Discrete Event Dynamic Systems (2022) 32:317–358
https://doi.org/10.1007/s10626-021-00354-0

Transformational supervisor synthesis for evolving systems

Sander Thuijsman[1] · Michel Reniers[1]

Received: 8 January 2021 / Accepted: 22 October 2021 / Published online: 23 February 2022

Abstract
Supervisory controller synthesis is a means to compute correct-by-construction controllers for discrete event systems. As these systems and their requirements evolve over time, an updated supervisor needs to be computed each time an adaptation takes place. We consider the case that a supervisor has been synthesized for a given model, after which this model is (slightly) adapted. We investigate how we can make use of the previous synthesis result, in order to more efficiently compute the supervisor for the adapted model. We introduce model deltas as a means to describe the difference between pairs of models. Using the model deltas, a notion of atomic adaptations is introduced. For these atomic adaptations, algorithms are provided to compute the supervisor for the adapted model in a transformational manner from the previous synthesis result, rather than performing a completely new synthesis. These atomic adaptations can be iterated over, to transformationally compute a supervisor for model deltas that contain a number of atomic adaptations. To improve efficiency, it is shown how atomic adaptations can be grouped together based on their required computations and be processed at the same time. A running example is used to support the explanations on the functioning of the algorithms. The efficiency of the method is evaluated by means of both an academic and an industrial use case.

Keywords Discrete event systems · Supervisory control theory · Supervisor synthesis · Software product line engineering · Delta modeling · Software evolution

1 Introduction

Supervisory control theory, as introduced by Ramadge and Wonham (1987) and Ramadge and Wonham (1989), is a model-based approach to control discrete event (dynamic)

This article belongs to the Topical Collection: *Topical Collection on Control 2022*
Guest Editors: Joerg Raisch, Carla Seatzu and Shigemasa Takai

Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement n° 826452 (project Arrowhead Tools) and from the partners national programs/funding authorities.

✉ Sander Thuijsman
s.b.thuijsman@tue.nl

1 Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, Netherlands

systems. Given a plant model (that defines all possible system behavior) and a requirement specification (which defines what plant behavior is allowed), a supervisor can be computed algorithmically *(synthesized)* that restricts the plant's behavior so that it is in accordance with the requirements. Depending on the synthesis algorithm, the supervised system has some useful properties, such as *safety*, *nonblockingness*, *controllability* and *maximal permissiveness*. The benefit of supervisory control theory has been shown in literature for varying fields of industry. Some examples where it is applied to controller design are; A patient support table of a magnetic resonance imaging scanner in Theunissen et al. (2014), chemical process control in Rawlings et al. (2014), lithography machines in van der Sanden et al. (2015), a waterway lock and movable bridge combination in Reijnen et al. (2020), construction robotics in Rosa et al. (2020), and tactical planning for automated vehicles in Krook et al. (2020). Despite the advantages of applying this technique, and the examples thereof shown in case studies, industrial acceptance is still scarce compared to other topics of control theory. Wonham et al. (2018) point to the *state space explosion* as one of the barriers to industrial acceptance. When the size of the system grows, the time and space (memory) required for synthesis grows exponentially.

We consider the situation sketched in Fig. 1; A supervisor has been synthesized for a particular specification of plant and requirements. Later, a (slight) adaptation is made to the specification, so that we are going to need a new supervisor. In state of practice, a completely new synthesis would be performed on the adapted model. We investigate how to reuse the initial model and synthesis result, in order to more efficiently synthesize a new supervisor, while the supervisor's desired properties are retained.

The reuse of artifacts during (software) development is considered in *(software) Product Line Engineering (PLE)*. Pohl et al. (2005) define: 'Software product line engineering is a paradigm to develop software applications using platforms and mass customisation.' By reusing domain artifacts and exploiting product line variability, companies can employ PLE to increase product individualization, reduce development costs, reduce time-to-market, and enhance product quality. Pohl et al. (2005) point to model-based software development as an ideal candidate for employing PLE.

Within the context of PLE, Schaefer et al. (2012) characterize *Delta Modeling* as a modular approach to model the variability of a system using transformations. A *model delta* explicitly specifies an adaptation that can be applied to some *base model*, in order to form a *variant model*. A particular variant model can be obtained by selecting one or more model deltas and applying them to the base model one-by-one.

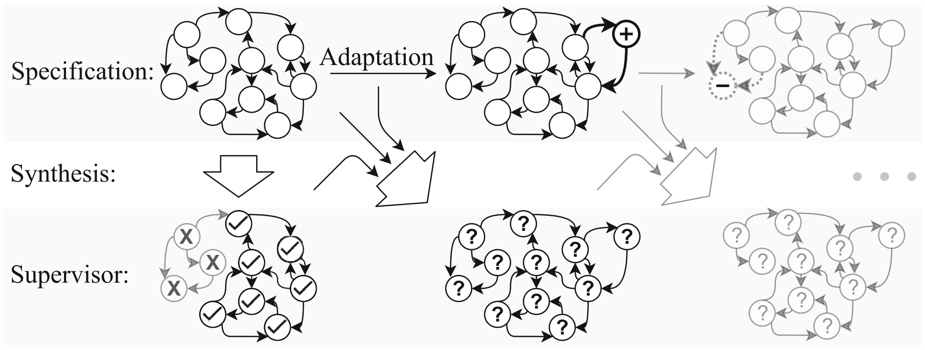

Fig. 1 Schematic overview synthesis for evolving system

Regarding adaptations that are made to software over time, Lehman (1996) has defined the *laws of software evolution*; these describe what changes typically occur during a software's lifetime. The laws themselves have evolved over the years, but *the law of continuing change* has consistently been a part of them. This law states that software controlling a cyber-physical system must continually be adapted, otherwise its functioning becomes progressively less satisfactory.

In this paper, we elaborate on a *Transformational Supervisor Synthesis (TSS)* method. This type of synthesis uses a base model, its synthesis result and model delta to obtain a supervisor. This supervisor is the same as the would-be supervisor if a completely new synthesis was performed for the variant model, which is defined by the base model and model delta. Note that in this problem statement the model delta is unknown before performing synthesis on the base model. This is a realistic constraint, following from Lehman's law of continuing change, as well as in the case of *iterative and incremental development* (Larman and Basili 2003), where system and requirement definitions are adapted during controller development. We introduce a supervisor synthesis algorithm that outputs relevant data that can be used for TSS. We present a notion for model delta, which defines adaptations made between two models, and use this notion to identify atomic adaptations, that are the smallest possible model deltas. For different types of atomic adaptations, we provide TSS algorithms that use the result from a previous synthesis to transformationally compute a supervisor. We show how we can iterate over these atomic adaptations to transformationally obtain a supervisor when multiple atomic adaptations specify the difference between any base and variant model. To improve the efficiency, we will then present an algorithm that groups atomic adaptations together based on their required computations and processes them at the same time. These algorithms are then first applied in an academic experiment in order to analyze their effectiveness. Next, an industrial case study is presented for evolution of a controller that is used in lithography machines. Finally, conclusions are provided based on these results.

1.1 Related work

This paper is strongly based on, and can be seen as an extension to, Thuijsman and Reniers (2020), where the TSS method was first introduced. The extension we present here includes more elaborate examples and explanations, an additional industrial case study, as well as theorems and their accompanying proofs. The algorithms we present here have been updated with respect to Thuijsman and Reniers (2020), some modifications were made on account of obtaining correct results, others for the sake of improving computational efficiency. Tijsse Claase (2020) is also closely related, in which a first attempt of applying TSS to symbolic supervisor synthesis is made, where binary decision diagrams are used to represent the system for efficient supervisor synthesis (Fei et al. 2014).

Within the research area of discrete event systems, PLE is mostly considered in the topic of formal verification or model checking. For example, efficient verification of linear-time temporal logic for variability-intensive systems in Classen et al. (2010) or feature-oriented modular verification of software product lines in ter Beek and de Vink (2014). Khan (2013) investigates evolving Algebraic Petri Nets, how to perform verification on the parts of the system that are affected by the property that is analyzed, and how to identify evolutions that require verification. In ter Beek et al. (2016) and Reniers and Thuijsman (2020), PLE has been applied in supervisory control. In these works a supervisory controller is synthesized for all possible product configurations given by a feature model. The output is one controller with multiple initial locations, were each initial location corresponds to a product configuration. In Reniers and Thuijsman (2020), runtime evolution of the system behavior over the configurations is studied. In contrast to this work, we do not assume a priori knowledge of the possible system configurations and the evolution takes place at design time.

2 Preliminaries

We consider finite state automaton A defined as a *5-tuple*: $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, where X is the finite set of states, of which $X_0 \subseteq X$ is the set of initial states and $X_m \subseteq X$ is the set of marked states. Σ is the finite set of events, also called the alphabet, which is partitioned into sets of controllable and uncontrollable events, respectively Σ_c and Σ_u. Σ^* denotes all possible finite strings using events in Σ. $\longrightarrow$ is the finite set of transitions, a transition is a *3-tuple*: $(x_{or}, \sigma, x_{tar}) \in X \times \Sigma \times X$, specifying a transition from origin state x_{or} to target state x_{tar} over event σ. We denote the existence of a transition $(x_{or}, \sigma, x_{tar}) \in \longrightarrow$ by: $x_{or} \xrightarrow{\sigma} x_{tar}$. Likewise, the existence of a sequence of transitions over intermediate states can be addressed by: $x_{or} \xrightarrow{s} x_{tar}$, for $s \in \Sigma^*$.

The *synchronous product* of automata $A_1 = (X_1, \Sigma_1, \longrightarrow_1, X_{0,1}, X_{m,1})$ and $A_2 = (X_2, \Sigma_2, \longrightarrow_2, X_{0,2}, X_{m,2})$ is defined as: $A_1||A_2 = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \longrightarrow_{12}, X_{0,1} \times X_{0,2}, X_{m,1} \times X_{m,2})$, where $\longrightarrow_{12}$ is constructed by:

$$\begin{aligned}
&((x_{or,1}, x_{or,2}), \sigma, (x_{tar,1}, x_{tar,2})) \in \longrightarrow_{12},\\
&\qquad \text{if } \sigma \in \Sigma_1 \cap \Sigma_2,\ x_{or,1} \xrightarrow{\sigma}_1 x_{tar,1},\ x_{or,2} \xrightarrow{\sigma}_2 x_{tar,2}\\
&((x_{or,1}, x_2), \sigma, (x_{tar,1}, x_2)) \in \longrightarrow_{12},\\
&\qquad \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2,\ x_{or,1} \xrightarrow{\sigma}_1 x_{tar,1}\\
&((x_1, x_{or,2}), \sigma, (x_1, x_{tar,2})) \in \longrightarrow_{12},\\
&\qquad \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1,\ x_{or,2} \xrightarrow{\sigma}_2 x_{tar,2}
\end{aligned} \tag{1}$$

For a given automaton A, we apply supervisor synthesis to generate a supervisor *subautomaton* S of A that is *reachable*, *coreachable*, *controllable*, and *maximally permissive*.

An automaton $S = (Y, \Sigma_S, \longrightarrow_S, Y_0, Y_m)$ is a subautomaton of $A = (X, \Sigma, \longrightarrow, X_0, X_m)$ if $Y \subseteq X$, $\Sigma_S = \Sigma$, $\longrightarrow_S \subseteq \longrightarrow$, $Y_0 \subseteq X_0$, and $Y_m \subseteq X_m$. In this work, the subautomata we encounter are restricted to $\longrightarrow_S = \longrightarrow \cap (Y \times \Sigma \times Y)$, $Y_0 = Y \cap X_0$, and $Y_m = Y \cap X_m$.

A state $x_r \in X$ is *reachable* if it can be reached from some initial state; $x_0 \xrightarrow{s} x_r$ for some $x_0 \in X_0,\ s \in \Sigma^*$. A state $x_{cr} \in X$ is *coreachable* if from it a marked state can be reached; $x_{cr} \xrightarrow{s} x_m$ for some $x_m \in X_m,\ s \in \Sigma^*$. Supervisor automaton S is called (co-) reachable for plant automaton A, if all its states can be defined as such. An automaton for which all reachable states are coreachable is commonly called nonblocking in literature. We say that for automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor automaton $S = (Y, \Sigma_S, \longrightarrow_S, Y_0, Y_m)$ is *controllable* if $\longrightarrow \cap (Y \times \Sigma_u \times X) \subseteq \longrightarrow_S$. If S is controllable, the states in Y are also called controllable. Maximally permissive says that S is the maximal subautomaton of A for which coreachability, reachability, and controllability are ensured. Meaning the supervisor does not disable any transitions that do not strictly need to be disallowed.

In addition to the properties of the supervisor mentioned above, problem formulations for supervisor synthesis often include a *safety* constraint; Along with the plant, some requirement specification on the plant's behavior is given. The supervisor should restrict the behavior of the plant so that the requirement specification is always satisfied. In such a case, a *plantified* requirement automaton can be constructed by introducing a non-coreachable

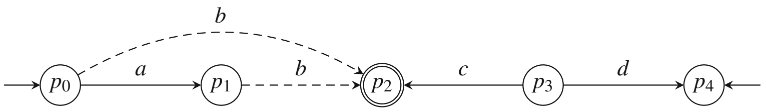

Fig. 2 Plant automaton P

sink state. Transitions that are not in accordance with the specification are redirected to this sink state. Given a requirement automaton $R = (X, \Sigma, \longrightarrow, X_0, X_m)$, the plantified requirement automaton is obtained as follows (Flordal et al. 2007):

$$R^{\perp} = (X \cup \{\perp\}, \Sigma, \longrightarrow^{\perp}, X_0, X_m), \tag{2}$$

where $\perp \notin X$ is the new sink state and

$$\longrightarrow^{\perp} = \longrightarrow \cup \{(x, \sigma_u, \perp) | x \in X, \sigma_u \in \Sigma_u, \nexists (x_{tar} \in X) x \xrightarrow{\sigma_u} x_{tar}\}.$$

A safe supervisor can be obtained by synthesizing a coreachable and controllable supervisor on the synchronous product of the plant automata and plantified requirement automata, as proven in Flordal et al. (2007). Other ways of specifying requirements can be plantified as well. For example state exclusion requirements discussed in Markovski et al. (2010) are plantified by removing excluded controllable transitions from the plant, and directing the excluded uncontrollable transitions to the sink state. Therefore, in this work we will consider synthesizing a coreachable supervisor for a single automaton, without loss in generality regarding safety constraints or networks of automata.

