SYNTHESIS EQUIVALENCE OF TRIPLES

Sahar Mohajerani, Robi Malik, Martin Fabian

September 19, 2012

©Sahar Mohajerani, Robi Malik, Martin Fabian

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, 3240
New Zealand
SYNTHESIS EQUIVALENCE OF TRIPLES

Sahar Mohajerani  
Department of Signals and Systems  
Chalmers University of Technology  
Göteborg, Sweden  
mohajera@chalmers.se

Robi Malik  
Department of Computer Science  
The University of Waikato  
Hamilton, New Zealand  
robi@waikato.ac.nz

Martin Fabian  
Department of Signals and Systems  
Chalmers University of Technology  
Göteborg, Sweden  
fabian@chalmers.se

September 19, 2012

Abstract

This working paper describes a framework for compositional supervisor synthesis, which is applicable to all discrete event systems modelled as a set of deterministic automata. Compositional synthesis exploits the modular structure of the input model, and therefore works best for models consisting of a large number of small automata. State-space explosion is mitigated by the use of abstraction to simplify individual components, and the property of synthesis equivalence guarantees that the final synthesis result is the same as it would have been for the non-abstracted model. The working paper describes synthesis equivalent abstractions and shows their use in an algorithm to compute supervisors efficiently. The algorithm has been implemented in the DES software tool Supremica and successfully computes modular supervisors, even for systems with more than $10^{14}$ reachable states, in less than 30 seconds.
1 Introduction

The supervisory control theory [28, 37] provides a general framework for the synthesis of reactive control functions. Given a model of the system, the plant, to be controlled, and a specification of the desired behaviour, it is possible to automatically compute, i.e. synthesise, a supervisor that restricts the plant behaviour while satisfying the specification.

Commonly, a supervisor is required to be controllable and nonblocking, i.e., it should not disable uncontrollable events, and the controlled system should always be able to complete some desired task [28]. In addition, it is typically required of a supervisor to achieve some minimum functionality. Most synthesis algorithms achieve this by producing the least restrictive supervisor, which restricts the system as little as possible while still being controllable and nonblocking [28]. Alternatives to least restrictiveness have been investigated [17, 34, 35]. They require additional analysis to guarantee minimum functionality, particularly when supervisors are synthesised automatically.

It is known [28] that for a given plant and specification, a unique least restrictive, controllable, and nonblocking supervisor exists. Straightforward synthesis algorithms explore the complete monolithic state space of the considered system, and are therefore limited by the well-known state-space explosion problem. The sheer size of the supervisor also makes it humanly incomprehensible, which hinders acceptance of the synthesis approach in industrial settings.

Various approaches for modular and compositional synthesis have been proposed to overcome these problems. Some of these approaches [32, 35] rely on structure provided by users and hence are hard to automate. Other early methods [1, 5] only consider the synthesis of a least restrictive controllable supervisors, ignoring nonblocking. Supervisor reduction [33] and supervisor localisation [7] greatly help to reduce synthesised supervisors in size, yet rely on a monolithic supervisor to be constructed first and thus remain limited by its size.

Compositional methods [12] use abstraction to remove states and transitions that are superfluous for the purpose of synthesis. The most common abstraction method is natural projection which, when combined with the observer property, produces a nonblocking but not necessarily least restrictive supervisor [35]. If output control consistency is added as an additional requirement, least restrictiveness can be ensured [10]. Output control consistency can be replaced by a weaker condition called local control consistency [30].

Conflict-preserving abstractions [17] and weak observation equivalence [34] are adequate abstractions for the synthesis of nonblocking supervisors. In these works it is assumed that, when an event is abstracted, supervisor components synthesised a later stage cannot use that event. This makes abstracted events unob-
servable and removes some possibilities of control.

The compositional methods [13, 18] allow for the abstraction of observable events through hiding. In [13, 18, 34], synthesis is considered in a nondeterministic setting, which leads to some problems when interpreting results and ensuring least restrictiveness. These problems are overcome to some extent by synthesis abstraction [20, 21, 24, 25]. Several compositional synthesis methods require all automata and their abstraction results to be deterministic, which makes some desirable abstractions impossible. Following ideas from [3, 31, 36], renaming is used in [20] to avoid nondeterminism after abstraction.

This working paper shows how the abstraction methods [13, 20, 21, 24, 25] can be brought together in a general framework for compositional synthesis, and presents an effective algorithm to compute modular supervisors that are least restrictive, controllable, and nonblocking.

In addition to halfway synthesis [13], the framework uses observation equivalence-based abstractions [21, 25], which have higher abstraction potential than methods based on natural projection [25]. These methods allow for the abstraction of observable events in such a way that abstracted events can still be used by supervisor components synthesised at a later stage. Nondeterminism after abstraction is avoided using renaming [3, 31, 36] as proposed in [20].

The proposed compositional synthesis algorithm is completely automatic. It is applicable to general discrete event systems, provided that they are represented as a set of deterministic finite-state automata, and uses no knowledge of the structure of the system to compute a solution. The algorithm has been implemented in the DES software tool Supremica [2] and applied to compute modular supervisors for several large industrial models. It successfully computes modular supervisors, even for systems with more than $10^{14}$ reachable states, within 30 seconds and using no more than 640 MB of memory.

In the following, section 2 gives a motivating example to informally illustrate compositional synthesis and abstraction. Sect. 3 briefly introduces the background of supervisory control theory, and section 4 explains compositional synthesis and the idea of synthesis equivalence underlying the compositional algorithm. Then, section 5 presents different ways of computing abstractions that preserve synthesis equivalence. The algorithm for the proposed compositional synthesis procedure is described in section 6, and section 7 applies the algorithm to several benchmark examples. Some concluding remarks are drawn in section 8. Formal proofs of technical results can be found in the appendix.
2 Motivating example

This section demonstrates compositional synthesis using the example of a simple manufacturing system shown in Figure 1. Two machines $M_1$ and $M_2$ are linked by two buffers $B_1$ and $B_2$ that can store one workpiece each. The first machine $M_1$ takes workpieces from outside the system (event $s_1$), processes them, and puts them into $B_1$ (event $!f_1$). $M_1$ also takes workpieces from $B_2$ (event $s_3$), processes them, and outputs them from the system (event $!o$). Machine $M_2$ takes workpieces from $B_1$ (event $s_2$), processes them, and puts them into $B_2$ (event $!f_2$). Using switches $W_1$ and $W_2$, the user can suspend (event $sus$) or resume (event $res$) production of $M_1$ or $M_2$, respectively.

Figure 2 shows an automata model of the system. All events are observable, and uncontrollable events are prefixed by an exclamation mark (!). Automata $M_1$, $M_2$, $W_1$, and $W_2$ are plants, while $B_1$ and $B_2$ are specifications to avoid buffer overflow and underflow. To satisfy these specifications, a supervisor must be synthesised for the system.

The compositional synthesis procedure presented in this working paper requires that the system only contains plant automata. Therefore, the specification automata $B_1$ and $B_2$ are transformed into plants $B_1^\perp$ and $B_2^\perp$, using a simple trans-
lation [13]. This is done by adding, for every uncontrollable event that is not enabled in a state, a transition to a new blocking state $\perp$. The switch model $W_2$ can also be considered as the result of this transformation, in that it models a requirement for the synthesised supervisor to prevent starting of $M_2$ in suspend mode. On the other hand, $W_1$ models a plant where it is physically impossible to start $M_1$ in suspend mode.

The compositional synthesis procedure is a sequence of small steps. At each step, automata are simplified and replaced by abstracted versions such that the supervisor synthesised from the abstracted system yields the same language when controlling the system as would the supervisor synthesised from the original system. Synchronous composition is computed step by step on the abstracted automata. In the end, the procedure results in a single abstracted automaton, which is simpler than the original system, and standard synthesis is applied to this abstracted automaton.

Initially, the system is $G_0 = \{W_1, W_2, M_1, M_2, B_1^\perp, B_2^\perp\}$. In the first step of compositional synthesis, individual automata are abstracted if possible. Events $\text{sus}_1$ and $\text{res}_1$ only appear in automaton $W_1$, and such events are referred to as local events. Exploiting local events, states $q_0$ and $q_1$ in $W_1$ can be merged, as synthesis will always remove either none or both of these states. Automaton $W_1$ can then be replaced by a synthesis equivalent automaton $\tilde{W}_1$ shown in figure 3. Automaton $W_1$ is a selfloop-only automaton that always enables all its events, so it can be disregarded in the synthesis.

Similarly, events $\text{sus}_2$ and $\text{res}_2$ are local to automaton $W_2$, so the same abstraction method can be applied. However, an attempt to compute an abstraction as before results in the nondeterministic automaton $\tilde{W}_2$ shown in figure 3. A correct supervisor needs to be aware of the states of $W_2$ in order to decide whether or not to enable event $s_2$, and it is not straightforward to construct such a supervisor only.

Figure 3: Abstraction results for switches in the manufacturing system example.
To solve the nondeterminism problem, event $s_2$ in $\tilde{W}_2$ is replaced by two new events $s_{21}$ and $s_{22}$. This procedure is referred to as renaming. Automaton $\tilde{W}_2$ is replaced by the renamed deterministic automaton $W'_2$ shown in Figure 3, and automaton $D$, which is the renamed version of $W_2$, is stored as a distinguisher in a set $S$ of collected supervisors. It is the first part of the supervisor to be computed in the end.

Having replaced $s_2$ in $W_2$, automata $M_2$ and $B_1^\perp$ need to be modified to use the new events $s_{21}$ and $s_{22}$. Therefore, $M_2$ and $B_1^\perp$ are replaced by $M'_2$ and $B'_1$ shown in Figure 3. These automata are constructed by replacing the $s_2$-transitions in $M_2$ and $B_1^\perp$ by transitions labelled $s_{21}$ and $s_{22}$.

After this, events $s_{us2}$ and $s_{res2}$ only appear in selfloops in the entire system, and as a result no state change is possible by executing these events. Thus, the self-loops associated with these events can be removed, which results in the abstracted automaton $W''_2$ shown in Figure 3.

Next, events $!o$ and $s_1$ are local events in $M_1$. States $q_0$ and $q_2$ can be merged. However, since $!f_1$ is not a local event, $q_0$ and $q_1$ are not equivalent since $q_1$ can be a blocking state if $!f_1$ is disabled by other components. Figure 4 shows the abstracted automaton $\tilde{M}_1$. Furthermore, event $!o$ now only appears in a selfloop in the entire system and thus, the selfloop associated with this event can be removed from $\tilde{M}_1$, resulting in the abstracted automaton $M'_1$ shown in Figure 4.

At this point, the system has been simplified to $G = \{W''_2, M'_1, M'_2, B'_1, B_2^\perp\}$. None of these automata can be simplified further, so the next step is to compose some of them. Figure 5 shows the composition of $M'_1$ and $B'_1$, which causes $!f_1$ to become a local event. Clearly, the blocking state $\perp$ in $M'_1 \parallel B'_1$ must be avoided, and since the uncontrollable event $!f_1$ only appears in this automaton, this means that state $q_3$ also must be avoided. Then controllable event $s_1$ must be disabled in $q_2$. Therefore, automaton $M'_1 \parallel B'_1$ is replaced by the synthesis equivalent abstraction $MB^H_1$ shown in figure 5. This abstraction method is called halfway synthesis [13]. The abstracted automaton $MB^H_1$ is added to the set $S$ of collected supervisors to enable the final supervisor to make the control decision for $s_1$. Furthermore, since $!f_1$ is a local uncontrollable event, states $q_1$ and $q_2$ in $MB^H_1$ can be merged,
Figure 5: $M'_1 \parallel B'_1$ and its abstraction result.

Figure 6: $M'_2 \parallel B'_2$ and its abstraction result.
which results in the synthesis equivalent automaton $MB_1$ shown in figure 5. Then event $!f_1$ only appears in a selfloop in $MB_1$ and nowhere else, so it can be removed, resulting in $MB'_1$ shown in figure 5.

A similar procedure is applied to $M_2' \parallel B_2'$. Exploiting the local event $!f_2$ results in the abstracted automata $MB^H_2$, $MB_2$, and $MB'_2$ shown in figure 6.

After all these abstractions, the uncontrolled plant model is $\tilde{G} = \{W''_2, MB'_1, MB'_2\}$, and the collected supervisor set is $S = \{D, MB^H_1, MB^H_2\}$. The final step is to calculate a supervisor for $\tilde{G} = W''_2 \parallel MB'_1 \parallel MB'_2$, which has 8 states and is shown in Figure 7. Synthesis results in the supervisor $S_1$ shown in Figure 7, which has 4 states. Adding it to the set $S$ results in the modular supervisor

$$S = \{D, MB^H_1, MB^H_2, S_1\},$$

which is the least restrictive, controllable and nonblocking supervisor, and produces the exact same controlled behaviour as would a monolithic supervisor calculated for the original system $G$. The largest component of the modular supervisor is $S_1$ with 4 states, and it has been computed by exploring the state space of $\tilde{G}$ with 8 states. In contrast, standard monolithic synthesis explores a state space of 138 states and produces a single supervisor with 52 states.

The example demonstrates how compositional synthesis works. In the sequel, section 4 explains the concepts more formally and shows how the renamed supervisor can control the unrenamed plant, and section 5 describes the individual abstraction methods.

3 Preliminaries

3.1 Events and Languages

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 finite alphabet $\Sigma$. For the purpose of supervisory control, this alphabet is partitioned into two disjoint subsets, the set $\Sigma_c$ of controllable events and the set $\Sigma_u$ of uncontrollable events. Controllable events can be disabled by a supervisor, while uncontrollable events may not be disabled by a supervisor. In addition, the special termination event $\omega \notin \Sigma$ is used, with the notation $\Sigma_\omega = \Sigma \cup \{\omega\}$.

$\Sigma^*$ is the set of all finite traces of events from $\Sigma$, including the empty trace $\varepsilon$. A subset $L \subseteq \Sigma^*$ is called a language. The concatenation of two traces $s, t \in \Sigma^*$ is written as $st$. A trace $s \in \Sigma^*$ is called a prefix of $t \in \Sigma^*$, written $s \sqsubseteq t$, if $t = su$ for some $u \in \Sigma^*$. For $\Omega \subseteq \Sigma$, the natural projection $P_\Omega : \Sigma^* \rightarrow \Omega^*$ is the operation that removes from traces $s \in \Sigma^*$ all events not in $\Omega$.

### 3.2 Finite-State Automata

Discrete system behaviours are typically modelled by deterministic automata, but in this paper nondeterministic automata may arise as intermediate results during abstraction.

**Definition 1** A finite-state automaton is a tuple $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$, where $\Sigma$ is a finite set of events, $Q$ is a finite set of states, $\rightarrow \subseteq Q \times \Sigma_\omega \times Q$ is the state transition relation, and $Q^0 \subseteq Q$ is the set of initial states. $G$ is deterministic, if $|Q^0| \leq 1$, and $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces in $\Sigma^*_\omega$ by letting $x \xrightarrow{s} x$ for all $x \in Q$, and $x \xrightarrow{s_1, \sigma} z$ if $x \xrightarrow{s_1} y$ 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 $x \xrightarrow{\sigma}$ means that $x \xrightarrow{\sigma} y$ for some $s \in \Sigma^*_\omega$. These notations also apply to state sets, $X \xrightarrow{s}$ for $X \subseteq Q$ means that $x \xrightarrow{s}$ for some $x \in X$, and to automata, $G \xrightarrow{s}$ means that $Q^0 \xrightarrow{s}$, etc. The language of automaton $G$ is $L(G) = \{ s \in \Sigma^*_\omega \mid G \xrightarrow{s} \}$. The termination event $\omega \notin \Sigma$ denotes completion of a task and does not appear anywhere else but to mark such completions. It is required that states reached by $\omega$ do not have any outgoing transitions, i.e., if $x \xrightarrow{\omega} y$ then there does not exist $\sigma \in \Sigma_\omega$ such that $y \xrightarrow{\sigma}$. This ensures that the termination event, if it occurs, is always the final event of any trace. The traditional set of marked states is $Q^\omega = \{ x \in Q \mid x \xrightarrow{\omega} \}$ in this notation. For graphical simplicity, states in $Q^\omega$ are shown shaded in the figures of this paper instead of explicitly showing $\omega$-transitions.

Most systems are modelled by several automata running in parallel. When these components are brought together to interact, lock-step synchronisation in the style of [15] is used.
Definition 2 Let $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^0 \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^0 \rangle$ be two automata. The synchronous composition of $G_1$ and $G_2$ is defined as

$$G_1 \parallel G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \rightarrow, Q_1^0 \times Q_2^0 \rangle \tag{2}$$

where

$$\begin{align*}
(x_1, x_2) &\xrightarrow{\sigma} (y_1, y_2) \text{ if } \sigma \in \Sigma_1 \cap \Sigma_2, x_1 \xrightarrow{\sigma_1} y_1, x_2 \xrightarrow{\sigma_2} y_2; \\
(x_1, x_2) &\xrightarrow{\sigma} (y_1, x_2) \text{ if } \sigma \in \Sigma_1 \setminus \Sigma_2, x_1 \xrightarrow{\sigma_1} y_1; \\
(x_1, x_2) &\xrightarrow{\sigma} (x_1, y_2) \text{ if } \sigma \in \Sigma_2 \setminus \Sigma_1, x_2 \xrightarrow{\sigma_2} y_2.
\end{align*}$$

Synchronous composition is associative, that is, $G_1 \parallel (G_2 \parallel G_3) = (G_1 \parallel G_2) \parallel G_3$.

Another common automaton operation is the quotient modulo an equivalence relation on the state set.

Definition 3 Let $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ be an automaton and let $\sim \subseteq Q \times Q$ be an equivalence relation. The quotient automaton of $G$ modulo $\sim$ is

$$G/\sim = \langle \Sigma, Q/\sim, \rightarrow/\sim, Q^0/\sim \rangle,$$  \tag{3}

where $\rightarrow/\sim = \{ [x] \xrightarrow{\sigma} [y] \mid x \xrightarrow{\sigma} y \}$ and $Q^0/\sim = \{ [x^0] \mid x^0 \in Q^0 \}$. Here, $[x] = \{ x' \in Q \mid x \sim x' \}$ denotes the equivalence class of $x \in Q$, and $Q/\sim = \{ [x] \mid x \in Q \}$ is the set of all equivalence classes modulo $\sim$.

