, then there exists $y_1 \in Q_1$ such that $x_1 \stackrel{\sigma}{\Rightarrow}_1 y_1$ and $y_1 \sim_w y_2$.

G_1 and G_2 are *weakly observation equivalent*, written $G_1 \sim_w G_2$, if there exists a weak observation equivalence $\sim_w$ between G_1 and G_2, such that for every initial state $x_1^\circ \in Q_1^\circ$ there exists $x_2 \in Q_2$ such that $Q_2^\circ \stackrel{\varepsilon}{\Rightarrow}_2 x_2$ and $x_1^\circ \sim_w x_2$, and vice versa.

The difference between this definition and weak bisimulation (Definition 3.15) is the restriction to $\sigma \in \Sigma \cup \{\omega\}$, i.e., not including τ. While two weakly bisimilar states with outgoing τ-transitions must be able to reach equivalent states via sequences of τ-transitions, weak observation equivalence only considers transition sequences that include a non-τ event. A coarsest weak observation equivalence relation can be computed using the bisimulation algorithm, but due to the exclusion of τ-transitions, a different quotient is needed to construct a reduced FSM.

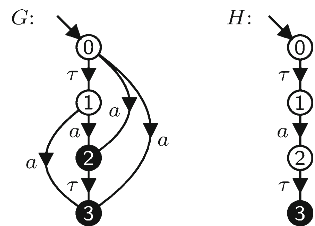


Fig. 6 Saturation and minimisation of transitions while preserving weak bisimulation

Definition 4.6 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim \subseteq Q \times Q$ be an equivalence relation. The *weak observation equivalence quotient* is $G/_{\mathrm{w}}\sim = \langle \Sigma, Q/\sim, \rightarrow_{\mathrm{w}}, Q^\circ_{\mathrm{w}} \rangle$ where $[x] \xrightarrow{\sigma}_{\mathrm{w}} [y]$ if $\sigma \in \Sigma \cup \{\omega\}$ and $x \overset{\sigma}{\Rightarrow} y$, and $Q^\circ_{\mathrm{w}} = \{\, [x] \in Q/\sim \mid Q^\circ \overset{\varepsilon}{\Rightarrow} x \,\}$.

Proposition 4.1 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim_{\mathrm{w}} \subseteq Q \times Q$ be a weak observation equivalence on G. Then $G \sim_{\mathrm{w}} (G/_{\mathrm{w}}\sim_{\mathrm{w}})$.*

Example 4.3 FSMs G and H in Fig. 7 are weakly observation equivalent. For example, considering states 1, it holds that $1 \overset{a}{\Rightarrow} 3$ and $1 \overset{b}{\Rightarrow} 3$ in both G and H. A relation $\sim_{\mathrm{w}}$ with $x \sim_{\mathrm{w}} x$ for $x \in \{0, 1, 2, 3\}$ is a weak observation equivalence between G and H, and $(G/_{\mathrm{w}}\sim_{\mathrm{w}}) = H$. While the number of states or transitions does not change in this case, an FSM that contains G and H as substructures can be reduced effectively using weak observation equivalence. It is worth noting that G and H are not weakly bisimilar because $1 \overset{\varepsilon}{\Rightarrow} 2$ in G, and no state equivalent to 2 can be reached by τ-transitions from state 1 in H.

Two weakly observation equivalent FSMs are also conflict equivalent (Malik and Leduc 2013), so weak observation equivalence can be used as abstraction in compositional nonblocking verification. While a weak observation equivalence relation is easier to compute than a weak bisimulation, the elimination of τ-transitions by the weak observation equivalence quotient can increase the number of transitions with other events substantially. A more careful quotient construction can limit the increase, as elimination of τ-transitions is only needed for states where weak bisimulation and weak observation equivalence differ. Su et al. (2010c) implement compositional nonblocking verification with weak observation equivalence as the only abstraction and use it to verify large discrete event systems.

Considering another variant of weak bisimulation, Pena et al. (2009) perform compositional nonblocking verification using deterministic FSMs and natural projection. Unlike hiding, natural projection does not preserve the nonblocking property, so that not all local events can be projected out in compositional nonblocking verification. The search for a projection that preserves conflict equivalence leads to the *observer property* (Wong and Wonham 1996).

Definition 4.7 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic FSM, and let $\Omega \subseteq \Sigma$. The natural projection $P_\Omega \colon \Sigma^* \rightarrow \Omega^*$ is said to have the *observer property (OP)* for G, if for all $s \in \mathcal{L}(G)$ and all $t \in \Omega^*$ such that $P_\Omega(s)t \in P_\Omega(\mathcal{L}(G))$, there exists $u \in \Sigma^*$ such that $P_\Omega(su) = P_\Omega(s)t$ and $su \in \mathcal{L}(G)$.

That is, if after some trace s the behaviour of G can be continued with a trace whose projection is t, then after every trace with equal projection as s the behaviour of G can be continued with some trace whose projection also is t. The observer property appears in a few variations in the literature. Definition 4.7 uses the prefix-closed language $\mathcal{L}(G)$ instead

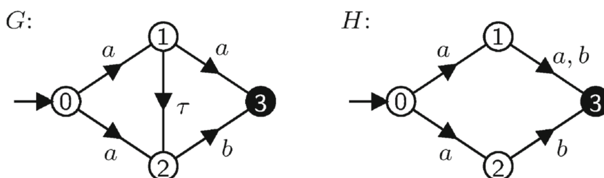


Fig. 7 Two weakly observation equivalent FSMs (Malik and Leduc 2013)

of the accepting language, ensuring that conflict equivalence is also preserved for a blocking FSM G.

Wong and Wonham (1996) show that, if a projection has the observer property, then the deterministic FSM resulting from the projection is weakly bisimilar to the original FSM. This ensures that the number of states cannot be increased, ruling out the exponential worst case of subset construction, and it also implies that conflict equivalence is preserved. Malik et al. (2007) show that the observer property is the weakest condition that can be imposed on natural projection if conflict equivalence is to be preserved.

It is easy to determine whether a given projection has the observer property (Wong and Wonham 1996; Pena et al. 2014). But sometimes hiding all local events does not ensure the observer property, in which case more complicated search algorithms (Schmidt and Moor 2006; Feng and Wonham 2010; Pena et al. 2010) are needed to determine an appropriate target event set Ω. The observer property is important when working with deterministic FSMs, but the use of nondeterministic FSMs allows for better abstraction at lower computational cost (Malik et al. 2007).

4.4.2 Transition-based abstraction rules

Conflict equivalence is a weaker relation than weak bisimulation or weak observation equivalence, and this implies the possibility to perform simplification beyond what has been described in the previous section. In the absence of general minimisation algorithms, Flordal and Malik (2009) propose a collection of abstraction rules that identify and simplify specific configurations of transitions. Most of these rules are concerned with τ-transitions, and another recurring concept is that of incoming equivalent states.

Definition 4.8 (Flordal and Malik 2009) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM. Two states $y_1, y_2 \in Q$ are *incoming equivalent*, written $y_1 \sim_{\text{inc}} y_2$, if

- $Q^\circ \stackrel{\varepsilon}{\Rightarrow} y_1$ if and only if $Q^\circ \stackrel{\varepsilon}{\Rightarrow} y_2$.
- For all $\sigma \in \Sigma$ and all $x \in Q$, it holds that $x \stackrel{\sigma}{\Rightarrow} y_1$ if and only if $x \stackrel{\sigma}{\Rightarrow} y_2$.

Two states are incoming equivalent if they both can be reached from the same states with the same non-τ events. This is a stronger requirement than weak observation equivalence which would imply reachability from equivalent rather than the same states. One way how incoming equivalent states arise is through nondeterministic branching: if some state has multiple successors reached by the same event, which are not also reached from other states, then these successors are incoming equivalent.

This idea is used by the *Active Events Rule*, which merges incoming equivalent states that also have the same active, or enabled, events. The idea is that, in order to preserve blocking, only the strings leading to accepting states are important. Therefore, if a nondeterministic choice leads to states with the same enabled events, the nondeterministic choice can be postponed by one step.

Active Events Rule (Flordal and Malik 2009) If two states are incoming equivalent and have the same non-τ events enabled, then they are conflict equivalent and can be merged. More precisely, if $x \sim_{\text{inc}} y$ and for all $\sigma \in \Sigma \cup \{\omega\}$ it holds that $x \stackrel{\sigma}{\Rightarrow}$ if and only if $y \stackrel{\sigma}{\Rightarrow}$, then x and y can be merged into a single state.

That is, two incoming equivalent states can be merged if they both allow continuations via the same events, possibly preceded with τ-transitions. As the termination event ω is

included, the two states must also have the same accepting state status—or more precisely either both or none must be able to reach an accepting state via a possibly empty sequence of τ-transitions.

Example 4.4 In Fig. 8, states 1 and 2 in G have incoming transitions from state 0 associated with event a and from state 1 associated with b, which establishes incoming equivalence. Furthermore, they both have the same active event b. Therefore, the Active Events Rule can be applied, states 1 and 2 are conflict equivalent and can be collapsed into a single state 12 as shown in H.

Another way how incoming equivalent states arise is through τ-transitions. The source and target states of a τ-transition are incoming equivalent provided that there are no other transitions to the target state. This is used by the next rule, the *Silent Continuation Rule*, which is the first of several abstraction rules that seek to collapse sequences of τ-transitions.

Silent Continuation Rule (Flordal and Malik 2009) Two states that are incoming equivalent and from which *stable* states, i.e., states without outgoing τ-transitions, can be reached via a nonempty sequence of τ-transitions, are conflict equivalent and can be merged into a single state.

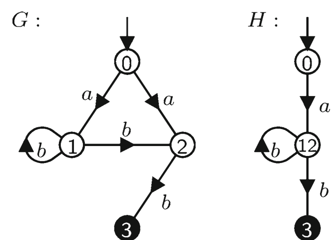
Example 4.5 In Fig. 9, states 0 and 1 in G are both considered initial since they can be reached silently from the initial state 0. This makes them incoming equivalent in this case, since neither state is reachable by any event other than τ. Moreover, both states can, by executing at least one silent transition, reach the stable state 3, which has no outgoing τ transitions. Thus, by the Silent Continuation Rule, states 0 and 1 in G are conflict equivalent and can be collapsed into state 01 as shown in H.

There are two further abstraction rules to reduce states with τ-transitions. The *Only Silent Incoming Rule* and *Only Silent Outgoing Rule* both apply to states with outgoing τ-transitions, in the first case seeking to merge them into predecessor states and in the second case into successor states. Both rules assume a *τ-loop free* FSM, i.e., an FSM without any cycles of τ-transitions. As all states on a cycle of τ-transitions are weakly bisimilar, such cycles can first be collapsed into a single state while preserving weak bisimulation and thus conflict equivalence.

Only Silent Incoming Rule (Flordal and Malik 2009) In a τ-loop free FSM, let x be a state with an outgoing τ-transition, for which all incoming transitions are τ-transitions. This state x can be removed if its outgoing transitions are copied to all its predecessor states.

Example 4.6 State 3 in G in Fig. 10 has only τ-transitions incoming and an outgoing τ-transition to state 0. This state can be removed while adding both its outgoing transitions with a and τ to its predecessor states 1 and 2, resulting in H.

Fig. 8 Example application of the Active Events Rule (Flordal and Malik 2009)



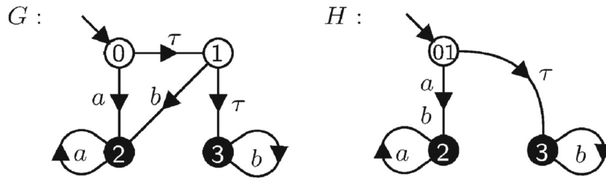


Fig. 9 Example application of the Silent Continuation Rule (Flordal and Malik 2009)

Only Silent Outgoing Rule (Flordal and Malik 2009) In a τ-loop free FSM, let x be a state whose outgoing transitions are all τ-transitions. This state x can be removed if its incoming transitions are redirected to all its successor states.

Example 4.7 State 1 in G in Fig. 11 has only τ-transitions outgoing. This state can be removed after redirecting its two incoming transitions to both the successor states 2 and 3, resulting in H.

By combining the four abstraction rules in this section, Flordal and Malik (2009) can collapse many sequences of τ-transitions into a single transition. This often reduces the number of states, while limiting the increase in the number of transitions that would result from saturation. The effect is similar to diamond elimination (see Section 4.3), but with conflict equivalence being more complicated, the abstractions are less systematic and only work under specific conditions.

Malik and Leduc (2013) propose a similar set of abstraction rules, which is derived differently based on the generalised nonblocking property. Malik (2015) relaxes some of the conditions for the above abstraction rules to make them applicable under more circumstances, particularly in FSMs that have selfloop transitions with events other than τ. Pilbrow and Malik (2015) propose yet another way to relax the conditions by not only considering local events but also taking into account *special events* that are always enabled or selfloop-only in the FSMs not currently being simplified.

4.4.3 Certain conflicts

The possibility for an FSM to include blocking states, i.e., states from where it is impossible to reach an accepting state, gives rise to a different kind of abstraction rules specifically for compositional nonblocking verification. Every FSM can be associated with a language of *certain conflicts* (Malik 2004), which contains all traces that, when executed by the FSM in any context, necessarily lead to a blocking situation. Traces that lead to blocking states are

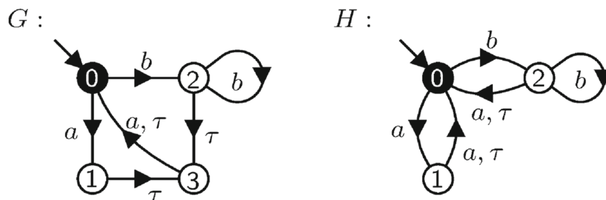


Fig. 10 Example application of the Only Silent Incoming Rule (Flordal and Malik 2009).

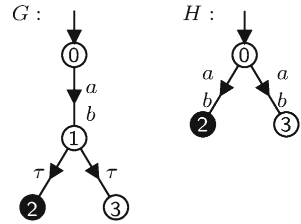


Fig. 11 Example application of the Only Silent Outgoing Rule (Flordal and Malik 2009).

traces of certain conflicts, and the following example shows that other traces can have this property as well.