We allow automata to be non-deterministic. In the case of non-determinism, we allow the supervisor to be able to disable individual controllable transitions as a result of the removal of unsafe states. So if in the plant a state has two outgoing transitions over the same controllable event, the supervisor subautomaton may contain this state with only one of these outgoing transitions. This is unlike some traditional supervisory control definitions, by for example (Ramadge and Wonham 1989) or (Cassandras and Lafortune 2008), where multiple outgoing events over the same event can not be disabled individually. The distinction between these paradigms is further discussed in Flordal et al. (2007).

2.1 Running example

We will consider the plant automaton P of Fig. 2 and requirement automaton R of Fig. 3 as a running example throughout this paper. A solid or dashed arrow respectively indicates a transition by a controllable or an uncontrollable event. The initial states are indicated by the incoming arrows, and the marked states are indicated by a double circle. Requirement automaton R has been plantified, resulting in the plantified requirement automaton $R^{\perp}$ in Fig. 4. Constructing the synchronous product $P||R^{\perp}$ yields automaton A of Fig. 5. Note that for the remainder of this paper, when we discuss model deltas to this example, they are always to automaton A directly, not to P and R with an implied delta on A.

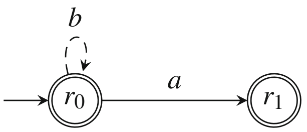

Fig. 3 Requirement automaton R

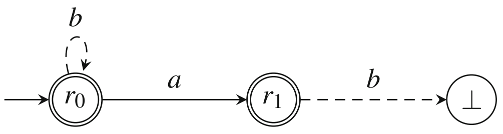

Fig. 4 Plantified requirement automaton $R^{\perp}$

Algorithm 1 Supervisor synthesis (SS).

```
Input: Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$
Output: Supervisor $S = (Y, \Sigma, \longrightarrow_S, Y_0, Y_m)$, good states $G$
1: $(Y, G) =$computeFixpoint$(A)$
2: $S = (Y, \Sigma, \longrightarrow \cap (Y \times \Sigma \times Y), X_0 \cap Y, X_m \cap Y)$
3: return $(S, G)$
```

2.2 Supervisor synthesis algorithm

In Algorithm 1 a supervisor synthesis algorithm is presented. It is based on the algorithm introduced in Ouedraogo et al. (2011). However, we consider Finite State Automata rather than Extended Finite Automata for sake of simplicity. We also omitted the use of forbidden states in the algorithm, as we plantify the requirements. In case the plant behavior is given by multiple plant automata, the input automaton for this algorithm can be obtained by calculating the synchronous product of the plant automata. We present this algorithm using function calls to other algorithms to facilitate reuse of these (sub-)algorithms later in this paper. The algorithm uses a fixpoint computation, provided in Algorithm 2, which iteratively calculates a set of *coreachable* states G, followed by a set of *bad* states B, that are non-coreachable or have a sequence of uncontrollable transitions to a non-coreachable state. The calculation to obtain G and B is done by the means of a *Backward Reachability Search (BRS)*, given in Algorithm 3, for which Lemma 1 holds. This is a Breadth First Search algorithm taken from Kleinberg and Tardos (2005) that has a linear runtime complexity. All found states are added to X_ω. The state space is searched in layers. For each state in the current layer, all undiscovered states that have a transition to this state are added to the next layer. After all states in the current layer have been evaluated, the algorithm moves to evaluating the states

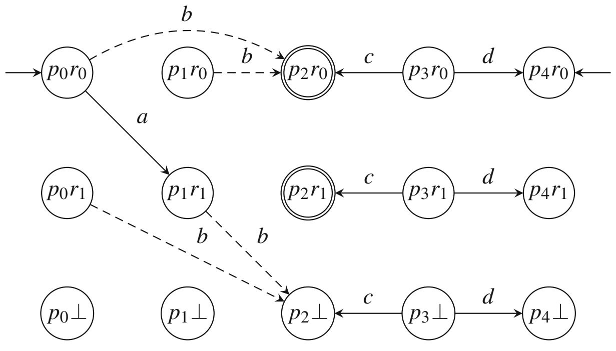

Fig. 5 Automaton $A = P||R^{\perp}$

in the next layer. These steps are repeated until no more new states are found. The algorithm is slightly adapted from Kleinberg and Tardos (2005) to allow a set of starting states, instead of a singular starting state. Also, at the start of the algorithm the transitions are pruned, so that only transitions between states in the input state set X are considered. Transitions from states in the starting set X_α are also removed, as these states are already discovered as the starting set, so analyzing these transitions is not necessary. The functioning of this algorithm is well known so Lemma 1 is not proven here.

Algorithm 2 Compute Fixpoint (`computeFixpoint`).

```
Input: Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$
Output: Supervisor states $Y$, good states $G$
1: $G' = X$
2: repeat
3:     $G = G'$
4:     $G' = \mathtt{BRS}(G, \Sigma, \longrightarrow, X_m)$
5:     $B = \mathtt{BRS}(G, \Sigma_{uc}, \longrightarrow, G \setminus G')$
6:     $G' = G \setminus B$
7: until $G' = G$
8: $Y = \mathtt{FRS}(G, \Sigma, \longrightarrow, X_0)$
9: return $(Y, G)$
```

Algorithm 3 Backward Reachability Search (`BRS`).

```
Input: State set $X$, alphabet $\Sigma$, finite set of transitions $\longrightarrow$, starting set $X_\alpha$
Output: State set $X_\omega$ in $X$ from which a sequence of transitions $\longrightarrow$ exists through states
    in $X$, using events in $\Sigma$, to a state in $X_\alpha \cap X$
1: $\longrightarrow_p = \{(x_{or}, \sigma, x_{tar}) \in \longrightarrow \mid (x_{or} \in X \wedge x_{tar} \in X) \wedge x_{or} \notin X_\alpha \wedge \sigma \in \Sigma\}$
2: $X_\omega = X_\alpha \cap X$
3: currentlayer $= X_\alpha \cap X$
4: while currentlayer $\neq \emptyset$
5:     nextlayer $= \emptyset$
6:     for all $x \in$ currentlayer do
7:         for all $\{(x_{or}, \sigma, x_{tar}) \in \longrightarrow_p \mid x_{tar} = x\}$ do
8:             if $x_{or} \notin X_\omega$
9:                 $X_\omega = X_\omega \cup \{x_{or}\}$
10:                nextlayer = nextlayer $\cup \{x_{or}\}$
11:            end if
12:        end for
13:    end for
14:    currentlayer $=$ nextlayer
15: end while
16: return $X_\omega$
```

Lemma 1 *For state set X, alphabet Σ, set of transitions $\longrightarrow$, and starting state set X_α; `BRS`$(X, \Sigma, \longrightarrow, X_\alpha)$ contains all states in X from which a state in $X_\alpha \cap X$ can be reached, using transitions in $\longrightarrow$, over states in X, that have an event in Σ.*

The bad states B are removed from G. The removal of these states can induce other states to become non-coreachable. Therefore, the algorithm repeats these steps until no further states get removed. At this point, the set of remaining states is defined as *good states* G, which is the maximal set of controllable and coreachable states, see Lemma 2. Proof for Lemma 2 is provided in Ouedraogo et al. (2011).

Lemma 2 *For automaton* $A = (X, \Sigma, \longrightarrow, X_0, X_m)$*, and* $(Y, G) =$ `computeFixpoint`(A)*; G is the maximal controllable and coreachable set of states in X.*

Algorithm 4 Forward reachability search (`FRS`).

```
Input: State set $X$, alphabet $\Sigma$, finite set of transitions $\longrightarrow$, starting set $X_\alpha$
Output: State set $X_\omega$ in $X$ to which a sequence of transitions $\longrightarrow$ exists through states in
    $X$, using events in $\Sigma$, from a state in $X_\alpha \cap X$
1: $\longrightarrow^{-1} = \{(x_{tar}, \sigma, x_{or}) | (x_{or}, \sigma, x_{tar}) \in \longrightarrow\}$
2: $X_\omega =$ BRS$(X, \Sigma, \longrightarrow^{-1}, X_\alpha)$
3: return $X_\omega$
```

Then, in order to generate a reachable supervisor, a *Forward Reachability Search (FRS)* (Algorithm 4, Lemma 3) is carried out from the set of initial states, resulting in states Y. Essentially `BRS` is performed with all transitions reversed to search forward instead of backward. Same as for `BRS`, the accompanying lemma is not proven here. Y is the maximal controllable, coreachable, and reachable subset of X, see Lemma 4.

Lemma 3 *For state set X, alphabet Σ, set of transitions $\longrightarrow$, and starting state set X_α;* `FRS`$(X, \Sigma, \longrightarrow, X_\alpha)$ *contains all states in X that can be reached from a state in $X_\alpha \cap X$, using transitions in $\longrightarrow$, over states in X, that have an event in Σ.*

Lemma 4 *For automaton* $A = (X, \Sigma, \longrightarrow, X_0, X_m)$*, and* $(Y, G) =$ `computeFixpoint`(A)*; Y is the maximal controllable, coreachable, and reachable set of states in X.*

Proof Y is the maximal reachable set in G, following from Lemma 3. It is shown by Lemma 2 that G is the maximal controllable and coreachable subset of X. Thus, the maximal reachable subset Y in G is the maximal controllable, coreachable, and reachable subset of X. □

Together, Y, alphabet Σ, the transitions of A between the states in Y, the initial states in Y, and the marked states in Y define the supervisor automaton S. Supervisor S is the maximal subautomaton of A that is reachable, coreachable, controllable; see Theorem 1. The algorithm always computes a supervisor automaton. If there are no reachable, coreachable, and controllable states then the supervisor automaton will contain no states, and hence no transitions.

Theorem 1 *For automaton A, and* $(S, G) =$ `SS`(A)*; S is maximally permissive, controllable, coreachable, and reachable with respect to A.*

Proof By construction, S is the maximal subautomaton of A over states in Y. Y is the maximal controllable, coreachable and reachable set of states in A (Lemma 4). It follows that S is maximally permissive, controllable, coreachable, and reachable with respect to A. □

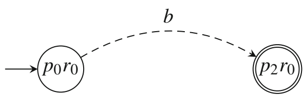

Fig. 6 Supervisor S, for $(S, G) = \mathtt{SS}(A)$

Next to supervisor S, the synthesis algorithm outputs good state set G, in order to facilitate reuse of this set in other computations. Note that this state set is computed anyways during synthesis, it is not computed specifically for the facilitation of reuse.

2.2.1 Example

When applying the supervisor synthesis algorithm to automaton A of Fig. 5, first the supervisor states and good states are calculated by `computeFixpoint` (Algorithm 2). The supervisor states are $\{p_0r_0, p_2r_0\}$, the good states are $\{p_0r_0, p_1r_0, p_2r_0, p_3r_0, p_2r_1, p_3r_1\}$. Next, the supervisor automaton is constructed, which provides the supervisor automaton given in Fig. 6.

For convenience we also provide a visualization of automaton A, where the states are color coded depending on their containment in the state sets resulting from synthesis, in Fig. 7. Supervisor states (Y) (that are also good states by definition) are displayed white, good states that are not supervisor states ($G \setminus Y$) are displayed grey, and non-good states ($X \setminus G$) are displayed black.

3 Model delta

For the purpose of TSS we wish to model the difference between the base and variant model. We can represent any adaptation from base to variant automaton as *model delta* as *10-tuple*: $\Delta = (X^+, X^-, \Sigma^+, \Sigma^-, \longrightarrow^+, \longrightarrow^-, X_0^+, X_0^-, X_m^+, X_m^-)$, which for each of the sets in the 5-tuple definition of automaton A, defines the added ($^+$) and removed ($^-$) elements of that set. Σ^+ and Σ^- are both partitioned into sets of controllable and uncontrollable events that are added or removed. The following constraints apply to the model delta:

- All removed elements within the model delta, must exist in the base model: $X^- \subseteq X,\ \Sigma^- \subseteq \Sigma, ...$
- All added elements within the model delta, must not yet exist in the base model: $X^+ \cap X = \emptyset,\ \Sigma^+ \cap \Sigma = \emptyset, ...$

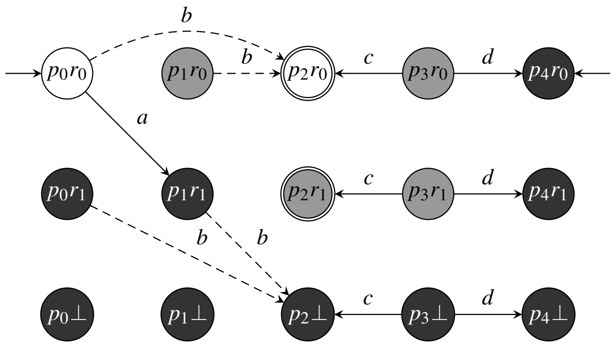

Fig. 7 Automaton A, color coded by synthesis result

- Elements can not simultaneously be added and removed $X^- \cap X^+ = \emptyset,\ \Sigma^- \cap \Sigma^+ = \emptyset, \ldots$.
- The initial and marked states of the variant model must exist in the variant state set: $X_0^+ \subseteq (X \cup X^+) \setminus X^-,\ X_m^+ \subseteq (X \cup X^+) \setminus X^-$.
- Transitions must go to-and-from states, by defined events: $\longrightarrow' \subseteq X' \times \Sigma' \times X'$, where $X' = (X \cup X^+) \setminus X^-,\ \Sigma' = (\Sigma \cup \Sigma^+) \setminus \Sigma^-$, and $\longrightarrow' = (\longrightarrow \cup \longrightarrow^+) \setminus \longrightarrow^-$.

When these constraints are met, we call the model delta *valid*.

Given base automaton A and model delta Δ, variant automaton A' is constructed by: $A' = (X', \Sigma', \longrightarrow', X_0', X_m')$, where: $X' = (X \cup X^+) \setminus X^-$, $\Sigma' = (\Sigma \cup \Sigma^+) \setminus \Sigma^-$, $\longrightarrow' = (\longrightarrow \cup \longrightarrow^+) \setminus \longrightarrow^-$, $X_0' = (X_0 \cup X_0^+) \setminus X_0^-$, $X_m' = (X_m \cup X_m^+) \setminus X_m^-$. For a valid model delta, and well-defined base automaton, the constructed variant automaton is well-defined.