3.3 Supervisory Control Theory

Given a plant automaton $G$ and a specification automaton $K$, a supervisor is a controlling agent that restricts the behaviour of the plant such that the specification is always fulfilled. Supervisory control theory [28] provides a method to synthesise a supervisor. Two common requirements for the supervisor are controllability and nonblocking.

Definition 4 Let $G$ and $K$ be two automata using the same alphabet $\Sigma$. $K$ is controllable with respect to $G$ if, for every trace $s \in \Sigma^*$, every state $x$ of $K$, and every uncontrollable event $v \in \Sigma_u$ such that $K \xrightarrow{s} x$ and $G \xrightarrow{s, v} x$, it holds that $x \xrightarrow{v} x'$ in $K$.

Definition 5 An automaton $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ is nonblocking, if for every state $x \in Q$ and every trace $s \in \Sigma^*$ such that $G \xrightarrow{s} x$ there exists $t \in \Sigma^*$ such that $x \xrightarrow{t} x'$.
For a deterministic plant $G$, it is well-known [28] that there exists a supremal controllable and nonblocking sublanguage of $L(G)$, which represents the least restrictive feasible supervisor. Algorithmically, it is more convenient to perform synthesis on the automaton $G$ instead of this language, or more precisely on the lattice of subautomata of $G$ [8]. This approach also works for nondeterministic automata.

**Definition 6** [18] $G_1 = \langle \Sigma, Q_1, \rightarrow_1, Q^\circ_1 \rangle$ is a subautomaton of $G_2 = \langle \Sigma, Q_2, \rightarrow_2, Q^\circ_2 \rangle$, written $G_1 \subseteq G_2$, if $Q_1 \subseteq Q_2$, $\rightarrow_1 \subseteq \rightarrow_2$, and $Q^\circ_1 \subseteq Q^\circ_2$.

**Theorem 1** [13] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton and $\Upsilon \subseteq \Sigma$. Then there exists a supremal controllable and nonblocking subautomaton,

$$
\text{supCN}_\Upsilon(G) = \text{sup} \{ G' \subseteq G \mid G' \text{ is controllable with respect to } G \text{ and non-blocking} \}.
$$

The subscript $\Upsilon$ is omitted if $\Upsilon = \Sigma_u$, i.e., $\text{supCN}(G) = \text{supCN}_{\Sigma_u}(G)$.

The supremal element is defined based on the subautomaton relationship (definition 6). The result is equivalent to that of traditional supervisory control theory [28]. That is, $\text{supCN}(G)$ represents the behaviour of the least restrictive supervisor that disables only controllable events in $G$ such that nonblocking is ensured.

The supervisor is typically modelled as a map $\Phi: \Sigma^* \rightarrow 2^{\Sigma_c}$ that assigns to each trace $s \in \Sigma^*$ a control decision $\Phi(s) \subseteq \Sigma_c$ consisting of the controllable events to be enabled after observing the trace $s$ [28]. Such a supervisor map can be implemented using a given automaton $S$,

$$
\Phi_S(s) = \{ \sigma \in \Sigma_c \mid s\sigma \in L(S) \}.
$$

The implementation is feasible if controllability and nonblocking are ensured, as is the case when $S = \text{supCN}(G)$. Based on this, supervisors are identified with automata in the following.

The synthesis result $\text{supCN}(G)$ can be computed by removing blocking and uncontrollable states from the plant, until a fixpoint is reached, and restricting the original automaton $G$ to these states.

**Definition 7** [18] The restriction of $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ to $X \subseteq Q$ is

$$
G_{\mid X} = \langle \Sigma, Q, \rightarrow_{\mid X}, Q^\circ \cap X \rangle,
$$

where $\rightarrow_{\mid X} = \{ (x, \sigma, y) \in \rightarrow \mid x, y \in X \} \cup \{ (x, \omega, y) \in \rightarrow \mid x \in X \}$. 

11
Note that restriction does not directly remove any states, and transitions with the termination event $\omega$ are retained even if their successor state is not contained in $X$. Typically, some states become unreachable after restriction, and these states can be removed, but this is not considered further in this working paper.

**Definition 8** [18] The synthesis step operator $\Theta_G : 2^Q \to 2^Q$ for $G = \langle \Sigma, Q, \to, Q^0 \rangle$ is defined as $\Theta_{G,\Upsilon}(X) = \Theta_{G,\Upsilon}^\text{cont}(X) \cap \Theta_{G}^\text{nonb}(X)$, where

$$\Theta_{G,\Upsilon}^\text{cont}(X) = \{ x \in X \mid \text{For all } \upsilon \in \Upsilon \text{ such that } x \overset{\upsilon}{\to} y \text{ it holds that } y \in X \} ;$$

$$\Theta_{G}^\text{nonb}(X) = \{ x \in X \mid x \overset{t\omega}{\to} \not\in X \text{ for some } t \in \Sigma^* \} .$$

Again it is defined that $\Theta_G = \Theta_{G,\Sigma_u}$ and $\Theta_{G}^\text{cont} = \Theta_{G,\Sigma_u}^\text{cont}$.

$\Theta_{G}^\text{cont}$ captures controllability, and $\Theta_{G}^\text{nonb}$ captures nonblocking. The synthesis result for $G$ is obtained by restricting $G$ to the greatest fixpoint of $\Theta_G$.

**Theorem 2** [18] Let $G = \langle \Sigma, Q, \to, Q^0 \rangle$ be a deterministic automaton, and let $\Upsilon \subseteq \Sigma$. The synthesis step operator $\Theta_{G,\Upsilon}$ has a greatest fixpoint $\text{gfp} \Theta_{G,\Upsilon} = \hat{\Theta}_{G,\Upsilon} \subseteq Q$, such that $G|_{\hat{\Theta}_{G,\Upsilon}}$ is the greatest subautomaton of $G$ that is both $\Upsilon$-controllable in $G$ and nonblocking, i.e.,

$$\text{sup} \mathcal{CN}_{\Upsilon}(G) = G|_{\hat{\Theta}_{G,\Upsilon}} . \quad (7)$$

If the state set $Q$ is finite, the sequence $X^0 = Q, X^{i+1} = \Theta_{G,\Upsilon}(X^i)$ reaches this fixpoint in a finite number of steps, i.e., $\hat{\Theta}_{G,\Upsilon} = X^n$ for some $n \geq 0$.

The operator $\text{sup} \mathcal{CN}$ only defines the synthesis result for a plant automaton $G$. In order to apply this synthesis to control problems that also involve specifications, the transformation proposed in [13] is used. A specification automaton is transformed into a plant by adding, for every uncontrollable event that is not enabled in a state, a transition to a new blocking state $\perp$. This essentially transforms all potential controllability problems into potential blocking problems.

**Definition 9** [13] Let $K = \langle \Sigma, Q, \to, Q^0 \rangle$ be a specification. The complete plant automaton $K^\perp$ for $K$ is

$$K^\perp = \langle \Sigma, Q \cup \{ \perp \}, \to^\perp, Q^0 \rangle \quad (8)$$

where $\perp \notin Q$ is a new state and

$$\to^\perp = \to \cup \{ (x, v, \perp) \mid x \in Q, v \in \Sigma_{\text{int}}, x \nott\to \} . \quad (9)$$
For example, automata $B_1^\perp$ and $B_2^\perp$ in the manufacturing system in section 2, shown in figure 2, are obtained by transforming the buffer specifications $B_1$ and $B_2$, respectively. In general, synthesis of the least restrictive nonblocking and controllable behaviour allowed by a specification $K$ with respect to a plant $G$ is achieved by computing $\text{supCN}(G \parallel K^\perp)$ [13].

4 Compositional Synthesis

This section describes the compositional synthesis framework. The data structure of synthesis triples is introduced, which represents partially solved synthesis problems in the algorithm including supervisors and renamings. Based on this, a control architecture is presented to implement the computed modular supervisors after renamings.

4.1 Basic Idea

The input to compositional synthesis is an arbitrary set of deterministic automata representing the plant to be controlled,

$$G = \{G_1, G_2, \ldots, G_n\}.$$  \hfill (10)

The objective is to calculate a least restrictive supervisor that constrains the behaviour of $G$ to its least restrictive nonblocking sub-behaviour, by disabling only controllable events.

Compositional synthesis works by repeated abstraction of system components $G_i$ based on local events; events that appear in $G_i$ and in no other automata $G_j$ with $j \neq i$ are local to $G_i$, and they are crucial to abstraction. In the following, the set of local events is denoted by $\Upsilon$, and $\Omega = \Sigma \setminus \Upsilon$ denotes the set of non-local or shared events.

Using abstraction, some components $G_i$ in (10) are replaced by simpler versions $G_i'$. If this is no longer possible, some components in (10) are selected and composed, i.e., replaced by their synchronous composition. This typically leads to new local events, making further abstraction possible.

When an abstraction $G_i'$ is computed, this may lead to the discovery of new supervisor decisions. For example, if $G_i$ contains a controllable transition leading to a blocking state, it is clear that this transition must be disabled by every supervisor. Therefore, as a result of abstraction a supervisor component $S_i$ may be produced in addition to the abstracted automaton $G_i'$. The algorithm collects these supervisor components in a set $S$, called the set of collected supervisors.
addition, abstraction may result in nondeterminism, which is avoided by applying a renaming.

Thus, compositional synthesis starts with the set of plant automata (10), no collected supervisors and no renaming. At each step, plant automata are abstracted or composed, adding supervisors to $\mathcal{S}$ and modifying the renaming. Plant automata can be replaced by supervisors through synthesis, and eventually the set $\mathcal{G}$ becomes empty. At this point, the supervisors $\mathcal{S}$, together with the renaming $\rho$, are used to form a least restrictive supervisor for the original synthesis problem.

4.2 Renaming

Nondeterminism is avoided in the compositional synthesis algorithm, because it is not straightforward to compute supervisors from nondeterministic abstractions. If an abstraction step results in a nondeterministic automaton, a renaming is applied first, introducing new events to disambiguate nondeterministic branching.

The use of renaming to disambiguate abstractions was proposed in [36]. In the following, a renaming is a map that relates the events of the current abstracted system $\mathcal{G}$ to the events in the original plant, so it works in the reverse direction compared to [36].

**Definition 10** Let $\Sigma_1$ and $\Sigma_2$ be two sets of events. A renaming $\rho: \Sigma_2 \rightarrow \Sigma_1$ is a controllability-preserving map, i.e., a map such that $\rho(\sigma)$ is controllable if and only if $\sigma$ is controllable.

For example, when event $s_2$ is disambiguated into $s_{21}$ and $s_{22}$ in automaton $\tilde{W}_2$ in figure 3 in the introductory example, the renaming $\rho$ is such that $\rho(s_{21}) = s_2$ and $\rho(s_{22}) = s_2$ and $\rho(\sigma) = \sigma$ for all other events. The definition of $\rho$ is extended to cover the termination event by letting $\rho(\omega) = \omega$. Renamings are extended to languages over $\Sigma_2^*$ and automata with alphabet $\Sigma_2$ in the standard way.

When new events are introduced, the compositional synthesis algorithm continues to operate using the new events and thus produces a supervisor based on an alphabet different from that of the original plant. To communicate correctly with the original plant, the supervisor needs to determine which of the new events ($s_{21}$ or $s_{22}$) is to be executed when the plants sends one of its original events ($s_2$). This is achieved by adding a so-called *distinguisher* [3, 36] to the synthesis result.

**Definition 11** An automaton $G = (\Sigma, Q, \rightarrow, Q^0)$ differentiates event $\gamma_1$ from $\gamma_2$, if $\gamma_1 \notin \Sigma$ and $\gamma_2 \in \Sigma$ or there exists a transition $x \xrightarrow{\gamma_1} y$ such that $x \xrightarrow{\gamma_2} y$ does not hold. $G$ differentiates between $\gamma_1$ and $\gamma_2$, if $G$ differentiates $\gamma_1$ from $\gamma_2$ or $G$ differentiates $\gamma_2$ from $\gamma_1$. 

14
Figure 8: Control architecture. $G$ is the original plant, $S$ are the computed modular supervisors, and $S_D \subseteq S$ are the distinguishers.

**Definition 12** Let $\rho: \Sigma_2 \to \Sigma_1$ be a renaming. An automaton $G_2$ with alphabet $\Sigma_2$ is a $\rho$-distinguisher if, for all traces $s, t \in \mathcal{L}(G_2)$ such that $\rho(s) = \rho(t)$, it holds that $s = t$.

For example, in the introductory example, automaton $D$ in figure 3 is a $\rho$-distinguisher that differentiates $s_{21}$ from $s_{22}$. This is because $D$ enables at most one of the events $s_{21}$ and $s_{22}$ in each state, so it can always make a choice between these two events.

Another operation is necessary in combination with renaming. After applying a renaming to an automaton $G_i$ in a system $G = \{G_1, \ldots, G_n\}$, the remaining automata $G_j$ with $j \neq i$ need to be modified to use the new events.

**Definition 13** Let $G = \langle \Sigma_1, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let $\rho: \Sigma_2 \to \Sigma_1$ be a renaming. Then $\rho^{-1}(G) = \langle \Sigma_2, Q, \rho^{-1}(\rightarrow), Q^\circ \rangle$ where $\rho^{-1}(\rightarrow) = \{ (x, \sigma, y) | x \xrightarrow{\rho(\sigma)} y \}$.

Automaton $\rho^{-1}(G)$ is obtained by replacing transitions labelled with the original event by new transitions labelled with each of the new events. For example, figure 3 in the introductory example shows $M_2' = \rho^{-1}(M_2)$ and $B_1' = \rho^{-1}(B_1^\circ)$, which replace the original plants $M_2$ and $B_1^\circ$ after the renaming. When a renaming is introduced, the distinguisher is the only automaton that differentiates between the renamed events, all others are constructed by $\rho^{-1}$.

The compositional synthesis algorithm proposed in the following repeatedly applies renamings as new abstractions are obtained. In the end, this results in a su-
Supervisor \( S \) using a modified alphabet \( \Sigma_S \) and a renaming \( \rho: \Sigma_S \to \Sigma \) that maps the renamed events back to the events of the original plant. The control architecture in figure 8 enables the renamed supervisor \( S \) to interact with the original unrenamed plant \( G \).

Assume that, after execution of a trace \( t \), an event \( \gamma \) occurs in the plant, and \( \gamma \) has been renamed and replaced by \( \gamma_1 \) and \( \gamma_2 \). Being unaware of the renaming, the plant will just communicate the occurrence of \( \gamma \) to the supervisor. When this happens, first the function \( \rho^{-1} \) replaces \( \gamma \) by the set \( \{ \gamma_1, \gamma_2 \} \), sending both possibilities to the distinguisher \( S_D \) which, following definition 12, enables only one of them. The selected event \( \gamma' \), either \( \gamma_1 \) or \( \gamma_2 \), is passed to the supervisor to update its state and issue a new control decision \( \Phi'(t'; \gamma') \subseteq \Sigma_S \). Here, \( t' \) is the renamed version of the history \( t \). The control decision is based on the renamed model and therefore contains renamed events, so the renaming \( \rho \) is applied to translate it back to a control decision \( \Phi(t; \gamma) \subseteq \Sigma \) using the original plant events.

### 4.3 Synthesis Triples

The compositional synthesis algorithm keeps track of three pieces of information:

- a set \( \mathcal{G} = \{ G_1, \ldots, G_n \} \) of uncontrolled plant automata;
- a set \( \mathcal{S} = \{ S_1, \ldots, S_m \} \) of collected supervisor automata;
- a renaming \( \rho \), to avoid nondeterminism through the introduction of new events.

This information is combined in a *synthesis triple*, which is the main data structure manipulated by the compositional synthesis algorithm.

**Definition 14** A *synthesis triple* is a triple \(( \mathcal{G}; \mathcal{S}; \rho )\), where \( \mathcal{G} \) and \( \mathcal{S} \) are sets of deterministic automata and \( \rho \) is a renaming, such that

1. \( \mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G}) \);
2. \( \mathcal{S} \) is a \( \rho \)-distinguisher.
3. for all events \( \gamma_1, \gamma_2 \) such that \( \rho(\gamma_1) = \rho(\gamma_2) \), there exists at most one automaton \( G_j \in \mathcal{G} \) that differentiates \( \gamma_1 \) from \( \gamma_2 \).

Here and in the following, sets \( \mathcal{G} \) and \( \mathcal{S} \) are also used to denote the synchronous composition of their elements, like \( \| \mathcal{G} = G_1 \| \cdots \| G_n \). For an empty set, \( \|\emptyset \) is the universal automaton that accepts the language \( \Sigma^* \).
A synthesis triple represents a partially solved control problem at an intermediate step of compositional synthesis. The set $G$ contains an abstracted plant model, and $S$ contains the supervisors collected so far, which must constrain the behaviour of the plant (i). The renaming $\rho$ maps the events found in the abstracted plant or collected supervisors back to events in the original plant. The synchronous composition of the supervisors is required to have the distinguisher property (ii) to ensure that it can be used with the control architecture in figure 8. Furthermore, if two events $\gamma_1$ and $\gamma_2$ are renamed to the same event, then there can be at most one automaton in the set $G$ that treats these events differently (iii).

The following notation associates with each synthesis triple a behaviour and a synthesis result.

**Definition 15** Let $(G; S; \rho)$ be a synthesis triple. Then

(i) $L(G; S; \rho) = L(\rho(G \parallel S))$;

(ii) $\text{sup}CN(G; S; \rho) = \rho(\text{sup}CN(G) \parallel S)$.

The behaviour of a synthesis triple is the behaviour of its plant and supervisor automata, after renaming it back to the original plant alphabet (i). Furthermore, (ii) defines a synthesis result for the partially solved control problem $(G; S; \rho)$. It is obtained by composing the monolithic supervisor for the remaining plants with the supervisors collected so far, and afterwards renaming.