Example 4.8 Consider the FSM G in Fig. 12. Trace ab is a trace of certain conflicts because it leads to the blocking state 2. Then a also is a trace of certain conflicts: after execution of a, the FSM can enter state 1, from where the only way to reach an accepting state is to execute b, and enablement of ab entails the potential to block by entering state 2. Therefore, G is blocking in composition with any FSM that enables ab or a, i.e., these are traces of certain conflicts of G.

For nonblocking verification, it is enough to determine that a blocking state can be reached, and there is no need to differentiate details of the blocking behaviour. Therefore, it is enough to identify shortest traces of certain conflicts, and represent all certain conflicts behaviour with a blocking state.

Malik (2010) describes an algorithm that effectively computes the language of certain conflicts of a given FSM and modifies the FSM in such a way that all traces of certain conflicts lead to a single blocking state. Unfortunately, the number of states of the FSM representation of the language of certain conflicts is in the worst case exponential in the number of states of the original FSM, possibly resulting in a much bigger abstracted FSM. An implementation of this abstraction by Lindsey (2012) fails to achieve faster compositional nonblocking verification.

Certain conflicts can improve compositional nonblocking verification if used in a more pragmatic way. Flordal and Malik (2006) introduce a couple of simple rules that identify and merge *states* of certain conflicts without computing the full language of certain conflicts. It is clear that blocking states are certainly conflicting. In addition to that, states with outgoing τ-transitions to a certainly conflicting state, and transitions with nondeterministic branches to a certainly conflicting state as in Example 4.8 are certainly conflicting. Malik and Ware (2020) combine these conditions in the *Limited Certain Conflicts Rule*.

Limited Certain Conflicts Rule An FSM G can be replaced by the limited certain conflicts abstraction $\mathrm{LCC}(G)$ in Definition 4.9, which is conflict equivalent to G.

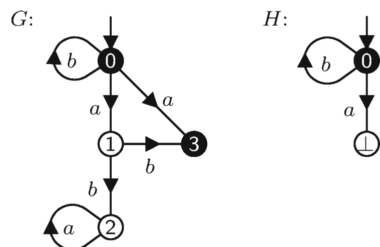


Fig. 12 Example application of the Limited Certain Conflicts Rule (Malik and Ware 2020).

Definition 4.9 (Malik and Ware 2020) Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an FSM. Define sets of *limited certain conflict states* inductively:

$$\mathrm{lcc}_G^0 = \{\, x \in Q \mid \text{ there does not exist } t \in (\Sigma \cup \{\tau\})^* \text{ such that } x \xrightarrow{t\omega} \,\}\,; \tag{4.15}$$

$$\mathrm{lcc}_G^{i+1} = \{\, x \in Q \mid \text{ for every path } x = x_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\omega} \text{ there exists } j \geq 0 \text{ such that } j \leq k \text{ and } x_j \overset{\varepsilon}{\Rightarrow} \mathrm{lcc}_G^i, \text{ or } j < k \text{ and } x_j \overset{\sigma_{j+1}}{\Rightarrow} \mathrm{lcc}_G^i \,\}\,; \tag{4.16}$$

$$\mathrm{lcc}_G = \bigcup_{i \geq 0} \mathrm{lcc}_G^i \,. \tag{4.17}$$

The *limited certain conflicts abstraction* of G is $\mathrm{L{\small CC}}(G) = \langle \Sigma, Q_{\mathrm{lcc}}, \to_{\mathrm{lcc}}, Q^\circ_{\mathrm{lcc}} \rangle$ where $Q_{\mathrm{lcc}} = (Q \setminus \mathrm{lcc}_G) \cup \{\bot\}$ (with $\bot \notin Q$); $x \xrightarrow{\sigma}_{\mathrm{lcc}} y$ if $x, y \neq \bot$ and $x \xrightarrow{\sigma} y$ and $x \overset{P(\sigma)}{\Longrightarrow} \mathrm{lcc}_G$ does not hold, or $x \neq \bot = y$ and $x \xrightarrow{\sigma} \mathrm{lcc}_G$; and $Q^\circ_{\mathrm{lcc}} = Q^\circ$ if $Q^\circ \cap \mathrm{lcc}_G = \emptyset$ and $Q^\circ_{\mathrm{lcc}} = \{\bot\}$ otherwise.

The set lcc_G^0 of *level-0* limited certain conflict states is the set of blocking states (4.15). Level $i + 1$ adds to this states that can only reach accepting states by passing through a state that can reach a level-i limited certain conflict state using τ-transitions, or using a transition that may lead to a level-i state (4.16). These sets form an increasing sequence, $\mathrm{lcc}_G^0 \subseteq \mathrm{lcc}_G^1 \subseteq \cdots$, which in the finite-state case converges against the set lcc_G. The abstraction $\mathrm{L{\small CC}}(G)$ is constructed by merging these states into a new state $\bot$, and deleting some transitions.

Example 4.9 Consider again the FSM G in Fig. 12. It holds that $\mathrm{lcc}_G^0 = \{2\}$ and $\mathrm{lcc}_G = \mathrm{lcc}_G^i = \{1, 2\}$ for $i \geq 1$. This results in the limited certain conflicts abstraction $\mathrm{L{\small CC}}(G) = H$ (the unreachable state 3 is not shown in the figure).

The *Limited Certain Conflicts Rule* has been implemented in a few variations for compositional nonblocking verification. Flordal and Malik (2009) use it together with weak bisimulation and the transition-based abstractions in Section 4.4.2, and Pilbrow and Malik (2015) combine it with weak observation equivalence and a different set of transition-based abstractions. Experiments suggest that the amount of state space reduction from certain conflicts is significant, slightly less than what is achieved by weak bisimulation or weak observation equivalence, but often more than by transition-based abstractions.

4.4.4 Normal forms

A *normal form* is a unique form equivalent to the original FSM, which can be computed by a well-defined algorithm. For the previously discussed cases of safety properties and deadlocks, it is relatively easy to identify such normal forms and use them as abstractions. Normal forms are more difficult to establish for conflict equivalence, and this has resulted in the large proliferation of abstraction rules for compositional nonblocking verification shown in the previous sections.

In search for a more uniform approach, Ware and Malik (2012) perform compositional nonblocking verification using generalised labelled transition systems (GLTS, Definition 4.3). By eliminating τ-transitions and recording minimal acceptance sets instead, several abstraction rules can be formulated more concisely and more generally for GLTS. The resulting conflict-preserving abstraction rules are more comprehensive and more general, without achieving a normal form.

Ware and Malik (2013) describe a normal form for the *generalised nonblocking* property and use it for compositional verification. The generalised nonblocking property introduces a set of *precondition* states and requires that an accepting state be reachable from every reachable precondition state (Malik and Leduc 2008). This is weaker than the standard nonblocking property considered here, which requires that an accepting state be reachable from *every* reachable state. The size of the generalised nonblocking normal form is in the worst case exponential in that of the original FSM, but this worst case seems rare in practice. Ware and Malik (2013) use this normal form to verify the generalised nonblocking property compositionally, and as the generalised nonblocking property includes the standard nonblocking property as a special case, the method can also be used for the standard nonblocking property.

The normal form of Ware and Malik (2013) is a unique equivalent FSM with respect to generalised nonblocking equivalence but not with respect to conflict equivalence. Ware (2014) describes a more complicated normal form that defines a unique conflict equivalent FSM. This normal form is more difficult to compute and has not yet been used for compositional verification.

4.5 Counterexamples

The verification approach based on compositional abstraction is effective at determining whether or not a property is satisfied. If the property is not satisfied, it is also common for model checkers to produce a *counterexample* to explain the cause of the detected problem. Counterexamples are a crucial feature of model checking that greatly helps users to understand and fix faults. While counterexamples are routinely computed by standard model checking algorithms, the loss of information after abstraction makes this difficult for compositional methods.

For the case of FSM models, a counterexample is a trace that takes the system to *bad* state, i.e., an unsafe, deadlock, or blocking state depending on the property being checked. If the final result of compositional abstraction is analysed and found not to satisfy the property, then most model checking algorithm also produce a counterexample trace in addition to the answer that the property is not satisfied. But this trace only applies to the final compositional abstraction, and after several steps of hiding and abstraction, it is unlikely to provide a helpful explanation for the designer of the original system. To be helpful, the trace needs to *expanded* to produce a trace with events and transitions the designer is familiar with.

The process of counterexample expansion involves going back through all the abstraction steps and bringing back the information that was abstracted away. Starting with the last abstraction step, the counterexample for the result of compositional abstraction is modified so that it applies to the system before the last step, and this is repeated for each abstraction step until the original system is reached. Precisely how the counterexample is expanded at each step depends on the particular kind of abstraction performed.

For the language-preserving abstractions used when verifying safety properties, it is enough to find a trace that uses the same sequence of events as the counterexample for the abstract system, possibly with inserted τ-transitions. Such a trace can be found by searching through the FSM before abstraction while using the trace accepted by the abstracted system as guidance (Yeh and Young 1993; Ware and Malik 2008). As each of these expansion steps involves only a single component, the process is fairly efficient.

In general, every abstraction method needs to be analysed individually to determine how it affects counterexamples. Malik and Ware (2020) investigate counterexamples in compo-

sitional nonblocking verification and devise a criterion that covers abstractions based on weak bisimulation and most transition-based abstraction rules. For these rules, the counterexample can be expanded using an algorithm similar to the above for language-preserving abstraction, which only considers the component that has been abstracted. Yet there are also abstraction rules—most importantly those based on certain conflicts—where it is necessary to also consider the unchanged remainder of the system to decide how a counterexample is to be expanded, and this may result in substantial computational overhead.

5 Compositional synthesis

5.1 Plants and specifications

As described in Section 3.2, the standard synthesis problem in supervisory control theory is defined by a *plant* language L and *specification* language K. The goal is to find the supremal controllable sublanguage $\sup\mathcal{C}(K, L)$. In a compositional setting, these languages are given through FSMs, e.g., $L = \mathcal{M}(H)$ and $K = \mathcal{M}(E)$, which in turn are represented as the synchronous composition of several FSMs,

$$\text{Plant:} \qquad H = H_1 \| \cdots \| H_k \; ; \tag{5.1}$$

$$\text{Specification:} \qquad E = E_1 \| \cdots \| E_m \; . \tag{5.2}$$

In the following, it is assumed that these plant and specification components are deterministic FSMs. They may then be transformed into nondeterministic abstractions during compositional synthesis.

The assumption of an initially deterministic model corresponds to *total observations*, i.e., the supervisor to be synthesised is always aware of the exact system state. Synthesis with nondeterministic plants and specifications has also been considered, e.g., by Heymann and Lin (1998) and Takai (2019). This leads to *partial observations* (Cieslak et al. 1988), where the supervisor is not fully aware of the plant's state. While Komenda and Masopust (2020) have recently reported results about projection-based abstraction for hierarchical control under partial observations, algorithms for compositional synthesis under partial observations are yet to be developed. This survey only considers total observations.

It is difficult to use compositional abstraction to simplify the plant (5.1) and specification (5.2) separately as interactions between specification and plant components are common. Instead, most methods compose a specification component E_i with some of the plant components H_j, then compute an abstraction and compose the result with more plant or specification components. This means that compositional synthesis algorithms must distinguish plants and specifications in some way.

On the other hand, a synthesis problem with plants and specifications can be transformed automatically into an equivalent problem to synthesise a nonblocking supervisor for a composition of plants only (Cassandras and Lafortune 2008). This transformation, called *plantification* by Flordal et al. (2007), can be used as a pre-processing step to replace specifications by plants and then use algorithms that do not have to distinguish plants and specifications.

Definition 5.1 (Flordal et al. 2007) Let $E = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM. The *complete plant FSM* for E is $E_\perp = \langle \Sigma, Q \cup \{\perp\}, \rightarrow_\perp, Q^\circ \rangle$ where $\perp \notin Q$ is a new state and

$$\rightarrow_\perp = \rightarrow \cup \{ \langle x, \mu, \perp \rangle \mid x \in Q \text{ and } \mu \in \Sigma \cap \hat{\Sigma}_\mathrm{u} \text{ and } x \overset{\mu}{\not\rightarrow} \} \, . \tag{5.3}$$

Plantification transforms a specification component E into a plant component $E_\perp$ by adding, for every uncontrollable event that is not enabled in a state, a transition to the new trap state $\perp$. This means that an uncontrollable system, i.e., one where the plant allows an uncontrollable event to occur while some specification component disables it, becomes blocking after the transformation. In this way, initial controllability problems are transformed into blocking problems. Still, the controllability of events remains important during synthesis as the supervisor cannot disable uncontrollable events.

Example 5.1 Figure 13 shows an example of plantification. The uncontrollable event !put is disabled in state 2 of specification E. Accordingly, plantification adds a transition $2 \xrightarrow{\text{!put}} \perp$, resulting in the complete plant FSM $E_\perp$.

Following Definition 5.1, the added trap state $\perp$ in $E_\perp$ is not accepting and has no outgoing transitions. Figure 13 also shows an equivalent form $E'_\perp$ where all events are enabled at the trap state $\perp$. This is equivalent in compositional synthesis as the blocking state $\perp$ cannot be reached under any form of nonblocking control. The standard form $E_\perp$ has fewer transitions, while the alternative $E'_\perp$ has the benefit of all uncontrollable events being enabled in all states.

Plantification can be realised as a fully automatic process. Thus, designers can model a system in terms of plants and specifications, while algorithms benefit from the uniform structure of a model consisting of plants only. After plantification, a synthesis problem becomes a composition

$$G = G_1 \| G_2 \| \cdots \| G_n \tag{5.4}$$

of plant components. The problem of synthesis is reduced to finding a supremal nonblocking sub-FSM of a plant FSM.