Furthermore, for any pair of well-defined automata ($A = (X, \Sigma, \longrightarrow, X_0, X_m)$, $A' = (X', \Sigma', \longrightarrow', X_0', X_m')$), a valid model delta is constructed as follows: $X^+ = X' \setminus X$, $X^- = X \setminus X'$, $\Sigma^+ = \Sigma' \setminus \Sigma$, $\Sigma^- = \Sigma \setminus \Sigma'$, $\longrightarrow^+ = \longrightarrow' \setminus \longrightarrow$, $\longrightarrow^- = \longrightarrow \setminus \longrightarrow'$, $X_0^+ = X_0' \setminus X_0$, $X_0^- = X_0 \setminus X_0'$, $X_m^+ = X_m' \setminus X_m$, $X_m^- = X_m \setminus X_m'$.

The change of controllability of an event can be modeled by removing all transitions that are labeled by this event, and adding these transitions back with an added event with modified controllability.

We may address a model delta with only its non-empty part. So if we mention a model delta with $X_0^+ = \{x^\delta\}$, and no other information, this implies that the other elements in the model delta tuple are empty. In the remainder of this paper, when considering a model delta Δ it is implied that this is a model delta from base automaton A to variant automaton A'.

4 Atomic adaptations

In this section we consider *atomic adaptations*, where the difference between the base and variant model can be described by a single, indivisible change in the automaton specification. Formally we can say that a model delta Δ is an *atomic adaptation* when only one of the tuple-elements is a set of size one, and all other elements are empty; $|X^+| + |X^-| + |\Sigma^+| + |\Sigma^-| + |\longrightarrow^+| + |\longrightarrow^-| + |X_0^+| + |X_0^-| + |X_m^+| + |X_m^-| = 1$.

We consider several types of atomic adaptations, e.g., removing a transition, or adding the marked property to a state, for which we provide an atomic TSS algorithm. The purpose of these algorithms is to calculate the supervisor states and good states of the variant automaton, using the base automaton, its synthesis result, and model delta. Theorem 2 holds for the algorithms. Essentially, the properties of the supervisor ((co-)reachability, controllability, and maximal permissiveness) are retained during atomic TSS. For sake of cohesion, proofs of Theorem 2 for each algorithm are given separately in Appendix A.

Theorem 2 *Given base automaton A, fixpoint result $(Y, G) = \texttt{computeFixpoint}(A)$, and atomic adaptation Δ for which the atomic TSS algorithm is given, the atomic TSS algorithm provides a supervisor state set Y' and good state set G' such that they are equal to the fixpoint result of the variant automaton; $(Y', G') = \texttt{computeFixpoint}(A')$.*

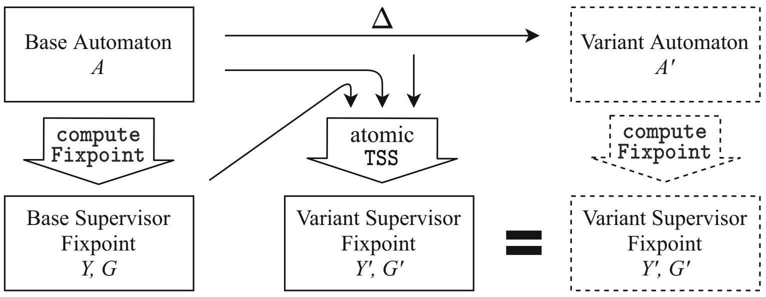

Fig. 8 Inputs and outputs for atomic TSS for atomic adaptation Δ

Figure 8 shows an overview of the atomic TSS method. It is similar to Fig. 1, only now the names of the artifacts and algorithms that have been introduced are shown. The figure shows that the fixpoint for the variant supervisor can be computed in two ways, either by (1) performing `computeFixpoint` on the variant automaton directly, or by (2) performing atomic TSS using the base automaton, base supervisor fixpoint, and atomic model adaptation. Either way, the fixpoint result is the same.

In the following subsections we consider adding or removing the initial property to some state, adding or removing the marked property to some state, and adding or removing a transition respectively. For the cases of removed or added states and events no algorithms are provided. These atomic adaptations are discussed in Section 4.7. After presenting the atomic TSS algorithms in this section, we will show how we can iterate over them in Section 5, where we are also going to group atomic adaptations together to process them at once.

The algorithms are strongly based on Thuijsman and Reniers (2020). In some places minor modifications are made for sake of correctness and efficiency. Some of these modifications are discussed in Tijsse Claase (2020). The authors note that these modifications influence the experimental results presented in Thuijsman and Reniers (2020), however only to a small enough extent that they do not influence the conclusions made on those results. All algorithms are also modified to compute the supervisor state set Y instead of the supervisor S, leading to shorter notations. S can simply be computed from Y, as in line 2 of Algorithm 1.

4.1 Added initial property

We assume the situation that $(Y, G) =$ `computeFixpoint`(A) has been calculated for base automaton A. Some state of base automaton A has been made an initial state, which is the only adaptation to create variant automaton A'. In Algorithm 5 the atomic TSS algorithm is provided to compute supervisor states Y' and good states G' for the variant model, given A, Y, G, and the state with added initial property x^{δ}. The algorithm uses a switch statement, where the value of a variable, in this case x^{δ}, is tested for multiple cases. Once a case match is found, the statements associated with the particular case are executed. In case no match is found, the default statements are executed. For all atomic TSS algorithms the switch cases are mutually exclusive, which means that for the given atomic adaptation only one switch case holds, or none and then the default statement is executed.

Algorithm 5 Atomic Transformational Supervisor Synthesis for Added Initial Property (TSSAIP).

```
Input: Base automaton $A = (X,\Sigma,\longrightarrow,X_0,X_m)$, supervisor states $Y$, good states $G$, state
    with added initial property $x^\delta \in X_0^+$
Output: Variant supervisor states $Y'$, variant good states $G'$
1: switch $x^\delta$
2:   case $x^\delta \in G \setminus Y$ do
3:     $Y' = \mathtt{FRS}(G,\Sigma,\longrightarrow,Y \cup \{x^\delta\})$
4:     $G' = G$
5:   default do
6:     $Y' = Y,\ G' = G$
7: end switch
8: return $(Y',G')$
```

In Algorithm 5 it can be seen that two cases are considered, the first being that x^δ is in $G \setminus Y$. A state in $G \setminus Y$ is coreachable and controllable in the base model. It was not reachable in the base model, as in that case it would be part of Y. Due to addition of the initial property, we now know that x^δ is reachable, so it should become part of Y'. It is possible that more states in $G \setminus Y$ have become reachable due to x^δ being reachable, so an FRS is carried out over states G. We already know that states in Y were reachable, and they will remain reachable in the variant model. As we do not want to reinvest computational effort in finding these states in Y again, the FRS is already initiated with states Y in the starting set along with x^δ, essentially we already start closer to the fixpoint that we wish to find. States in $X \setminus G$ are not considered, as they remain non-coreachable or non-controllable in the variant model. In our running example, we can consider the adaptation to make p_3r_1 initial, which would fit under this particular case. As p_2r_1 is a reachable good state from p_3r_1, both p_3r_1 and p_2r_1 will be added to the supervisor states Y to construct the supervisor states for the variant model Y'.

Alternatively, x^δ may be in $X \setminus G$. As we just noted, the adaptation of initial states does not influence the set of coreachable and controllable states. So in this case, we already performed the FRS over the same set G to compute Y. As G did not change, the supervisor and good states remain the same for the variant model. In our example we can consider making p_4r_1 an initial state as such an adaptation. In that variant model, p_4r_1 will remain a non-good state in $X \setminus G'$.

Finally, x^δ may be in Y. If this is the case, just like in the previous example the default statement will be executed. x^δ was already found in the FRS of the base model, so also all reachable states from x^δ in G are included in Y. Thus, the supervisor states and good states remain the same for the variant model. In the running example, making p_2r_0 an initial state would be of this case.

4.2 Removed initial property

We consider a similar situation as the previous section, only this time the initial property has been removed from a state instead of added. The atomic TSS algorithm for this case is shown in Algorithm 6.

Algorithm 6 Atomic Transformational Supervisor Synthesis for Removed Initial Property (TSSRIP).

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor states $Y$, good states $G$, state
    with removed initial property $x^\delta \in X_0^-$
Output: Variant supervisor states $Y'$, variant good states $G'$
1: switch $x^\delta$
2:   case $x^\delta \in Y$ do
3:     $Y' = \texttt{FRS}(Y, \Sigma, \longrightarrow, X_0 \setminus \{x^\delta\})$
4:     $G' = G$
5:   default do
6:     $Y' = Y$, $G' = G$
7: end switch
8: return $(Y', G')$
```

In the first case of the algorithm, an initial state in Y is removed. This might lead to some states in Y being unreachable in the variant model. However the good states G are not influenced by the initial states, so they remain the same. Also unreachable states will remain unreachable. So an FRS is carried out over the previously reachable states Y, from the new set of initial states for the variant model to compute Y'. In the running example, the removal of initial property from p_0r_0 would fit in this case. Consequently, there are no initial states left in Y, so for the variant model there are no supervisor states; $Y' = \emptyset$. The good states remain the same.

The other case is that the removed initial state is not in Y, considered under the default statement of Algorithm 6. Actually we know that in this case the removed initial state is in $X \setminus G$, as states in $G \setminus Y$ could not be an initial state, they would already have been in Y as a reachable state in G. As x^δ is not in the maximal controllable and coreachable set in this case, it does not matter if it is reachable, or initial. It will not be part of the good states and supervisor states. In the running example this could be demonstrated by the removal of the initial property from state p_4r_0.

Algorithm 7 Atomic Transformational Supervisor Synthesis for Added Marked Property (TSSAMP).

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor states $Y$, good states $G$, state
    with added marked property $x^\delta \in X_m^+$
Output: Variant supervisor states $Y'$, variant good states $G'$
1: switch $x^\delta$
2:   case $x^\delta \in X \setminus G$ do
3:     $(Y', G') = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow, Y \cup X_0, Y \cup X_m \cup \{x^\delta\}))$
4:   default do
5:     $Y' = Y$, $G' = G$
6: end switch
7: return $(Y', G')$
```

4.3 Added marked property

In Algorithm 7 the atomic TSS algorithm is provided for the case that a state was given the marked property in the model delta from A to A'.

In the first case the circumstance is considered that a non-good state is now marked. This means that some non-good states may become good and/or supervisor states because they are coreachable in the variant model. All states that were supervisor states and good states will remain so. So a fixpoint computation is instantiated where the supervisor states are already added as marked states and initial states, so this part of the states does not have to be found again in the reachability searches. In the running example we could consider the case that p_1r_1 was made a marked state. During the first iteration of the fixpoint computation it will be found as a good state, as it is marked. It will however be removed from the good states since it has an uncontrollable transition to a bad state. So for this specific example the supervisor and good states will remain the same from base to variant model.

If the state with added marked property is a good state, it was already coreachable, as well as all good states that can reach this state. So giving it a marked property is not going to change the coreachability. Thus the supervisor states and the good states remain the same. In the running example, this would be the case if for instance p_1r_0 was added to the set of marked states.

4.4 Removed marked property

Now the case is considered that a marked state in the base model, is not a marked state in the variant model. The atomic TSS algorithm for this case is given in Algorithm 8.

Algorithm 8 Atomic Transformational Supervisor Synthesis for Removed Marked Property (TSSRMP).

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor states $Y$, good states $G$, state
    with removed property $x^\delta \in X_m^-$
Output: Variant supervisor states $Y'$, variant good states $G'$
1: switch $x^\delta$
2:   case $x^\delta \in G \setminus Y$
3:     $(Y', G') = \mathtt{computeFixpoint}((G, \Sigma, \longrightarrow, Y \cup X_0, (Y \cup X_m) \setminus \{x^\delta\}))$
4:   case $x^\delta \in Y$
5:     $(Y', G') = \mathtt{computeFixpoint}((G, \Sigma, \longrightarrow, X_0, X_m \setminus \{x^\delta\}))$
6:   default do
7:     $Y' = Y,\ G' = G$
8: end switch
9: return $(Y', G')$
```

In the first case the removal of a marked state in $G \setminus Y$ is considered. The supervisor states Y will remain the same, as x^δ is not reachable from such a state, the removal of its marked property does not influence their coreachability. Some good states might not be good states for the variant model, as they may not be coreachable anymore. Therefore a fixpoint computation is performed, with all supervisor states already added as initial states and marked states, so this part of the state space does not have to be searched anymore. In the running example we can consider removing the marked property from p_2r_1. Consequently, the supervisor for the variant model will remain the same, but p_2r_1 and p_3r_1 are not good states for the variant model as they are not coreachable anymore.

The second case considers the situation that the removed marked state is in Y. A new fixpoint computation is performed for the variant model, only the non-good states are not taken into account, as the removal of a marked state will not increase the maximal set of coreachable and controllable states. In the example we can consider removing the marked property of state p_2r_0. As a result, there will be no supervisor states in the variant model, $Y' = \emptyset$, and only p_2r_1 and p_3r_1 remain in G'.

The final case will occur when the marked property is removed from a non-good state. This does not influence the coreachability and controllability of the good states, as they must be coreachable for marked states in G. Also the reachable part of the good states remains the same. So the variant model has the same supervisor states and good states as the base model.

4.5 Added transition

Now we consider the case that only a single transition has been added to the base automaton A to create variant automaton A'. So all elements in Δ are empty except $\longrightarrow^+$ which only contains the transition $(x_{or}, \sigma, x_{tar})$. Algorithm 9 computes the supervisor and good states for the variant model according to the type of adaptation that is made.

Algorithm 9 Atomic Transformational Supervisor Synthesis for Added Transition (`TSSAT`).

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor states $Y$, good states $G$, added
    transition $(x_{or}, \sigma, x_{tar}) \in \longrightarrow^+$
Output: Variant supervisor states $Y'$, variant good states $G'$
1: switch $(x_{or}, \sigma, x_{tar})$
2:   case $x_{or} \in Y \wedge x_{tar} \in G \setminus Y$ do
3:     $Y' = \texttt{FRS}(G, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, Y)$
4:     $G' = G$
5:   case $x_{or} \in G \wedge x_{tar} \in X \setminus G \wedge \sigma \in \Sigma_u$ do
6:     $(Y', G') = \texttt{computeFixpoint}((G, \Sigma, \longrightarrow \cap ((G \setminus \{x_{or}\}) \times \Sigma \times G), X_0 \setminus \{x_{or}\}, X_m \setminus$
     $\{x_{or}\}))$
7:   case $x_{or} \in X \setminus G \wedge x_{or} \neq x_{tar}$
8:     $(Y', G') = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, Y \cup X_0, Y \cup X_m))$
9:   default do
10:    $Y' = Y, G' = G$
11: end switch
12: return $(Y', G')$
```

The first case considers an added transition from a state in Y to a state in $G \setminus Y$. The target and origin state where already coreachable and controllable, so this is not influenced by the addition of the transition. However the target state is now reachable, which it wasn't before. To find all good states that are now reachable, an FRS is performed to compute the variant supervisor states. The good states remain the same. In the running example we can consider the addition of a transition from p_0r_0 to p_3r_1. In that case, p_3r_1 and p_2r_1 would become supervisor states in the variant model in the addition to the states already in Y. The good states remain unchanged.