While manipulating synthesis triples, the compositional synthesis algorithm maintains the invariant that all generated triples have the same synthesis result, which is equivalent to the least restrictive solution of the original control problem. Every abstraction step must ensure that the synthesis result is the same as it would have been for the non-abstracted components. This property is called *synthesis equivalence*.

**Definition 16** Two triples $(G_1; S_1; \rho_1)$ and $(G_2; S_2; \rho_2)$ are said to be synthesis equivalent, written $(G_1; S_1; \rho_1) \simeq_{\text{synth}} (G_2; S_2; \rho_2)$, if

$$L(\text{sup}CN(G_1; S_1; \rho_1)) = L(\text{sup}CN(G_2; S_2; \rho_2)).$$

(11)

The compositional synthesis algorithm calculates a modular supervisor for a modular system $G = G_0$. Initially no renaming has been applied and no supervisor or distinguisher has been collected. Thus, this input is converted to the initial synthesis triple $(G; G; \text{id})$, where $\text{id}: \Sigma \to \Sigma$ is the identity map, i.e., $\text{id}(\sigma) = \sigma$ for all $\sigma \in \Sigma$. Afterwards, the initial triple is abstracted repeatedly such that synthesis equivalence is preserved,

$$(G; G; \text{id}) = (G_0; S_0; \rho_0) \simeq_{\text{synth}} (G_1; S_1; \rho_1) \simeq_{\text{synth}} \cdots \simeq_{\text{synth}} (G_k; S_k; \rho_k).$$

(12)
Some of these steps replace an automaton in $G_k$ by an abstraction, others reduce the number of automata in $G_k$ by synchronous composition or by replacing an automaton in $G_k$ with a supervisor in $S_{k+1}$. The algorithm terminates when $G_k = \emptyset$, at which point $S_k$ together with $\rho_k$ forms the modular supervisor. The following result confirms that this results in the same supervised behaviour as a monolithic supervisor for the original system.

**Theorem 3** Let $G = \{G_1, \ldots, G_n\}$ be a set of automata, and let $(G; S; \rho) \simeq_{\text{synth}} (\emptyset; S; \rho)$. Then $L(\rho(S)) = L(\text{supCN}(\emptyset; S; \rho)) = L(\text{supCN}(G))$.

**Proof.** It follows directly from definitions 15 (ii) and 16 that $L(\rho(S)) = L(\rho(\emptyset || S)) = L(\rho(\text{supCN}(\emptyset) || S)) = L(\text{supCN}(\emptyset; S; \rho)) = L(\text{supCN}(G; G; \text{id})) = L(\text{id}(\text{supCN}(G)) || G) = L(\text{supCN}(G))$. □

5 **Synthesis Triple Abstraction Operations**

The idea of compositional synthesis is to continuously rewrite synthesis triples such that synthesis equivalence is preserved. Therefore, this section gives an overview of different ways to simplify automata that can be used in the framework of this paper. Further details and formal proofs of correctness can be found in [22].

5.1 **Basic Rewrite Operations**

The simplest methods to rewrite synthesis triples are synchronous composition and monolithic synthesis. It is always possible to compose two automata in the set $G$ of uncontrolled plants, or to place their monolithic synthesis result into the set $S$ of supervisors. These basic methods are included here for the sake of completeness. They do not contribute to simplification, and are only needed when no other abstraction is possible.

**Theorem 4** Let $G_1 = \{G_1, \ldots, G_n\}$ and $G_2 = \{G_1 || G_2, G_3, \ldots, G_n\}$, let $\rho$ be a renaming, and let $S$ be a $\rho$-distinguisher. Then $(G_1; S; \rho) \simeq_{\text{synth}} (G_2; S; \rho)$.

**Proof.** By definition 15, it holds that

\[
L(\text{supCN}(G_1; S; \rho)) = L(\rho(\text{supCN}(G_1) || S)) = L(\rho(\text{supCN}(G_1 || \cdots \ || G_n) || S)) = L(\rho(\text{supCN}(G_2) || S)) = L(\text{supCN}(G_2; S; \rho)),
\]

(13)
so the claim follows from definition 16.

\[\square\]

**Theorem 5**  Let \((G; S; \rho)\) be a synthesis triple. Then \((G; S; \rho) \simeq_{\text{synth}} (\emptyset; S \cup \{\sup CN(G)\}, \rho)\).

**Proof.** Clearly by definition 15 (ii), \(L(\sup CN(G; S; \rho)) = L(\rho(\sup CN(G) \parallel S)) = L(\rho(\sup CN(\emptyset) \parallel \sup CN(G) \parallel S)) = L(\rho(\sup CN(\emptyset) \parallel \sup CN(G) \parallel S)). \)

Another way of rewriting a synthesis triple is by renaming. As explained in section 4, an automaton \(G_1\) can be rewritten into \(H_1\) using a renaming \(\rho\) such that \(\rho(H_1) = G_1\) and \(H_1\) is a \(\rho\)-distinguisher. Then \(H_1\) is added to the set \(S\) of supervisors as a distinguisher, and the renaming \(\rho\) is composed with the previous renamings. The proof of the following result can be found in appendix A.

**Theorem 6** Let \((G_1; S; \rho_1)\) be a synthesis triple with \(G_1 = \{G_1, \ldots, G_n\}\), let \(\rho\) be a renaming, and let \(H_1\) be a \(\rho\)-distinguisher such that \(\rho(H_1) = G_1\) and \(G_2 = \{H_1, \rho^{-1}(G_2), \ldots, \rho^{-1}(G_n)\}\). Then

\[(G_1; S; \rho_1) \simeq_{\text{synth}} (G_2; \{H_1\} \cup \rho^{-1}(S); \rho_1 \circ \rho).\]

In compositional verification, events used in only one automaton can immediately be removed from the model [12]. This is not always possible in compositional synthesis. Even if no other automata use an event, the synthesised supervisor may still need to use it for control decisions that are not yet apparent. Therefore, events can only be removed if it is clear that no further supervisor decision depends on them.

An event \(\lambda\) can be removed from a synthesis triple, if it causes no state change, which means that it appears only on selfloop transitions in the automata model. In this case, \(\lambda\) can be removed from all automata. This abstraction step is formally described in theorem 7, and the proof can be found in appendix A.

**Definition 17**  An automaton \(G = (\Sigma, Q, \rightarrow, Q^c)\), is selfloop-only for \(\lambda \in \Sigma\) if \(x \xrightarrow{\lambda} y\) implies \(x = y\). Automaton \(G\) is selfloop-only for \(\Lambda \subseteq \Sigma\) if \(G\) is selfloop-only for each \(\lambda \in \Lambda\).

**Definition 18**  The restriction of \(G = (\Sigma, Q, \rightarrow, Q^c)\) to \(\Omega \subseteq \Sigma\) is \(G|_{\Omega} = (\Omega, Q, \rightarrow|_{\Omega}, Q^c)\) where \(\rightarrow|_{\Omega} = \{(x, \sigma, y) \in \rightarrow \mid \sigma \in \Omega\}\). The restriction of \(G = \{G_1, \ldots, G_n\}\) is \(G|_{\Omega} = \{G_1|_{\Omega}, \ldots, G_n|_{\Omega}\}\).

**Theorem 7** Let \((G; S; \rho)\) be a synthesis triple such that \(G\) is selfloop-only for \(\Lambda \subseteq \Sigma\). Then \((G; S; \rho) \simeq_{\text{synth}} (G|_{\Sigma \setminus \Lambda}; S; \rho)\).
5.2 Abstraction Based on Observation Equivalence

This section gives an overview of previous results on observation equivalence-based abstractions for synthesis purposes. Bisimulation and observation equivalence [19] provide well-known abstraction methods that work well in compositional verification [12]. Both can be implemented efficiently [11]. They are known to preserve all temporal logic properties [6], but unfortunately this does not help for synthesis [25]. Synthesis equivalence is preserved when an automaton is replaced by a bisimilar automaton, while observation equivalence must be strengthened to achieve the same result. This is achieved by synthesis observation equivalence [25] and weak synthesis observation equivalence [21].

**Definition 19** [19] Let $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ be an automaton. An equivalence relation $\sim \subseteq Q \times Q$ is called a bisimulation on $G$, if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$: if $x_1 \xrightarrow{\sigma} y_1$ for some $\sigma \in \Sigma_\omega$, then there exists $y_2 \in Q$ such that $x_2 \xrightarrow{\sigma} y_2$ and $y_1 \sim y_2$.

**Theorem 8** [25] Let $(G; S; \rho)$ be a synthesis triple with $G = \{G_1, \ldots, G_n\}$, and let $\sim$ be a bisimulation on $G_1$ and $\tilde{G} = \{G_1/\sim, G_2, \ldots, G_n\}$. Then it holds that $(G; S; \rho) \cong_{\text{synth}} (\tilde{G}; S; \rho)$.

Bisimulation is the strongest of the branching process equivalences. Two states are treated as equivalent if they have exactly the same outgoing transitions to the same or equivalent states. Theorem 8 confirms that it is possible to merge bisimilar states in a plant automaton in a synthesis triple while preserving synthesis equivalence.

Bisimulation treats transitions with all events alike. For better abstraction, it is desirable to differentiate between local and shared events. This is the idea of observation equivalence, which considers two states as equivalent if they can reach equivalent states by the same sequences of shared events.

**Definition 20** [19] Let $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ be an automaton with $\Sigma = \Omega \cup \Upsilon$. An equivalence relation $\sim \subseteq Q \times Q$ is called an observation equivalence on $G$ with respect to $\Upsilon$, if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$: if $x_1 \xrightarrow{s_1} y_1$ for some $s_1 \in \Sigma_\omega^s$, then there exist $y_2 \in Q$ and $s_2 \in \Sigma_\omega^s$ such that $P_{1 \cup \Upsilon}(s_1) = P_{1 \cup \Upsilon}(s_2), x_2 \xrightarrow{s_2} y_2$, and $y_1 \sim y_2$.

**Example 1** In automaton $G$ in figure 9, states $q_0$ and $q_1$ can be considered as observation equivalent with respect to $\Upsilon = \{\alpha, \beta\}$. Merging these states results in $\tilde{G}$, also shown in figure 9.
Figure 9: Example automata to demonstrate observation equivalence. Uncontrollable events are prefixed with !, and local events have parentheses around them.

Figure 10: Two observation equivalent automata that are not synthesis equivalent.

Unfortunately, observation equivalence in general does not imply synthesis equivalence, so theorem 8 cannot be generalised for observation equivalence [25].

**Example 2** Consider again the observation equivalent automata in figure 9, with $\Sigma_c = \{\alpha, \beta\}$ and $\Sigma_u = \{!\upsilon\}$. The triples $\langle \{G\}; \{G\}; \text{id} \rangle$ and $\langle \{\tilde{G}\}; \{G\}; \text{id} \rangle$ are not synthesis equivalent. With $G$, a supervisor can disable the local controllable event $\alpha$ to prevent entering state $q_1$ and thus the occurrence of the undesirable uncontrollable $!\upsilon$, but this is not possible with $\tilde{G}$. It holds that $\omega \in L(\sup CN(G))$ while $L(\sup CN(\tilde{G})) = \emptyset$.

There are different ways how observation equivalence can be restricted for use in compositional synthesis. The problem in example 2 does not arise if the local events $\alpha$ and $\beta$ are uncontrollable. In fact, a result similar to theorem 8 can be shown if observation equivalence is restricted to uncontrollable events [25]. With controllable events, abstraction is also possible, but two other issues need to be taken into account.

**Example 3** Consider automaton $G$ in figure 10 with $\Sigma_u = \{!\mu, !\upsilon\}$ and $\Upsilon = \{\beta, \gamma, !\mu, !\upsilon\}$. Merging of observation equivalent states results in $\tilde{G}$, but states $q_1$
and $q_2$ should not be merged for synthesis purposes. Although both states can reach the same states via the controllable event $\alpha$, possibly preceded and followed by the local event $\nu$, the transition $q_4 \xrightarrow{\alpha} q_6$ must always be disabled to prevent blocking via the uncontrollable event $\nu$, while the transition $q_1 \xrightarrow{\alpha} q_8$ may be enabled. When used in a system that requires $\alpha$ to occur for correct behaviour, such as $T$ in figure 10, state $q_1$ is retained in synthesis while $q_2$ is removed. The triples $T = (\{G, T\}; \{G, T\}; \text{id})$ and $\tilde{T} = (\{\tilde{G}, T\}; \{G, T\}; \text{id})$ are not synthesis equivalent as $L(\text{supCN}(T)) = \emptyset$ but $\nu \in L(\text{supCN}(\tilde{T}))$.

Example 4 Consider automaton $G$ in figure 11 with $\Sigma_u = \{\nu, \mu\}$ and $\Upsilon = \{\alpha, \beta\}$. Merging of observation equivalent states results in $\tilde{G}$, but states $q_1$ and $q_2$ should not be merged for synthesis purposes. In $G$, states $q_3$ and $q_4$ should be avoided to prevent blocking in state $q_5$ via the uncontrollable event $\nu$. Thus, $\alpha$ should be disabled in $q_1$ and $q_2$, making $q_2$ a blocking state, while $q_1$ remains nonblocking due to the transition $q_1 \xrightarrow{\beta} q_6$. The triples $T = (\{G\}; \{G\}; \text{id})$ and $\tilde{T} = (\{\tilde{G}\}; \{G\}; \text{id})$ are not synthesis equivalent as $\nu /\in L(\text{supCN}(T))$ but $\nu \in L(\text{supCN}(\tilde{T}))$.

The problem in example 3 is caused by considering the path $q_2 \xrightarrow{\nu \alpha \mu} q_9$ as equivalent to $q_1 \xrightarrow{\alpha} q_8$ to justify states $q_1$ and $q_2$ to be merged. However, the path $q_2 \xrightarrow{\nu \alpha \mu} q_9$ passes through the unsafe state $q_6$, while $q_1 \xrightarrow{\alpha} q_8$ does not pass through any unsafe states. This situation can be avoided by only allowing local events before a controllable event. That is, for $x_1 \xrightarrow{\sigma} y_1$ and $x_1 \sim x_2$ it is required that there exists $t \in \Upsilon^*$ such that $x_2 \xrightarrow{t \sigma} y_2$ and $y_1 \sim y_2$. In example 3, the local events in $t$ are all uncontrollable. Controllable events can lead to the problem in example 4. They can be allowed under the additional condition that their target states are equivalent to the start state of the path.
Imposing such conditions on observation equivalence results in synthesis observation equivalence, which preserves synthesis results in a way similar to theorem 8 [25].

Definition 21 [25] Let \( G = (\Sigma, Q, \rightarrow, Q^0) \) be an automaton with \( \Sigma = \Omega \cup \Upsilon \). An equivalence relation \( \sim \subseteq Q \times Q \) is a synthesis observation equivalence on \( G \) with respect to \( \Upsilon \), if the following conditions hold for all \( x_1, x_2 \in Q \) such that \( x_1 \sim x_2 \):

(i) if \( x_1 \xrightarrow{\sigma} y_1 \) for \( \sigma \in \Sigma_c \cup \{ \omega \} \), then there exists a path \( x_2 = x_2^0 \tau_1 \cdots \tau_n \) \( x_2^0 \xrightarrow{P_{0u}(\omega)(\sigma)} y_2 \) such that \( y_1 \sim y_2 \) and \( \tau_1, \ldots, \tau_n \in \Upsilon \), and whenever \( \tau_i \in \Sigma_c \) then \( x_1 \sim x_2^i \);

(ii) if \( x_1 \xrightarrow{u} y_1 \) for \( u \in \Sigma_u \), then there exist \( t_2, u_2 \in (\Upsilon \cap \Sigma_u)^* \) such that \( x_2 \xrightarrow{t_2 P_2(u) u_2} y_2 \) and \( y_1 \sim y_2 \).

Condition (i) allows for a state \( x_1 \) with an outgoing controllable event to be equivalent to another state \( x_2 \), if that state allows the same controllable event, possibly after a sequence of local events. If that sequence includes a controllable transition \( x_2^{i-1} \rightarrow x_2^i \), its target state \( x_2^i \) must be equivalent to the start states \( x_1 \sim x_2 \).

Condition (ii) is similar to observation equivalence, but restricted to uncontrollable events. Projection \( P_2 \) is used in the definition to ensure that the conditions (i) and (ii) apply to both local and shared events.

Example 5 Consider automaton \( G \) in figure 12, with all events controllable and \( \Upsilon = \{ \beta \} \). An equivalence relation with \( q_1 \sim q_3 \) and \( q_4 \sim q_7 \) is a synthesis observation equivalence on \( G \). Merging the equivalent states results in the deterministic automaton \( G' \) shown in figure 12. Note that \( q_1 \) and \( q_2 \) in \( G \) are not synthesis observation equivalent, because for \( q_2 \xrightarrow{\alpha} q_6 \) but only \( q_1 \xrightarrow{\alpha} q_7 \xrightarrow{\beta} q_6 \), and the local event \( \beta \) occurs after the shared event \( \alpha \) on the path.

Synthesis observation equivalence does not allow local events after a controllable event. This condition can be further relaxed, allowing local events after controllable events, provided that it can be guaranteed that the states visited by the local transition after a controllable event are all present in the synthesis result.