Definition 5.2 Let G be an FSM. The *supremal nonblocking sub-FSM* of G is

$$\sup\mathcal{C}(G) = \sup\mathcal{C}(G, G) . \tag{5.5}$$

Here, $\sup\mathcal{C}(G, G)$ is the supremal controllable and nonblocking sub-FSM according to Definition 3.10. In cases where plants and specifications are distinguished, the synthesis result can be characterised using plantification as

$$\sup\mathcal{C}(E, H) = \sup\mathcal{C}(E_\perp \| H) , \tag{5.6}$$

which gives the correct result even if the specification E is not a sub-FSM of the plant H. Some of the compositional synthesis methods described in this survey work with plantified specifications, while others require separate plants and specifications. The following subsection describes frameworks for both cases, which can be converted into each other using relations such as (5.6).

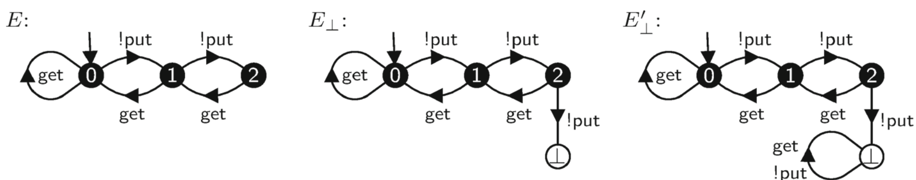


Fig. 13 Example of plantification. $E_\perp$ is the complete plant FSM of specification E, and $E'_\perp$ is an equivalent form where the uncontrollable event !put is always enabled.

5.2 Frameworks for compositional synthesis

This subsection explores options for a general formal framework for compositional synthesis similar to the verification framework in Section 2.4. That framework is defined with a set Φ of properties to be verified and a relation $\models$ to determine whether a system satisfies a property. This does not directly fit the needs of synthesis where the objective is to compute a supervisor rather than check whether some property is true or false.

One possible solution is to use the set of all languages as the set of properties, $\Phi = 2^{\hat{\Sigma}^*}$, and define that a system G satisfies a property $L \subseteq \hat{\Sigma}^*$ if the supremal controllable and nonblocking behaviour is equal to L,

$$G \models L \qquad \text{if and only if} \qquad \mathcal{L}(\sup\mathcal{C}(G)) = L \ . \tag{5.7}$$

This captures the synthesis result through verification, converting the synthesis problem to the problem of checking whether the result of synthesis is equal to a given language L. The next step for a compositional framework is to identify an appropriate process equivalence, which leads on to abstraction and hiding operations. The process equivalence corresponding to (5.7) is *synthesis equivalence* (Flordal et al. 2007)

$$\begin{gathered} G \simeq_{\text{synth}} H \\ \text{if and only if} \\ \mathcal{L}(\sup\mathcal{C}(G \| T)) = \mathcal{L}(\sup\mathcal{C}(H \| T)) \quad \text{for all FSMs } T \ . \end{gathered} \tag{5.8}$$

Two FSMs are synthesis equivalent if the supremal controllable and nonblocking behaviours are equal after synthesis in composition with any arbitrary FSM T.

At this point it can be confirmed that synthesis equivalence is a congruence with respect to synchronous composition and preserves properties, so that conditions (CV1) and (CV3) of the compositional framework are satisfied. Therefore, compositional synthesis without hiding is feasible using synchronous composition and any abstraction that preserves synthesis equivalence.

It is possible to define hiding operations such that synthesis equivalence is a congruence with respect to hiding and satisfies (CV2), but synthesis results are not preserved by hiding and (CV4) does not hold when using (5.7). Even if the hidden events Υ are local and regardless of how exactly hiding is defined, $\mathcal{L}(\sup\mathcal{C}(G)) = \mathcal{L}(\sup\mathcal{C}(G \setminus \Upsilon))$ does not hold in general, because the synthesis result $\sup\mathcal{C}(G)$ usually depends on the events in G. Then it is not clear whether hiding is possible in a compositional framework based on synthesis equivalence.

Another idea is to use an equivalence that considers only the nonemptiness of the synthesis result, such as

$$\begin{gathered} G \simeq H \\ \text{if and only if} \\ \mathcal{L}(\sup\mathcal{C}(G \| T)) \neq \emptyset \Leftrightarrow \mathcal{L}(\sup\mathcal{C}(H \| T)) \neq \emptyset \quad \text{for all FSMs } T \ . \end{gathered} \tag{5.9}$$

In this case, synthesis is replaced by a verification problem to determine whether or not a controllable and nonblocking supervisor exists. It is possible to define hiding operations that preserve this equivalence and derive a compositional framework where local events can be hidden. This approach can effectively determine whether or not a synthesis problem has a solution, but unfortunately there is no easy way to construct a supervisor if a solution is found to exist. The supervisor has to be obtained by processing all abstraction steps backwards similarly to counterexample computation in compositional verification (Section 4.5), and this is more difficult for supervisors than for counterexamples.

To avoid these complications, most current methods for compositional synthesis follow the verification framework only partially. Synthesis equivalence (5.8) or stronger equivalences are used instead of (5.9), with additional assumptions and reasoning to facilitate the construction of a supervisor. To capture such reasoning, Mohajerani et al. (2014) describe a framework that incorporates abstraction and supervisor construction in a uniform notation. A partially abstracted system is represented as a *synthesis pair*[5]

$$\langle \mathcal{G}, \mathcal{S} \rangle \ , \tag{5.10}$$

where $\mathcal{G}$ is a set of deterministic plant FSMs representing the partially abstracted system and $\mathcal{S}$ is a set of deterministic FSMs that form a partially constructed supervisor. An initial system (5.4) is represented as $\langle \{G_1, \ldots G_n\}, \emptyset \rangle$ with the plants $G_1, \ldots G_n$ and no supervisor components, and compositional abstraction is used to rewrite this into the form $\langle \emptyset, \{S_1, \ldots S_m\} \rangle$ such that the synchronous composition of the supervisors $S_1, \ldots S_m$ solves the original synthesis problem, i.e.,

$$\mathcal{L}(\sup\mathcal{C}(G_1 \| \cdots \| G_n)) = \mathcal{L}(S_1 \| \cdots \| S_m) \ . \tag{5.11}$$

To characterise requirements for sound abstraction steps, synthesis equivalence is defined for synthesis pairs by

$$\begin{gathered} \langle \mathcal{G}_1, \mathcal{S}_1 \rangle \simeq_{\text{synth}} \langle \mathcal{G}_2, \mathcal{S}_2 \rangle \\ \text{if and only if} \\ \mathcal{L}(\sup\mathcal{C}(\|(\mathcal{G}_1)) \| \|(\mathcal{S}_1)) = \mathcal{L}(\sup\mathcal{C}(\|(\mathcal{G}_2)) \| \|(\mathcal{S}_2)) \ . \end{gathered} \tag{5.12}$$

That is, two synthesis pairs are equivalent if the synthesis result for the plants composed with the components of the partial supervisors yields the same closed-loop behaviour in both cases. If all abstraction steps adhere to this equivalence, then the correctness (5.11) of the final result is ensured.

The rewrite operations in Section 2.2 can be formalised and analysed in this notation. It is clear that synchronous composition preserves synthesis equivalence of pairs. For example, two components G_1 and G_2 can be replaced by their synchronous composition $G_1 \| G_2$,

$$\langle \{G_1, G_2, \ldots, G_n\}, \mathcal{S} \rangle \simeq_{\text{synth}} \langle \{G_1 \| G_2, \ldots, G_n\}, \mathcal{S} \rangle \ . \tag{5.13}$$

Abstraction of a single component subject to synthesis equivalence of FSMs (5.8) also preserves synthesis equivalence of pairs. For example, a component G_1 can be replaced by a synthesis equivalent abstraction $\tilde{G}_1$,

$$\langle \{G_1, G_2, \ldots, G_n\}, \mathcal{S} \rangle \simeq_{\text{synth}} \langle \{\tilde{G}_1, G_2, \ldots, G_n\}, \mathcal{S} \rangle \quad \text{if} \quad G_1 \simeq_{\text{synth}} \tilde{G}_1 \ . \tag{5.14}$$

If two plant components are synthesis equivalent FSMs, they can be substituted with each other within the plant set of a synthesis pair.

The inclusion of the partial supervisors in the pairs allows for abstraction operations that do not respect synthesis equivalence of FSMs such as hiding and partial synthesis operations. The simplest example of this is *monolithic synthesis* where a supervisor for all remaining components G_i is computed and added to the set $\mathcal{S}$ of supervisors. This supervisor replaces the components G_i to produce a final result,

$$\langle \{G_1, \ldots, G_n\}, \mathcal{S} \rangle \simeq_{\text{synth}} \langle \emptyset, \{\sup\mathcal{C}(G_1 \| \cdots \| G_n)\} \cup \mathcal{S} \rangle \ . \tag{5.15}$$

[5] Mohajerani et al. (2014) define *synthesis triples* that also include a *renaming*, which is specific to their synthesis procedure and not always needed.

Monolithic synthesis is typically used as the last step of compositional synthesis to compute a supervisor for the final result of compositional abstraction.

So far, synthesis pairs have been assumed to consist of sets of deterministic FSMs. For abstractions that produce nondeterministic FSMs, $\mathcal{G}$ and $\mathcal{S}$ must be treated as multisets, and additional reasoning may be necessary to interpret the contents of $\mathcal{S}$ as a deterministic supervisor.

The above also assumes that all components in $\mathcal{G}$ are plants or plantified specifications. Alternatively, specifications and plants can be distinguished using *synthesis triples*

$$\langle \mathcal{E}, \mathcal{H}, \mathcal{S} \rangle \ , \tag{5.16}$$

where $\mathcal{E} = \{E_1, \dots, E_m\}$ consists of specification components, $\mathcal{H} = \{H_1, \dots, H_k\}$ consists of plant components, and $\mathcal{S}$ are the components of a partially computed supervisor as above. The equivalence of such triples can be expressed as synthesis equivalence of pairs using plantification,

$$\begin{gathered} \langle \mathcal{E}_1, \mathcal{H}_1, \mathcal{S}_1 \rangle \simeq_{\text{synth}} \langle \mathcal{E}_2, \mathcal{H}_2, \mathcal{S}_2 \rangle \\ \text{if and only if} \\ \langle (\mathcal{E}_1)_\perp \cup \mathcal{H}_1, \mathcal{S}_1 \rangle \simeq_{\text{synth}} \langle (\mathcal{E}_2)_\perp \cup \mathcal{H}_2, \mathcal{S}_2 \rangle \end{gathered} \tag{5.17}$$

where $(\mathcal{E}_i)_\perp = \{\, E_\perp \mid E \in \mathcal{E}_i \,\}$ for $i = 1, 2$.

Using synthesis triples, plantification can be expressed as the replacement of a specification component by its complete plant FSM, which is then treated as a plant. Following Flordal et al. (2007), this results in an equivalent triple,

$$\langle \{E_1, E_2, \dots, E_m\}, \mathcal{H}, \mathcal{S} \rangle \simeq_{\text{synth}} \langle \{E_2, \dots, E_m\}, \{(E_1)_\perp\} \cup \mathcal{H}, \mathcal{S} \rangle \ . \tag{5.18}$$

With synthesis pairs or triples, it is also possible to describe more advanced abstractions that simplify components while producing parts of the supervisor at the same time. This includes partial synthesis and hiding operations, which are the subject of the following sections.

5.3 Local synthesis

Many compositional methods perform synthesis locally, for example by composing one of the specification components with some of the plants and then computing a supervisor. In a compositional framework, this can be described as replacing a component such as G_1 in (5.4) by its supremal controllable and nonblocking sub-FSM $\sup\mathcal{C}(G_1)$, resulting in

$$\sup\mathcal{C}(G_1) \| G_2 \| \cdots \| G_n \ . \tag{5.19}$$

The result $\sup\mathcal{C}(G_1)$ becomes a component of a modular supervisor that forms the final synthesis result, and is also used to participate in further abstraction steps. In synthesis pair notation,

$$\begin{gathered} \langle \{G_1, G_2, \dots, G_n\}, \mathcal{S} \rangle \\ \text{is transformed into} \\ \langle \{\sup\mathcal{C}(G_1), G_2, \dots, G_n\}, \{\sup\mathcal{C}(G_1)\} \cup \mathcal{S} \rangle \ . \end{gathered} \tag{5.20}$$

That is, the local synthesis result $\sup\mathcal{C}(G_1)$ is included both in the plants $\mathcal{G}$ and in the supervisors $\mathcal{S}$ after the abstraction. In settings where plants and specifications are distinguished, $\sup\mathcal{C}(G_1)$ is treated as plant after abstraction as its control decisions are now part of the supervisor $\mathcal{S}$ being computed and do not need to be enforced again by further synthesis steps.

The question is whether local synthesis preserves the final result of compositional synthesis, or equivalently whether synthesis equivalence of pairs (5.12) is preserved by the transformation (5.20). This is the case if

$$\mathcal{L}(\sup\mathcal{C}(\sup\mathcal{C}(G_1)\|G_2\|\cdots\|G_n)) = \mathcal{L}(\sup\mathcal{C}(G_1\|\cdots\|G_n)) . \tag{5.21}$$

This is clearly true if G_1 and $\sup\mathcal{C}(G_1)$ are synthesis equivalent (5.8), but $G_1 \simeq_{\text{synth}} \sup\mathcal{C}(G_1)$ does not hold in general. Yet it is known that $\mathcal{L}(\sup\mathcal{C}(G_1)) \subseteq \mathcal{L}(G_1)$, which implies $\mathcal{L}(\sup\mathcal{C}(\sup\mathcal{C}(G_1)\|G_2\|\cdots\|G_n)) \subseteq \mathcal{L}(G_1\|\cdots\|G_n)$, and the final synthesis result $\sup\mathcal{C}(\sup\mathcal{C}(G_1)\|G_2\|\cdots\|G_n)$ is also controllable and nonblocking by construction. Therefore, local synthesis guarantees a final synthesis result that constrains the original behaviour (5.4) in a controllable and nonblocking way—a correct supervisor.

Yet, the resulting supervisor is not maximally permissive in general. If G_1 includes a transition with an uncontrollable event μ that leads to an unsafe state, then the source state of this transition is considered as unsafe and deleted when computing $\sup\mathcal{C}(G_1)$. However, if another plant component within $G_2\|\cdots\|G_n$ disables this uncontrollable event μ, then the transition in G_1 cannot occur and its source state does not need to be deleted.