Next, the addition of an uncontrollable transition from a good state to a non-good state is considered. The target state was non-coreachable or non-controllable, so transitions to this state need to be disabled. Because an uncontrollable transition is added, it can not be disabled by the supervisor. This means that the origin state is not a good state anymore, and we wish to remove it for the variant model. The state is made non-coreachable by removing all outgoing transitions from it, and removing it from the set of marked states in case it was a marked state. The fixpoint computation is then performed on the state space spanned by the good states, with the origin state of the added transition as non-coreachable. In the running example this could be an added uncontrollable transition from p_2r_1 to $p_2\perp$. For that adaptation p_2r_1 and p_3r_1 would be removed from the good states to construct the variant good states, and the supervisor states remain the same.

In case the origin state is not a good state, adding the transition that is not a self-loop ($x_{or} \neq x_{tar}$) might influence the coreachability of this and other non-good states. States that were supervisor states in the base model will remain so. Thus, a fixpoint computation is performed over the entire state set, where the supervisor states are added to the marked and initial states so that this part of the state space does not need to be searched to reduce computational effort. For the running example we can consider the case that a transition is added from p_4r_1 to p_2r_0, in that case the supervisor states will remain the same, and p_4r_1 is added to the good states to construct the variant good states.

In all other cases, the supervisor states and good states remain the same. For example the addition of a transition from-and-to a supervisor state, all states that are good states will remain coreachable and controllable, and states in $G \setminus Y$ will remain non-reachable. In the example this could be a transition from p_2r_0 to p_0r_0. Another example is adding a self-loop to a non-good state not influencing its non-coreachability or non-controllability. In the running example this may be a transition from p_4r_1 to p_4r_1.

4.6 Removed transition

Here we consider the case that only a single transition has been removed from base automaton A to create variant automaton A'. Algorithm 10 computes the supervisor states and good states for the variant model according to the state sets the origin and target state of this transition belong to, and the controllability of the transition.

Algorithm 10 Atomic Transformational Supervisor Synthesis for Removed Transition (TSSRT).

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor states $Y$, good states $G$, removed transition $(x_{or}, \sigma, x_{tar}) \in \longrightarrow^-$
Output: Variant supervisor states $Y'$, variant good states $G'$
1: switch $(x_{or}, \sigma, x_{tar})$
2:   case $x_{or} \in Y \wedge x_{tar} \in Y \wedge x_{or} \neq x_{tar}$ do
3:     $(Y', G') = \texttt{computeFixpoint}((G, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m))$
4:   case $x_{or} \in G \setminus Y \wedge x_{tar} \in G \wedge x_{or} \neq x_{tar}$ do
5:     $(Y', G') = \texttt{computeFixpoint}((G, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, Y \cup X_0, Y \cup X_m))$
6:   case $x_{or} \in X \setminus G \wedge x_{tar} \in X \setminus G \wedge \sigma \in \Sigma_u \wedge x_{or} \neq x_{tar}$ do
7:     $(Y', G') = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, Y \cup X_0, G \cup X_m))$
8:   default do
9:     $Y' = Y, G' = G$
10: end switch
11: return $(Y', G')$
```

Let us consider a removed transition, that is not a self-loop, for which x_{or} and x_{tar} both were in Y. This case is considered first in Algorithm 10. It is possible that due to the removal of this transition, x_{or} and other states in G might not be coreachable anymore. Also x_{tar} might not be reachable anymore. However, bad states and non-reachable states will remain as such. Therefore, a fixpoint computation is performed for the variant model, only for the states in G of the base model. As this synthesis is on a reduced state-set, it will require less effort to perform than a completely new synthesis on the variant model. In the running example this could be the removal of transition (p_0r_0, b, p_2r_0), which would lead to no supervisor states for the variant automaton. p_0r_0 would also be removed from the good state set to construct the good states G'.

Next, we consider a removed transition from a state x_{or} in $G \setminus Y$ to a state x_{tar} in G, that is not a self-loop. x_{or}, and other good states, might become non-coreachable after removal of this transition. However, as these states are not reachable from the supervisor states, these will remain the same for the variant model. To find the set of good states for the variant model, a fixpoint computation is performed over the good states, where the supervisor states are added to the marked and initial states so that this part of the state space is not searched. For the running example, removing transition (p_3r_1, c, p_2r_1) would fall under this case. As a result the supervisor states remain the same from the base to the variant model, but p_3r_1 is removed from the good states to construct the variant good states G'.

As a third case in Algorithm 10, an uncontrollable transition is removed with origin and target state as not good states. It is possible that the origin state and other states were not good states due to the existence of this transition. We need to perform some additional fixpoint computation in order to find these states. However, we know that all good states that have been found already, will remain good states for the variant model. The same goes for supervisor states. Therefore, `computeFixpoint` is instantiated with the good states added as marked states, and the supervisor states added as initial states. In the running example the removal of transition $(p_0r_1, b, p_2\bot)$ would fall under this case. In that circumstance, the supervisor states and good states would remain the same for the variant model as the base model.

Finally, for all other cases the supervisor states and good states remain the same between variant and base model. For example, we remove transition $(x_{or}, \sigma, x_{tar})$ from base model A, for which $x_{or} \in X \setminus G$ and $x_{tar} \in Y$. We know that in A, the state x_{or} was coreachable, as the removed transition existed to a state in Y. As (coreachable state) x_{or} does not exist in the set of good states G, it must be non-controllable. We can reason that the removal of this transition is not going to make it controllable. Thus, Y and G of the base automaton remain the same for the variant automaton. This is also observed in Algorithm 10.

4.7 Other atomic adaptations

Some atomic adaptations were not discussed in the algorithms above. These atomic adaptations are: adding a state, removing a state, adding an event, and removing an event. When these model deltas occur as an atomic adaptation, they do not influence the supervisor states or good states. For example, an added state only influences the synthesis result if there are added transitions towards or from it. Or an event can only be removed, if there are no transitions that are labeled by that event. Otherwise the model delta is not an atomic adaptation, or it is not a valid model delta. Proofs for Lemma 5 are provided in Appendix A.

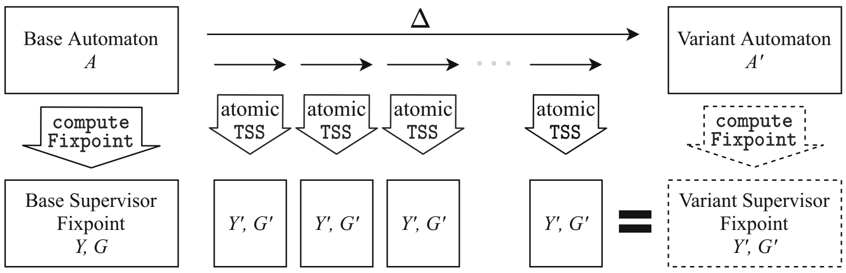

Fig. 9 ITSS for non-atomic Δ

Lemma 5 *For an atomic adaptation that is an: added state, removed state, added event, or removed event, the supervisor states Y' of the variant model are equal to the supervisor states Y of the base model, and the good states G' of the variant model are equal to the good states G of the base model.*

5 Transformational Supervisor Synthesis for any model delta

In this section we will not restrict the model delta to atomic cases anymore; any valid model delta of any size is allowed. A method that iterates over all atomic adaptations is discussed in Section 5.1. A method that groups these atomic adaptations together based on their required computation and processes them at the same time is shown in Section 5.2.

5.1 Iterative Transformational Supervisor Synthesis

In Fig. 9 a modified version of Fig. 8 is shown, which provides a visualization of the idea on which Iterative TSS (ITSS) is based. A non-atomic model delta describes the difference between the base and the variant model. This model delta is split into atomic adaptations, on which the atomic TSS algorithms can be applied, and the variant supervisor fixpoint is computed by iterating over these atomic adaptations.

Algorithm 11 provides an ITSS algorithm, that iterates over the atomic adaptations in the model delta one-by-one. Each time an atomic adaptation is applied using the results of Section 4, and Y' and G' are computed accordingly. After an adaptation has been applied, the tuples X'_0, X'_m, $\longrightarrow'$ are updated, so that if we were to construct an intermediate automaton $A' = (X', \Sigma', \longrightarrow', X'_0, X'_m)$, this automaton is up-to-date for the adaptations applied until that point. This intermediate automaton is then used in the input for the next atomic TSS algorithm. Before iterating over the atomic TSS algorithms, the added states and events are added to the base supervisor and automaton. After the iterations, the set of removed events is removed from the supervisor, as at this point no transitions with this event are left in the supervisor, following from the restrictions specified in Section 3. Because A' will not have transitions to-or-from the removed states, and the removed states are not initial or marked, the set of removed states will not be a part of Y' (or G') at this point of the algorithm. So they do not need to be removed from the supervisor. Algorithm 11 also outputs the variant automaton A' of which it produces the synthesis result.

Algorithm 11 Iterative transformational supervisor synthesis (ITSS).

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor states $Y$, good states $G$,
    model delta $\Delta = (X^+, X^-, \Sigma^+, \Sigma^-, \longrightarrow^+, \longrightarrow^-, X_0^+, X_0^-, X_m^+, X_m^-)$
Output: Variant supervisor $S' = (Y', \Sigma', \longrightarrow'_S, Y'_0, Y'_m)$, good states $G'$, variant automaton $A' = (X', \Sigma', \longrightarrow', X'_0, X'_m)$
 1: $X' = X \cup X^+,\ \Sigma' = \Sigma \cup \Sigma^+,\ \longrightarrow' = \longrightarrow,\ X'_0 = X_0,\ X'_m = X_m$
 2: $Y' = Y,\ G' = G$
 3: for all $x^\delta \in X_0^+$ do
 4:     $(Y', G') =$ TSSAIP$((X', \Sigma', \longrightarrow', X'_0, X'_m), Y', G', x^\delta)$
 5:     $X'_0 = X'_0 \cup \{x^\delta\}$
 6: end for
 7: for all $x^\delta \in X_0^-$ do
 8:     $(Y', G') =$ TSSRIP$((X', \Sigma', \longrightarrow', X'_0, X'_m), Y', G', x^\delta)$
 9:     $X'_0 = X'_0 \setminus \{x^\delta\}$
10: end for
11: for all $x^\delta \in X_m^+$ do
12:     $(Y', G') =$ TSSAMP$((X', \Sigma', \longrightarrow', X'_0, X'_m), Y', G', x^\delta)$
13:     $X'_m = X'_m \cup \{x^\delta\}$
14: end for
15: for all $x^\delta \in X_m^-$ do
16:     $(Y', G') =$ TSSRMP$((X', \Sigma', \longrightarrow', X'_0, X'_m), Y', G', x^\delta)$
17:     $X'_m = X'_m \setminus \{x^\delta\}$
18: end for
19: for all $(x_{or}, \sigma, x_{tar}) \in \longrightarrow^+$ do
20:     $(Y', G') =$ TSSAT$((X', \Sigma', \longrightarrow', X'_0, X'_m), Y', G', (x_{or}, \sigma, x_{tar}))$
21:     $\longrightarrow' = \longrightarrow' \cup \{(x_{or}, \sigma, x_{tar})\}$
22: end for
23: for all $(x_{or}, \sigma, x_{tar}) \in \longrightarrow^-$ do
24:     $(Y', G') =$ TSSRT$((X', \Sigma', \longrightarrow', X'_0, X'_m), Y', G', (x_{or}, \sigma, x_{tar}))$
25:     $\longrightarrow' = \longrightarrow' \setminus \{(x_{or}, \sigma, x_{tar})\}$
26: end for
27: $S' = (Y', \Sigma' \setminus \Sigma^-, \longrightarrow \cap (Y' \times \Sigma \times Y'), X'_0 \cap Y', X'_m \cap Y')$
28: $A' = (X' \setminus X^-, \Sigma' \setminus \Sigma^-, \longrightarrow', X'_0, X'_m)$
29: return $(S', G', A')$
```

Theorem 3 holds for Algorithm 11. It is similar to Theorem 2 for the atomic model adaptations, only modified to apply for a supervisor automaton S, rather than supervisor states Y. It also considers that the algorithm provides the correct variant automaton as output. The proof for Theorem 3 on Algorithm 11 can be found in Appendix B.

Theorem 3 *Given base automaton A, model delta Δ, synthesis result $(Y, G) =$ computeFixpoint(A), and $(\hat{S}, \hat{G}, \hat{A}) =$ ITSS(A, Y, G, Δ); then $\hat{S} = S'$, $\hat{G} = G'$, and $\hat{A} = A'$, for $(S', G') =$ SS(A').*

5.1.1 Order of applying atomic adaptations

There are some restrictions to the order in which the atomic adaptations can be applied during iterative TSS. Essentially, when an atomic adaptation is applied, this atomic adaptation needs to be a valid model delta. Let us consider the following model delta for the running example:

- $X^+ = \{p_5r_0\}$,
- $\Sigma^+ = \{e\}$,
- $\longrightarrow^+ = \{(p_4r_0, e, p_5r_0)\}$,
- $X^- = \emptyset$, $\Sigma^- = \emptyset$, $\longrightarrow^- = \emptyset$, $X_0^+ = \emptyset$, $X_0^- = \emptyset$, $X_m^+ = \emptyset$, $X_m^- = \emptyset$.