Definition 22 [21] Let \( G = (\Sigma, Q, \rightarrow, Q^0) \) be an automaton with \( \Sigma = \Omega \cup \Upsilon \). An equivalence relation \( \sim \subseteq Q \times Q \) is a weak synthesis observation equivalence on \( G \) with respect to \( \Upsilon \), if the following conditions hold for all \( x_1, x_2 \in Q \):

(i) If \( x_1 \xrightarrow{\sigma} y_1 \) for \( \sigma \in \Sigma_c \cup \{ \omega \} \), then there exists a path \( x_2 = x_2^0 \tau_1 \cdots \tau_n \) \( x_2^0 \xrightarrow{P_{0u}(\omega)(\sigma)} y_2^0 \tau_{n+1} \cdots \tau_m \) \( y_2^m = y_2 \) such that \( y_1 \sim y_2 \) and \( \tau_1, \ldots, \tau_m \in \Upsilon \) and,
Figure 12: Example of synthesis observation equivalence and weak synthesis observation equivalence.

a) whenever $\tau_i \in \Sigma_c$ for some $i \leq n$ then $x_1 \sim x_2$;

b) whenever $y_i^j \xrightarrow{u} z$ for some $u \in (\Sigma_u \cap \Upsilon)^*$ then $z \sim y_j^i$ for some $0 \leq j \leq m$;

c) whenever $y_i^j \xrightarrow{u} z$ for some $u \in \Sigma_u^*$ such that $P_{\Omega}(u) \in \Sigma_u \setminus \Upsilon$, then there exists $u' \in \Sigma_u^*$ such that $P_{\Omega}(u) = P_{\Omega}(u')$ and $y_2 \xrightarrow{u'} z'$ for some $z' \sim z$.

(ii) If $x_1 \xrightarrow{v} y_1$ for $v \in \Sigma_u$, then there exist $t, u_2 \in (\Upsilon \cap \Sigma_u)^*$ such that $x_2 \xrightarrow{t', \Pi(v)u_2} y_2$ and $y_1 \sim y_2$.

Condition (i) weakens the condition for controllable events in that it allows for a path of local events after a controllable event, if local uncontrollable transitions outgoing from the path lead to a state equivalent to a state on the path, and shared uncontrollable transitions are also possible in the end state of the path. Condition (ii) is the same as for synthesis observation equivalence.

Example 6 Consider again automaton $G$ in figure 12, with all events controllable and $\Upsilon = \{\beta\}$. An equivalence relation with $q_1 \sim q_2 \sim q_3$ and $q_4 \sim q_7$ is a weak synthesis observation equivalence on $G$, producing the abstraction $\tilde{G} = G/\sim$. For example, states $q_1$ and $q_2$ can be equivalent as $q_2 \xrightarrow{\alpha} q_6$ and $q_1 \xrightarrow{\alpha} q_7 \xrightarrow{\beta} q_6$. The nondeterminism in $\tilde{G}$ can be avoided using a renaming $\rho$: $\{\alpha_1, \alpha_2, \gamma, \beta\} \rightarrow \{\alpha, \gamma, \beta\}$, which leads to the deterministic automaton $\tilde{G}'$ in figure 12.
Both synthesis observation equivalence and weak synthesis observation equivalence can be used for abstraction steps in compositional synthesis. After computing an appropriate equivalence relation \( \sim \) on a renamed automaton \( \rho(G_1) \), the automaton \( G_1 \) can be replaced by its quotient \( G_1/\sim \).

**Theorem 9** [21] Let \((G; S; \rho)\) be a synthesis triple with \( G = \{G_1, \ldots, G_n\} \) and \( G_i = (\Sigma_i, Q_i, \rightarrow_i, Q_i^o) \). Let \( \Upsilon \subseteq \Sigma_1 \) such that \((\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset \). Let \( \sim \) be a synthesis observation equivalence or a weak synthesis observation equivalence relation on \( \rho(G_1) \) with respect to \( \Upsilon \) such that \( G_1/\sim \) is deterministic, and let \( \hat{G} = \{G_1/\sim, G_2, \ldots, G_n\} \). Then \((\hat{G}; S; \rho) \cong_{\text{synth}} (G; S; \rho)\).

**Complexity.** Observation equivalence-based abstractions can be computed in polynomial time. The time complexity to compute a bisimulation is \( O(|\rightarrow| \log |Q|) \) [11]. Synthesis observation equivalence and weak synthesis observation equivalence are computed by a modified version of the same algorithm in \( O(|\rightarrow||Q|^4) \) and \( O(|\rightarrow||Q|^5) \) time, respectively [21].

### 5.3 Halfway Synthesis

**Halfway synthesis** is an abstraction method that works well in compositional synthesis [13]. Sometimes it is clear that certain states in an automaton must be removed in synthesis, no matter what the behaviour of the rest of the system is. Clearly, blocking states can never become nonblocking. Moreover, local uncontrollable transitions to blocking states must be removed, because no other component nor the supervisor can disable a local uncontrollable transition.

**Definition 23** Let \( G = (\Sigma, Q, \rightarrow, Q^o) \) and \( \Upsilon \subseteq \Sigma \). The halfway synthesis result for \( G \) with respect to \( \Upsilon \) is

\[
\text{hsupCN}_\Upsilon(G) = (\Sigma, Q \cup \{\perp\}, \rightarrow_{\text{hsup}}, Q^o),
\]

where \( \text{supCN}_\Upsilon(G) = (\Sigma, Q, \rightarrow_{\text{sup}}, Q^o), \perp \notin Q, \) and

\[
\rightarrow_{\text{hsup}} = \rightarrow_{\text{sup}} \cup \{ (x, \sigma, \perp) \mid \sigma \in \Sigma_u \setminus \Upsilon, x \xrightarrow{\sigma}, \text{ and } x \xrightarrow{\sigma_{\text{sup}}} \text{ does not hold} \}.
\]

Halfway synthesis is calculated like ordinary synthesis, but considering only local events as uncontrollable. Shared uncontrollable transitions to blocking states do not necessarily cause blocking, as some other plant component may yet disable them. Therefore, these transitions are retained and redirected to the blocking state \( \perp \) instead.
Example 7 Consider automaton $G$ in Figure 13 with $\Sigma_u = \{!\lambda, !\mu, !\upsilon\}$ and $\Upsilon = \{\gamma, !\lambda\}$. State $q_3$ is blocking, so $q_2$ is also considered as unsafe, because the uncontrollable $!\lambda$-transition cannot be disabled by the supervisor nor by any other plant component. Every nonblocking supervisor can and will disable the controllable transitions $q_1 \xrightarrow{\alpha} q_3$ and $q_1 \xrightarrow{\beta} q_2$. State $q_0$ may still be safe, because some other plant component may disable the shared events $!\mu$ and $!\upsilon$. The blocking state $\bot$ is added and the $!\mu$- and $!\upsilon$-transitions are redirected to $\bot$ in the halfway synthesis result $\text{hsupCN}_{\{!\lambda\}}(G)$, see Figure 13. This ensures that later synthesis is aware of the potential problem regarding $!\mu$ or $!\upsilon$.

The following theorem extends a result about halfway synthesis for supervision equivalence using state labels [13] to the more general framework of synthesis triples. The proof can be found in appendix C.

Theorem 10 Let $(G; S; \rho)$ be a synthesis triple with $G = \{G_1, \ldots, G_n\}$, and let $\Upsilon \subseteq \Sigma_1 \cap \Sigma_u$ such that $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset$. Then

$$(G; S; \rho) \simeq_{\text{synth}} \left(\{\text{hsupCN}_{\Upsilon}(G_1), G_2, \ldots, G_n\}; \text{hsupCN}_{\Upsilon}(G_1) \right) \cup S; \rho \right).$$

Complexity. Halfway synthesis can be achieved using a standard synthesis algorithm and runs in time complexity $O(|Q|\rightarrow|)$, where $|Q|$ and $|\rightarrow|$ are the numbers of states and transitions of the input automaton.

6 Compositional Synthesis Algorithm

Given a set of plant automata $\mathcal{G}$, the compositional synthesis algorithm repeatedly composes automata and applies abstraction rules. While doing so, it modifies a synthesis triple $(\mathcal{G}; S; \rho)$, collecting supervisors in $S$ and updating the renaming $\rho$, and continues until only one automaton that cannot be further abstracted is left. Then a standard synthesis algorithm is used to compute a final supervisor. This principle, which is justified by theorem 3, is shown in Algorithm 1.
Algorithm 1 Compositional synthesis

1: input $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$
2: $S \leftarrow \mathcal{G}$, $\rho \leftarrow \text{id}$
3: while $|\mathcal{G}| > 1$ do
4: $\mathcal{G} \leftarrow \text{selfloopRemoval}(\mathcal{G})$
5: subsys $\leftarrow \text{selectSubSystem}(\mathcal{G})$
6: $\mathcal{G} \leftarrow \mathcal{G} \setminus \text{subsys}$
7: $A \leftarrow \text{synchronousComposition}(\text{subsys})$
8: $\Upsilon \leftarrow \Sigma_A \setminus \Sigma_{\mathcal{G}}$
9: $A \leftarrow \text{hsupCN}_{\Upsilon \cap \Sigma_u}(A)$
10: $S \leftarrow S \cup \{A\}$
11: $A \leftarrow \text{bisimulation}(A)$
12: $\bar{A} \leftarrow \text{WSOE}_{\Upsilon}(A)$
13: if $\bar{A}$ is deterministic then
14: $\mathcal{G} \leftarrow \mathcal{G} \cup \{\bar{A}\}$
15: else
16: $\langle \rho_D, \bar{D}, D \rangle \leftarrow \text{makeDistinguisher}(\bar{A}, A)$
17: $\mathcal{G} \leftarrow \rho_D^{-1}(\mathcal{G}) \cup \{\bar{D}\}$, $S \leftarrow \rho_D^{-1}(S) \cup \{D\}$, $\rho \leftarrow \rho \circ \rho_D$
18: end if
19: end while
20: $S \leftarrow S \cup \{\text{supCN}(\mathcal{G})\}$
During each iteration of the main loop, a series of steps is applied to simplify the set \( G \) of plant automata. First, line 4 applies selfloop removal to the entire plant \( G \) according to theorem 7. This quick operation improves the performance of the following steps.

The next step is to choose a subsystem of \( G \) for simplification. If no automaton can be simplified individually, a group of automata is selected for composition. The \texttt{selectSubSystem()} method in line 5 selects an appropriate subsystem, which is then removed from \( G \) and composed. Different methods to select this subsystem have been investigated in previous work [12,14]. Here, the strategy \texttt{MustL} is used, which facilitates the exploitation of local events. For each event \( \sigma \), a subsystem is formed by considering all automata with \( \sigma \) in the alphabet, so \( \sigma \) becomes a local event after composing the subsystem. This gives several candidate subsystems, one for each event, so a second step applies a strategy called \texttt{MinSync}, which chooses the subsystem with the smallest number of states in its synchronous composition.

After identification and composition of a subsystem, the set \( \Upsilon \) of local events is formed in line 8, which contains the events used only in the subsystem to be simplified. Based on the local events, the abstraction rules given in Theorems 8–10 are applied in lines 9–12. Rules of lower complexity are applied first, so halfway synthesis is followed by bisimulation and weak synthesis observation equivalence. If halfway synthesis produces a new supervisor, it is added to the set \( S \) of supervisors. If weak synthesis observation equivalence results in a deterministic abstracted automaton, this automaton is added back into the set \( G \) of uncontrolled plants.

Weak synthesis observation equivalence may also result in nondeterminism, if some states in an equivalence class have successor states reached by the same event, but belonging to different equivalence classes. In this case, a renaming is introduced. The \texttt{makeDistinguisher()} method in line 16 replaces the events of any transitions causing nondeterminism in the abstracted automaton \( \tilde{A} \) by new events and records the target states of these transitions. Using the recorded target states, the same modification to corresponding transitions is applied to the original automaton \( A \). The \texttt{makeDistinguisher()} method returns a renaming map \( \rho_\text{D} \), the deterministic abstracted automaton \( \tilde{D} \), and an appropriate distinguisher \( D \). In line 17, the inverse renaming \( \rho_\text{D}^{-1} \) is applied to the entire system \( G \) and the collected supervisors \( S \), the abstracted automaton \( \tilde{D} \) and the distinguisher \( D \) are added to the resultant automata sets, and the renaming \( \rho \) is updated to include \( \rho_\text{D} \). This is equivalent to the application of theorem 6 followed by theorem 9.

The loop terminates when the set \( G \) of uncontrolled plants contains only a single automaton, which is passed to standard synthesis in line 20. According to theorem 5, the result is added to the set \( S \), which in combination with the final renaming \( \rho \) gives the least restrictive, controllable and nonblocking supervisor for the original system \( G \).
7 Experimental Results

The compositional synthesis algorithm has been implemented in the DES software tool Supremica [2]. The algorithm is completely automatic and does not use any prior knowledge about the structure of the system. The implementation has successfully computed modular supervisors for several large discrete event systems models. The test cases include the following complex industrial models and case studies, which are taken from different application areas such as manufacturing systems and automotive body electronics:

- **agv** Automated guided vehicle coordination based on the Petri net model in [27]. To make the example blocking in addition to uncontrollable, there is also a variant, **agvb**, with an additional zone added at the input station.

- **aip** Automated manufacturing system of the Atelier Inter-établissement de Productique [4].

- **fencaiwon09** Model of a production cell in a metal-processing plant from [9].

- **fms** Large-scale flexible manufacturing system based on [38].

- **tbed** Model of a toy railroad system based on [16]. Two versions present different control objectives.

- **verriegel** Models of the central locking system of a BMW car. There are two variants, a three-door model **verriegel3**, and a four-door model **verriegel4**. These models are derived from the KORSYS project [29].

- **6link** Models of a cluster tool for wafer processing previously studied for synthesis in [34].

All the test cases considered have at least $10^7$ reachable states in their synchronous product and are either uncontrollable, blocking, or both. Algorithm 1 has been used to compute modular supervisors for each of these models. In addition to section 6, the algorithm is controlled by a state limit of 5000 states: if the synchronous composition of a subsystem in line 7 exceeds 5000 states, that subsystem is discarded and another subsystem is chosen instead. All experiments have been run on a standard desktop PC using a single 2.66 GHz microprocessor.

The results of the experiments are shown in Table 1. For each model, the table shows the number of automata (Aut), the number of reachable states (Size), and whether the model is nonblocking (Nonb.) or controllable (Cont.). Next, the table shows the size of the largest synchronous composition encountered during
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>agv</td>
<td>16</td>
<td>2.6·10^7</td>
<td>true</td>
<td>false</td>
<td>856</td>
<td>3.11</td>
<td>27.9</td>
<td>6</td>
<td>12339</td>
<td>0</td>
<td>30</td>
<td>208</td>
</tr>
<tr>
<td>agvb</td>
<td>17</td>
<td>2.3·10^7</td>
<td>false</td>
<td>false</td>
<td>562</td>
<td>0.81</td>
<td>61.3</td>
<td>7</td>
<td>9380</td>
<td>0</td>
<td>30</td>
<td>187</td>
</tr>
<tr>
<td>aip0alps</td>
<td>35</td>
<td>3.0·10^8</td>
<td>false</td>
<td>true</td>
<td>502</td>
<td>0.43</td>
<td>84.3</td>
<td>3</td>
<td>17</td>
<td>2</td>
<td>53</td>
<td>3</td>
</tr>
<tr>
<td>fencaiwon09b</td>
<td>29</td>
<td>8.9·10^7</td>
<td>false</td>
<td>true</td>
<td>182</td>
<td>0.27</td>
<td>118.4</td>
<td>6</td>
<td>917</td>
<td>4</td>
<td>56</td>
<td>3</td>
</tr>
<tr>
<td>fencaiwon09s</td>
<td>29</td>
<td>2.9·10^8</td>
<td>false</td>
<td>false</td>
<td>525</td>
<td>0.44</td>
<td>150.2</td>
<td>11</td>
<td>436</td>
<td>5</td>
<td>59</td>
<td>186</td>
</tr>
<tr>
<td>fms2003s</td>
<td>31</td>
<td>1.4·10^7</td>
<td>false</td>
<td>true</td>
<td>2596</td>
<td>23.6</td>
<td>332.8</td>
<td>4</td>
<td>59109</td>
<td>36</td>
<td>52</td>
<td>64</td>
</tr>
<tr>
<td>tbed-noderaib</td>
<td>84</td>
<td>3.1·10^12</td>
<td>false</td>
<td>true</td>
<td>4989</td>
<td>6.22</td>
<td>265.2</td>
<td>17</td>
<td>26</td>
<td>0</td>
<td>12</td>
<td>158</td>
</tr>
<tr>
<td>tbed-uncont</td>
<td>84</td>
<td>3.6·10^12</td>
<td>true</td>
<td>false</td>
<td>4479</td>
<td>5.34</td>
<td>491.6</td>
<td>10</td>
<td>19737</td>
<td>1</td>
<td>1</td>
<td>190</td>
</tr>
<tr>
<td>verriegel3b</td>
<td>52</td>
<td>1.3·10^9</td>
<td>false</td>
<td>true</td>
<td>1367</td>
<td>1.80</td>
<td>218.2</td>
<td>1</td>
<td>4</td>
<td>77</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>verriegel4b</td>
<td>64</td>
<td>6.2·10^10</td>
<td>false</td>
<td>true</td>
<td>1382</td>
<td>4.86</td>
<td>250.5</td>
<td>1</td>
<td>4</td>
<td>21</td>
<td>71</td>
<td>189</td>
</tr>
<tr>
<td>6linka</td>
<td>53</td>
<td>2.4·10^14</td>
<td>false</td>
<td>true</td>
<td>3614</td>
<td>19.52</td>
<td>515.3</td>
<td>13</td>
<td>2073</td>
<td>15</td>
<td>48</td>
<td>1754</td>
</tr>
<tr>
<td>6linki</td>
<td>53</td>
<td>2.7·10^14</td>
<td>false</td>
<td>true</td>
<td>2925</td>
<td>1372</td>
<td>635.4</td>
<td>12</td>
<td>4017</td>
<td>12</td>
<td>49</td>
<td>1205</td>
</tr>
<tr>
<td>6linkp</td>
<td>48</td>
<td>4.2·10^14</td>
<td>false</td>
<td>true</td>
<td>3614</td>
<td>26.62</td>
<td>538.3</td>
<td>17</td>
<td>2073</td>
<td>25</td>
<td>45</td>
<td>1731</td>
</tr>
<tr>
<td>6linkre</td>
<td>59</td>
<td>6.2·10^14</td>
<td>false</td>
<td>true</td>
<td>240</td>
<td>1.01</td>
<td>584.9</td>
<td>19</td>
<td>375</td>
<td>10</td>
<td>51</td>
<td>221</td>
</tr>
</tbody>
</table>
abstraction (Peak States), the total runtime (Time), the total amount of memory used (Mem.), the number of modular supervisors computed (Num.) and the number of states of the largest supervisor automaton (Largest). The table furthermore shows the number of events replaced by renaming (Ren.) and the number of events removed by selfloop removal (SR), and finally the number of states removed by halfway synthesis (HS), bisimulation (Bis.), and weak synthesis observation equivalence (WSOE).