Fortunately, simple assumptions can be imposed on the input system to ensure a maximally permissive result of compositional synthesis while using local synthesis. Maximal permissiveness only fails when some plant component disables an uncontrollable event that appears in the component subjected to local synthesis. This can be ruled out by an assumption of event disjointness.

Definition 5.3 Two FSMs $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ, Q_1^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$ are *event-disjoint* if $\Sigma_1 \cap \Sigma_2 = \emptyset$.

Two event-disjoint FSMs do not have any events in common. If all the components in a system (5.4) are pairwise event-disjoint, then local synthesis can be performed for each component separately, and the composition of the resulting supervisors gives the maximally permissive controllable and nonblocking solution. However, all components being pairwise event-disjoint is a strong assumption that rarely holds in practice.

It is more reasonable to assume that only the *plant* components are pairwise event-disjoint. For example, Hill and Tilbury (2008) compose a specification with all the plant components it shares events with and perform local synthesis for the resulting subsystem. Under the assumption that the plants are pairwise event-disjoint, this subsystem cannot share events with other plants; it may share uncontrollable events with other specifications, but this cannot break maximal permissiveness as specifications cannot disable uncontrollable events without violating controllability. If a plant shares events with more than one specification, this plant is reused for more than one composition, which is sound for an initially deterministic model. Therefore, under the assumption of event-disjoint deterministic plants, a specification that is composed with all components it shares events with can be abstracted using local synthesis while preserving maximal permissiveness. In the notation of synthesis triples, the results of Hill and Tilbury (2008) imply the following proposition.

Proposition 5.1 *Let $\mathcal{H}$, $\mathcal{E}$, and $\mathcal{S}$ be sets of deterministic FSMs where all elements of $\mathcal{H}$ are pairwise event-disjoint, let $E' \in \mathcal{E}$, and let*

$$\mathcal{H}' = \{\, H' \in \mathcal{H} \mid H' \textit{ and } E' \textit{ are not event-disjoint} \,\} \tag{5.22}$$

be the set of FSMs in $\mathcal{H}$ that share events with E'. Then

$$\langle \mathcal{E}, \mathcal{H}, \mathcal{S} \rangle \simeq_{\text{synth}} \langle \mathcal{E} \setminus \{E'\}, (\mathcal{H} \setminus \mathcal{H}') \cup \{S'\}, \mathcal{S} \cup \{S'\} \rangle , \tag{5.23}$$

where $S' = \sup\mathcal{C}(E', \|(\mathcal{H}'))$ *is the local synthesis result.*

Thus, after selecting a specification E' and all the plant components in $\mathcal{H}'$ that share at least one event with E', local synthesis is performed with the plants in $\mathcal{H}'$ and the specification E'. The resultant supervisor S' replaces the components it was synthesised from, and after the abstraction is considered both as a plant and a supervisor. Under the assumption of pairwise event-disjoint plants, this abstraction preserves the correctness and maximal permissiveness of the final synthesis result.

The assumption of event-disjointness can be relaxed. Åkesson et al. (2002) show that only shared *uncontrollable* events can affect maximal permissiveness, so it is enough to assume that plants do not share uncontrollable events and specifications are composed with plants sharing uncontrollable events.

Definition 5.4 Two FSMs $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ, Q_1^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$ are *uncontrollable event-disjoint* if $\hat{\Sigma}_{\mathrm{u}} \cap \Sigma_1 \cap \Sigma_2 = \emptyset$.

The requirement of event disjointness in Proposition 5.1 can be replaced by uncontrollable event disjointness. Event disjointness can be relaxed further by only considering an event as shared with another component if that component disables the event in a state where it would otherwise remain enabled. This leads to the idea of *mutual controllability*.

Definition 5.5 (Lee and Wong 2002) Let $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ, Q_1^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$ be two FSMs. G_1 and G_2 are *mutually controllable* if G_1 is $(\hat{\Sigma}_{\mathrm{u}} \cap \Sigma_1 \cap \Sigma_2)$-controllable with respect to G_2 and G_2 is $(\hat{\Sigma}_{\mathrm{u}} \cap \Sigma_1 \cap \Sigma_2)$-controllable with respect to G_1.

Two plants are mutually controllable if neither disables a shared uncontrollable event in a state where it would be enabled by the other. Schmidt and Breindl (2011) show that least restrictiveness is preserved by local synthesis if all plant components are pairwise mutually controllable and specifications are composed with the plants they share events with. Combining this with the idea of uncontrollable event disjointness, Proposition 5.1 can be relaxed as follows.

Proposition 5.2 *Let* $\mathcal{H}$, $\mathcal{E}$, *and* $\mathcal{S}$ *be sets of deterministic FSMs where all elements of* $\mathcal{H}$ *are pairwise mutually controllable, let* $E' \in \mathcal{E}$, *and let*

$$\mathcal{H}' = \{\, H' \in \mathcal{H} \mid H' \text{ and } E' \text{are not uncontrollable event-disjoint}\} \,. \tag{5.24}$$

Then (5.23) *holds.*

Here, after selecting a specification E' and all the plant components in $\mathcal{H}'$ that share at least one uncontrollable event with E', local synthesis is performed with the plants in $\mathcal{H}'$ and the specification E'. The resultant supervisor $\sup\mathcal{C}(E'_\perp \,\|\, \|(\mathcal{H}'))$ replaces the components it was synthesised from, and after the abstraction is considered both as a plant and a supervisor. Under the assumption of mutually controllable plants, this abstraction preserves the correctness and maximal permissiveness of the final synthesis result.

Mutual controllability is a weaker condition than uncontrollable event disjointness, but it can only be applied to original plants and does not support plantification. Both mutual controllability and uncontrollable event disjointness are global assumptions that must be satisfied by the entire model before compositional synthesis can start. If such an assumption is not satisfied, it is suggested to compose components until it is satisfied, although this may result in large components and defeat the purpose of the compositional approach.

In an attempt to avoid global assumptions, Flordal et al. (2007) propose to change the algorithm to compute local synthesis by only considering events as uncontrollable when least restrictiveness is known to be preserved. This leads to the idea of *halfway synthesis*.

Definition 5.6 (Flordal et al. 2007) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\Upsilon_u \subseteq \Sigma$. The result of *halfway synthesis* of G with respect to Υ_u is

$$\mathrm{hsup}\mathcal{C}(G, \Upsilon_u) = \bigcup \{\, G' \subseteq G \mid G' \text{ is } \Upsilon_u\text{-controllable in } G \text{ and nonblocking} \,\} . \quad (5.25)$$

Given a plant or plantified specification G and a set Υ_u of uncontrollable events, halfway synthesis computes the maximally permissive controllable and nonblocking sub-FSM under the assumption that only the events in Υ_u are uncontrollable while all other events are controllable. This over-approximates the standard synthesis result and may fail to be controllable by disabling an uncontrollable event from the full set $\hat{\Sigma}_u$ that is not included in the considered subset Υ_u. Therefore, unlike local synthesis with $\sup\mathcal{C}$, the result of halfway synthesis cannot be used as an abstraction, but this can be rectified using a modified version of plantification.

Definition 5.7 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, let $\Upsilon_u \subseteq \Sigma$ be a set of events, and let $\mathrm{hsup}\mathcal{C}(G, \Upsilon_u) = \langle \Sigma, Q, \rightarrow_{\mathrm{hsup}\mathcal{C}}, Q^\circ \rangle$. The *halfway synthesis abstraction* of G with respect to Υ_u is $\mathrm{hsup}\mathcal{C}_\perp(G, \Upsilon_u) = \langle \Sigma, Q \cup \{\perp\}, \rightarrow_\perp, Q^\circ \rangle$ where $\perp \notin Q$ is a new state and $\rightarrow_\perp = \rightarrow_{\mathrm{hsup}\mathcal{C}} \cup \{\, \langle x, \mu, \perp \rangle \mid x \in Q \text{ and } \mu \in (\Sigma \cap \hat{\Sigma}_u) \setminus \Upsilon_u \text{ and } x \stackrel{\mu}{\not\rightarrow}_{\mathrm{hsup}\mathcal{C}} \,\}$.

The halfway synthesis abstraction is obtained by plantifying the halfway synthesis result from Definition 5.6 in a way similar to Definition 5.1, with the difference that only uncontrollable events not in Υ_u are used to generate transitions to the trap state $\perp$.

Example 5.2 Figure 14 shows the differences between local synthesis and halfway synthesis. The plant component G contains two uncontrollable events !u and !v. Standard synthesis removes states 1 and 2, both of which can uncontrollably reach the blocking state 3. This is the result of local synthesis, shown as $\sup\mathcal{C}(G)$.

Now assume that event !u is shared with another plant component while !v is not. Then !u could be disabled by the plant while G is in state 1. That would mean that state 1 can be reached by a maximally permissive supervisor, and using $\sup\mathcal{C}(G)$ as a supervisor component is more restrictive than necessary.

If halfway synthesis is performed with $\Upsilon_u = \{!\mathrm{v}\}$, then the uncontrollable event !u is treated as controllable for the purpose of synthesis, the transition $1 \xrightarrow{!u} 3$ can be disabled, and state 1 is not deleted. The result is shown as $\mathrm{hsup}\mathcal{C}(G, \{!\mathrm{v}\})$ in Fig. 14. Disabling the

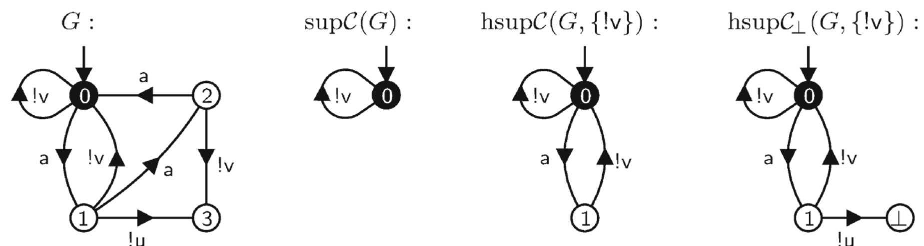


Fig. 14 Examples of local and halfway synthesis. Given plant G and uncontrollable events $\hat{\Sigma}_u = \{!\mathrm{u}, !\mathrm{v}\}$, local synthesis results in $\sup\mathcal{C}(G)$. Assuming only !v is local, $\Upsilon_u = \{!\mathrm{v}\}$, halfway synthesis results in the supervisor $\mathrm{hsup}\mathcal{C}(G, \{!\mathrm{v}\})$ and the abstraction $\mathrm{hsup}\mathcal{C}_\perp(G, \{!\mathrm{v}\})$

uncontrollable event !u means that this FSM may fail to be controllable if !u was to occur while in state 1. The halfway synthesis abstraction $\mathrm{hsup}\mathcal{C}_\perp(G, \{!\mathsf{v}\})$ with its transition $1 \xrightarrow{!\mathsf{u}} \perp$ retains this possibility so that state 1 can be removed later and only if !u is enabled.

Flordal et al. (2007) show that halfway synthesis preserves the maximal permissiveness of the final synthesis result if the set Υ_u of events considered as uncontrollable consists of the local uncontrollable events. That is, events are only considered as uncontrollable if they do not appear in any other component outside of the one being simplified. Uncontrollable events that are used in some other component are treated as controllable during halfway synthesis, and if their transitions get disabled, the halfway synthesis abstraction includes a transition to the blocking state $\perp$ to inform future steps about the possible uncontrollability.

Newer results make it possible to relax the requirement for the set Υ_u of events considered as uncontrollable. Malik and Teixeira (2020) point out that only plant components that *disable* an uncontrollable event can affect maximal permissiveness, so shared uncontrollable events that are enabled in all states of all components outside of the one being simplified can also be included in Υ_u. Moreover, if plantification is used in such a way that uncontrollable events are enabled in all states of a plantified specification, then specifications are transformed into FSMs with all uncontrollable events always enabled, which means halfway synthesis can treat uncontrollable events in plants as local independently of specifications. Combination of the result of Flordal et al. (2007) with the idea of always enabled uncontrollable events leads to the following proposition.

Proposition 5.3 *Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ contain FSMs $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ \rangle$, and let $\Upsilon_\mathrm{u} \subseteq \hat{\Sigma}_\mathrm{u} \cap \Sigma_1$ be a set of uncontrollable events such that, every event $\mu \in \Upsilon_\mathrm{u}$ is always enabled in $G_2, \ldots, G_n$. Then*

$$\langle \mathcal{G}, \mathcal{S} \rangle \simeq_{\mathrm{synth}} \langle \{\mathrm{hsup}\mathcal{C}_\perp(G_1, \Upsilon_\mathrm{u}), G_2, \ldots, G_n\}, \{\mathrm{hsup}\mathcal{C}(G_1, \Upsilon_\mathrm{u})\} \cup \mathcal{S} \rangle \, . \tag{5.26}$$

According to Proposition 5.3, to abstract a component G_1 using halfway synthesis, the first step is to determine which events can be considered as uncontrollable. Originally uncontrollable events that are always enabled by all components outside of G_1 are assigned to the set Υ_u, and all other events are treated as controllable. The resulting supervisor is added to the synthesis result $\mathcal{S}$, and the plant component G_1 is replaced by the halfway synthesis abstraction obtained by adding the transitions to $\perp$ to the supervisor.

Halfway synthesis avoids global assumptions of event disjointness or mutual controllability by adjusting the set of uncontrollable events, increasing flexibility at the expense of a more conservative abstraction. As the halfway synthesis abstraction cannot be treated purely as a plant or specification, the method does not distinguish plants and specifications and relies on plantification instead.

Ware et al. (2013) propose a further improvement to the algorithm of halfway synthesis by identifying *certainly uncontrollable states*. By considering that certain uncontrollable transitions must remain enabled for the system to be nonblocking, they can treat additional transitions as uncontrollable and remove states beyond those removed by standard synthesis algorithms.

To summarise, the synthesis operator $\mathrm{sup}\mathcal{C}$ can be used as an abstraction during compositional synthesis while ensuring a correct but not necessarily maximally permissive result. Maximal permissiveness can be maintained under additional assumptions of event disjointness or mutual controllability of plants, if specifications are composed with certain plants beforehand. Alternatively, halfway synthesis guarantees a maximally permissive result

provided that events are only treated as uncontrollable if they are always enabled in all components except for the one being simplified.