This model delta represents an added transition with an added event (e) to an added state (p_5r_0) from an existing state (p_4r_0). The model delta can be split into three atomic parts. We observe that the added state and added event need to be added to the automaton first, before the added transition can be added. Otherwise, the transition goes to an undefined state, or uses an undefined event, which means the model delta is not valid.

Therefore, the added states and added events are added first. Now, any state with added or removed initial property and any state with added or removed marked property will exist in this intermediate automaton. Also all added and removed transitions will go between defined states by defined events. Thus these atomic adaptations can be applied in any order. Once these adaptations have been applied, there will be no more transitions towards removed states, or transitions over removed events. Then finally the removed states and removed events can be removed, resulting in the final variant automaton.

The functioning of the atomic TSS algorithms is based on the assumption that the atomic adaptation is a valid model delta by the definitions in Section 3. To support the correct functioning of Algorithm 11, proof that each atomic adaptation that is applied is a valid model delta is given in Appendix B.

The authors note that even though the atomic adaptations of added/removed initial property, added/removed marked property, and added/removed transition can be applied in any order to come up with the same supervisor, the order in which they are applied may impact the computational efficiency. We do not optimize this order here, as the optimal order is likely highly dependent on the particular model and model delta. Therefore the adaptations are applied in the order shown in Algorithm 11, where the atomic TSS algorithms appear in the order in which they are introduced in Section 4.

5.2 Grouped Transformational Supervisor Synthesis

We observe that Algorithm 11 might not be very efficient when many atomic adaptations need to be considered. The main issue is that the `SS` and `FRS` algorithms are repeatedly called for input sets that are considerably similar to each other. This may notably occur when the plant description is given by a set of automata $\{P_1, P_2, ..., P_n\}$. For our synthesis purpose, we take the synchronous product $A = P_1||P_2||...||P_n$, as mentioned in Section 2. If one of the automata P_i is adapted in an atomic manner, this might result in the model delta Δ to contain many atomic adaptations, due to synchronicity. A lot of these atomic adaptations in the synchronous system will be of the same type, e.g., an added transition in P_i can induce many added transitions in A. Therefore, we want to consider some adaptations at the same time as a group, rather than applying them one-by-one.

We partition the model delta into two disjoint subsets; $\Delta^\times \uplus \Delta^\circ = \Delta$, where $\uplus$ denotes the disjoint union of the sets that are in the same field of both tuples. $\Delta^\times$ contains all atomic

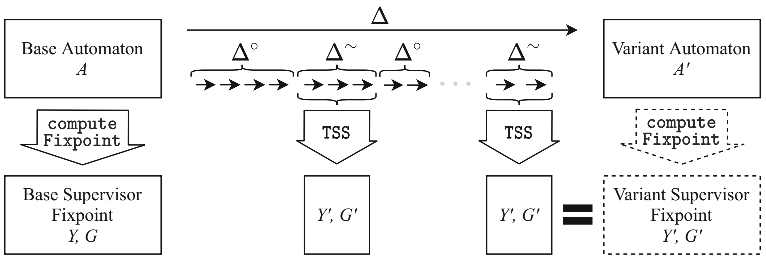

Fig. 10 GTSS for non-atomic $\varDelta$

adaptations that, when applying the respective atomic TSS algorithm, require the SS or FRS algorithm. $\varDelta^\circ$ contains all other possible atomic adaptations outside $\varDelta^\times$, these require no reachability searches. As a result of the construction of the atomic TSS algorithms, all atomic adaptations in $\varDelta^\circ$ fit under the default cases of these algorithms, and all adaptations in $\varDelta^\times$ match one of the (non-default) case statements in these algorithms. Formally, we can compute $\varDelta^\times = (X^{+,\times}, X^{-,\times}, \varSigma^{+,\times}, \varSigma^{-,\times}, \longrightarrow^{+,\times}, \longrightarrow^{-,\times}, X_0^{+,\times}, X_0^{-,\times}, X_m^{+,\times}, X_m^{-,\times})$ for a model delta $\varDelta = (X^+, X^-, \varSigma^+, \varSigma^-, \longrightarrow^+, \longrightarrow^-, X_0^+, X_0^-, X_m^+, X_m^-)$ as follows:

- $X^{+,\times} = \emptyset$, $X^{-,\times} = \emptyset$, $\varSigma^{+,\times} = \emptyset$, $\varSigma^{-,\times} = \emptyset$
- $\longrightarrow^{+,\times} = \{(x_{or}, \sigma, x_{tar}) | (x_{or} \in Y \wedge x_{tar} \in G \setminus Y) \vee (x_{or} \in G \wedge x_{tar} \in X \setminus G \wedge \sigma \in \varSigma_u) \vee (x_{or} \in X \setminus G \wedge x_{or} \neq x_{tar})\}$
- $\longrightarrow^{-,\times} = \{(x_{or}, \sigma, x_{tar}) | (x_{or} \in Y \wedge x_{tar} \in Y \wedge x_{or} \neq x_{tar}) \vee (x_{or} \in G \setminus Y \wedge x_{tar} \in G \wedge x_{or} \neq x_{tar}) \vee (x_{or} \in X \setminus G \wedge x_{tar} \in X \setminus G \wedge \sigma \in \varSigma_u \wedge x_{or} \neq x_{tar})\}$
- $X_0^{+,\times} = X_0^+ \cap (G \setminus Y)$
- $X_0^{-,\times} = X_0^- \cap Y$
- $X_m^{+,\times} = X_m^+ \cap (X \setminus G)$
- $X_m^{-,\times} = X_m^- \cap G$

$\varDelta^\circ$ is then constructed by all atomic adaptations in $\varDelta$ outside $\varDelta^\times$.

When performing Grouped TSS (GTSS), we first want to apply all atomic adaptations in $\varDelta^\circ$, as we can observe in the atomic TSS algorithms that no reachability searches need to be performed, and the supervisor states and good states remain unchanged. After all atomic adaptations in $\varDelta^\circ$ have been applied, we find the first atomic adaptation in $\varDelta^\times$ for which, if we were to apply ITSS with model delta $\varDelta^\times$, one of the case conditions in Algorithms 5-10 holds. Instead of performing the corresponding case statements on only this atomic adaptation, within the case operation we first construct a set $\varDelta^\sim$, that contains all atomic adaptations in $\varDelta^\times$ for which that same case condition in the same atomic TSS algorithm holds. So for example, if we have $\varDelta^\times$ with nonempty set $X_0^{+,\times} \cap (G \setminus Y)$, then $X_0^{+,\times} \cap (G \setminus Y)$ is a set $\varDelta^\sim$. In addition to just atomic adaptations, the rationale applied in Algorithms 5-10 still holds for sets $\varDelta^\sim$. So, we take set $\varDelta^\sim$, and apply the respective case operation on this set, rather than doing this for each atomic adaptation one-by-one. This might influence the supervisor states and good states. Because the partitioning in $\varDelta^\times$ and $\varDelta^\circ$ of $\varDelta$ depends on those state sets, $\varDelta^\circ$ might be nonempty if we recompute it for the atomic adaptations in $\varDelta$ that have not been applied yet. Thus, we once more apply adaptations in $\varDelta^\circ$, and reiterate until all adaptations have been applied.

Figure 10 visualizes the grouped TSS method. First, all atomic adaptations are applied that require no reachability search. Then, a set of atomic adaptations $\varDelta^\sim$ is applied at once.

This repetition continues until the entire model delta is applied. At this point, the fixpoint for the variant supervisor has been found.

Algorithm 12 Grouped Transformational Supervisor Synthesis (GTSS).

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_m, X_0)$, base supervisor $S = (Y, \Sigma, \longrightarrow_S, Y_0, Y_m)$,
    good states $G$, model delta $\Delta = (X^+, X^-, \Sigma^+, \Sigma^-, \longrightarrow^+, \longrightarrow^-, X_0^+, X_0^-, X_m^+, X_m^-)$
Output: Variant supervisor $S' = (Y', \Sigma', \longrightarrow'_S, Y'_0, Y'_m)$, good states $G'$, variant automaton
    $A' = (X', \Sigma', \longrightarrow', X'_0, X'_m)$
1:  $X' = X \cup X^+,\ \Sigma' = \Sigma \cup \Sigma^+,\ \longrightarrow' = \longrightarrow,\ X'_0 = X_0,\ X'_m = X_m$
2:  $\Delta = (\emptyset, X^-, \emptyset, \Sigma^-, \longrightarrow^+, \longrightarrow^-, X_0^+, X_0^-, X_m^+, X_m^-)$
3:  $Y' = Y,\ G' = G$
4:  repeat
5:      Compute $(\Delta^\times, \Delta^\circ = (X^{+,\circ}, X^{-,\circ}, \Sigma^{+,\circ}, \Sigma^{-,\circ}, \longrightarrow^{+,\circ}, \longrightarrow^{-,\circ}, X_0^{+,\circ}, X_0^{-,\circ}, X_m^{+,\circ}, X_m^{-,\circ}))$
        for $\Delta$, $Y'$, and $G'$
6:      $\longrightarrow' = (\longrightarrow \cup \longrightarrow^{+,\circ}) \setminus \longrightarrow^{-,\circ},\ X'_0 = (X_0 \cup X_0^{+,\circ}) \setminus X_0^{-,\circ},\ X'_m = (X_m \cup X_m^{+,\circ}) \setminus X_m^{-,\circ}$
7:      $\Delta = (\emptyset, X^-, \emptyset, \Sigma^-, \longrightarrow^+ \setminus \longrightarrow^{+,\circ}, \longrightarrow^- \setminus \longrightarrow^{-,\circ}, X_0^+ \setminus X_0^{+,\circ}, X_0^- \setminus X_0^{-,\circ}, X_m^+ \setminus$
        $X_m^{+,\circ}, X_m^- \setminus X_m^{-,\circ})$
8:      Compute $(Y', G', \longrightarrow', X'_0, X'_m)$ by applying one set of adaptations $\Delta^\sim$ in $\Delta^\times$ at once
9:      Remove $\Delta^\sim$ from $\Delta$
10: until $\Delta = (\emptyset, X^-, \emptyset, \Sigma^-, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
11: $S' = (Y', \Sigma' \setminus \Sigma^-, \longrightarrow \cap (Y' \times \Sigma \times Y'), X'_0 \cap Y', X'_m \cap Y')$
12: $A' = (X' \setminus X^-, \Sigma' \setminus \Sigma^-, \longrightarrow', X'_0, X'_m)$
13: return $(S', G', A')$
```

Algorithm 12 functions as described above. The added events and added states are added in line 1. After these have been applied, they are removed from the model delta. As stated in Section 4.7, these adaptations do not influence the supervisor states and good states, seen in line 3. The model delta is partitioned in $\Delta^\times$ and Δ° as specified earlier in this section. All adaptations in Δ° are applied in line 6, and subsequently removed from the model delta in line 7. Note that Y' and G' remain unchanged. In line 8 a set of atomic adaptations $\Delta^\sim$ is applied; Y' and G' are calculated accordingly, and $\longrightarrow'$, X'_0, and X'_m are consequently updated. The calculation of Y' and G' is done by using slightly modified versions of Algorithms 5-10 that accept sets of atomic adaptations. To avoid redundancy, we only provide the modified version of Algorithm 5 in the example below. The other atomic TSS algorithms are converted in the same manner. The set of atomic adaptations that has been applied, is removed from Δ in line 9. Now $\Delta^\times$ and Δ° are calculated once again, because the partitioning of the model delta is dependent on Y' and G' which are now modified. The steps are repeated until all atomic adaptations to $\longrightarrow'$, X'_0, and X'_m have been applied. Finally, the supervisor automaton for the variant model and the variant automaton are constructed in lines 11 and 12 respectively.

Theorem 4 is proven for Algorithm 12 in Appendix C. In Appendix C also the proof is given that Algorithm 12 respects the order of applying adaptations discussed in Section 5.1.

Theorem 4 *Given base automaton A, model delta Δ, synthesis result (Y, G) =* `computeFixpoint`(A)*, and $(\hat{S}, \hat{G}, \hat{A})$ =*`GTSS`(A, Y, G, Δ)*; then $\hat{S} = S'$, $\hat{G} = G'$, and $\hat{A} = A'$, for (S', G') =*`SS`(A').

5.2.1 Example

Let us consider the case that Δ contains three atomic adaptations, all are states with added initial property; $X_0^+ = \{x^{\delta,1}, x^{\delta,2}, x^{\delta,3}\}$, with $x^{\delta,1} \in G \setminus Y$, $x^{\delta,2} \in G \setminus Y$, $x^{\delta,3} \in Y$. In our running example this could be states p_3r_0, p_3r_1, and p_2r_0 respectively. $x^{\delta,1}$ and $x^{\delta,2}$ require FRS, seen in line 3 of Algorithm 5, so they are in $\Delta^\times$. $x^{\delta,3}$ triggers the default case in Algorithm 5, and thus it is in Δ°. Therefore, $x^{\delta,3}$ is applied first, resulting in:

- $X_0' = X_0' \cup \{x^{\delta,3}\}$
- $X_0^+ = X_0^+ \setminus \{x^{\delta,3}\}$

Note that Y' and G' are not influenced as $x^{\delta,3}$ is in Δ°.

Now all adaptations in Δ° have been applied. $X_0^+ = \{x^{\delta,1}, x^{\delta,2}\}$ remains in the model delta. As $x^{\delta,2}$ and $x^{\delta,3}$ are both of the same case (line 2 of Algorithm 5), they are in a set $\Delta^\sim = \{x^{\delta,1}, x^{\delta,2}\}$. Consequently, these atomic adaptations will simultaneously be applied. Y' and G' are calculated in a modified version of Algorithm 5, given in Algorithm 13. X_0' and X_0^+ are updated as follows:

- $X_0' = X_0' \cup \Delta^\sim$
- $X_0^+ = X_0^+ \setminus \Delta^\sim$

Algorithm 13 Grouped Transformational Supervisor Synthesis for Added Initial Property.