All examples have been solved successfully with no more than 30 seconds runtime, and never using more than 640 MB of memory, even for models with more than $10^{14}$ reachable states. It is worth mentioning that other methods for selecting subsystems give smaller supervisors for the agv and tbed examples. However, persistently good results can be achieved for all the examples in this test with the considered strategy MustL/MinSync.

Figure 14 shows some data concerning the performance of the abstraction rules. For each example, it shows the ratio of the number of states removed by each rule over the total number of states removed, and the ratio of the runtime consumed by each rule over the total runtime of all abstraction rules. Particularly for large models, halfway synthesis and also bisimulation run much faster than weak synthesis observation equivalence, as is expected from the higher complexity class. However, weak synthesis observation equivalence also has the highest percentage of states removed and typically contributes most of the states removed by abstraction. The data suggests a correlation between the percentage of runtime and the percentage of states removed by each rule. By this measure, the three abstraction rules have similar performance in practice.

The compositional synthesis algorithm is also applied to the transfer line example [37]. The model consists of a parametrised number of serially connected cells, each consisting of a machine, a test unit, and two buffers. The output of one cell is the input of the next cell. This model can easily be scaled up to arbitrary size. Its state space grows exponentially, and the number of reachable states of the controlled system is approximately $1.2 \cdot 14.62^n$ where $n$ is the number of cells [5]. Yet, the cells are identical and the real complexity of the system is small.

Although the compositional synthesis algorithm has no knowledge of the symmetry of the model and treats each subsystem as if it was unique, it successfully computes modular supervisors for transfer lines with up to 1000 serially connected cells. Figure 15 shows a linear relation between the number of connected cells and the total number of supervisor states. The algorithm never constructs a supervisor component with more than 79 states. The relation between the number of cells and the execution time is quadratic. This behaviour is due to the complexity of evaluating and choosing subsystems from growing lists. This experiment shows that the compositional synthesis algorithm automatically discovers that the cells
are identical and produces identical supervisors accordingly.

8 Conclusions

A general framework for compositional synthesis in supervisory control has been presented, which supports the synthesis of least restrictive, controllable, and non-blocking supervisors for large models consisting of several automata that synchronise in lock-step synchronisation. The framework supports compositional reasoning using different kinds of abstractions that are guaranteed to preserve the final synthesis result, even when applied to individual components. Hiding and non-determinism are avoided, solving problems in previous related work. The computed supervisor is modular in that it typically consists of several interacting components, which means that it is easy to understand and implement. The algorithm
has been implemented, and experimental results show that the method successfully computes modular supervisors for a set of large industrial models.

In future work, the authors would like to generalise the framework to consider unobservable events. Furthermore, finite-state machines augmented with bounded integer variables show good modelling potential, and it is of interest to adapt the described compositional synthesis approach to work directly with this type of modelling formalism.

References


[29] **KORSys Project:** [http://www4.in.tum.de/proj/korsys/](http://www4.in.tum.de/proj/korsys/)


A Proofs for Renaming and Selfloop Removal

This appendix contains proofs for theorem 6 and theorem 7 in section 5.1. As a prerequisite for theorem 6, it is first confirmed that every renaming step

\[ (\mathcal{G}_1; \rho_1) \simeq_{\text{synth}} (\mathcal{G}_2; \{H_1\} \cup \rho^{-1}(S); \rho_1 \circ \rho) \]  

produces a proper synthesis triple.

**Lemma 11** Let \((\mathcal{G}_1; \rho_1)\) be a synthesis triple with \(\mathcal{G}_1 = \{G_1, \ldots, G_n\}\), let \(\rho\) be a renaming, and let \(H_1\) be a \(\rho\)-distinguisher such that \(\rho(H_1) = G_1\) and \(\mathcal{G}_2 = \{H_1, \rho^{-1}(G_2), \ldots, \rho^{-1}(G_n)\}\). Then \((\mathcal{G}_2; \{H_1\} \cup \rho^{-1}(S); \rho_1 \circ \rho)\) is a synthesis triple.

**Proof.** It is necessary to prove properties (i), (ii), and (iii) in definition 14.

(i) As \((\mathcal{G}_1; \rho_1)\) is a synthesis triple, it holds that \(\mathcal{L}(S) \subseteq \mathcal{L}(\mathcal{G}_1)\). Then it follows that \(\mathcal{L}(\{H_1\} \cup \rho^{-1}(S)) = \mathcal{L}(H_1 \parallel \rho^{-1}(S)) \subseteq \mathcal{L}(H_1 \parallel \rho^{-1}(\mathcal{G}_1)) = \mathcal{L}(H_1 \parallel \rho^{-1}(G_1) \parallel \cdots \parallel \rho^{-1}(G_n)) = \mathcal{L}(\mathcal{G}_2)\).

(ii) It needs to be shown that \(H_1 \parallel \rho^{-1}(S)\) is a \((\rho_1 \circ \rho)\)-distinguisher. Let \(s, t \in \mathcal{L}(H_1 \parallel \rho^{-1}(S))\) such that \(\rho_1(\rho(s)) = \rho_1(\rho(t))\). Then \(s, t \in \mathcal{L}(\rho^{-1}(S)) = \rho^{-1}(\mathcal{L}(S))\), and thus \(\rho(s), \rho(t) \in \rho(\rho^{-1}(\mathcal{L}(S))) = \mathcal{L}(S)\). Since \(\rho_1(\rho(s)) = \rho_1(\rho(t))\) and \(S\) is a \(\rho_1\)-distinguisher, it follows that \(\rho(s) = \rho(t)\). Further, since also \(s, t \in \mathcal{L}(H_1)\) and \(H_1\) is a \(\rho\)-distinguisher, it follows that \(s = t\).

(iii) Let \(\gamma_1, \gamma_2\) such that \((\rho_1 \circ \rho)(\gamma_1) = (\rho_1 \circ \rho)(\gamma_2)\). It needs to be shown that there exists at most one automaton in \(\mathcal{G}_2\) that differentiates between \(\gamma_1\) and \(\gamma_2\). This is clear when \(\gamma_1 = \gamma_2\), so assume that \(\gamma_1 \neq \gamma_2\). Since \((\mathcal{G}_1; \rho_1)\) is a synthesis triple and \(\rho_1(\rho(\gamma_1)) = \rho_1(\rho(\gamma_2))\), there exists at most one automaton \(G_i \in \mathcal{G}_1\) that differentiates between \(\rho(\gamma_1)\) and \(\rho(\gamma_2)\). Write \(H_{1_j} = \rho^{-1}(G_j)\) for \(j = 2, \ldots, n\), so that \(\mathcal{G}_2 = \{H_1, \ldots, H_n\}\). It is shown that the automata \(H_{1_j}\) with \(j \neq i\) do not differentiate between \(\gamma_1\) and \(\gamma_2\).

First consider the case \(j = 1\), so assume that \(G_1\) does not differentiate between \(\rho(\gamma_1)\) and \(\rho(\gamma_2)\). Then the following are equivalent. It holds that \(x \xrightarrow{\gamma_1} y\) in \(H_1\), if and only if \(x \xrightarrow{\rho(\gamma_1)} y\) in \(G_1 = \rho(H_1)\), if and only if \(x \xrightarrow{\rho(\gamma_2)} y\) in \(G_1\) as \(G_1\) does not differentiate between \(\rho(\gamma_1)\) and \(\rho(\gamma_2)\), if
Lemma 12 Let $\rho : \Sigma' \rightarrow \Sigma$ be a renaming, let $A'$ be an automaton with alphabet $\Sigma_A \subseteq \Sigma'$, and let $B$ be an automaton with alphabet $\Sigma_B \subseteq \Sigma$. Then $\rho(A') \parallel B = \rho(A' \parallel \rho^{-1}(B))$.

Proof. It is enough to show that the automata $\rho(A') \parallel B$ and $\rho(A' \parallel \rho^{-1}(B))$ have the same transition relations.

First let $(x_A, x_B) \xrightarrow{\sigma} \rho(A') \parallel B (y_A, y_B)$. Consider three cases. If $\sigma \in \Sigma_{\rho(A')} \cap \Sigma_B$ then $x_A \xrightarrow{\sigma} \rho(A') y_A$ and $x_B \xrightarrow{\sigma} B y_B$. This means that there exists $\sigma' \in \Sigma'$ such that $\rho(\sigma') = \sigma$ and $x_A \xleftarrow{\sigma'} A' y_A$. Since $x_B \xrightarrow{\sigma_B} y_B$, by definition 13 it holds that $x_B \xrightarrow{\sigma_B} \rho^{-1}(B) y_B$ which implies $(x_A, x_B) \xrightarrow{\sigma} A' \parallel \rho^{-1}(B) (y_A, y_B)$. If $\sigma \in \Sigma_{\rho(A')} \setminus \Sigma_B$ then $x_B = y_B$ and $x_A \xrightarrow{\sigma} \rho(A') y_A$. This means that there exists $\sigma' \in \Sigma_A \setminus \Sigma_B$ such that $\rho(\sigma') = \sigma$ and $x_A \xleftarrow{\sigma'} A' y_A$, which implies $(x_A, x_B) \xrightarrow{\sigma} A' \parallel \rho^{-1}(B) (y_A, y_B)$. If $\sigma \in \Sigma_B \setminus \Sigma_{\rho(A')}$ then $x_A = y_A$ and $x_B \xrightarrow{\sigma} B y_B$. This means that there exists $\sigma' \in \Sigma_{\rho^{-1}(B)} \setminus \Sigma_A$ such that $\rho(\sigma') = \sigma$, and by definition 13 it holds that $x_B \xrightarrow{\sigma'} \rho^{-1}(B) y_B$, which implies $(x_A, x_B) \xrightarrow{\sigma} A' \parallel \rho^{-1}(B) (y_A, y_B)$. Thus, in all cases $(x_A, x_B) \xrightarrow{\sigma} A' \parallel \rho^{-1}(B) (y_A, y_B)$. Then it follows that $(x_A, x_B) \xrightarrow{\rho(\sigma')} \rho(A' \parallel \rho^{-1}(B)) (y_A, y_B)$, which furthermore implies $(x_A, x_B) \xrightarrow{\rho(\sigma')} \rho(A' \parallel \rho^{-1}(B)) (y_A, y_B)$.

Conversely, let $(x_A, x_B) \xrightarrow{\rho(\sigma')} \rho(A' \parallel \rho^{-1}(B)) (y_A, y_B)$. Then there exists $\sigma' \in \Sigma'$ such that $\rho(\sigma') = \sigma$ and $(x_A, x_B) \xrightarrow{\sigma'} A' \parallel \rho^{-1}(B) (y_A, y_B)$. There are three possibilities. If $\sigma' \in \Sigma_A \cap \Sigma_{\rho^{-1}(B)}$ then $x_A \xrightarrow{\sigma'} A' y_A$, which implies $x_A \xrightarrow{\rho(\sigma')} \rho(A') y_A$, and also $x_B \xrightarrow{\sigma'} \rho^{-1}(B) y_B$, which implies $x_B \xrightarrow{\rho(\sigma')} \rho^{-1}(B) y_B$ by definition 13. Therefore, $(x_A, x_B) \xrightarrow{\rho(\sigma')} \rho(A') \parallel B (y_A, y_B)$. If $\sigma' \in \Sigma_A \setminus \Sigma_{\rho^{-1}(B)}$ then $x_B = y_B$ and $x_A \xrightarrow{\sigma'} A' y_A$, which implies $x_A \xrightarrow{\rho(\sigma')} \rho(A') y_A$. Also $\rho(\sigma') \notin \Sigma_B$ as $\sigma' \notin \Sigma_{\rho^{-1}(B)}$, and thus
Thus, the claim follows from definition 16. □

By lemma 12 and 13, it holds that

\[(x, y) \xrightarrow{\sigma'} y \] if and only if \( x \xrightarrow{\rho(\sigma')} y \) holds. Also \(\rho(\sigma') \notin \Sigma\), as \(\sigma' \notin \Sigma\), and thus \( (x, y) \xrightarrow{\rho(\sigma')} \). Thus, in all cases \( (x, y) \xrightarrow{\rho(\sigma')} \). □

Lemma 13 Let \( G \) be an automaton with alphabet \( \Sigma \), and let \( \rho : \Sigma \to \Sigma' \) be a renaming. Then \(\rho(\supCN(G)) = \supCN(\rho(G))\).

Proof. Since \(\rho\) preserves controllability, it follows from definition 8 that \(\Theta = \Theta_{\rho(G)}\). Thus by theorem 2,

\[\rho(\supCN(G)) = \rho(G)_{\rho(G)} = \supCN(\rho(G)).\] □

Theorem 6 Let \((G_1; S; \rho_1)\) be a synthesis triple with \(G_1 = \{G_1, \ldots, G_n\}\), let \(\rho\) be a renaming, and let \(H_1\) be a \(\rho\)-distinguisher such that \(\rho(H_1) = G_1 \) and \(G_2 = \{H_1, \rho^{-1}(G_2), \ldots, \rho^{-1}(G_n)\}\). Then

\[(G_1; S; \rho_1) \sim_{\text{synth}} (G_2; \{H_1\} \cup \rho^{-1}(S); \rho_1 \circ \rho).\]

Proof. By definition 15, it holds that

\[\supCN(G_1; S; \rho_1) = \rho_1(\supCN(G_1) \parallel S) = \rho_1(\supCN(G_1 \parallel \cdots \parallel G_n \parallel S)).\] (17)

By lemma 12 and 13, it holds that

\[\supCN(G_1 \parallel \cdots \parallel G_n) = \supCN(H_1 \parallel G_2 \parallel \cdots \parallel G_n)\]

\[= \supCN(H_1 \parallel \rho^{-1}(G_2) \parallel \cdots \parallel \rho^{-1}(G_n))\]

\[= \rho(\supCN(H_1 \parallel \rho^{-1}(G_2) \parallel \cdots \parallel \rho^{-1}(G_n))).\] (18)

Combining these equations gives

\[\mathcal{L}(\supCN(G_1; S; \rho_1))\]

\[= \mathcal{L}(\rho_1(\supCN(G_1 \parallel \cdots \parallel G_n \parallel S))\]

\[= \mathcal{L}(\rho_1(\rho(\supCN(H_1 \parallel \rho^{-1}(G_2) \parallel \cdots \parallel \rho^{-1}(G_n)) \parallel S))\]

\[= \mathcal{L}(\rho_1(\rho(\supCN(H_1 \parallel \rho^{-1}(G_2) \parallel \cdots \parallel \rho^{-1}(G_n)) \parallel H_1 \parallel \rho^{-1}(S))))\]

\[= \mathcal{L}(\supCN(G_2; \{H_1\} \cup \rho^{-1}(S); \rho_1 \circ \rho)).\] (19)

Thus, the claim follows from definition 16. □
This completes the proof for the correctness of renaming. Next, considering selfloop removal, the proof for theorem 7 uses two lemmas that show the relationship between selfloop removal and synthesis.

**Lemma 15** Let automaton $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ with $\Sigma = \Omega \cup \Lambda$ be selfloop-only for $\Lambda$. Then $\hat{\Theta}_G = \hat{\Theta}_{G,\Omega}$.

**Proof.** In the following, let $\hat{\Theta}_G = \hat{\Theta}_{G,\Omega}$. First, it is shown by induction on $n \geq 0$ that $\hat{\Theta}_G \subseteq X^n_{\Omega} = \Theta^n_{\Omega}(Q)$.

**Base case.** $n = 0$. Clearly $\hat{\Theta}_G \subseteq Q = \Theta^0_{\Omega}(Q) = X^0_{\Omega}$.

**Inductive step.** Let $x \in \hat{\Theta}_G \subseteq X^n_{\Omega}$ by inductive assumption. It must be shown that $x \in X^{n+1}_{\Omega} = \Theta^\text{conf}_{\Omega}(X^n_{\Omega}) \cap \Theta^\text{nonb}_{\Omega}(X^n_{\Omega})$.

To see that $x \in \Theta^\text{conf}_{\Omega}(X^n_{\Omega})$, let $v \in \Sigma_n$ and $x \xrightarrow{v} y$. Since every transition in $G_{\Omega}$ also is in $G$, it holds that $x \xrightarrow{v} y$. Since $x \in \hat{\Theta}_G$, it follows by controllability that $y \in \hat{\Theta}_G$. By inductive assumption $y \in X^n_{\Omega}$, which implies $x \in \Theta^\text{conf}_{\Omega}(X^n_{\Omega})$.

Next it is shown that $x \in \Theta^\text{nonb}_{\Omega}(X^n_{\Omega})$. Since $x \in \hat{\Theta}_G$, there exists a path

$$x = x_0 \xrightarrow{\sigma_1} \hat{\Theta}_G x_1 \xrightarrow{\sigma_2} \hat{\Theta}_G \cdots \xrightarrow{\sigma_k} \hat{\Theta}_G x_k \xrightarrow{\omega} \hat{\Theta}_G x_{k+1}. \quad (20)$$

Consider the first transition in (20). If $\sigma_1 \in \Lambda$ then $x_0 = x_1 \in \hat{\Theta}_G$. If $\sigma_1 \notin \Lambda$ then $x_0 \xrightarrow{\sigma_1} x_1$ where $x_1 \in \hat{\Theta}_G$. In both cases, $x_1 \in \hat{\Theta}_G \subseteq X^n_{\Omega}$ by inductive assumption. By induction, it follows that

$$x = x_0 \xrightarrow{P_1(\sigma_1)} X^n_{\Omega} x_1 \xrightarrow{P_1(\sigma_2)} X^n_{\Omega} \cdots \xrightarrow{P_1(\sigma_k)} X^n_{\Omega} x_k \xrightarrow{\omega} X^n_{\Omega} x_{k+1}. \quad (21)$$

Thus, $x \in \Theta^\text{nonb}_{\Omega}(X^n_{\Omega})$.