Compositional synthesis is possible using local or halfway synthesis as the only means of abstraction. In this case, the final result of compositional abstraction produces exactly the same supervisor as monolithic synthesis does, with the same number of states. Still, the compositional abstraction process results in early elimination of unsafe or blocking states, which must otherwise be visited and removed during the final synthesis, and this can improve performance (Flordal et al. 2007). More substantial state space reduction is possible by hiding of local events, which is considered in the following subsections.

5.4 Projection

Hiding of events poses several challenges for compositional synthesis. Even though a local event is not used in any other component, it is conceivable that a synthesised supervisor observes such an event and uses it to make control decisions. A controllable local event may even be disabled by the supervisor, and the need for this disablement may be discovered several steps after hiding. More generally, hiding results in nondeterministic state machines and, while nondeterminism is only a minor issue in compositional verification, it can cause major problems when attempting to construct a deterministic supervisor from abstractions.

Several of these problems are avoided by restricting all abstractions to be deterministic. In this case, hiding is performed by natural projection. This section considers the abstraction of a component G_1 whose alphabet is separated into hidden events Υ and shared events Ω, and then G_1 is replaced by $P_\Omega(G_1)$, which is a minimal deterministic FSM that accepts the projection $P_\Omega(\mathcal{L}(G_1))$ of the language of G_1 with the events in Υ removed. In synthesis pair notation,

$$\begin{gathered} \langle\{G_1, G_2, \ldots, G_n\}, \mathcal{S}\rangle \\ \text{is transformed into} \\ \langle\{P_\Omega(G_1), G_2, \ldots, G_n\}, \{G_1\} \cup \mathcal{S}\rangle \;. \end{gathered} \tag{5.27}$$

The component G_1 is replaced by its abstraction within the system $\mathcal{G}$ and also added to the supervisors $\mathcal{S}$. As events in Υ are erased in $P_\Omega(G_1)$ and do not appear in $G_2, \ldots, G_n$, it is clear that no supervisor component constructed after this step includes these events. By including G_1 as a supervisor in $\mathcal{S}$, it is ensured that the final supervisor includes any necessary disablements of controllable events in Υ. The inclusion of G_1 in $\mathcal{S}$ may not always be needed in practice, as G_1 may be the result of local or halfway synthesis and already in $\mathcal{S}$ following (5.20).

In general, the naive transformation (5.27) does not ensure a maximally permissive or even correct synthesis result. As explained in Section 4.4, natural projection is not a sound abstraction for compositional nonblocking verification, and it does not ensure a nonblocking result of compositional synthesis either. In the same way as with verification, the issue can be addressed with the observer property.

Wong and Wonham (1996) show that the observer property ensures nonblocking control after abstraction. According to their result, if the set Υ of hidden event is chosen such that it only contains events local to G_1 and such that the projection P_Ω has the observer property for G_1 (Definition 4.7), then the transformation (5.27) ensures that the final supervisor is nonblocking.

Hill and Tilbury (2008) use this result for compositional synthesis. They hide local events subject to the observer property and perform abstraction by natural projection. All plant components are abstracted individually in this way, and afterwards local synthesis is performed

by composing abstracted specifications with all abstracted plants they share events with. The result from this local synthesis becomes a supervisor component as well as a plant whose local events can be projected out again. This method produces a correct supervisor that is controllable and nonblocking, but not necessarily maximally permissive.

Example 5.3 Figure 15 shows an example where projection of local events subject to the observer property fails to ensure maximal permissiveness. The FSM G_1 shares events c and !u with the rest of the system, G_2, suggesting that events d and e are local and can be hidden. Projecting them out results in $P_{\{c,!u\}}(G_1)$, which satisfies the observer property because, independently of whether the hidden events occur or not, it remains possible to reach the accepting state by executing c or !u. The abstraction $P_{\{c,!u\}}(G_1)$ enables the shared uncontrollable event !u in its initial state, meaning that the synthesis result $\sup\mathcal{C}(P_{\{c,!u\}}(G_1)\|G_2)$ is empty. However, the synthesis result $\sup\mathcal{C}(G_1\|G_2)$ for the system before abstraction is nonempty because the unsafe occurrence of !u can be prevented by disabling the controllable event e, leaving three safe states.

The reason why the observer property is insufficient to ensure a maximally permissive result is because it does not distinguish adequately between controllable and uncontrollable events, and the removal of controllable transitions can remove opportunities for the supervisor to prevent subsequent uncontrollable transitions. This suggests that a maximally permissive result can be achieved by allowing only uncontrollable events to be projected out. Feng and Wonham (2008) identify the weaker condition of *output control consistency* that can be imposed on the projection while still achieving maximal permissiveness.

Definition 5.8 (Feng and Wonham 2008) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic FSM and let $\Sigma = \Omega \dot{\cup} \Upsilon$. The natural projection P_Ω is *output control consistent (OCC)* for G, if for all $s \in \Sigma^*$ and all $u \in \Upsilon^*$ and all $\mu \in \Omega \cap \hat{\Sigma}_u$ such that $su\mu \in \mathcal{L}(G)$, it holds that $u \in \hat{\Sigma}_u^*$.

In words, for the projection to be output control consistent, for every subtrace $u\mu$ allowed by the system that consists of local events u followed by a shared uncontrollable event μ, the local events u must all be uncontrollable. Feng and Wonham (2008) show that maximally permissive nonblocking control is achieved if the projection has the observer property and is output control consistent.

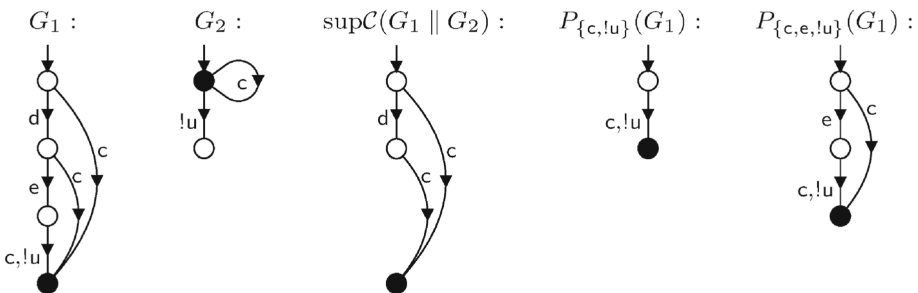


Fig. 15 Observer property with and without output control consistency. G_1 and G_2 share events c and !u, and $\hat{\Sigma}_u = \{!u\}$. The projection $P_{\{c,!u\}}(G_1)$ has the observer property, but produces an empty synthesis result. The projection $P_{\{c,e,!u\}}(G_1)$ is additionally output control consistent and gives a maximally permissive synthesis result

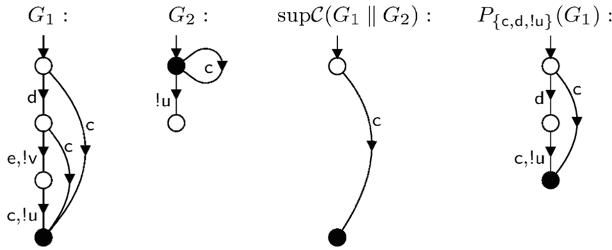


Fig. 16 Output control consistency vs. local control consistency. G_1 and G_2 share events c and !u, and $\hat{\Sigma}_u = \{!u, !v\}$. The projection $P_{\{c,d,!u\}}(G_1)$ is locally control consistent but not output control consistent

Example 5.4 Considering G_1 in Fig. 15 again, the projection $P_{\{c,!u\}}$ is not output control consistent because d e !u $\in \mathcal{L}(G)$ with d e $\in \Upsilon^*$ and !u $\in \Omega \cap \hat{\Sigma}_u$ but d e $\notin \hat{\Sigma}_u^*$. The projection $P_{\{c,e,!u\}}$, which also has the observer property, is output control consistent, because the only event that can precede the uncontrollable event !u is e, which is retained by this projection. Synthesis with this abstraction and the remainder G_2 of the system results in the disablement of e while leaving d enabled, correctly retaining the behaviour of $\sup\mathcal{C}(G_1 \parallel G_2)$.

Schmidt and Breindl (2008) introduce *local control consistency*, which is weaker than output control consistency and achieves the same results.

Definition 5.9 (Schmidt and Breindl 2008) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic FSM and let $\Sigma = \Omega \dot{\cup} \Upsilon$. The natural projection P_Ω is *locally control consistent (LCC)* for G, if for all $s \in \Sigma^*$ and all $u \in \Upsilon^*$ and all $\mu \in \Omega \cap \hat{\Sigma}_u$ such that $su\mu \in \mathcal{L}(G)$, there exists a trace $v \in (\Upsilon \cap \hat{\Sigma}_u)^*$ such that $sv\mu \in \mathcal{L}(G)$.

In words, a natural projection is locally control consistent if for any state with a shared uncontrollable event μ feasible following some local events u, the same uncontrollable event μ is also feasible after some local uncontrollable events v. The difference to output control consistency is that local control consistency allows for shared uncontrollable transitions to be preceded with controllable local transitions, as long as there is an alternative path of local uncontrollable transitions.

Example 5.5 Consider the FSM G_1 in Fig. 16, which shares events c and !u with the rest of the system G_2. This leaves three local events d, e, and !v, but d cannot be projected out for similar reasons as in Example 5.3. Considering $\Upsilon = \{e, !v\}$, the projection $P_{\{c,d,!u\}}(G_1)$ is not output control consistent, because d e !u $\in \mathcal{L}(G)$ with e $\in \Upsilon^*$ and !u $\in \Omega \cap \hat{\Sigma}_u$ but e $\notin \hat{\Sigma}_u^*$. Yet this projection is locally control consistent. For d e !u $\in \mathcal{L}(G)$ there exists d !v !u $\in \mathcal{L}(G)$ with !v $\in (\Upsilon \cap \hat{\Sigma}_u)^*$. Synthesis with this abstraction and the remainder G_2 of the system results in the disablement of d, which is equivalent to synthesis with the unabstracted system.

Feng and Wonham (2008) and Schmidt and Breindl (2011) describe methods of compositional synthesis using projection as abstraction. After composing each specification with all plants it shares events with and performing local synthesis, the resulting supervisor components are simplified using a projection that satisfies the observer property and either output control consistency (Feng and Wonham 2008) or local control consistency (Schmidt and Breindl 2011); further synthesis steps are performed afterwards. These methods make additional assumptions about the plants for the local synthesis steps to ensure maximal permissiveness, for example Schmidt and Breindl (2011) assume mutually controllable plants.

Fortunately, such assumptions are not needed when the abstraction is isolated in synthesis pair notation. The following is a consequence of the results of Schmidt and Breindl (2011).

Proposition 5.4 *Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ be a set of deterministic FSMs $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ \rangle$, and let $\Upsilon \subseteq \Sigma_1$ be such that $\Upsilon \cap (\Sigma_2 \cup \cdots \cup \Sigma_n) = \emptyset$. If the projection $P_{\Sigma_1 \setminus \Upsilon}$ has the observer property and is locally control consistent for G_1, then*

$$\langle \mathcal{G}, \mathcal{S} \rangle \simeq_{\text{synth}} \langle \{P_{\Sigma_1 \setminus \Upsilon}(G_1), G_2, \ldots, G_n\}, \{G_1\} \cup \mathcal{S} \rangle \ . \tag{5.28}$$

Accordingly, if only local events are projected out by a projection that satisfies the observer property and local control consistency, then the resulting abstraction can be used in compositional synthesis while ensuring a maximally permissive result. As output control consistency implies local control consistency (Schmidt and Breindl 2011), Proposition 5.4 also holds if local control consistency is replaced by output control consistency. This shows that local or halfway synthesis can be combined with natural projection for a comprehensive method of compositional synthesis.

A problem that remains with projection-based methods is the need to identify an appropriate subset Υ of events to be projected out. While it is easy to identify local events, it is in general not possible to project out all local events. The observer property and output or local control consistency must be satisfied, and it is not straightforward to find an appropriate and ideally large set of events that satisfies these conditions. A variety of search algorithms have been proposed to identify projections that satisfy the needed properties (Schmidt and Moor 2006; Feng and Wonham 2010).

5.5 Nondeterministic abstractions with partial observations

More general approaches to the removal of local events in compositional synthesis are possible by at least temporarily allowing for nondeterministic state machines as abstractions. Then process-algebraic hiding can be used to remove all local events. In the terminology of synthesis pairs, the projection $P_\Omega(G_1)$ in (5.27) is replaced by the result of hiding, $G_1 \setminus \Upsilon$, where Υ is the set of all events local to G_1.

Hiding means that events are replaced by the silent event τ, and it needs to be determined how the silent event is treated by synthesis. The standard interpretation of silent transitions is that no other component can synchronise with them, and this includes the synthesised supervisor. Based on this assumption, the silent event τ is not only treated as uncontrollable but also as *unobservable* (Lin and Wonham 1988). Treating τ as uncontrollable means that supervisors synthesised after abstraction cannot disable hidden events, and treating τ as unobservable means that supervisors cannot change their state to base other control decisions on the occurrence of hidden events. Unfortunately, this interpretation does not preserve the maximal permissiveness of synthesis results.

Example 5.6 Consider plants G_1 and G_2 with controllable events $\hat{\Sigma}_c = \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$ in Fig. 17. The supremal controllable and nonblocking sub-behaviour is shown as $\sup\mathcal{C}(G_1 \| G_2)$. The controllable event b must be disabled, because plant G_2 does not allow the two consecutive occurrences of c needed to reach the accepting state after b. Hiding the local events !a and b from G_1 results in $G_1 \setminus \{!\mathsf{a}, \mathsf{b}\}$. If the silent event τ is uncontrollable, a supervisor cannot disable it. As one of its transitions leads to a blocking state, a nonblocking supervisor must prevent its source state from being reached. As the source state is the initial state, the maximally permissive supervisor $\sup\mathcal{C}((G_1 \setminus \{!\mathsf{a}, \mathsf{b}\}) \parallel G_2)$ is empty.