```
Input: Base automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, supervisor states $Y$, good states $G$, added
    initial states $X_0^+$
Output: Variant supervisor states $Y'$, variant good states $G'$
1: switch $X_0^+$
2:   case $X_0^+ \cap (G \setminus Y) \neq \emptyset$ do
3:     $\Delta^\sim = X_0^+ \cap (G \setminus Y)$
4:     $Y' = \mathtt{FRS}(G, \Sigma, \longrightarrow, Y \cup \Delta^\sim)$
5:     $G' = G$
6:   default do
7:     $Y' = Y$, $G' = G$
8: end switch
9: return $(Y', G')$
```

After applying these adaptations, all atomic model adaptations in the model have been applied and the model delta is empty, and Y' and G' are the fixpoint result for the variant automaton A'. For the running example, the good states would remain the same, $G' = G$, and Y' would be $\{p_0r_0, p_2r_0, p_3r_0, p_3r_1, p_2r_1\}$.

6 Experiments

As stated before, we can deal with any valid base automaton and model delta, so the TSS algorithm will always find a supervisor for the variant model. However, there are no guarantees that by applying TSS, we will find the supervisor more efficiently relative to simply performing a completely new synthesis. Therefore, we perform some experiments to investigate the potential reduction in computational effort by applying TSS.

For the experiments, a proof-of-concept implementation of the above synthesis algorithms and models of the case studies we describe below have been made in Matlab.[1]

Before discussing case studies, we provide some practical notes in Section 6.1. In Section 6.2 we consider the Transfer Line model as an academic case study, and in Section 6.3 we consider a Lithography Machine Wafer Logistics controller as an industrial case study.

6.1 Practical notes

A proof-of-concept Matlab implementation of the above algorithms has been made. One modification from Thuijsman and Reniers (2020) to this paper, is that now a linear complexity reachability search algorithm is used instead of quadratic. This is to enable a more realistic effort comparison between SS and TSS since linear search algorithms are more widely used than quadratic. In Thuijsman and Reniers (2020), a counter in the reachability search algorithms was introduced to express and compare the time effort of performing synthesis. After implementing the new reachability search algorithms, it was found that the previously used metric was not representative anymore of the time effort of performing synthesis. Experiments were performed with counters at several locations in the code, however none of these counters gave a proper representation of the time effort. Therefore, wall-clock time is used here to represent the time effort of performing synthesis. In order to enable fair comparisons between running times, obvious inefficient parts of the algorithm were improved. Still, the authors note that by using wall clock time instead of a counter makes the results more dependent on the implementation. The experiments were performed on an HP ZBook Studio G4 laptop, using an Intel i7 processor clocked at 2.8 GHz. Regarding memory, Matlab used around 1 GB of memory, regardless of the model size. Filesizes to store the automata and model deltas ranged from a few KB to a few MB.

Each synthesis algorithm requires a monolithic automaton as input, and the transformational synthesis algorithms require a model delta to the variant automaton. In practice, these inputs may not be readily available. E.g., the monolithic automaton A needs to be constructed from a component-wise specification, as discussed in Section 2. This is also the case for the conducted experiments. The inputs are computed in preparation of the experiments. We assume that for the transformational method, the input automaton A is maintained for the next iteration. For each transformational synthesis, A' is constructed beforehand in order to compute the model delta. Note that A' also needs to be constructed for the baseline case of performing a completely new synthesis. Computing the model delta is done by simple matrix subtractions and requires negligible computational effort. The preparatory computations are not included in the computational effort measurements, because this matches the experiments to the monolithic level discussed in the theoretical part and because computing the synchronous composition is required for all methods.

The construction of A can be done by computing the complete synchronous composition as shown in Eq. 1. Practically however, often only the reachable part of A is constructed. Note that the computed supervisor is the same, as it only contains reachable states. The good state set G is influenced by only using the reachable part of A. Even when all states in X are reachable, it is still beneficial to store set G next to Y. When states are removed from G during synthesis, not all states in G may be reachable anymore through states in G. The model delta may be impacted by considering only the reachable parts of the automata or not. Also

[1]The algorithms and models can be found here: https://github.com/sbthuijsman/JDEDS_TSS

the computational effort of performing transformational or non-transformational supervisor synthesis may be impacted. Note that the transformational synthesis method works for both cases, as it allows for any pair of automata A and A', independent of how they are constructed. We consider both options in the case studies below.

6.2 Transfer Line

We first consider the Transfer Line model from Wonham and Cai (2019) as an academic case study. In this model, products are being processed by two machines. Machine M1 takes products from the environment, and processes them. After processing, M1 places the product in buffer B1, which can hold up to three products. Machine M2 takes products from B1, processes them, and places them in buffer B2, which can hold only one product. Test unit TU takes products from B2, and tests them. If the product is accepted, it is released from the system. If the product is rejected, it goes back to B1. M1, M2, and TU start by a controllable event, and terminate by an uncontrollable event. Same as in Wonham and Cai (2019), M1, M2, and TU are modeled by plant automata and B1 and B2 by requirement automata. These automata are shown in Fig. 11. The synchronous product over all automata is taken. For plantification, a single sink state is added to this synchronous product, to which all uncontrollable transitions are created whenever one of the requirement automata blocks an uncontrollable event of the plant. The resulting base automaton TL has 65 states and 200 transitions. Supervisor synthesis (Algorithm 1) for this base automaton requires 1.4 milliseconds, which is the mean runtime of 100 executions of `SS`.

The following five variant automata, TL'_1 to TL'_5, have been generated by making adaptations to TL.

- TL'_1: Reduced capacity of B1 to two products; state x_3 of automaton $B1$ removed, and transitions $(x_2, 2, x_3)$, $(x_2, 8, x_3)$, $(x_3, 3, x_2)$ removed.
- TL'_2: Increased capacity of B2 to two products; added a state x_2 and transitions $(x_1, 4, x_2)$, $(x_2, 5, x_1)$ to B2.
- TL'_3: B1 initially holds one product instead of zero; removed initial property of state x_0, and added initial property to state x_1 in automaton B1.
- TL'_4: TU may send the product to B2 upon completion; added uncontrollable event 9, added transition $(x_1, 9, x_2)$ to TU, and added transition $(x_0, 9, x_1)$ to B2.
- TL'_5: Capacity of B1 and B2 is two products each; removed state x_3 and transitions $(x_2, 2, x_3)$, $(x_2, 8, x_3)$, $(x_3, 3, x_2)$ from B1, state x_2 and transitions $(x_1, 4, x_2)$, $(x_2, 5, x_1)$ are added to B2.

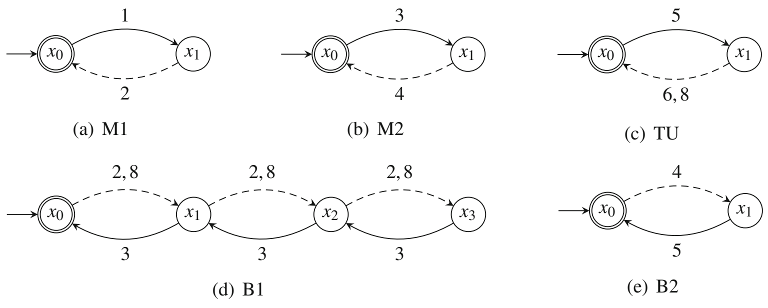

Fig. 11 Transfer Line automata; (a), (b), and (c) are plant automata, (d) and (e) are requirement automata

As the TSS methods are on a monolithic state space, these adaptations for the individual automata are converted to adaptations on the synchronous state space for the experiments, as discussed in Section 6.1. For the base model and each variant model, all states constructed during the synchronous composition (1) are reachable. Therefore, for these results it does not matter if A is completely constructed as in Eq. 1, or only the reachable part. The experiment cases we presented are the same as presented in Thuijsman and Reniers (2020), however the experiments are executed differently as pointed out in Section 6.1.

For each of the variant automata, the number of states and transitions, as well as the sizes of the nonempty sets in the model delta, are given in Table 1. One might expect that for constructing TL'_1 only states and transitions would be removed. However, it can be observed that some transitions have been added. These are new transitions towards the sink state, as the buffer now may overflow from different states. For each variant model, a supervisor has been synthesized three times. First, doing a completely new synthesis by applying `SS` given in Algorithm 1, the second and third by using the synthesis result of the base automaton and applying `ITSS` (Algorithm 11), and `GTSS` (Algorithm 12) respectively. For each model, the three synthesized supervisors have the exact same automaton specification. For each of the syntheses, the runtime is shown in milliseconds in Table 1. This is the mean over 100 runs for each synthesis. The Coefficient of Variation (CV), calculated by dividing the sample standard deviation over the mean, is shown in between brackets underneath each runtime value as a percentage. Because this example is small, the runtimes are low. Therefore slight absolute variations in runtime result in a high relative variation. The rightmost column shows the percentage change in runtime of using `GTSS` compared to `SS`. A positive value indicates an increase in computational effort, and a negative value indicates a reduction. Note that the efficiency of the TSS algorithms will be influenced by the order in which

Table 1 Experimental results of performing `SS`, `ITSS`, and `GTSS` on five variant models of the Transfer Line model

Evolution	Variant model size	Model delta size	Var. model runtime [ms] (Sample CV %)			% change GTSS from SS
			SS	ITSS	GTSS	
TL to TL'_1	$\lvert X'\rvert = 49$	$\lvert X^-\rvert = 16$	1.1	43.2	2.4	115
	$\lvert \longrightarrow' \rvert = 148$	$\lvert \longrightarrow^+ \rvert = 16$	*(18)*	*(6)*	*(9)*	
		$\lvert \longrightarrow^- \rvert = 68$				
TL to TL'_2	$\lvert X'\rvert = 97$	$\lvert X^+\rvert = 32$	1.6	140.2	2.9	83
	$\lvert \longrightarrow' \rvert = 308$	$\lvert \longrightarrow^+ \rvert = 124$	*(18)*	*(4)*	*(11)*	
		$\lvert \longrightarrow^- \rvert = 16$				
TL to TL'_3	$\lvert X'\rvert = 65$	$\lvert X_0^+\rvert = 1$	1.4	0.2	0.2	−84
	$\lvert \longrightarrow' \rvert = 200$	$\lvert X_0^-\rvert = 1$	*(19)*	*(27)*	*(20)*	
TL to TL'_4	$\lvert X'\rvert = 65$	$\lvert \Sigma^+\rvert = 1$	1.3	19.3	1.6	27
	$\lvert \longrightarrow' \rvert = 232$	$\lvert \longrightarrow^+ \rvert = 32$	*(19)*	*(7)*	*(12)*	
TL to TL'_5	$\lvert X'\rvert = 73$	$\lvert X^+\rvert = 24$	1.3	154.2	3.8	180
	$\lvert \longrightarrow' \rvert = 228$	$\lvert X^-\rvert = 16$	*(17)*	*(5)*	*(12)*	
		$\lvert \longrightarrow^+ \rvert = 108$				
		$\lvert \longrightarrow^- \rvert = 80$				

the adaptations are applied, this has not been optimized for the experiments as this would entail a complete new study.

We observe that in some cases applying `ITSS` requires considerably more runtime than applying `SS`. This was already addressed in Section 5, and led to the introduction of the `GTSS` algorithm. We observe that for this use case, for most variant models `GTSS` requires considerably less runtime than `ITSS`, but still requires more runtime than `SS` in most cases. The aim of `GTSS` is to use smaller synthesis calls and reachability searches by using the model delta and previous synthesis result. To do so, `GTSS` prepares these smaller synthesis calls and reachability searches by operations such as iterating over the model delta and evaluating the switch case statements. Because the model is very small, relatively speaking `GTSS` requires a lot of time to do the preparation steps, and little time performing the smaller synthesis calls and reachability searches. Therefore `GTSS` performs poorly compared to `SS`, that can just perform synthesis directly to the small model. It is expected that for a larger system, `GTSS` spends relatively less time on the preparation steps, and more on the smaller synthesis calls and reachability searches. Therefore `GTSS` could be more competitive to `SS` for these larger systems, since `SS` will require more runtime as well for the larger systems. We will study a larger system next, in Section 6.3. From a user perspective the runtime differences for the Transfer Line case would be insignificant.

6.3 Lithography Machine Wafer Logistics

Next we present an industrial case study. This case study is performed using models from ASML. ASML is the world-leading manufacturer of lithography machines, which are used in the semiconductor industry to produce integrated circuits. These circuits are printed on silicon wafers. The movement of these wafers through the machine is called the Wafer Logistics, which is studied in van der Sanden et al. (2015) and (van der Schriek 2018). The controller of the Wafer Logistics is constructed using Analytical Software Design (ASD) ((Broadfoot and Hopcroft 2003)). van der Schriek (2018) presents a study on how these ASD models of the components of the Wafer Logistics controller evolve over time. In this study equivalent automata models are constructed in CIF ((van Beek et al. 2014), CIF is part of the Eclipse Supervisory Control Engineering Toolkit®), that are suitable for supervisory controller synthesis. These automata models are constructed for the variation points that the ASD models evolved to over a number of years. We use these automata models here, to investigate the efficiency of TSS in this industrial setting.