Conversely, it is shown by induction on $n \geq 0$ that $\hat{\Theta}_G \subseteq X^n = \Theta^n_G(Q)$.

**Base case.** $n = 0$. Clearly $\hat{\Theta}_G \subseteq Q = \Theta^0_G(Q) = X^0$.

**Inductive step.** Let $x \in \hat{\Theta}_G \subseteq X^n$ by inductive assumption. It must be shown that $x \in X^{n+1} = \Theta^\text{conf}_G(X^n) \cap \Theta^\text{nonb}_G(X^n)$.

To see that $x \in \Theta^\text{conf}_G(X^n)$, let $v \in \Sigma_n$ and $x \xrightarrow{v} y$. If this transition is not in $G_{\Omega}$, it follows that $v \in \Lambda$ and $y = x \in X^n$. If $x \xrightarrow{v} y$, since $x \in \hat{\Theta}_G$, it follows by controllability that $y \in \hat{\Theta}_G$. By inductive assumption $y \in X^n$, which implies $x \in \Theta^\text{conf}_G(X^n)$.

Next it is shown that $x \in \Theta^\text{nonb}_G(X^n)$. Since $x \in \hat{\Theta}_G$, there exists a path $x = x_0 \xrightarrow{\omega} X^n$. Since every transition in $G_{\Omega}$ also is in $G$ and by inductive assumption, it follows that $x = x_0 \xrightarrow{\omega} X^n$. Hence, $x \in \Theta^\text{nonb}_G(X^n)$. 

\[\square\]
Lemma 16 Let $G = \langle \Sigma, Q, \to, Q^o \rangle$ with $\Sigma = \Omega \cup \Lambda$ be a deterministic automaton that is selfloop-only for $\Lambda$. Then $\supCN(G) = \supCN(G|_{\Omega}) \parallel G$.

Proof. By definition 17, $G|_{\Omega} = \langle \Omega, Q, \to|_{\Omega}, Q^o \rangle$ where $\to|_{\Omega} = \{ (x, \sigma, y) \in \to \mid \sigma \in \Omega \}$. Let $\Theta|_{\Omega} = \Theta_{G|_{\Omega}}$. The following proof exploits the fact that $G$ and thus also $\supCN(G)$ are deterministic, and shows that the automaton $\supCN(G|_{\Omega}) \parallel G$ contains the transition $x \xrightarrow{\sigma} y$ if and only if the automaton $\supCN(G|_{\Omega}) \parallel G$ contains the transition $(x, x) \xrightarrow{\sigma} (y, y)$.

First let $x \xrightarrow{\sigma} y$ in $\supCN(G)$, i.e., $x \xrightarrow{\sigma}_{\Theta_G} y$ and $x \xrightarrow{\sigma} y$ in $G$. If $\sigma \in \Omega$, then $P_{\Omega}(\sigma) = \sigma$ and $x \xrightarrow{\sigma}_{\Omega} y$. Otherwise $\sigma \in \Lambda$ and $P_{\Omega}(\sigma) = \varepsilon$, and $x = y$ since $G$ is selfloop-only for $\Lambda$. In both cases, $x \xrightarrow{P_{\Omega}(\sigma)}_{\Omega} y$. Given $x, y \in \Theta_G = \Theta|_{\Omega}$ by lemma 15, it follows that $x \xrightarrow{P_{\Omega}(\sigma)}_{\Omega} y$ in $\supCN(G|_{\Omega})$. This implies $(x, x) \xrightarrow{\sigma} (y, y)$ in $\supCN(G|_{\Omega}) \parallel G$.

Conversely, let $(x, x) \xrightarrow{\sigma} (y, y)$ in $\supCN(G|_{\Omega}) \parallel G$. This means $x \xrightarrow{\sigma} y$ and $x \xrightarrow{P_{\Omega}(\sigma)}_{\Theta|_{\Omega}} y$, i.e., $x \xrightarrow{P_{\Omega}(\sigma)}_{\Theta_G} y$ by lemma 15. This implies $x, y \in \Theta_G$ and thus $x \xrightarrow{\sigma} y$ in $\supCN(G)$.

Theorem 7 Let $(G; S; \rho)$ be a synthesis triple such that $G$ is selfloop-only for $\Lambda \subseteq \Sigma$. Then $(G; S; \rho) \simeq_{\text{synth}} (G|_{\Sigma \setminus \Lambda}; S; \rho)$.

Proof. By definition 15 it follows that,

$$
\begin{align*}
\mathcal{L}(\supCN(G; S; \rho)) &= \mathcal{L}(\rho(\supCN(G) \parallel S)) \\
&= \mathcal{L}(\rho(\supCN(G|_{\Sigma \setminus \Lambda}) \parallel G \parallel S)) \quad \text{by lemma 16} \\
&= \mathcal{L}(\rho(\supCN(G|_{\Sigma \setminus \Lambda}) \parallel S)) \quad \text{as } \mathcal{L}(S) \subseteq \mathcal{L}(G) \text{ by definition 14 (i)} \\
&= \mathcal{L}(\supCN(G|_{\Sigma \setminus \Lambda}; S; \rho)) .
\end{align*}
$$

The claim follows from definition 16. □

B Proofs for Abstractions based on Observation Equivalence

This appendix contains the proofs for theorem 8 and theorem 9 in section 5.2, which state that bisimulation, synthesis observation equivalence, and weak synthesis observation equivalence preserve synthesis equivalence. The common feature of these abstractions is that they are obtained by merging equivalent states, and
can be represented as an automaton quotient modulo an equivalence relation. This observation leads to the following state-based definition, which is a sufficient condition for abstractions preserving synthesis equivalence [26].

**Definition 24** Let \( G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle \) be an automaton. An equivalence relation \( \sim \subseteq Q \times Q \) is a state-wise synthesis equivalence relation on \( G \) with respect to \( \mathcal{Y} \subseteq \Sigma \), if for all \( x \in Q \), all deterministic automata \( T = \langle \Sigma_T, Q_T, \rightarrow_T, Q^0_T \rangle \) such that \( \Sigma_T \cap \mathcal{Y} = \emptyset \), and for all states \( x_T \in Q_T \) the following relations hold,

(i) if \((x, x_T) \in \hat{\Theta}_{G/\sim} T\), then \(([x], x_T) \in \hat{\Theta}_{G/\sim} T\);

(ii) if \(([x], x_T) \in \hat{\Theta}_{G/\sim} T\), then \((x, x_T) \in \hat{\Theta}_{G/\sim} T\).

**Lemma 18** Let \((G; S; \rho)\) be a synthesis triple with \( G = \{G_1, \ldots, G_n\} \), and let \( T = G_2 \parallel \cdots \parallel G_n \). Then it holds that \( \rho(G_1 \parallel T) = \rho(G_1) \parallel \rho(T) \).

**Proof.** It is enough to show that \( \rho(G_1 \parallel T) \) and \( \rho(G_1) \parallel \rho(T) \) have the same transition relations.

First, let \((x_G, x_T) \xrightarrow{\gamma} (y_G, y_T)\) in \( \rho(G_1 \parallel T) \). Then there exists \( \gamma_0 \in \rho^{-1}(\gamma) \) such that \((x_G, x_T) \xrightarrow{\gamma_0} (y_G, y_T)\) in \( G_1 \parallel T \), which implies \( \gamma_0 \in \rho(G_1) \parallel \rho(T) \). There are three possibilities. If \( \gamma_0 \in \Sigma_{G_1} \cap \Sigma_T \) then \( x_G \xrightarrow{\gamma_0} y_G \) and \( x_T \xrightarrow{\gamma_0} y_T \), which implies \( x_G \xrightarrow{\gamma} y_G \) and \( x_T \xrightarrow{\gamma} y_T \), i.e., \((x_G, x_T) \xrightarrow{\gamma} (y_G, y_T)\) in \( \rho(G_1 \parallel T) \). If \( \gamma_0 \in \Sigma_T \setminus \Sigma_{G_1} \) then \( x_G = y_G \) and \( x_T \xrightarrow{\gamma_0} y_T \), which implies \( x_T \xrightarrow{\gamma_0} y_T \) and thus \((x_G, x_T) \xrightarrow{\gamma_0} (x_G, y_T) \xrightarrow{\gamma} (y_G, y_T)\) in \( \rho(G_1 \parallel T) \). If \( \gamma_0 \in \Sigma_{G_1} \setminus \Sigma_T \) then \( x_G \xrightarrow{\gamma_0} y_G \) and \( x_T = y_T \), which implies \( x_G \xrightarrow{\gamma_0} y_G \) and thus \((x_G, x_T) \xrightarrow{\gamma_0} (y_G, y_T)\) in \( \rho(G_1 \parallel T) \). Thus in all cases, \((x_G, x_T) \xrightarrow{\gamma} (y_G, y_T)\) in \( \rho(G_1 \parallel T) \).

Conversely, let \((x_G, x_T) \xrightarrow{\gamma} (y_G, y_T)\) in \( \rho(G_1) \parallel \rho(T) \). There are three cases. If \( \gamma \in \Sigma_{\rho(G_1)} \cap \Sigma_{\rho(T)} \) then \( x_G \xrightarrow{\gamma} y_G \) in \( \rho(G_1) \) and \( x_T \xrightarrow{\gamma} y_T \) in \( \rho(T) \). Then there exist \( \gamma_G, \gamma_T \in \Sigma_{G_1} \cap \Sigma_T \) such that \( \rho(\gamma_G) = \rho(\gamma_T) = \gamma \) and \( x_G \xrightarrow{\gamma_G} y_G \) and \( x_T \xrightarrow{\gamma_T} y_T \). By definition 14 (iii), at most one of \( G_1 \) or \( T \) differentiates between \( \gamma_G \) and \( \gamma_T \). Thus, it holds that \( x_G \xrightarrow{\gamma} y_G \) and \( x_T \xrightarrow{\gamma} y_T \) in \( G_1 \parallel T \), where \( \gamma_0 = \gamma_G \) or \( \gamma_0 = \gamma_T \), and thus \((x_G, x_T) \xrightarrow{\gamma} (y_G, y_T)\) in \( \rho(G_1 \parallel T) \). If \( \gamma \in \Sigma_{\rho(G_1)} \setminus \Sigma_{\rho(T)} \) then \( x_T = y_T \), and there exists \( \gamma_T \in \Sigma_T \) such that \( \rho(\gamma_T) = \gamma \) and \( x_G \xrightarrow{\gamma_T} y_G \). Also \( \gamma_G \notin \Sigma_T \) as \( \rho(\gamma_G) \neq \rho(\gamma_T) \) and thus \((x_G, x_T) \xrightarrow{\gamma_T} (y_G, y_T)\) in \( G_1 \parallel T \). If \( \gamma \in \Sigma_{\rho(T)} \setminus \Sigma_{\rho(G_1)} \) then \( x_G = y_G \), and there exists \( \gamma_T \in \Sigma_T \) such that \( \rho(\gamma_T) = \gamma \) and \( x_T \xrightarrow{\gamma_T} y_T \). Also \( \gamma_G \notin \Sigma_{G_1} \) as \( \rho(\gamma_G) \neq \rho(\gamma_T) \) and thus \((x_G, x_T) \xrightarrow{\gamma_T} (y_G, y_T)\) in \( G_1 \parallel T \). Thus, in all cases \((x_G, x_T) \xrightarrow{\gamma} (y_G, y_T)\) in \( \rho(G_1 \parallel T) \). \( \square \)
Proposition 19 Let \((\mathcal{G}; \mathcal{S}; \rho)\) be a synthesis triple with \(\mathcal{G} = \{G_1, \ldots, G_n\}\) and \(G_i = (\Sigma_i, Q_i, \delta_i, Q_0^i)\). Let \(\Upsilon \subseteq \Sigma_1\) such that \((\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset\). Let \(\sim\) be a state-wise synthesis equivalence relation on \(\rho(G_1)\) with respect to \(\Upsilon\) such that \(G_1/\sim\) is deterministic, and let \(\hat{\mathcal{G}} = \{G_1/\sim, G_2, \ldots, G_n\}\). Then \((\hat{\mathcal{G}}; \hat{\mathcal{S}}; \rho) \simeq_{\text{synth}} (\mathcal{G}; \mathcal{S}; \rho)\).

Proof. Let \(T = G_2 \parallel \cdots \parallel G_n\). First it is shown that
\[
\mathcal{L}(G_1 \parallel \sup \mathcal{CN}(G_1 \parallel T)) = \mathcal{L}(G_1 \parallel \sup \mathcal{CN}((G_1/\sim) \parallel T)). \tag{23}
\]

Let \(s \in \mathcal{L}(G_1 \parallel \sup \mathcal{CN}(G_1 \parallel T))\). This means \(G_1 \parallel \sup \mathcal{CN}(G_1 \parallel T) \xrightarrow{\rho} (y_G; y_G, y_T)\). Let \(s = \sigma_1 \cdots \sigma_n\). Then there exists a path
\[
(y_0^G, y_0^T) \xrightarrow{\sigma_1} (y_1^G, y_1^T) \cdots \xrightarrow{\sigma_n} (y_n^G, y_n^T) = (y_G, y_T) \tag{24}
\]
with \((y_k^G, y_k^T) \in \hat{\Theta}_{G_1\parallel T}\) or \(\sigma_k = \omega\) for \(k = 0, \ldots, n\). Since \(\rho\) preserves controllability, it follows from definition 8 that \(\Theta_{G_1\parallel T} = \Theta_{\rho(G_1)\parallel \rho(T)}\), and by lemma 18 \(\Theta_{\rho(G_1)\parallel \rho(T)} = \Theta_{\rho(G_1)\parallel \rho(T)}\). Thus,
\[
(y_0^G, y_0^T) \xrightarrow{\rho(\sigma_1)} (y_1^G, y_1^T) \cdots \xrightarrow{\rho(\sigma_n)} (y_n^G, y_n^T). \tag{25}
\]
By definition 24 (i), it holds that \(((y_k^G, y_k^T) \in \hat{\Theta}_{\rho(G_1)/\sim \parallel \rho(T)}\) or \(\sigma_k = \omega\) for \(k = 0, \ldots, n\), and thus
\[
((y_0^G, y_0^T) \xrightarrow{\rho(\sigma_1)} (y_1^G, y_1^T) \cdots \xrightarrow{\rho(\sigma_n)} (y_n^G, y_n^T)). \tag{26}
\]
Note that \(\rho(G_1)/\sim = \rho(G_1/\sim)\) and thus \(\rho(G_1/\sim) \parallel T = \rho(G_1/\sim) \parallel T = \rho(G_1/\sim) \parallel T\) by lemma 18. Given (24), it follows that
\[
((y_0^G, y_0^T) \xrightarrow{\sigma_1} (y_1^G, y_1^T) \cdots \xrightarrow{\sigma_n} (y_n^G, y_n^T) = ([y_G], [y_T]). \tag{27}
\]
Therefore, \(G_1 \parallel \sup \mathcal{CN}(G_1/\sim \parallel T) \xrightarrow{\rho} (y_G; [y_G], y_T),\) which means that \(s \in \mathcal{L}(G_1 \parallel \sup \mathcal{CN}(G_1/\sim \parallel T)).\)

Conversely, let \(s \in \mathcal{L}(G_1 \parallel \sup \mathcal{CN}(G_1/\sim \parallel T)).\) Since \(G_1\) and \(G_1/\sim\) are deterministic, there exists a path \(G_1 \parallel \sup \mathcal{CN}(G_1/\sim \parallel T) \xrightarrow{\sigma_1} (x_1^G, [x_1^G], x_T^T) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} (x_n^G, [x_n^G], x_T^T)\) where \(s = \sigma_1 \cdots \sigma_n\) and \([x_k^G], x_T^T\) \(\in \hat{\Theta}_{G_1/\sim \parallel T}\) or \(\sigma_k = \omega\) for \(k = 0, \ldots, n\). Since \(\rho\) preserves controllability, it follows from definition 8 and lemma 18 that \(\Theta_{G_1/\sim \parallel T} = \Theta_{\rho(G_1)\parallel \rho(T)} = \Theta_{\rho(G_1)/\sim \parallel \rho(T)}\), which implies \(([x_k^G], x_T^T) \in \hat{\Theta}_{\rho(G_1)/\sim \parallel \rho(T)}\). By definition 24 (ii), it follows that \((x_k^G, x_T^T) \in \hat{\Theta}_{\rho(G_1)\parallel \rho(T)}\). This means \((x_k^G, x_T^T) \in \hat{\Theta}_{G_1\parallel T}\) or \(\sigma_k = \omega\) for \(k =
there exist $t \in \Sigma$. Therefore, $G_1 \supseteq \supCN(G_1 \parallel T) \stackrel{\sigma_1}{\rightarrow} (x^G_1, x^G_2, x^T_1) \stackrel{\sigma_2}{\cdots} \stackrel{\sigma_n}{\rightarrow} (x^G_n, x^G_n, x^T_n)$, and thus $s \in L(G_1 \parallel \supCN(G_1 \parallel T))$.

Given (23), it follows from definition 15 that

$$L(\supCN(G; S; \rho)) = \mathcal{L}(\rho(\supCN(G) \parallel S))$$

$$= \rho(\mathcal{L}(\supCN(G_1 \parallel T) \cap \mathcal{L}(S))$$

$$= \rho(\mathcal{L}(G_1 \parallel \supCN((G_1 \parallel T) \cap \mathcal{L}(S))$$

$$= \rho(\mathcal{L}(\supCN((G_1 \parallel T) \cap \mathcal{L}(G_1 \parallel T) \cap \mathcal{L}(S))$$

$$= \rho(\mathcal{L}(\supCN((G_1 \parallel T) \cap \mathcal{L}(S))$$

$$(\text{as } \mathcal{L}(S) \subseteq \mathcal{L}(G) = \mathcal{L}(G_1 \parallel T) \text{ by definition 14 (ii)}$$

$$= \rho(\mathcal{L}(\supCN(\tilde{G})) \cap \mathcal{L}(S))$$

$$= \mathcal{L}(\rho(\supCN(\tilde{G}) || S))$$

$$= \mathcal{L}(\supCN(\tilde{G}; S; \rho)),$$  \hspace{1cm} (28)

so the claim follows from definition 16.