Despite the inability to produce maximally permissive supervisors, Su et al. (2010a, b) and Hill et al. (2010) propose compositional methods with hiding and unobservable silent events to

synthesise controllable and nonblocking supervisors. To handle the unobservable transitions resulting from abstraction, synthesis needs to respect *observability* (Lin and Wonham 1988) in addition to controllability and the nonblocking property. In this case the only unobservable event, τ, is also uncontrollable, and observability is equivalent to *normality* (Wonham 2013).

This suggests that local synthesis can be performed by computing a normal supervisor. In synthesis pair notation, the local synthesis operation (5.20) is modified by replacing $\sup\mathcal{C}(G_1)$ with $\sup\mathcal{CN}(G_1)$, the supremal controllable, nonblocking, and normal sub-behaviour of G_1. It is computed as a deterministic FSM, which can be used both as a supervisor component and as a plant to continue the compositional synthesis process. The algorithm to synthesise a supremal normal supervisor is exponential in the number of states (Brandt et al. 1990). To avoid the exponential complexity, Su et al. (2010a, b) and Hill et al. (2010) replace $\sup\mathcal{CN}(G_1)$ by approximations which are less permissive than the supremal controllable, nonblocking, and normal supervisor but can be computed in polynomial time.

In addition to hiding and local synthesis, the above methods perform abstraction to reduce the number of states further. In the case of uncontrollable and unobservable silent events, where the maximal permissiveness is not guaranteed, it is enough for abstraction to preserve the nonblocking property of the final synthesis result. Hill et al. (2010) compute abstractions using two conflict-preserving rules from Flordal and Malik (2009). Similarly, Su et al. (2010a, b) use a special form of weak observation equivalence (Definition 4.5), which also implies conflict equivalence (Definition 4.4).

Indeed, if a component G_1 is replaced by a conflict equivalent abstraction $\tilde{G}_1 \simeq_{\text{conf}} G_1$, it follows from the congruence of conflict equivalence with respect to synchronous composition that any nonblocking supervisor for the abstraction is also nonblocking when composed with the original system. Although conflict equivalence does not preserve the language, it does preserve the nonconflicting part of the language (Malik et al. 2006), and therefore the controlled behaviour after synthesis from a conflict equivalent abstraction is also preserved. It follows that a plant component in a synthesis pair can be replaced by a conflict equivalent abstraction while ensuring a controllable and nonblocking synthesis result whose behaviour is contained in that of the original specifications.

5.6 Nondeterministic abstractions with observable silent events

Flordal et al. (2007) propose a different approach to hide events and handle nondeterminism after abstraction in compositional synthesis. They replace the silent event τ by two events:

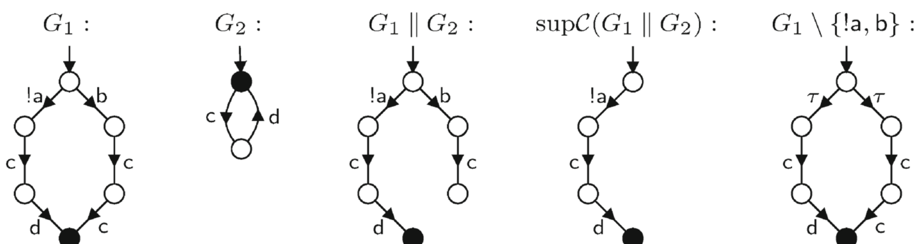


Fig. 17 Hiding in compositional synthesis. Synthesis with plants G_1 and G_2 and controllable events $\hat{\Sigma}_c = \{b, c, d\}$ gives a nonempty result, but if the local events !a and b in G_1 are replaced by an uncontrollable event τ, the synthesis result becomes empty

the silent controllable event $\tau_c \in \hat{\Sigma}_c$ and the silent uncontrollable event $\tau_u \in \hat{\Sigma}_u$. The natural projection is changed to $P\colon \hat{\Sigma}^* \to (\hat{\Sigma} \setminus \{\tau_c, \tau_u\})^*$, deleting both these events from traces.

Definition 5.10 (Flordal et al. 2007) Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an FSM, and let $\Upsilon \subseteq \Sigma$. The result of *controllability-preserving hiding* of Υ from G is $G \backslash_! \Upsilon = \langle Q, \Sigma_!, \to_!, Q^\circ \rangle$ where $\Sigma_! = \Sigma \setminus \Upsilon$ and $\to_! \subseteq Q \times (\Sigma_! \cup \{\tau_c, \tau_u\}) \times Q$ is obtained from $\to$ by replacing each transition $x \xrightarrow{\sigma} y$ with $\sigma \in \Upsilon$ by $x \xrightarrow{\tau_c} y$ if $\sigma \in \hat{\Sigma}_c$ or by $x \xrightarrow{\tau_u} y$ if $\sigma \in \hat{\Sigma}_u$.

Controllability-preserving hiding replaces hidden events in Υ by one of the silent events τ_c or τ_u depending on controllability status. This makes it possible to hide events while retaining the information whether a silent transition is controllable or uncontrollable.

Example 5.7 Consider again plants G_1 and G_2 in Fig. 17. Controllability-preserving hiding of the local events $\Upsilon = \{!\mathsf{a}, \mathsf{b}\}$ from G_1 means to replace the uncontrollable event $!\mathsf{a}$ by τ_u and the controllable event b by τ_c. The result $G_1 \backslash_! \{!\mathsf{a}, \mathsf{b}\}$ is shown in in Fig. 18. If this FSM is composed with the second plant component G_2, its blocking states are reached by a controllable transition labelled τ_c. Assuming this transition can be disabled by a supervisor, the synthesis result $\sup\mathcal{C}((G_1 \backslash_! \{!\mathsf{a}, \mathsf{b}\}) \| G_2)$ has the same states as $\sup\mathcal{C}(G_1 \| G_2)$ obtained from the unabstracted system in Fig. 17.

Retaining the controllability status of silent transitions makes it possible to represent the capabilities of supervisors more accurately. Additionally, Flordal et al. (2007) treat the silent transitions resulting from controllability-preserving hiding as observable rather than unobservable. The idea is that, given an originally fully observable and deterministic model, a supervisor can always determine the current system state. With some effort, the supervisor can trace this state through all abstraction steps and determine the current state of each abstracted FSM even if it is nondeterministic (Mohajerani et al. 2017).

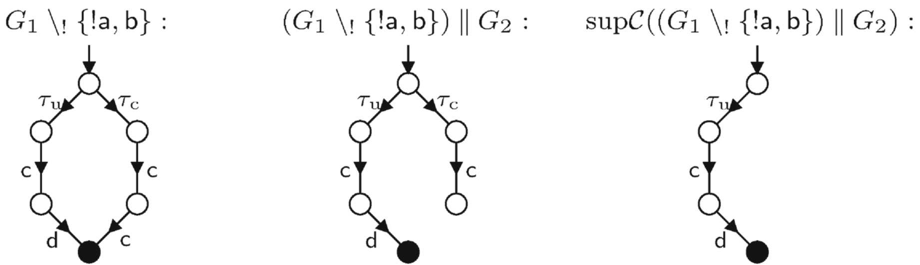


Fig. 18 Controllability-preserving hiding of $\Upsilon = \{!\mathsf{a}, \mathsf{b}\}$ from G_1 in Fig. 17 retains the maximal permissiveness.

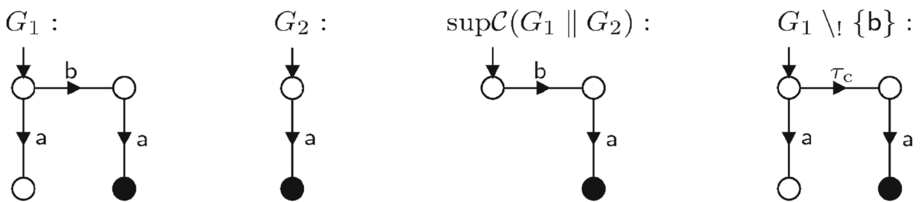


Fig. 19 Observability of hidden events. With $\hat{\Sigma}_c = \{\mathsf{a}, \mathsf{b}\}$, the synthesis result after hiding the local event b from G_1 is only nonempty if the silent event τ_c is observable

Example 5.8 Consider plants G_1 and G_2 in Fig. 19, where a and b are controllable events. A maximally permissive supervisor $\sup\mathcal{C}(G_1\|G_2)$ for the composition $G_1\|G_2 = G_1$ ensures that a is only allowed after b. Event b is local, and controllability-preserving hiding results in $G_1\backslash_!\{\mathsf{b}\}$, which in this case is equal to $(G_1\backslash_!\{\mathsf{b}\})\|G_2$. If the τ_c-transition is treated as unobservable, a maximally permissive normal supervisor must disable a both before and after τ_c, producing an empty synthesis result. If the τ_c-transition is treated as observable, a supervisor can disable a in the initial state and enable it after τ_c, ensuring a synthesis result equivalent to that obtained without hiding.

While controllability-preserving hiding preserves the set of states reached by a maximally permissive supervisor, it is not immediately clear how to construct a supervisor for the original system from such an abstraction. A supervisor synthesised based on the abstraction may disable a silent controllable transition, which needs to be related to a controllable event of the original system. To solve this problem, Mohajerani et al. (2014) retain the labels of silent transitions after hiding. In other words, events are marked as local and their transitions are treated as silent when simplifying FSMs, but the original events are still known and used to construct supervisor components.

A different solution is to avoid the construction of supervisors in the form of state machines and use abstractions to determine the set of safe states only. A supervisor can be represented as a map

$$S\colon Q \to 2^{\Sigma} \tag{5.29}$$

that assigns to each state a set of events to be enabled in that state (Ramadge and Wonham 1989). Given a subset Q' of the state set of the original system, a supervisor can be defined that disables any transition that leads to a state outside of that subset Q'. This control is feasible if it only disables controllable transitions, which is guaranteed if the subset Q' is known to be that reached by a controllable supervisor.

In the compositional synthesis of Flordal et al. (2007) and Mohajerani et al. (2017), abstractions are constructed in such a way that every state x of the original system is mapped to a state x' of the final compositional abstraction. If the state x' is a state of the maximally permissive supervisor for the compositional abstraction, then the original state x is safe; otherwise, if x' is removed in synthesis, then x is an unsafe state and the least restrictive supervisor for the original system disables transitions to this state x. In this way, a maximally permissive supervisor can be implemented without constructing its state machine representation.

Flordal et al. (2007) describe an implementation of compositional synthesis that uses controllability-preserving hiding and halfway synthesis, with halfway synthesis treating the silent uncontrollable event τ_u as the only uncontrollable event. They also propose a collection of abstraction rules to simplify components, which are inspired by compositional nonblocking verification (Flordal and Malik 2009) and closely linked to specific configurations of silent transitions. More general abstractions such as weak bisimulation and conflict equivalence ensure the nonblocking property of the final synthesis result, but they do not preserve maximal permissiveness. Mohajerani et al. (2011) strengthen weak bisimulation to *synthesis observation equivalence*, and Mohajerani et al. (2012a) improve this to *weak synthesis observation equivalence*. These two relations can be used in compositional synthesis in a framework with observable silent transitions and nondeterminism.

Synthesis observation equivalence and weak synthesis observation equivalence are variations of weak bisimulation (Definition 3.15). For two states x_1 and x_2 to be weakly bisimilar, it is enough that for every transition $x_1 \xrightarrow{\sigma} y_1$ there exists a path $x_2 \overset{\sigma}{\Rightarrow} y_2$ such that y_1 and y_2 are again weakly bisimilar. For weak synthesis observation equivalence, the relation $\Rightarrow$ is

replaced by two more specific relations $\Rightarrow_{\mathrm{u}}$ and $\Rightarrow_{\mathrm{c}}$ depending on whether or not the event σ is controllable.

Definition 5.11 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\Sigma_{\mathrm{u}} = (\Sigma \cap \hat{\Sigma}_{\mathrm{u}}) \cup \{\tau_{\mathrm{u}}\}$. The *extended uncontrollable transition relation* $\Rightarrow_{\mathrm{u}} \subseteq Q \times \Sigma_{\mathrm{u}} \times Q$ is defined by $x \overset{\mu}{\Rightarrow}_{\mathrm{u}} y$ if there exists $v \in \Sigma_{\mathrm{u}}^*$ such that $P(v) = P(\mu)$ and $x \overset{v}{\rightarrow} y$.

Definition 5.12 (Mohajerani et al. 2012a) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, let $\Sigma_{\mathrm{c}} = (\Sigma \cap \hat{\Sigma}_{\mathrm{c}}) \cup \{\tau_{\mathrm{c}}, \omega\}$, and let $\sim\, \subseteq Q \times Q$ be an equivalence relation. The *extended controllable transition relation* $\Rightarrow_{\mathrm{c}} \subseteq Q \times \Sigma_{\mathrm{c}} \times Q$ is defined such that $x \overset{\sigma}{\Rightarrow}_{\mathrm{c}} y$ if there exists a path

$$x = x_0 \overset{\tau_1}{\rightarrow} \cdots \overset{\tau_n}{\rightarrow} x_n \overset{\sigma}{\rightarrow} y_0 \xrightarrow{\tau_{n+1}} \cdots \xrightarrow{\tau_{n+m}} y_m = y \tag{5.30}$$

where $\tau_1, \ldots, \tau_{n+m} \in \{\tau_{\mathrm{c}}, \tau_{\mathrm{u}}\}$ and the following conditions hold:

(i) if $\sigma = \tau_{\mathrm{c}}$, then $n = 0$;
(ii) if $\tau_i = \tau_{\mathrm{c}}$ for some $1 \leq i \leq n$, then $x \sim x_i$;
(iii) if $y_i \overset{\tau_{\mathrm{u}}}{\rightarrow} z$ for some $1 \leq i \leq m$, then $z \sim y_j$ for some $1 \leq j \leq m$;
(iv) if $y_i \overset{\mu}{\rightarrow} z$ for some $1 \leq i \leq m$ and $\mu \in \Sigma \cap \hat{\Sigma}_{\mathrm{u}}$, then $y \overset{\mu}{\Rightarrow}_{\mathrm{u}} z'$ for some $z' \sim z$.