Component B of (van der Schriek 2018) is selected to perform the experiments on, based on its large but manageable state space size, the number of variation points, and the variety within the model deltas. The first 11 variation points of this model are taken, to investigate 10 adaptations. Opposed to the Transfer Line experiment, where each variant model was an adaptation of the same base model, we now consider incremental adaptations. So we start with the evolution from $B1$ to $B2$, then from $B2$ to $B3$, from $B3$ to $B4$, and so on. To provide an indication of the model size, $B1$ contains 5 plant automata, that respectively have:

1. 2 states and 2 transitions,
2. 2 states and 1 transition,
3. 2 states and 3 transitions,
4. 15 states and 72 transitions,
5. 16 states and 128 transitions.

Model $B1$ also contains 4 requirement automata, that respectively have:

1. 2 states and 2 transitions,
2. 2 states and 2 transitions,
3. 3 states and 5 transitions,
4. 3 states and 5 transitions.

Additionally 3 state transition exclusion invariant requirements (Markovski et al. 2010) are specified. After computing the synchronous composition (1), and adding a sink state for plantification, $B1$ has 69 121 states and 1 038 228 transitions. Performing SS (Algorithm 1) requires 46.2 seconds, which is the mean over 20 executions of SS.

Unlike the Transfer Line model, for the models of Component B not all states are reachable when automaton A is constructed by Eq. 1. We perform two studies, in Section 6.3.1 we consider the case that the complete automaton is used as an input to the synthesis algorithms and in Section 6.3.2 we consider that only the reachable part of the automaton is used as input.

6.3.1 Complete state space

In this section we perform synthesis on the complete state space that is constructed by performing synchronous composition on the Component B Wafer Logistics model. The experimental results are presented in Table 2. ITSS has not been performed, as its inefficiency compared to GTSS has been discussed in Section 5.2, and shown in the Transfer Line experiments. Note that the runtime is now displayed in seconds instead of milliseconds. Each runtime value is the mean over 20 syntheses. Because of the higher absolute runtimes compared to the Transfer Line use case, the CV is lower in these experiments. Compared to the Transfer Line experiment, we observe that the number of states for Component B increased by three orders of magnitude, for the number of transitions it is four orders of magnitude. Due to this increase in model size, the required runtime has also increased by four orders of magnitude. We observe that for all evolutions in this case study, applying GTSS is more efficient than applying SS to compute the supervisor for the variant model. The efficiency gain ranges from 9 to 29%, and is 20% on average.

6.3.2 Reachable state space

In this section we perform synthesis using only the reachable part of synchronous composition of the Component B Wafer Logistics model. The complete state space of model B1 consists of 69 121 states and 1 038 228 transitions. The reachable part of this automaton has 59 185 states and 888 780 transitions. Note that also the model delta sizes are influenced by only considering the reachable part of the base and variant model automata. The results are shown in Table 3. Each runtime value is the mean over 20 syntheses. Once more, for all evolutions in this experiment applying GTSS is more efficient than applying SS to compute the supervisor for the variant model. Compared to the experiment on the complete state space, discussed in Section 6.3.1, the absolute computational effort to perform synthesis is reduced for both SS and GTSS. Furthermore, the relative efficiency of using GTSS compared to SS was also improved. The efficiency gain ranges from 17 to 36%, and is 27% on average. The authors note that even though in this case study the transformational synthesis method works better when only the reachable states are considered, this may not be so for other models.

Table 2 Experimental results of performing SS and GTSS for evolution of the complete Component B Wafer Logistics model

Evolution	Variant model size	Model delta size	Var. model runtime [s] (Sample CV %) SS	GTSS	% change GTSS from SS
$B1$ to $B2$	$\|X'\| = 6\,921$	$\|\Sigma^+\| = 1$	49.4	35.0	−29
	$\|\longrightarrow'\| = 1\,042\,548$	$\|\longrightarrow^+\| = 4\,320$	*(1.2)*	*(1.1)*	
$B2$ to $B3$	$\|X'\| = 77\,761$	$\|X^+\| = 8\,640$	62.9	44.8	−29
	$\|\longrightarrow'\| = 1\,154\,772$	$\|\Sigma^+\| = 8$	*(0.8)*	*(0.7)*	
		$\|\longrightarrow^+\| = 112\,224$			
$B3$ to $B4$	$\|X'\| = 73\,441$	$\|X^-\| = 4\,320$	56.4	48.0	−15
	$\|\longrightarrow'\| = 1\,102\,980$	$\|\Sigma^-\| = 1$	*(0.9)*	*(0.9)*	
		$\|\longrightarrow^-\| = 51\,792$			
$B4$ to $B5$	$\|X'\| = 73\,441$	$\|\Sigma^+\| = 4$	57.3	40.5	−29
	$\|\longrightarrow'\| = 1\,121\,412$	$\|\longrightarrow^+\| = 18\,432$	*(0.9)*	*(0.8)*	
$B5$ to $B6$	$\|X'\| = 73441$	$\|\Sigma^+\| = 1$	57.3	49.2	−14
	$\|\longrightarrow'\| = 1\,121\,412$	$\|\Sigma^-\| = 1$	*(0.8)*	*(0.7)*	
		$\|\longrightarrow^+\| = 4\,320$			
		$\|\longrightarrow^-\| = 4\,320$			
$B6$ to $B7$	$\|X'\| = 73\,441$	$\|\Sigma^+\| = 3$	57.7	49.6	−14
	$\|\longrightarrow'\| = 1\,130\,052$	$\|\Sigma^-\| = 1$	*(0.8)*	*(0.9)*	
		$\|\longrightarrow^+\| = 12\,960$			
		$\|\longrightarrow^-\| = 4\,320$			
$B7$ to $B8$	$\|X'\| = 78\,337$	$\|X^+\| = 4\,896$	65.7	60.1	−9
	$\|\longrightarrow'\| = 1\,200\,980$	$\|\Sigma^+\| = 5$	*(0.9)*	*(0.7)*	
		$\|\Sigma^-\| = 1$			
		$\|\longrightarrow^+\| = 75\,824$			
		$\|\longrightarrow^-\| = 4\,896$			
$B8$ to $B9$	$\|X'\| = 73\,441$	$\|X^-\| = 4\,896$	56.2	48.6	−14
	$\|\longrightarrow'\| = 1\,200\,980$	$\|\Sigma^-\| = 10$	*(0.8)*	*(1.1)*	
		$\|\longrightarrow^-\| = 98\,000$			
$B9$ to $B10$	$\|X'\| = 78\,337$	$\|X^+\| = 4\,896$	65.7	50.1	−24
	$\|\longrightarrow'\| = 1\,200\,980$	$\|\Sigma^-\| = 10$	*(0.9)*	*(1.0)*	
		$\|\longrightarrow^+\| = 98\,000$			
$B10$ to $B11$	$\|X'\| = 83\,233$	$\|X^+\| = 4\,896$	74.2	56.0	−25
	$\|\longrightarrow'\| = 1\,267\,012$	$\|\Sigma^+\| = 4$	*(0.9)*	*(0.9)*	
		$\|\longrightarrow^+\| = 66\,032$			

Table 3 Experimental results of performing `SS` and `GTSS` for evolution of the reachable part of the Component B Wafer Logistics model

Evolution	Variant model size	Model delta size	Var. model runtime [s] (Sample CV %)		% change GTSS from SS
			SS	GTSS	
$B1$ to $B2$	$\lvert X'\rvert = 59\,185$	$\lvert \Sigma^+\rvert = 1$	2.36	15.7	−33
	$\lvert \longrightarrow' \rvert = 892\,460$	$\lvert \longrightarrow^+ \rvert = 3\,680$	*(1.4)*	*(1.6)*	
$B2$ to $B3$	$\lvert X'\rvert = 66\,545$	$\lvert X^+\rvert = 7\,360$	29.8	20.7	−30
	$\lvert \longrightarrow' \rvert = 988\,252$	$\lvert \Sigma^+\rvert = 8$	*(1.0)*	*(1.1)*	
		$\lvert \longrightarrow^+ \rvert = 95\,792$			
$B3$ to $B4$	$\lvert X'\rvert = 62\,865$	$\lvert X^-\rvert = 3\,680$	26.7	22.1	−17
	$\lvert \longrightarrow' \rvert = 944\,036$	$\lvert \Sigma^-\rvert = 1$	*(0.7)*	*(1.6)*	
		$\lvert \longrightarrow^- \rvert = 44\,216$			
$B4$ to $B5$	$\lvert X'\rvert = 62\,865$	$\lvert \Sigma^+\rvert = 4$	27.1	18.1	−33
	$\lvert \longrightarrow' \rvert = 959\,732$	$\lvert \longrightarrow^+ \rvert = 15\,696$	*(0.9)*	*(1.6)*	
$B5$ to $B6$	$\lvert X'\rvert = 62\,865$	$\lvert \Sigma^+\rvert = 1$	27.2	20.6	−24
	$\lvert \longrightarrow' \rvert = 959\,732$	$\lvert \Sigma^-\rvert = 1$	*(0.8)*	*(1.2)*	
		$\lvert \longrightarrow^+ \rvert = 3\,680$			
		$\lvert \longrightarrow^- \rvert = 3\,680$			
$B6$ to $B7$	$\lvert X'\rvert = 62\,865$	$\lvert \Sigma^+\rvert = 3$	27.3	20.6	−24
	$\lvert \longrightarrow' \rvert = 967\,092$	$\lvert \Sigma^-\rvert = 1$	*(0.7)*	*(2.0)*	
		$\lvert \longrightarrow^+ \rvert = 11\,040$			
		$\lvert \longrightarrow^- \rvert = 3\,680$			
$B7$ to $B8$	$\lvert X'\rvert = 67\,033$	$\lvert X^+\rvert = 4\,168$	31.1	23.9	−23
	$\lvert \longrightarrow' \rvert = 1\,027\,520$	$\lvert \Sigma^+\rvert = 5$	*(0.6)*	*(1.3)*	
		$\lvert \Sigma^-\rvert = 1$			
		$\lvert \longrightarrow^+ \rvert = 64\,596$			
		$\lvert \longrightarrow^- \rvert = 4\,168$			
$B8$ to $B9$	$\lvert X'\rvert = 62\,865$	$\lvert X^-\rvert = 4\,168$	26.8	22.2	−17
	$\lvert \longrightarrow' \rvert = 944\,036$	$\lvert \Sigma^-\rvert = 10$	*(1.6)*	*(2.2)*	
		$\lvert \longrightarrow^- \rvert = 83\,484$			
$B9$ to $B10$	$\lvert X'\rvert = 67\,033$	$\lvert X^+\rvert = 4\,168$	31.2	21.3	−32
	$\lvert \longrightarrow' \rvert = 1\,027\,520$	$\lvert \Sigma^+\rvert = 10$	*(1.2)*	*(1.0)*	
		$\lvert \longrightarrow^+ \rvert = 83\,484$			
$B10$ to $B11$	$\lvert X'\rvert = 71\,201$	$\lvert X^+\rvert = 4\,168$	37.1	23.7	−36
	$\lvert \longrightarrow' \rvert = 1\,083\,780$	$\lvert \Sigma^+\rvert = 4$ *(1.0)*	*(1.1)*		
		$\lvert \longrightarrow^+ \rvert = 56\,260$			

7 Conclusions

Supervisory controller synthesis is a means to compute correct-by-construction controllers for discrete event systems. As these systems evolve over time, we want to be able to efficiently generate a supervisor each time the system is adapted. We consider the case that a supervisor has been synthesized for a given base model, after which this base model is adapted to some variant model. Model deltas are used to describe the difference between the base and the variant model. A notion of atomic adaptations is introduced, where the model delta can be described by a single, indivisible change in the automaton specification. For these atomic model adaptations, algorithms are provided to compute the supervisor for the variant model in a transformational manner. The atomic model adaptations can be iterated over to transformationally compute a supervisor for any model delta that contains a number of atomic model adaptations. It is discussed why, and shown in experiments that, only purely iterating over these atomic adaptations is not efficient. Therefore a method is presented where groups of adaptations are considered. By means of both an academic and an industrial case study, we show that in some, but not all, cases the method of GTSS can more efficiently compute the variant model supervisor than SS. The best results for GTSS were found for the larger, industrial, use case.

The methods we presented are based on monolithic synthesis of a fully enumerated state space. To tackle industrial-sized systems, commonly modular methods that split the synthesis problem into smaller sub-problems are applied. It would be interesting to see how the transformational synthesis method we present here translates to those methods. The performance of transformational synthesis could further be improved by, for example, evaluating the adaptations in a smart order, or by imposing restrictions on the model delta. It might also be possible to obtain better results by relaxing the problem constraints, for example not requiring full equality between the results of TSS and SS, but only requiring them to be bisimilar. At the moment we have no way to tell if TSS will be more efficient than simply performing a new synthesis. Optionally, one could run both algorithms in parallel, and stop as soon as on of them is finished. It might also be valuable to have a method that predicts which algorithm will be more efficient. Moreover, perhaps it is possible to have a TSS algorithm that is guaranteed to require less effort than a new synthesis.

Appendix A

In this appendix, we provide the proofs of Theorem 2 for the atomic TSS algorithms provided in this paper. Subsequently the proofs for Lemma 5 are provided for an added state, removed state, added event, and removed event.

A.1 Added initial property (Algorithm 5)

We denote automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, $A' = (X, \Sigma, \longrightarrow, X_0 \cup \{x^\delta\}, X_m)$. Additionally, we denote $(Y, G) = \texttt{computeFixpoint}(A)$, $(Y', G') = \texttt{computeFixpoint}(A')$, and $(\hat{Y}, \hat{G}) = \texttt{TSSAIP}(A, Y, G, x^\delta)$.

- For any $(X_0, X_0') \subseteq X \times X$ it holds that $G = G'$ when computing $(Y, G) = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0, X_m)$, $(Y', G') = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0', X_m)$, as the initial state does not influence the computation of G. We also observe that for all switchcases in Algorithm 5, $\hat{G} = G$ is computed. It follows that $\hat{G} = G'$.

- Y' are all reachable states in G'. Since we have proven that $\hat{G} = G'$, it suffices to prove that $\hat{Y}$ are all reachable states in $\hat{G}$ to show that $\hat{Y} = Y'$.
  - In case that x^δ is in Y, the state x^δ will already have been found in the FRS (line 8 Algorithm 2), so in this case $\hat{Y} = Y = Y'$, which is also found by Algorithm 5.
  - In case that x^δ is in $X \setminus G$, it will also be in $X \setminus G'$. As x^δ is not in G', the change of initial property can not influence the reachable part of G', so $Y = Y'$. This is also found in Algorithm 5. So $\hat{Y} = Y = Y'$.
  - In case that x^δ is in $G \setminus Y$, for the supervisor synthesis of A', the reachable part is determined by $Y = \texttt{FRS}(G', \Sigma, \longrightarrow, X_0')$, where we know that $G' = G$ and $X_0' = X_0 \cup \{x^\delta\}$. $\hat{Y}$ is calculated by $\texttt{FRS}(G, \Sigma, \longrightarrow, Y \cup \{x^\delta\})$. The result of these FRSs is the same, as we know that all states in Y are reachable in G from X_0 from the base synthesis. So $\hat{Y} = Y'$.

We can conclude that $\hat{Y} = Y'$ and $\hat{G} = G'$ for any $x^\delta \in X \setminus X_0$, so Theorem 2 holds for Algorithm 5. □