To prove the main results of this section, theorems 8 and 9, it is now enough to show that every bisimulation relation, every synthesis observation equivalence relation, and every weak synthesis observation equivalence relation is a state-wise synthesis equivalence relation.

The most general of these relations is weak synthesis observation equivalence. Therefore, lemma 21 below establishes the crucial result that every weak synthesis observation equivalence relation is a state-wise synthesis equivalence relation.

Lemma 20  Let $G = (\Sigma, Q, \rightarrow, Q^o)$ and $T = (\Sigma_T, Q_T, \rightarrow_T, Q^o_T)$ be two automata with $\Sigma \cup \Sigma_T = \Omega \cup T$ and $\Omega \cap \Sigma_T = \emptyset$, and let $\sim$ be a weak synthesis observation equivalence on $G$ with respect to $T$. Let $X \subseteq Q \times Q_T$ such that $([x], x_T) \in \Theta_G/\sim_T$ always implies $(x, x_T) \in X$. Furthermore, let $(x_1, x^T_1) \stackrel{\sigma}{\rightarrow} (x_2, x^T_2)$ such that $([x_1], x^T_1) \stackrel{\sigma}{\rightarrow}_{\Theta_G/\sim_T} ([x_2], x^T_2)$. Then for all states $y_1 \in Q$ such that $x_1 \sim y_1$, there exist $t_1, t_2 \in T^*$ and $y_2 \in Q$ such that $(y_1, x^T_1) \stackrel{t_1\rho_2(\sigma)t_2}{\rightarrow}_{\Theta_G/\sim_T} (y_2, x^T_2)$ and $x_2 \sim y_2$.

Proof. Let $x_1, x_2, y_1 \in Q$ and $x^T_1, x^T_2 \in Q_T$ and $\sigma \in \Sigma_\omega \cup \Sigma_T$ such that $(x_1, x^T_1) \stackrel{\sigma}{\rightarrow} (x_2, x^T_2)$, $([x_1], x^T_1) \stackrel{\sigma}{\rightarrow}_{\Theta_G/\sim_T} ([x_2], x^T_2)$, and $x_1 \sim y_1$. Consider
three cases.

(i) If $\sigma \notin \Sigma_{\omega}$, then $\sigma \in \Sigma_{T} \setminus \Sigma \subseteq \Omega$ and $x_{1} = x_{2}$ and $x_{1}^T \xrightarrow{\sigma} x_{2}^T$. Given $([x_1], x_1^T) \xrightarrow{\sigma}_{\Theta_{G/\sim T}} ([x_2], x_2^T)$, it follows that $([y_1], x_1^T) = ([x_1], x_1^T) \in \Theta_{G/\sim T}$ and $([y_1], x_1^T) = ([x_1], x_1^T) \in \Theta_{G/\sim T}$, and therefore $(y_1, x_1^T), (y_1, x_1^T) \in X$ by assumption. This implies that $(y_1, x_1^T) \xrightarrow{P_{\Omega}(\sigma)}_{\sim T} (y_1, x_2^T)$.

(ii) If $\sigma \in \Sigma \cap \Sigma_{\Omega}$, then $x_{1} \xrightarrow{\sigma} x_{2}$ and $x_{1} \sim y_{1}$, so by definition 22 (ii) there exist $T_{1}, T_{2} \subseteq (\Upsilon \cap \Sigma_{\Omega})^{*}$ and $T_{2} \in Q$ such that $y_{1} \xrightarrow{T_{1} P_{\Omega}(\sigma) T_{2}} y_{2}$. Let $r \subseteq T_{1} P_{\Omega}(\sigma) T_{2}$ such that $y_{1} \xrightarrow{r} z$. Then $[x_1] = [y_1] \xrightarrow{T_{1}} [z]$, and since $\Sigma_{T} \cap \Upsilon = \emptyset$, it follows that $([x_1], x_1^T) \xrightarrow{T_{1}} ([z], x_d^T)$ for some $d \in \{1, 2\}$. Since $r \in \Sigma_{\Omega}$ and $([x_1], x_1^T) \in \Theta_{G/\sim T}$, it follows that $([z], x_d^T) \in \Theta_{G/\sim T}$. This implies $(z, x_d^T) \in X$ by assumption. This argument holds for all prefixes $r \subseteq T_{1} P_{\Omega}(\sigma) T_{2}$, and therefore $(y_1, x_1^T) \xrightarrow{T_{1} P_{\Omega}(\sigma) T_{2}}_{\sim T} (y_2, x_2^T)$.

(iii) If $\sigma \in \Sigma \cap \Sigma_{\omega}$ or $\sigma = \omega$, then $x_{1} \xrightarrow{\sigma} x_{2}$ and $x_{1} \sim y_{1}$, so by definition 22 (i) there exists a path

$$y_{1} = z_{0} \xrightarrow{T_{1}} \ldots \xrightarrow{T_{k}} z_{k} \xrightarrow{P_{\Omega}(\sigma)} \ldots \xrightarrow{T_{\omega - 1}} z_{\omega - 1} \xrightarrow{T_{\omega}} z_{\omega} = y_{2}$$

(29) such that $x_{2} \sim y_{2}$ and $T_{1}, \ldots, T_{\omega - 1} \in \Upsilon$. The first part of this path satisfies (i)a) and the second part satisfies (i)b) and (i)c) in definition 22. Since $T_{1}, \ldots, T_{\omega - 1} \in \Upsilon$ and $\Sigma_{T} \cap \Upsilon = \emptyset$, it holds that

$$(y_1, x_1^T) = ([z_0], x_1^T) \xrightarrow{T_1} \ldots \xrightarrow{T_k} ([z_k], x_1^T) \xrightarrow{P_{\Omega}(\sigma)} ([z_{k+1}], x_2^T) \xrightarrow{T_{k+1}} \ldots \xrightarrow{T_{\omega - 1}} ([z_{\omega}], x_2^T) = (y_2, x_2^T)$$

(30)

It follows that

$$([z_0], x_1^T) \xrightarrow{T_1} \ldots \xrightarrow{T_k} ([z_k], x_1^T) \xrightarrow{P_{\Omega}(\sigma)} ([z_{k+1}], x_2^T) \xrightarrow{T_{k+1}} \ldots \xrightarrow{T_{\omega - 1}} ([z_{\omega}], x_2^T).$$

(31)

It is shown in the following that this path also exists in the restriction of $G/\sim T$ to $\Theta_{G/\sim T}$.

For the first part of the path, it is shown by induction on $i$ that $([z_{i}], x_{i}^T) \in \Theta_{G/\sim T}$. For $i = 0, \ldots, k$ if $\sigma \in \Omega \cup \{\omega\}$, and for $i = 0, \ldots, k - 1$ if $\sigma \in \Upsilon$. Base case. For $i = 0$, it follows by assumption that $([z_{0}], x_{1}^T) = ([y_{1}], x_{1}^T) = ([x_1], x_1^T) \in \Theta_{G/\sim T}$.
**Inductive step.** Assume the claim holds for some \( i \geq 0 \), i.e., \( ([z_i], x_T^i) \in \tilde{\Theta}_{G/\sim T} \). It must be shown that \( ([z_{i+1}], x_T^{i+1}) \in \tilde{\Theta}_{G/\sim T} \). There are two possibilities for \( \tau_{i+1} \in \Upsilon \):

a) \( \tau_{i+1} \in \Sigma_e \). In this case, it follows from definition 22 (i) that \( z_{i+1} \sim x_1 \), and thus \( ([z_{i+1}], x_T^{i+1}) = ([z_i], x_T^i) \in \tilde{\Theta}_{G/\sim T} \) by assumption.

b) \( \tau_{i+1} \in \Sigma_u \). As \((z_i, x_T^i) \overset{\tau_{i+1}}{\rightarrow} ([z_{i+1}], x_T^{i+1}) \), it holds that \( ([z_i], x_T^i) \overset{\tau_{i+1}}{\rightarrow} ([z_{i+1}], x_T^{i+1}) \), and \( ([z_i], x_T^i) \in \tilde{\Theta}_{G/\sim T} \) by inductive assumption. Then \( ([z_{i+1}], x_T^{i+1}) \in \tilde{\Theta}_{G/\sim T} \) because \( \tau_{i+1} \in \Sigma_u \).

If \( \sigma = \omega \), the second part of the path (31) is empty and the claim follows. Otherwise note that by assumption,

\[
([x_2], x_T^2) \in \tilde{\Theta}_{G/\sim T}.
\]

It is shown that \( ([z_i], x_T^i) \in \tilde{\Theta}_{G/\sim T} \) for \( k < i < l \). Let \( \Upsilon_u^T = \Sigma_u \cap (\Sigma_T \setminus \Sigma) \) and \( Y^T = \{ y^T \in Q_T \mid x_T^2 \overset{u}{\sim_T} y^T \text{ for some } u \in (\Upsilon_u^T)^* \} \).

As \( x_T^2 \in Y^T \), it is enough to show that \( ([z_i], y^T) \in \tilde{\Theta}_{G/\sim T} \) for all \( y^T \in Y^T \). It is shown by induction on \( n \geq 0 \) that for all \( k < i < l \) and for all \( y^T \in Y^T \) it holds that \( ([z_i], y^T) \in \hat{\Theta}^n = \Theta^0_{G/\sim T}(Q/\sim \times Q_T) \).

**Base case.** \( n = 0 \). Clearly \( ([z_i], y^T) \in Q/\sim \times Q_T = \Theta^0_{G/\sim T}(Q/\sim \times Q_T) = \hat{X}^0 \).

**Inductive step.** Let \( k < i < l \) and \( y^T \in Y^T \). It must be shown that \( ([z_i], y^T) \in \Theta^0_{G/\sim T}(\hat{X}^n) \cap \Theta^0_{G/\sim T}(\hat{X}^n) \).

To see that \( ([z_i], y^T) \in \Theta^0_{G/\sim T}(\hat{X}^n) \), let \( u \in \Sigma_u \) and \( ([z_i], y^T) \overset{u}{\sim_T} ([z], z^T) \). Consider three cases.

a) \( u \in \Sigma \cap \Upsilon \). In this case \( y^T = z^T \) and \( [z_i] \overset{u}{\sim_T} [z] \), so there exist \( z'_i \sim z_i \) and \( z' \sim z \) such that \( z'_i \overset{u}{\sim_T} z' \). By definition 22 (ii), there exist \( u_1, u_2 \in (\Sigma_u \cap \Upsilon)^* \) and \( z'' \sim z' \) such that \( z_i \overset{u_1 u_2}{\sim_T} z'' \). As \( z_i \) is on the path (29), it follows from definition 22 (i) that \( z'' \sim z_i \) for some \( k < j \leq l \). If \( j < l \), then \( ([z], z^T) = ([z'], z^T) = ([z''], z^T) = ([z_i], z^T) \in \hat{X}^n \) by inductive assumption. If \( j = l \), then note that \( ([x_2], x_T^2) \overset{u_1 u_2}{\sim_T} ([x_2], z''^T) \) for some \( u \in (\Upsilon_u^T)^* \) as \( z''^T = y^T \in Y^T \), and given (32) it follows that \( ([y_2], z''^T) = ([x_2], z''^T) \in \tilde{\Theta}_{G/\sim T} \). Then \( ([z], z^T) = ([z'], z^T) = ([z''], z^T) = ([z_i], z^T) \in \tilde{\Theta}_{G/\sim T} \subseteq \hat{X}^n \).
Lemma 21

Let $\sim$ be a weak synthesis observation equivalence on $G = (\Sigma, Q, \rightarrow, Q^0)$ with respect to $\Upsilon \subseteq \Sigma$. Then $\sim$ is a state-wise synthesis equivalence on $G$ with respect to $\Upsilon$. 

47
Proof. Let $T = \langle \Sigma_T, Q_T, \rightarrow_T, Q^0_T \rangle$ with $\Sigma_T \cap \Upsilon = \emptyset$ and $\Sigma_T \cup \Sigma_T = \Omega \cup \Upsilon$. The conditions of state-wise synthesis equivalence in definition 24 must be confirmed.

(i) It is shown by induction on $n \geq 0$ that $(x, x_T) \in \hat{\Theta}_{G\parallel T}$ implies $([x], x_T) \in \hat{X}^n = \hat{\Theta}_{G\parallel T}^n(Q/\sim \times Q_T)$.

Base case. $([x], x_T) \in Q/\sim \times Q_T = \hat{\Theta}_{G\parallel T}^0(Q/\sim \times Q_T) = \hat{X}^0$.

Inductive step. Assume the claim holds for some $n \geq 0$, i.e., if $(x, x_T) \in \hat{\Theta}_{G\parallel T}$ then $([x], x_T) \in \hat{X}^n$. Now let $(x, x_T) \in \hat{\Theta}_{G\parallel T}$. It must be shown that $([x], x_T) \in \hat{X}^{n+1} = \hat{\Theta}_{G\parallel T}^{n+1}(\hat{X}^n) = \Theta_{G\parallel T}^{\text{cont}}(\hat{X}^n) \cap \Theta_{G\parallel T}^{\text{nonb}}(\hat{X}^n)$.

To see that $([x], x_T) \in \Theta_{G\parallel T}^{\text{cont}}(\hat{X}^n)$, let $v \in \Sigma_u$ and $([x], x_T) \overset{v}{\rightarrow} ([y], y_T)$. Consider two cases.

a) $v \notin \Sigma$. In this case, $[x] = [y]$ and $(x, x_T) \overset{v}{\rightarrow} (x, y_T)$, and it follows from $(x, x_T) \in \hat{\Theta}_{G\parallel T}$ and $v \in \Sigma_u$ that $(x, y_T) \in \hat{\Theta}_{G\parallel T}$. Then by inductive assumption $([y], y_T) = ([x], y_T) \in \hat{X}^n$.

b) $v \in \Sigma$. In this case, there exist $x' \in [x]$ and $y' \in [y]$ such that $x' \overset{v}{\rightarrow} y'$. By definition 22 (ii), there exist $t_1, t_2 \in (\Upsilon \cap \Sigma_u)^*$ and $y'' \sim y'$ such that $x \overset{t_1P_3(v)t_2}{\rightarrow} y''$. As $t_1, t_2 \in \Upsilon^*$, it follows that $(x, x_T) \overset{t_1P_3(v)t_2}{\rightarrow} (y'', y_T)$. Since $(x, x_T) \in \hat{\Theta}_{G\parallel T}$ and $t_1P_3(v)t_2 \in \Sigma_u^*$, it follows that $(y'', y_T) \in \hat{\Theta}_{G\parallel T}$. Then by inductive assumption $([y], y_T) = ([y''], y_T) \in \hat{X}^n$.

Thus $([y], y_T) \in \hat{X}^n$ can be shown for all $v \in \Sigma_u$, and it follows that $([x], x_T) \in \Theta_{G\parallel T}^{\text{cont}}(\hat{X}^n)$.

Next, it is shown that $([x], x_T) \in \Theta_{G\parallel T}^{\text{nonb}}(\hat{X}^n)$. Since $(x, x_T) \in \hat{\Theta}_{G\parallel T}$, there exists a path

$$
(x, x_T) = (x_0, x_{0, T}) \overset{\sigma_1}{\rightarrow}_{\hat{\Theta}_{G\parallel T}} \cdots \overset{\sigma_k}{\rightarrow}_{\hat{\Theta}_{G\parallel T}} (x_k, x_{k, T}) \in \hat{\Theta}_{G\parallel T}((x_{k+1}, x_{k+1, T})).
$$

Then $(x_l, x_{l, T}) \in \hat{\Theta}_{G\parallel T}$ for $l = 0, \ldots, k$. By inductive assumption, it follows that $([x_l], x_{l, T}) \in \hat{X}^n$ for $l = 0, \ldots, k$. Thus,

$$
([x], x_T) = ([x_0], x_{0, T}) \overset{\sigma_1}{\rightarrow}_{\hat{X}^n} \cdots \overset{\sigma_k}{\rightarrow}_{\hat{X}^n} ([x_k], x_{k, T}) \overset{\omega}{\rightarrow}_{\hat{X}^n} ([x_{k+1}], x_{k+1, T}),
$$

which implies $([x], x_T) \in \Theta_{G\parallel T}^{\text{nonb}}(\hat{X}^n)$.

Thus, it has been shown that $([x], x_T) \in \Theta_{G\parallel T}^{\text{cont}}(\hat{X}^n) \cap \Theta_{G\parallel T}^{\text{nonb}}(\hat{X}^n) = \hat{X}^{n+1}$. 

48
(ii) Now it is shown by induction on \( n \geq 0 \) that \( ([x], x_T) \in \hat{\Theta}_{G/\sim||T} \) implies 
\((x, x_T) \in X^n = \hat{\Theta}_{G||T}^n(Q \times Q_T)\).

**Base case.** \((x, x_T) \in Q \times Q_T = \Theta_{G||T}^0(Q \times Q_T) = X^0\).

**Inductive step.** Assume the statement holds for \( n \geq 0 \), i.e., if \( ([x], x_T) \in \hat{\Theta}_{G/\sim||T} \) then \((x, x_T) \in X^n\). Let \((x, x_T) \in \hat{\Theta}_{G/\sim||T} \). It must be shown that \((x, x_T) \in X^{n+1} = \Theta_{G||T}(X^n) = \Theta_{G||T}^{cont}(X^n) \cap \Theta_{G||T}^{nonb}(X^n)\).

To see that \((x, x_T) \in \Theta_{G||T}^{cont}(X^n)\), let \(v \in \Sigma_u \) and \((x, x_T) \xrightarrow{v} (y, y_T)\). This implies \([x], x_T) \xrightarrow{v} ([y], y_T)\). Since \((x, x_T) \in \hat{\Theta}_{G/\sim||T} \) and \(v \in \Sigma_u\), it follows that \((y, y_T) \in \hat{\Theta}_{G/\sim||T}\). Then by inductive assumption \((y, y_T) \in X^n\), and thus \((x, x_T) \in \Theta_{G||T}^{cont}(X^n)\).