The extended uncontrollable transition relation $\Rightarrow_{\mathrm{u}}$ is the same as the extended transition relation $\Rightarrow$, but restricted to uncontrollable events. The extended controllable transition relation $\Rightarrow_{\mathrm{c}}$ depends on an equivalence relation, and is more complicated so as to preserve the existence of supervisors that rely on a particular controllable transition. For $x \overset{\sigma}{\Rightarrow}_{\mathrm{c}} y$ to hold, there must exist a path $x \overset{P(\sigma)}{\Rightarrow} y$ of the form (5.30). Silent transitions before the event σ are only allowed if σ is a shared event (i), and they can only be controllable if the target state is equivalent to the start state x of the path (ii). This means that any supervisor that can reach the equivalence class of x will leave such silent controllable transitions enabled. The sequence of silent transitions after σ imposes conditions on outgoing uncontrollable transitions: silent uncontrollable transitions must lead to a state equivalent to some state on the path (iii), and shared uncontrollable transitions must be enabled at the path's end state y and lead to an equivalent state (iv). Based on these definitions, weak synthesis observation equivalence is an equivalence relation on the state set of an FSM that respects $\Rightarrow_{\mathrm{u}}$ and $\Rightarrow_{\mathrm{c}}$ for all events.

Definition 5.13 (Mohajerani et al. 2012a) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM. An equivalence relation $\sim\, \subseteq Q \times Q$ is a *weak synthesis observation equivalence* on G, if the following conditions hold for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$:

(i) if $x_1 \overset{\mu}{\Rightarrow}_{\mathrm{u}} y_1$ for some $\mu \in (\Sigma \cap \hat{\Sigma}_{\mathrm{u}}) \cup \{\tau_{\mathrm{u}}\}$, then $x_2 \overset{\mu}{\Rightarrow}_{\mathrm{u}} y_2$ for some $y_1 \sim y_2$;
(ii) if $x_1 \overset{\sigma}{\Rightarrow}_{\mathrm{c}} y_1$ for some $\sigma \in (\Sigma \cap \hat{\Sigma}_{\mathrm{c}}) \cup \{\tau_{\mathrm{c}}, \omega\}$, then $x_2 \overset{\sigma}{\Rightarrow}_{\mathrm{c}} y_2$ for some $y_1 \sim y_2$.

Synthesis observation equivalence (Mohajerani et al. 2011) is defined similarly, with the only difference that there can be no silent transitions after the controllable event σ in the path (5.30) defining $\Rightarrow_{\mathrm{c}}$, so it is a special case of weak synthesis observation equivalence. Given a weak synthesis observation equivalence relation on an FSM, a synthesis equivalent abstraction is obtained using a τ-loop free quotient[6].

Proposition 5.5 (Mohajerani et al. 2012a) *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim\, \subseteq Q \times Q$ be a weak synthesis observation equivalence on G. Then $G \simeq_{\mathrm{synth}} (G/_{\circ}\!\sim)$.*

[6] Adapted from Definition 3.16, with selfloops of both silent events τ_{c} and τ_{u} suppressed.

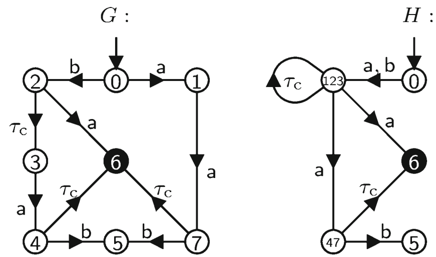


Fig. 20 Two weakly synthesis observation equivalent FSMs (Mohajerani et al. 2012a)

Example 5.9 Consider FSM G in Fig. 20, where all events are controllable. An equivalence relation $\sim$ with $1 \sim 2 \sim 3$ and $4 \sim 7$ is a weak synthesis observation equivalence on G. For example, the transition $2 \xrightarrow{a} 6$ implies $2 \overset{a}{\Rightarrow}_c 6$, which is matched by $1 \overset{a}{\Rightarrow}_c 6$ as $1 \xrightarrow{a} 7 \xrightarrow{\tau_c} 6$ and state 7 has no uncontrollable transitions outgoing. $H = (G/_{\circ}\!\sim)$ is synthesis equivalent to G by Proposition 5.5.

While the computation of most general weak synthesis observation equivalence relations has exponential time complexity, Mohajerani et al. (2012b) describe polynomial algorithms that compute appropriate approximations. Care needs to be taken as the quotient FSM may be nondeterministic in such a way that a state has more than one successor state by transitions with the same event. Similarly to the above case of silent transitions, this nondeterminism needs to be treated as benign to ensure a maximally permissive synthesis result. That is, a supervisor is thought to be aware of the exact state a nondeterministic FSM is in, and if a state has more than one outgoing transition labelled by the same controllable event, the supervisor can disable each of these transitions individually.

While nondeterministic abstractions with the benign interpretation are sufficient to compute the state space of the maximally permissive supervisor, they cause problems when trying to compute supervisor FSMs from the abstraction: if an abstract supervisor disables one of the controllable transitions at a particular state while leaving another transition with the same event enabled, it is not immediately clear how to interpret this as a control decision for the original system. Mohajerani et al. (2014) solve this problem with *event renaming* and *distinguishers*. Any nondeterministic abstraction is converted to a deterministic FSM using new distinct events whenever more than one transition with the same event is enabled in a state. At the same time, a distinguisher is constructed from the deterministic FSM before abstraction as an additional supervisor component to determine which of the newly introduced events are enabled based on the unabstracted state. In this way, it is ensured that all abstractions are deterministic and supervisors can be computed. A disadvantage of this approach is that, while the number of states decreases with abstraction, the number of events may increase exponentially.

Mohajerani et al. (2011) show that synthesis observation equivalence is weaker than the projection-based abstractions described in Section 5.4, in the sense that every abstraction that can be described by a projection with the observer and local control consistency properties can also be described as a synthesis observation equivalent abstraction. On the other hand, it is clear that (weak) synthesis observation equivalence is stronger than weak bisimulation: particularly the condition for controllable events is restrictive and reduces the abstraction potential compared to what is possible in compositional nonblocking verification.

Mohajerani et al. (2012c) consider possibilities for removing transitions while preserving relations like weak synthesis observation equivalence. While the conditions to preserve synthesis results are more complicated than for weak bisimulation, it is still possible to

remove several transitions. This reduces the memory requirements for large state machines and enables abstractions that are not immediately applicable otherwise. On the other hand, transition removal makes it even more difficult to link the remaining transitions of an abstract FSM to transitions and events of the original system, causing more problems when constructing supervisor FSMs. Therefore, transition removal is currently only used in the compositional synthesis of Mohajerani et al. (2017), where only the state space of the maximally permissive supervisor is computed.

To summarise, the compositional synthesis methods of Mohajerani et al. (2014) and Mohajerani et al. (2017) produce maximally permissive nonblocking supervisors. Both methods use controllability-preserving hiding, halfway synthesis, and weak synthesis observation equivalence abstraction. Mohajerani et al. (2014) retain event names after hiding and use renaming to ensure that all abstractions are deterministic, making it possible to produce a modular supervisor in the form of several composed FSMs. Mohajerani et al. (2017) fully hide local events, allow nondeterministic abstractions, and perform transition removal as an additional abstraction. This method works with simpler and possibly smaller abstract FSMs, while its supervisor is returned in the form of a control map without attempting to provide an FSM representation.

6 Conclusions

This survey gives an overview of compositional algorithms in supervisory control. Compositional algorithms operate by simplifying individual components before considering them in the context of a large system, in an attempt to reduce the global state space and mitigate state-space explosion. How exactly components can be simplified depends on the specific type of verification or synthesis task at hand. The survey describes and compares a variety of algorithms for compositional verification and synthesis from the recent literature.

Compositional algorithms have been used successfully for many real-life applications, where it would otherwise not have been possible to verify or synthesise. As the size and complexity of industrial applications continues to increase, research continues to develop new algorithms that can handle even larger state spaces, for example by combining compositional methods with other approaches.

Acknowledgements The research of the second and third authors is partially funded by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

Compliance with ethical standards

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Åkesson K, Flordal H, Fabian M (2002) Exploiting modularity for synthesis and verification of supervisors. IFAC Proc 35(1):175–180. https://doi.org/10.3182/20020721-6-ES-1901.00517

Åkesson K, Fabian M, Flordal H, Malik R (2006) Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: 8th Int. Workshop on Discrete Event Systems, WODES '06. IEEE, pp 384–385. https://doi.org/10.1109/WODES.2006.382401

Akers S (1978) Binary decision diagrams. IEEE Trans Comput 27(06):509–516. https://doi.org/10.1109/TC.1978.1675141

Andersen HR, Stirling C, Winskel G (1994) A compositional proof system for the modal μ-calculus. In: 9th Annual Symp. Logic in Computer Science. pp 144–153. https://doi.org/10.1109/LICS.1994.316076

Arnold A (1994) Finite Transition Systems: Semantics of Communicating Systems. Prentice-Hall, Hertfordshire

Aziz A, Singhal V, Brayton R, Swamy GM (1994) Minimizing interacting finite state machines: a compositional approach to language containment. In: 1994 IEEE Int. Conf. Computer Design: VLSI in Computers and Processors. IEEE, pp 255–261. https://doi.org/10.1109/ICCD.1994.331900

Baier C, Katoen JP (2008) Principles of Model Checking. MIT Press

Balemi S (1992) Input/output discrete event processes and system modeling. In: Int. Workshop on Discrete Event Systems, WODES '92, pp 15–27. https://doi.org/10.1007/978-3-0348-9120-2_2

Bérard B, Bidoit M, Finkel A, Laroussinie F, Petit A, Petrucci L, Schnoebelen P (2001) Systems and Software Verification. Springer

Bergstra JA, Klop JW (1984) Process algebra for synchronous communication. Inf Control 60(1–3):109–137. https://doi.org/10.1016/S0019-9958(84)80025-X

Brandin BA, Malik R, Malik P (2004) Incremental verification and synthesis of discrete-event systems guided by counter-examples. IEEE Trans Control Syst Technol 12(3):387–401. https://doi.org/10.1109/TCST.2004.824795

Brandt RD, Garg V, Kumar R, Lin F, Marcus SI, Wonham WM (1990) Formulas for calculating supremal controllable and normal sublanguages. Syst Control Lett 15:111–117. https://doi.org/10.1016/0167-6911(90)90004-E

Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput 35(8):677–691. https://doi.org/10.1109/TC.1986.1676819

Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ (1992) Symbolic model checking: 10^{20} states and beyond. Inf Comput 98(2):142–170. https://doi.org/10.1016/0890-5401(92)90017-A

Cassandras CG, Lafortune S (2008) Introduction to Discrete Event Systems, 2nd edn. Springer Science & Business Media, New York

Cheung SC, Kramer J (1999) Checking safety properties using compositional reachability analysis. ACM Trans Softw Eng Methodol 8(1):49–78. https://doi.org/10.1145/295558.295570

Cieslak R, Desclaux C, Fawaz AS, Varaiya P (1988) Supervisory control of discrete-event processes with partial observations. IEEE Trans Autom Control 33(3):249–260. https://doi.org/10.1109/9.402

Clarke EM, Emerson EA, Sistla AP (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst 8(2):244–263. https://doi.org/10.1145/5397.5399

Clarke EM, Long DE, McMillan KL (1989) Compositional model checking. In: 4th Annual Symp. Logic in Computer Science. pp 353–362. https://doi.org/10.1109/LICS.1989.39190

Clarke EM, Grumberg O, Long DE (1994) Model checking and abstraction. ACM Trans Program Lang Syst 16(5):1512–1542. https://doi.org/10.1145/186025.186051

Clarke EM Jr, Grumberg O, Peled DA (1999) Model Checking. MIT Press

De Nicola R, Hennessy MCB (1984) Testing equivalences for processes. Theor Comput Sci 34(1–2):83–133. https://doi.org/10.1016/0304-3975(84)90113-0

Eloranta J (1991) Minimizing the number of transitions with respect to observation equivalence. BIT 31(4):397–419. https://doi.org/10.1007/BF01933173

Emerson EA, Halpern JY (1986) "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. J ACM 33(1):151–178. https://doi.org/10.1145/4904.4999

Feng L, Wonham WM (2006) TCT: A computation tool for supervisory control synthesis. In: 8th Int. Workshop on Discrete Event Systems, WODES '06. IEEE, pp 388–389. https://doi.org/10.1109/WODES.2006.382399

Feng L, Wonham WM (2008) Supervisory control architecture for discrete-event systems. IEEE Trans Autom Control 53(6):1449–1461. https://doi.org/10.1109/TAC.2008.927679

Feng L, Wonham WM (2010) On the computation of natural observers in discrete-event systems. Discret Event Dyn Syst 20:63–102. https://doi.org/10.1007/s10626-008-0054-3

Fernandez JC (1990) An implementation of an efficient algorithm for bisimulation equivalence. Sci Comput Program 13:219–236. https://doi.org/10.1016/0167-6423(90)90071-K
Flordal, H. and Malik, R. (2006). Modular nonblocking verification using conflict equivalence. In: 8th Int. Workshop on Discrete Event Systems, WODES '06. IEEE, pp 100–106. https://doi.org/10.1109/WODES.2006.1678415
Flordal H, Malik R (2009) Compositional verification in supervisory control. SIAM J Control Optim 48(3):1914–1938. https://doi.org/10.1137/070695526
Flordal H, Malik R, Fabian M, Åkesson K (2007) Compositional synthesis of maximally permissive supervisors using supervision equivalence. Discret Event Dyn Syst 17(4):475–504. https://doi.org/10.1007/s10626-007-0018-z
Gohari P, Wonham WM (2000) On the complexity of supervisory control design in the RW framework. IEEE Trans Syst Man Cybern 30(5):643–652. https://doi.org/10.1109/3477.875441
Goorden M, van de Mortel-Fronczak J, Reniers M, Fokkink W, Rooda J (2020) Structuring multilevel discrete-event systems with dependency structure matrices. IEEE Trans Autom Control 65(4):1625–1639. https://doi.org/10.1109/TAC.2019.2928119
Graf S, Steffen B (1990) Compositional minimization of finite state systems. In: 1990 Workshop on Computer-Aided Verification, volume 531 of LNCS. Springer, pp 186–196. https://doi.org/10.1007/BFb0023732
Heymann M, Lin F (1998) Discrete-event control of nondeterministic systems. IEEE Trans Autom Control 43(1). https://doi.org/10.1109/9.654883
Hill RC, Tilbury DM (2008) Incremental hierarchical construction of modular supervisors for discrete-event systems. Int J Control 81(9):1364–1281. https://doi.org/10.1080/00207170701799365
Hill RC, Tilbury DM, Lafortune S (2010) Modular supervisory control with equivalence-based abstraction and covering-based conflict resolution. Discret Event Dyn Syst 20(1):139–185. https://doi.org/10.1007/s10626-009-0070-y
Hoare CAR (1985) Communicating Sequential Processes. Prentice-Hall
Hopcroft JE, Motwani R, Ullman JD (2001) Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston
Komenda J, Masopust T (2020) Conditions for hierarchical supervisory control under partial observation. IFAC PapersOnLine 53(4):303–308. https://doi.org/10.1016/j.ifacol.2021.04.029
Krook J, Kianfar R, Zita A, Mohajerani S, Fabian M (2018) Modeling and synthesis of the lane change function of an autonomous vehicle. IFAC PapersOnLine 51(7):133–138. https://doi.org/10.1016/j.ifacol.2018.06.291
Leduc RJ, Brandin BA, Lawford M, Wonham WM (2005) Hierarchical interface-based supervisory control–part I: Serial case. IEEE Trans Autom Control 50(9):1322–1335. https://doi.org/10.1109/TAC.2005.854586
Lee SH, Wong KC (2002) Structural decentralised control of concurrent discrete-event systems. Eur J Control 8:477–491. https://doi.org/10.3166/ejc.8.477-491
Li Y (1997) On deadlock-free modular supervisory control of discrete-event systems. IEEE Trans Autom Control 42(12). https://doi.org/10.1109/9.650022
Li Z, Zhou M, Wu N (2008) A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems. IEEE Trans Syst Man Cybern 38(2):173–188. https://doi.org/10.1109/TSMCC.2007.913920
Lin F, Wonham WM (1988) On observability of discrete-event systems. Inform Sci 44(3):173–198. https://doi.org/10.1016/0020-0255(88)90001-1
Lindsey J (2012) The set of certain conflicts. Honours project report, Dept. of Computer Science, University of Waikato
Malik R (2004) On the set of certain conflicts of a given language. In: 7th Int. Workshop on Discrete Event Systems. WODES '04. IFAC, pp 277–282. https://doi.org/10.1016/S1474-6670(17)30757-7
Malik R (2010) The language of certain conflicts of a nondeterministic process. Working Paper 05/2010, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand. http://hdl.handle.net/10289/4108
Malik R (2015) Advanced selfloop removal in compositional nonblocking verification of discrete event systems. In: 11th Int. Conf. Automation Science and Engineering, CASE 2015. https://doi.org/10.1109/CoASE.2015.7294182
Malik R (2016) Programming a fast explicit conflict checker. In: 13th Int. Workshop on Discrete Event Systems, WODES '16. IEEE, pp 464–469. https://doi.org/10.1109/WODES.2016.7497885
Malik R, Flordal H, Pena PN (2007) Conflicts and projections. IFAC PapersOnLine 40(6):205–210. https://doi.org/10.3182/20070613-3-FR-4909.00037
Malik R, Leduc R (2008) Generalised nonblocking. In: 9th Int. Workshop on Discrete Event Systems, WODES '08. IEEE, pp 340–345. https://doi.org/10.1109/WODES.2008.4605969

Malik R, Leduc R (2013) Compositional nonblocking verification using generalised nonblocking abstractions. IEEE Trans Autom Control 58(8):1–13. https://doi.org/10.1109/TAC.2013.2248255

Malik R, Streader D, Reeves S (2006) Conflicts and fair testing. Int J Found Comput Sci 17(4):797–813. https://doi.org/10.1142/S012905410600411X

Malik R, Teixeira M (2020) Synthesis of least restrictive controllable supervisors for extended finite-state machines with variable abstraction. Discret Event Dyn Syst 30(2):211–241. https://doi.org/10.1007/s10626-019-00302-z

Malik R, Ware S (2020) On the computation of counterexamples in compositional nonblocking verification. Discret Event Dyn Syst 30(2):301–334. https://doi.org/10.1007/s10626-019-00305-w

Milner R (1989) Communication and concurrency. Series in Computer Science. Prentice-Hall

Mohajerani S, Malik R, Fabian M (2012) An algorithm for weak synthesis observation equivalence for compositional supervisor synthesis. IFAC PapersOnLine 45(29):239–244. https://doi.org/10.3182/20121003-3-MX-4033.00040

Mohajerani S, Malik R, Fabian M (2012b) Synthesis observation equivalence and weak synthesis observation equivalence. Working Paper 03/2012, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand. http://hdl.handle.net/10289/6585

Mohajerani S, Malik R, Fabian M (2012c) Transition removal for compositional supervisor synthesis. In: 8th Int. Conf. Automation Science and Engineering, CASE 2012. pp 690–695. https://doi.org/10.1109/CoASE.2012.6386447

Mohajerani S, Malik R, Fabian M (2014) A framework for compositional synthesis of modular nonblocking supervisors. IEEE Trans Autom Control 59(1):150–162. https://doi.org/10.1109/TAC.2013.2283109

Mohajerani S, Malik R, Fabian M (2017) Compositional synthesis of supervisors in the form of state machines and state maps. Automatica 76:277–281. https://doi.org/10.1016/j.automatica.2016.10.012

Mohajerani S, Malik R, Ware S, Fabian M (2011) On the use of observation equivalence in synthesis abstraction. In: 3rd IFAC Workshop on Dependable Control of Discrete Systems, DCDS 2011. pp 84–89. https://doi.org/10.1109/DCDS.2011.5970323

Pena PN, Cury JER, Lafortune S (2009) Verification of nonconflict of supervisors using abstractions. IEEE Trans Autom Control 54(12):2803–2815. https://doi.org/10.1109/TAC.2009.2031730

Pena PN, Cury JER, Malik R, Lafortune S (2010) Efficient computation of observer projections using OP-Verifiers. In: 10th Int. Workshop on Discrete Event Systems, WODES '10. pp 416–421. https://doi.org/10.3182/20100830-3-DE-4013.00067

Pena PN, Bravo HJ, da Cunha AEC, Malik R, Lafortune S, Cury JER (2014) Verification of the observer property in discrete event systems. IEEE Trans Autom Control 59(8):2176–2181. https://doi.org/10.1109/TAC.2014.2298985

Pilbrow C, Malik R (2015) An algorithm for compositional nonblocking verification using special events. Sci Comput Programm 113(2):119–148. https://doi.org/10.1016/j.scico.2015.05.010

Pnueli A (1977) The temporal logic of programs. In: 18th Annual Symp. Found. of Computer Science. pp 46–57. https://doi.org/10.1109/SFCS.1977.32

Ramadge PJ (1983) Control and Supervision of Discrete Event Processes. Ph.D. thesis, Dept. of Electrical Engineering, University of Toronto, ON, Canada

Ramadge PJG, Wonham WM (1989) The control of discrete event systems. Proc IEEE 77(1):81–98. https://doi.org/10.1109/5.21072

Reijnen FFH, Goorden MA, van de Mortel-Fronczak JM, Rooda JE (2020) Modeling for supervisor synthesis – a lock-bridge combination case study. Discret Event Dyn Syst 30(2):499–532. https://doi.org/10.1007/s10626-020-00314-0

Roscoe AW (1994) Model-checking CSP. In: Roscoe AW (ed) A Classical Mind: Essays in Honour of C. A. R, Hoare. Prentice-Hall

Roscoe AW, Gardiner PHB, Goldsmith M, Hulance JR, Jackson DM, Scattergood JB (1995) Hierarchical compression for model-checking CSP or how to check 10^{20} dining philosophers for deadlock. In: Workshop on Tools and Algorithms for The Construction and Analysis of Systems, TACAS '95, volume 1019 of LNCS. Springer, pp 133–152. https://doi.org/10.1007/3-540-60630-0_7

Rudie K, Wonham W (1992) Think globally, act locally: Decentralized supervisory control. IEEE Trans Autom Control 37(11):1692–1708. https://doi.org/10.1109/9.173140

Schmidt K, Breindl C (2008) On maximal permissiveness of hierarchical and modular supervisory control approaches for discrete event systems. In: 9th Int. Workshop on Discrete Event Systems, WODES '08. IEEE, pp 462–467. https://doi.org/10.1109/WODES.2008.4605990

Schmidt K, Breindl C (2011) Maximally permissive hierarchical control of decentralized discrete event systems. IEEE Trans Autom Control 56(4):723–737. https://doi.org/10.1109/TAC.2010.2067250

Schmidt K, Moor T (2006) Marked-string accepting observers for the hierarchical and decentralized control of discrete event systems. In: 8th Int. Workshop on Discrete Event Systems, WODES '06. IEEE, pp 413–418. https://doi.org/10.1109/WODES.2006.382509
Su R, van Schuppen JH, Rooda JE (2010) Aggregative synthesis of distributed supervisors based on automaton abstraction. IEEE Trans Autom Control 55(7):1267–1640. https://doi.org/10.1109/TAC.2010.2042342
Su R, van Schuppen JH, Rooda JE (2010) Model abstraction of nondeterministic finite-state automata in supervisor synthesis. IEEE Trans Autom Control 55(11):2527–2541. https://doi.org/10.1109/TAC.2010.2046931
Su R, van Schuppen JH, Rooda JE, Hofkamp AT (2010) Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. Automatica 46(6):968–978. https://doi.org/10.1016/j.automatica.2010.02.025
Takai S (2019) Bisimilarity enforcing supervisory control of nondeterministic discrete event systems with nondeterministic specifications. Automatica 108:108470. https://doi.org/10.1016/j.automatica.2019.06.022
Tanenbaum AS (1992) Modern Operating Systems. Prentice-Hall
Ware S (2007) Modular finite-state machine analysis. Honours project report, Dept. of Computer Science, University of Waikato
Ware S (2014) On Conflicts in Concurrent Systems. Ph.D. thesis, Dept. of Computer Science, University of Waikato. http://hdl.handle.net/10289/8545
Ware S, Malik R (2008) The use of language projection for compositional verification of discrete event systems. In: 9th Int. Workshop on Discrete Event Systems, WODES '08. IEEE, pp 322–327. https://doi.org/10.1109/WODES.2008.4605966
Ware S, Malik R (2012) Conflict-preserving abstraction of discrete event systems using annotated automata. Discret Event Dyn Syst 22(4):451–477. https://doi.org/10.1007/s10626-012-0133-3
Ware S, Malik R (2013) Compositional verification of the generalized nonblocking property using abstraction and canonical automata. Int J Found Comput Sci 24(8):1183–1208. https://doi.org/10.1142/S0129054113500287
Ware S, Malik R, Mohajerani S, Fabian M (2013) Certainly unsupervisable states. In: 2nd Int. Workshop on Formal Techniques for Safety-Critical Systems, FTSCS 2013. pp 3–18. https://doi.org/10.1007/978-3-319-05416-2_18
Willner Y, Heymann M (1991) Supervisory control of concurrent discrete-event systems. Int J Control 54(5):1143–1169. https://doi.org/10.1080/00207179108934202
Wong KC, Wonham WM (1996) Hierarchical control of discrete-event systems. Discret Event Dyn Syst 6(3):241–273. https://doi.org/10.1007/BF01797154
Wonham WM (2013) Supervisory control of discrete-event systems. Systems Control Group, Dept. of Electrical Engineering, University of Toronto, ON, Canada
Wonham WM, Ramadge PJ (1988) Modular supervisory control of discrete event systems. Math Control Signals Syst 1(1):13–30. https://doi.org/10.1007/BF02551233
Yeh WJ, Young M (1993) Hierarchical tracing of concurrent programs. In: 3rd Irvine Software Symp., ISS '93
Zhang ZH, Wonham WM (2002) STCT: An efficient algorithm for supervisory control design. In: Caillaud B, Darondeau P, Lavagno L, Xie X (Eds) Synthesis and Control of Discrete Event Systems, 77–100. Kluwer, Dordrecht, the Netherlands. https://doi.org/10.1007/978-1-4757-6656-1_5
Zhong H, Wonham WM (1990) On the consistency of hierarchical supervision in discrete-event systems. IEEE Trans Autom Control 35(10):1125–1134. https://doi.org/10.1109/9.58555

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Robi Malik received the M.S. and Ph.D. degree in computer science from the University of Kaiserslautern, Germany, in 1993 and 1997, respectively. From 1998 to 2002, he worked at Siemens Corporate Research in Munich, Germany, where he was involved in the development and application of modelling and analysis software for discrete event systems. Since 2003, he is lecturing Computer Science at the University of Waikato in Hamilton, New Zealand. He is participating in the development of the Supremica software for modelling and analysis of discrete event systems. His current research interests are in the area of model checking and synthesis of large discrete event systems and other finite-state machine models.

Sahar Mohajerani received her M.S. degree in Systems, Control, and Mechatronics from and her Ph.D. in Automation from Chalmers University of Technology, in 2009 and 2015, respectively. She worked at Volvo Cars Corporation on function verification for two years, until 2017. She was a post-doctoral fellow at the University of Michigan from 2017 to 2019, where she worked on security and privacy verification of cyber-physical systems. She received the Best Student Paper Award at IFAC-IEEE International Workshop on Discrete Event Systems in 2014 (for a paper co-authored with R. Malik and M. Fabian). She is currently an assistant professor in the Department of Electrical Engineering at Chalmers University of Technology. Her research interests include formal methods for verification and synthesis of large discrete event systems and cyber-physical systems.

Martin Fabian was born in Gothenburg, Sweden, in 1960. He received his Ph.D. degree in control engineering in 1995 from Chalmers University of Technology, Gothenburg, Sweden. He is currently Full Professor with the Department of Electrical Engineering, Chalmers University of Technology, and head of the Automation Research group. His research interests involve modelling and supervisory control of discrete event systems, modular and compositional methods for complex systems, and verification of autonomous systems. He is also a member of the WASP Faculty.