Next it is shown that \((x, x_T) \in \Theta_{G/\sim||T}^{nonb}(X^n)\). Since \((x, x_T) \in \hat{\Theta}_{G/\sim||T} \), there exists a path
\[
([x], x_T) = ([x_0], x_0^{T}) \xrightarrow{\sigma_1} \hat{\Theta}_{G/\sim||T} \cdots \xrightarrow{\sigma_k} \hat{\Theta}_{G/\sim||T} ([x_{k+1}], x_{k+1}^{T}) .
\]

Consider the first transition in (35). Since \([x_0] \xrightarrow{P_{\Sigma_u}((\omega_1)} [x_1]\), there exists \(x_0' \in [x_0] \) and \(x_1' \in [x_1] \) such that \(x_0' \xrightarrow{P_{\Sigma_u}((\sigma_1)} x_1'\). The conditions of lemma 20 apply to this transition: by inductive assumption, \(X^n\) can be used as the set \(X\) in the lemma, and \(([x_0'], x_0^{T}) = ([x_0], x_0^{T}) \in \hat{\Theta}_{G/\sim||T}\), \(([x_1'], x_1^{T}) = ([x_1], x_1^{T}) \in \hat{\Theta}_{G/\sim||T}\) or \(\sigma_1 = \omega, (x_0', x_0^{T}) \xrightarrow{\omega} (x_1', x_1^{T})\), and \(x_0' \sim x_0\). So there exist \(t_1, u_1 \in \Upsilon^*\) and \(x_1'' \in Q\) such that
\[
(x_0, x_0^{T}) \xrightarrow{t_1P_{\Sigma_u}((\omega_1)} [x_1''(x_1', x_1^{T})] .
\]

and \(x_1' \sim x_1''\). Since \(x_1'' \in [x_1'] = [x_1]\), the same logic also applies to the second transition in (35). Therefore, there exist \(t_2, u_2 \in \Upsilon^*\) and \(x_2'' \in Q\) such that
\[
(x_1'', x_1^{T}) \xrightarrow{t_2P_{\Sigma_u}((\omega_2)} [x_2''(x_2', x_2^{T})] \text{ and } x_2' \sim x_2'' .
\]
By induction, it follows that there exist \(t_1, u_1, \ldots, t_k, u_k, t_{k+1} \in \Upsilon^*\) and \(x_1'', \ldots, x_k'' \in Q\) such that
\[
(x, x_T) = (x_0, x_0^{T}) \xrightarrow{t_1P_{\Sigma_u}((\sigma_1)} [x_1''(x_1', x_1^{T})] \xrightarrow{t_2P_{\Sigma_u}((\sigma_2)} [x_2''(x_2', x_2^{T})] \cdots \xrightarrow{t_kP_{\Sigma_u}((\sigma_k)} [x_k''(x_k', x_k^{T})] \xrightarrow{\sigma_{k+1} \omega} [x_n''] .
\]

Therefore, \((x, x_T) \in \Theta_{G||T}^{nonb}(X^n)\).
Thus, it has been shown that $(x, x_T) \in \Theta^\text{cont}_{G\parallel T}(X^n) \cap \Theta^\text{monh}_{G\parallel T}(X^n) = X^{n+1}$.  
□

**Theorem 8** Let $(G; S; \rho)$ be a synthesis triple with $G = \{G_1, \ldots, G_n\}$, and let $\sim$ be a bisimulation on $G_1$ and $\tilde{G} = \{G_1/\sim, G_2, \ldots, G_n\}$. Then it holds that $(G; S; \rho) \simeq_{\text{synth}} (\tilde{G}; S; \rho)$.

**Proof.** Clearly, if $\sim$ is a bisimulation on $G_1$, then $\sim$ also is a weak synthesis observation equivalence on $G_1$ with respect to $\Omega = \Sigma$. By lemma 21, it follows that $\sim$ is a state-wise synthesis equivalence on $G_1$ with respect to $\Sigma$. Then the claim follows from proposition 19.  
□

**Theorem 9** Let $(G; S; \rho)$ be a synthesis triple with $G = \{G_1, \ldots, G_n\}$ and $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^0 \rangle$. Let $\Upsilon \subseteq \Sigma_1$ such that $\Sigma_T \cap \Upsilon = \emptyset$. Let $\sim$ be a synthesis observation equivalence or a weak synthesis observation equivalence relation on $\rho(G_1)$ with respect to $\Upsilon$ such that $G_1/\sim$ is deterministic, and let $\tilde{G} = \{G_1/\sim, G_2, \ldots, G_n\}$. Then $(G; S; \rho) \simeq_{\text{synth}} (\tilde{G}; S; \rho)$.

**Proof.** If $\sim$ is a weak synthesis observation equivalence on $G_1$ with respect to $\Upsilon$, then it follows from lemma 21 that $\sim$ is a state-wise synthesis equivalence on $G_1$ with respect to $\Upsilon$, so the claim follows from proposition 19.

If $\sim$ is a synthesis observation equivalence on $G_1$ with respect to $\Upsilon$, then it is shown in [23] that $\sim$ is a weak synthesis observation equivalence on $G_1$ with respect to $\Upsilon$, and the claim follows as above.  

□

**C Proof for Halfway Synthesis**

This appendix contains a proof for theorem 10 in section 5.3. The proof is based on two lemmas, which show how halfway synthesis preserves synthesis results in synchronous composition.

**Lemma 24** Let $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ and $T = \langle \Sigma_T, Q_T, \rightarrow_T, Q_T^0 \rangle$, and let $\Upsilon \subseteq \Sigma \cap \Sigma_0$ such that $\Sigma_T \cap \Upsilon = \emptyset$. Then for all $x \in Q$ and $x_T \in Q_T$ such that $(x, x_T) \in \Theta^T_{G\parallel T}$, it holds that $x \in \Theta_{G\parallel T, \Upsilon}$.

**Proof.** It is shown by induction on $n \geq 0$ that $(x, x_T) \in \Theta^T_{G\parallel T}$ implies $x \in X^n = \Theta^0_{G, \Upsilon}(Q)$.

**Base case.** Clearly $x \in Q = \Theta^0_{G, \Upsilon}(Q) = X^0$.  

50
Inductive step. Assume that \((x, x_T) \in \hat{\Theta}_{G||T}\) implies \(x \in X^n\) for some \(n \geq 0\), and let \((x, x_T) \in \hat{\Theta}_{G||T}\). It is to be shown that \(x \in X^{n+1} = \Theta_{G,T}(X^n) = \Theta_{G,T}^\text{cont}(X^n) \cap \Theta_{G,T}^\text{nonb}(X^n)\).

First, to see that \(x \in \Theta_{G,T}^\text{cont}(X^n)\), let \(v \in \Upsilon\) and \(x \xrightarrow{v} y\). As \(\Sigma_T \cap \Upsilon = \emptyset\), it follows that \((x, x_T) \xrightarrow{v} G||T\ (y, x_T)\). As \((x, x_T) \in \hat{\Theta}_{G||T}\) and \(v \in \Upsilon \subseteq \Sigma_u\), it follows by controllability that \((y, x_T) \in \hat{\Theta}_{G||T}\), and then \(y \in X^n\) by inductive assumption. As \(v \in \Upsilon\) was chosen arbitrarily, it follows that \(x \in \Theta_{G,T}^\text{cont}(X^n)\).

Next it is shown that \(x \in \Theta_{G,T}^\text{nonb}(X^n)\). As \((x, x_T) \in \hat{\Theta}_{G||T}\), there exists a trace \(t = \sigma_1 \cdots \sigma_n\) such that
\[
(x, x_T) = (x_0, x_0^T) \xrightarrow{\sigma_1} \hat{\Theta}_{G||T} (x_1, x_1^T) \cdots \xrightarrow{\sigma_n} \hat{\Theta}_{G||T} (x_n, x_n^T) \xrightarrow{\omega} \hat{\Theta}_{G||T}.
\]

Then by inductive assumption \(x_0, \ldots, x_n \in X^n\), which implies \(x \xrightarrow{\omega} \|_X\) and therefore \(x \in \Theta_{G,T}^\text{nonb}(X^n)\).

\[\square\]

Lemma 25 Let \(G = (\Sigma, Q, \rightarrow, Q^0)\) and \(T = (\Sigma_T, Q_T, \rightarrow_T, Q_T^0)\), and let \(\Upsilon \subseteq \Sigma \cap \Sigma_u\) such that \(\Sigma_T \cap \Upsilon = \emptyset\). Then \(\sup\CN(G \parallel T) = \sup\CN(H \parallel T)\) where \(H = \hsup\CN_T(G)\).

Proof. By definition 23, \(H = (\Sigma, Q_H, \rightarrow_{\hsup}, Q_H^0)\) where \(Q_H = Q \cup \{\bot\}\). It is enough to show \(\hat{\Theta}_{G||T} = \hat{\Theta}_{H||T}\).

Let \((x, x_T) \in \hat{\Theta}_{G||T}\). It is shown by induction on \(n \geq 0\) that \(\hat{\Theta}_{G||T} \subseteq X^n_{H||T} = \Theta_{H||T}^0(Q_H \times Q_T)\).

Base case. By definition 23, \(\hat{\Theta}_{G||T} \subseteq Q_H \times \Theta_{H||T}^0(Q_H \times Q_T) = X^n_{H||T}\).

Inductive step. Assume \(\hat{\Theta}_{G||T} \subseteq X^n_{H||T}\) for some \(n \geq 0\), and let \((x, x_T) \in \hat{\Theta}_{G||T}\). It is to be shown that \((x, x_T) \in X^{n+1}_{H||T} = \Theta_{H||T}(X^n_{H||T}) = \Theta_{H||T}^\text{cont}(X^n_{H||T}) \cap \Theta_{H||T}^\text{nonb}(X^n_{H||T})\).

First, to see that \((x, x_T) \in \Theta_{H||T}^\text{cont}(X^n_{H||T})\), let \(v \in \Sigma_u\) and \((x, x_T) \xrightarrow{v} H||T\ (y, y_T)\). It is next shown that \((x, x_T) \xrightarrow{v} G||T\ (y, y_T)\). Assume this is not the case. Then \(v \in \Sigma\), and by construction of \(H = \hsup\CN_T(G)\) and definition 23 also \(y = \bot\), which again by definition 23 implies that \(x \xrightarrow{v}\) does not hold in \(\sup\CN_T(G)\), and \(x \xrightarrow{v} y'\) in \(G\) for some \(y' \in Q\). Then \((x, x_T) \xrightarrow{v} G||T\ (y', y_T)\), and given \((x, x_T) \in \hat{\Theta}_{G||T}\) it follows that \((y', y_T) \in \hat{\Theta}_{G||T}\). Then \(x, y' \in \hat{\Theta}_{G,T}\) by lemma 24, and thus \(x \xrightarrow{v} y'\) in \(\sup\CN_T(G)\). This contradicts the above statement that \(x \xrightarrow{v}\) does not hold in \(\sup\CN_T(G)\). Therefore, \((x, x_T) \xrightarrow{v} G||T\ (y, y_T)\), and since \((x, x_T) \in \hat{\Theta}_{G||T}\), it follows by controllability that \((y, y_T) \in \hat{\Theta}_{G||T}\). By inductive assumption \((y, y_T) \in X^n_{H||T}\), which implies \((x, x_T) \in \Theta_{H||T}^\text{cont}(X^n_{H||T})\).
Next it is shown that \((x, x_T) \in \Theta^{nonb}_{H\|T}(X^0_H)\). Since \((x, x_T) \in \hat{\Theta}_{G\|T}\), there exists a path
\[
(x, x_T) = (x_0, x^0_T) \xrightarrow{\sigma_1}_{|\hat{\Theta}_{G\|T}} \cdots \xrightarrow{\sigma_k}_{|\hat{\Theta}_{G\|T}} (x_k, x^k_T) \xrightarrow{\omega}_{|\hat{\Theta}_{G\|T}} (x_{k+1}, x^k_{k+1}).
\]
Then \((x_l, x^l_T) \in \hat{\Theta}_{G\|T}\) for \(l = 0, \ldots, k\). By inductive assumption \((x_l, x^l_T) \in X^n_{G\|T}\) for \(l = 0, \ldots, k\), and thus
\[
(x, x_T) = (x_0, x^0_T) \xrightarrow{\sigma_1}_{|X^n_{G\|T}} \cdots \xrightarrow{\sigma_k}_{|X^n_{G\|T}} (x_k, x^k_T) \xrightarrow{\omega}_{|X^n_{G\|T}} (x_{k+1}, x^k_{k+1}),
\]
which implies \((x, x_T) \in \Theta^{nonb}_{H\|T}(X^0_H)\).

Conversely, to show that \(\hat{\Theta}_{H\|T} \subseteq \hat{\Theta}_{G\|T}\), it is shown by induction on \(n \geq 0\) that \(\Theta^{cont}_{H\|T} \subseteq X^n_{G\|T} = \Theta^{cont}_{G\|T}(Q \times Q_T)\).

**Base case.** Let \((x, x_T) \in \hat{\Theta}_{H\|T}\). Clearly \(x \neq \bot\), as \((\bot, x_T) \notin \Theta^{nonb}_{H\|T}(Q_H \times Q_T)\). Therefore, \((x, x_T) \in Q \times Q_T = \Theta^0_{G\|T}(Q \times Q_T) = X^0_{G\|T}\).

**Inductive step.** Assume \(\hat{\Theta}_{H\|T} \subseteq X^n_{G\|T}\) for some \(n \geq 0\), and let \((x, x_T) \in \hat{\Theta}_{H\|T}\). It must be shown that \((x, x_T) \in X^{n+1}_{G\|T} = \Theta^{cont}_{G\|T}(X^n_{G\|T}) = \Theta^{cont}_{G\|T}(X^n_{G\|T}) \cap \Theta^{nonb}_{G\|T}(X^n_{G\|T})\).

First, to see that \((x, x_T) \in \Theta^{cont}_{G\|T}(X^n_{G\|T})\), let \(v \in \Sigma_u\) such that \((x, x_T) \xrightarrow{v}_{|\hat{\Theta}_{G\|T}} (y, y_T)\). Then there are three possibilities for \(v\). If \(v \notin \Sigma\) then \((x, x_T) \xrightarrow{v}_{|\hat{\Theta}_{H\|T}} (x, y_T)\). If \(v \in \Omega\) then since \(v \in \Sigma_u\), either \(x \xrightarrow{v}_{|\hat{\Theta}_{H\|T}} y\) or \(x \xrightarrow{\bot}_{|\hat{\Theta}_{H\|T}} \perp\) by definition 23. If \(v \notin \Omega\) then \(x_T = y_T\) and by \(\Upsilon\)-controllability of \(H = \text{hsupCN}_{G\|T}(G)\) it can be concluded that \((x, x_T) \xrightarrow{v}_{|\hat{\Theta}_{H\|T}} (y, y_T)\). In all cases, there exists \(y' \in Q_H\) such that \((x, x_T) \xrightarrow{v}_{|\hat{\Theta}_{H\|T}} (y', y_T)\). Since \(v \in \Sigma_u\), it follows by controllability of \(\text{supCN}(H \| T)\) that \((y', y_T) \in \hat{\Theta}_{H\|T}\). By inductive assumption \((y', y_T) \in X^n_{G\|T}\), which implies \((x, x_T) \in \Theta^{cont}_{G\|T}(X^n_{G\|T})\).

Next, it is shown that \((x, x_T) \in \Theta^{nonb}_{H\|T}(X^n_{G\|T})\). Since \((x, x_T) \in \hat{\Theta}_{H\|T}\), there exist a path
\[
(x, x_T) = (x_0, x^0_T) \xrightarrow{\sigma_1}_{|\hat{\Theta}_{H\|T}} \cdots \xrightarrow{\sigma_k}_{|\hat{\Theta}_{H\|T}} (x_{k+1}, x^k_{k+1}).
\]
Then \((x_l, x^l_T) \in \hat{\Theta}_{H\|T}\) for \(l = 0, \ldots, k\). Thus, by inductive assumption \((x_l, x^l_T) \in X^n_{G\|T}\) for \(l = 0, \ldots, k\). Therefore,
\[
(x, x_T) = (x_0, x^0_T) \xrightarrow{\sigma_1}_{|X^n_{G\|T}} \cdots \xrightarrow{\sigma_k}_{|X^n_{G\|T}} (x_{k+1}, x^k_{k+1}),
\]
which implies \((x, x_T) \in \Theta^{nonb}_{G\|T}(X^n_{G\|T})\).
Theorem 10  Let \((G; S; \rho)\) be a synthesis triple with \(G = \{G_1, \ldots, G_n\}\), and let \(\Upsilon \subseteq \Sigma_1 \cap \Sigma_u\) such that \((\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset\). Then

\[(G; S; \rho) \approx_{\text{synth}} (\{\text{hsup}CN_\Upsilon(G_1), G_2, \ldots, G_n\}; \{\text{hsup}CN_\Upsilon(G_1)\} \cup S; \rho) .\]

**Proof.** Let \(H_1 = \text{hsup}CN_\Upsilon(G_1)\). By definition 15 and lemma 25, it holds that

\[
L(\text{supCN}(G; S; \rho)) = L(\rho(\text{supCN}(G_1 \parallel G_2 \parallel \cdots \parallel G_n) \parallel S))
= L(\rho(\text{supCN}(H_1 \parallel G_2 \parallel \cdots \parallel G_n) \parallel S))
= L(\rho(\text{supCN}(H_1 \parallel G_2 \parallel \cdots \parallel G_n) \parallel H_1 \parallel S))
= L(\text{supCN}(\{H_1, G_2, \ldots, G_n\}; \{H_1\} \cup S; \rho)).
\]

Using \(H_1 = \text{hsup}CN_\Upsilon(G_1)\), the claim follows from definition 16. \(\square\)