A.2 Removed initial property (Algorithm 6)

Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0 \setminus \{x^\delta\}, X_m)$. Additionally, we denote $(Y, G) = \texttt{computeFixpoint}(A)$, $(Y', G') = \texttt{computeFixpoint}(A')$, and $(\hat{Y}, \hat{G}) = \texttt{TSSRIP}(A, Y, G, x^\delta)$.

- For any $(X_0, X_0') \subseteq X \times X$ it holds that $G = G'$ when computing $(Y, G) = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0, X_m)$, $(Y', G') = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0', X_m)$, as the initial state does not influence the computation of G. We also observe that for all switchcases in Algorithm 6, $\hat{G} = G$ is computed. It follows that $\hat{G} = G'$.
- Y' are all reachable states in G'. Since we have proven that $\hat{G} = G'$, it suffices to prove that $\hat{Y}$ are all reachable states in $\hat{G}$ to show that $\hat{Y} = Y'$.
  - In case that x^δ is not in Y, the state x^δ was not in the reachable part of G. Thus, the set of reachable states in G is not influenced by x^δ being initial. So in this case $\hat{Y} = Y = Y'$, which is also found by Algorithm 6.
  - In case that x^δ is in Y, for the supervisor synthesis of A', the reachable part is determined by $Y = \texttt{FRS}(G', \Sigma, \longrightarrow, X_0')$, where we know that $G' = G$ and $X_0' = X_0 \setminus \{x^\delta\}$. $\hat{Y}$ is calculated by $\texttt{FRS}(Y, \Sigma, \longrightarrow, X_0 \setminus \{x^\delta\})$. The result of these FRSs is the same, as we know that all states in $G \setminus Y$ are not reachable from the base synthesis. So $\hat{Y} = Y'$.

We can conclude that $\hat{Y} = Y'$ and $\hat{G} = G'$ for any $x^\delta \in X_0$, so Theorem 2 holds for Algorithm 6. □

A.3 Added marked property (Algorithm 7)

Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0, X_m \cup \{x^\delta\})$. Additionally, we denote $(Y, G) = \texttt{computeFixpoint}(A)$, $(Y', G') = \texttt{computeFixpoint}(A')$, and $(\hat{Y}, \hat{G}) = \texttt{TSSAMP}(A, Y, G, x^\delta)$.

- In case that x^δ is in G, it was already in the maximal coreachable and controllable set of states. It will remain so after making it marked, so $G' = G$. We observe $\hat{G} = G$ is computed in Algorithm 7, so $\hat{G} = G'$. As the initial states did not change, and $G' = G$, the reachable part Y will remain the same. So $\hat{Y} = Y = Y'$.
- In case that x^δ is in $X \setminus G$, states in Y remain reachable, coreachable, and controllable. So, `computeFixpoint`$((X, \Sigma, \longrightarrow, Y \cup X_0, Y \cup X_m \cup \{x^\delta\}))$ = `computeFixpoint`$((X, \Sigma, \longrightarrow, X_0, X_m \cup \{x^\delta\}))$.

We can conclude that $\hat{Y} = Y'$ and $\hat{G} = G'$ for any $x^\delta \in X \setminus X_m$, so Theorem 2 holds for Algorithm 7. □

A.4 Removed marked property (Algorithm 8)

Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0, X_m \setminus \{x^\delta\})$. Additionally, we denote $(Y, G) =$ `computeFixpoint`(A), $(Y', G') =$ `computeFixpoint`(A'), and $(\hat{Y}, \hat{G}) =$ `TSSRMP`(A, Y, G, x^δ).

- We know that $x^\delta \in X_m$ (model delta is valid). And all states in X_m are coreachable by definition. G was the maximal controllable coreachable set to X_m. After removing x^δ as a marked state; $G' \subseteq G$ and $Y' \subseteq Y$. So $X \setminus G \subseteq X \setminus G'$. Therefore `computeFixpoint`$((G, \Sigma, \longrightarrow, X_0, X_m \setminus \{x^\delta\}))$ = `computeFixpoint`$((X, \Sigma, \longrightarrow, X_0, X_m \setminus \{x^\delta\}))$. Thus, in case $x^\delta \in Y$, then $\hat{Y} = Y'$ and $\hat{G} = G'$.
- In case that x^δ is in $G \setminus Y$, all states in Y are coreachable and controllable for $X_m \setminus \{x^\delta\}$, as x^δ is not reachable from Y, otherwise it would be contained in Y. Therefore `computeFixpoint`$((G, \Sigma, \longrightarrow, X_0, (Y \cup X_m) \setminus \{x^\delta\}))$ = `computeFixpoint`$((X, \Sigma, \longrightarrow, X_0, X_m \setminus \{x^\delta\}))$. So in case $x^\delta \in G \setminus Y$, $\hat{Y} = Y'$ and $\hat{G} = G'$.
- In case that $x^\delta \notin G$, x^δ must have an uncontrollable path to a non-coreachable state, as it is coreachable as a marked state, it would have been in G if it were controllable. States that uncontrollably reach x^δ were removed from G in the base synthesis. G remains the same, and consequently Y will also remain the same, so $\hat{Y} = Y = Y'$ and $\hat{G} = G = G'$, which is also found by Algorithm 8.

We can conclude that $\hat{Y} = Y'$ and $\hat{G} = G'$ for any $x^\delta \in X_m$, so Theorem 2 holds for Algorithm 8. □

A.5 Added transition (Algorithm 9)

Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \cup\{(x_{or}, \sigma, x_{tar})\}, X_0, X_m)$. Additionally, we denote $(Y, G) =$ `computeFixpoint`(A), $(Y', G') =$ `computeFixpoint`(A'), and $(\hat{Y}, \hat{G}) =$ `TSSAT`$(A, Y, G, (x_{or}, \sigma, x_{tar}))$.

- In case $x_{or} \in Y$ and $x_{tar} \in Y$, all states in Y remain reachable, coreachable, reachable, and controllable. Also the (co-)reachability and controllability of the states in $X \setminus Y$ remains the same. So $\hat{Y} = Y = Y'$ and $\hat{G} = G = G'$.
- In case $x_{or} \in Y$ and $x_{tar} \in G \setminus Y$, the coreachability an controllability of all states remains unchanged. So $\hat{G} = G = G'$. All states in Y remain reachable, so `FRS`$(G, \Sigma, \longrightarrow \cup\{(x_{or}, \sigma, x_{tar})\}, Y)$ = `FRS`$(G, \Sigma, \longrightarrow \cup\{(x_{or}, \sigma, x_{tar})\}, X_0)$.

- In case x_{or} in $X \setminus G$ and $x_{or} \neq x_{tar}$, then all states in Y remain (co-)reachable and controllable. Therefore, $\texttt{computeFixpoint}((X, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, Y \cup X_0, Y \cup X_m)) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m))$.
- In case $x_{or} \in G$ and $x_{tar} \in X \setminus G$. The states in $X \setminus G$ remain non-coreachable or non-controllable. In case that
  - $\sigma \in \Sigma_c$. The supervisor can disable the added transition. So $\hat{Y} = Y = Y'$ and $\hat{G} = G = G'$.
  - $\sigma \in \Sigma_u$. The supervisor can not disable the added transition. Thus x_{or} is non-controll-able. Because also states in $X \setminus G$ remain non-coreachable or non-controllable, $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow \cap((G \setminus \{x_{or}\}) \times \Sigma \times G), X_0 \setminus \{x_{or}\}, X_m \setminus \{x_{or}\})) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m))$.
- In case $x_{or} \in G \setminus Y$ and $x_{tar} \in G$ the coreachability and controllability of all states does not change; $\hat{G} = G = G'$. x_{or} is non-reachable, so the added transition does not change the reachability of any state. Thus, $\hat{G} = G = G'$.
- In case $x_{or} = x_{tar}$, the (co-)reachability and controllability of any state does not change. So $\hat{G} = G = G'$ and $\hat{Y} = Y = Y'$.

We can conclude that $\hat{Y} = Y'$ and $\hat{G} = G'$ for any $(x_{or}, \sigma, x_{tar}) \in (X \times \Sigma \times X) \setminus \longrightarrow$, so Theorem 2 holds for Algorithm 9. $\square$

A.6 Removed transition (Algorithm 10)

Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m)$. Additionally, we denote $(Y, G) = \texttt{computeFixpoint}(A)$, $(Y', G') = \texttt{computeFixpoint}(A')$, and $(\hat{Y}, \hat{G}) = \texttt{TSSRT}(A, Y, G, (x_{or}, \sigma, x_{tar}))$.

- In case $x_{or} \in Y$, $x_{tar} \in Y$, and $x_{or} \neq x_{tar}$, states in $X \setminus G$ remain non-coreachable or non-controllable. Therefore $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m)) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m))$.
- In case $x_{or} \in G \setminus Y$, $x_{tar} \in G$, and $x_{or} \neq x_{tar}$, states in $X \setminus G$ remain non-coreachable or non-controllable. Also, states in Y remain (co-)reachable and controllable. Therefore, $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, Y \cup X_0, Y \cup X_m)) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m))$.
- In case $x_{or} \in X \setminus G$, $x_{tar} \in X \setminus G$, and $x_{or} \neq x_{tar}$, states in Y remain (co-)reachable and controllable. States in G remain coreachable and controllable. In case that
  - $\sigma \in \Sigma_c$, the non-coreachablity or non-controllability of x_{or} and x_{tar} do not change. So $\hat{G} = G = G'$ and $\hat{Y} = Y = Y'$.
  - $\sigma \in \Sigma_u$, then x_{or} may have been non-controllable, but may be controllable in the variant model. Because states in Y remain (co-)reachable and controllable, and states in G remain coreachable and controllable; $Y \subseteq Y'$ and $G \subseteq G'$. Therefore $\texttt{computeFixpoint}((X, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, Y \cup X_0, G \cup X_m)) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \setminus \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m))$.
- A removed transition from $x_{or} \in Y$ to $x_{tar} \in G \setminus Y$ can not exist, as x_{tar} was reachable in the base model by this transition, and x_{tar} would have existed in Y.
- In case $x_{or} \in Y$ and $x_{tar} \in X \setminus G$, then states in Y remain (co-)reachable and controllable, states in G remain coreachable and controllable, but not reachable, and states

in $X \setminus G$ remain non-coreachable or non-controllable. Therefore $\hat{G} = G = G'$ and $\hat{Y} = Y = Y'$.

- In case $x_{or} \in X \setminus G$ and $x_{tar} \in G$, then x_{or} was coreachable in the base model. As it is not in G, x_{or} must be non-controllable. It will remain as such after removal of the transition $(x_{or}, \sigma, x_{tar})$. Thus, states in Y remain (co-)reachable and controllable, states in G remain coreachable and controllable, but not reachable, and states in $X \setminus G$ remain non-coreachable or non-controllable. Therefore $\hat{G} = G = G'$ and $\hat{Y} = Y = Y'$.
- In case $x_{or} = x_{tar}$, the (co-)reachability and controllability of any state does not change. So $\hat{G} = G = G'$ and $\hat{Y} = Y = Y'$.

We can conclude that $\hat{Y} = Y'$ and $\hat{G} = G'$ for any $(x_{or}, \sigma, x_{tar}) \in \longrightarrow$, so Theorem 2 holds for Algorithm 10. □

A.7 Added state

In case a state x^δ is added as an atomic model adaptation, there are no transitions to or from this state, because for the base model it holds that $\longrightarrow \subseteq X \times \Sigma \times X$, and $x^\delta \notin X$ It is also not an initial state or marked state because for the base model it holds that $X_0 \subseteq X$ and $X_m \subseteq X$. Therefore the added state x^δ is non-coreachable and non-reachable in the variant model. Therefore $Y' = Y$, and $G' = G$, proving Lemma 5 for an added state. □

A.8 Removed state

In case a state x^δ is removed as an atomic model adaptation, there are no transitions to or from this state, because for the variant model it holds that $\longrightarrow \subseteq (X \setminus \{x^\delta\}) \times \Sigma \times (X \setminus \{x^\delta\})$. It is also not an initial state or marked state because for the variant model it holds that $X'_0 \subseteq X \setminus \{x^\delta\}$ and $X'_m \subseteq X \setminus \{x^\delta\}$. Therefore the removed state x^δ is non-coreachable and non-reachable in the base model. Therefore $Y' = Y$, and $G' = G$, proving Lemma 5 for a removed state. □

A.9 Added event

In case an event σ is added as an atomic adaptations, there are no transitions over this event, because for the base model it holds that $\longrightarrow \subseteq X \times \Sigma \times X$, and $\sigma \notin \Sigma$. Therefore, adding the event is not going to influence the (co-)reachability or controllability of any state. Thus, $Y' = Y$, and $G' = G$, proving Lemma 5 for an added event. □

A.10 Removed event

In case an event σ is removed as an atomic adaptations, there are no transitions over this event in the base model, because for the variant model it holds that $\longrightarrow' \subseteq X \times (\Sigma \setminus \{\sigma\}) \times X$, and $\longrightarrow' = \longrightarrow$. Therefore, removing the event is not going to influence the (co-)reachability or controllability of any state. Thus, $Y' = Y$, and $G' = G$, proving Lemma 5 for a removed event. □

Appendix B

The lemmas in this appendix support Theorem 3 for Algorithm 11. Following from the lemmas, Theorem 3 is proven at the end of this appendix.

Lemma 6 *Following each line in Algorithm 11 that directly follows a call to an atomic TSS algorithm (i.e., lines 5,9,13,17,21,25), it holds for the intermediate automaton that is formed by* $\hat{A} = (X', \Sigma', \longrightarrow', X'_0, X'_m)$ *that* `computeFixpoint`$(\hat{A}) = (Y', G')$, *where* (Y', G') *is the result of the atomic TSS algorithm in the line above.*

Proof The correct result (by Theorem 2) of each atomic TSS algorithm is proven in Appendix A. $\hat{A}$ correctly constructs the variant model after applying the atomic model delta, following the definitions in Section 3. □

Lemma 7 *Each time an atomic TSS algorithm is initiated by Algorithm 11, the atomic adaptation is a valid model delta for the automaton that is input.*

Proof We subdivide the proof over all calls to the atomic TSS algorithms, Algorithms 5 - 10:

- From Section 3, we know that $X_0^+ \subseteq (X \cup X^+) \setminus X^-$, and $X_0^+ \cap X_0 = \emptyset$. At the time Algorithm 5 (`TSSAIP`) is initiated with $x^\delta \in X_0^+$, this is with an automaton with state set $X' = X \cup X^+$ and initial state set X'_0. It holds that $\{x^\delta\} \subseteq X'$. It holds that $\{x^\delta\} \cap X'_0 = \emptyset$, as x^δ is only added to X'_0 after the call to Algorithm 5 with x^δ as added initial state. We can conclude that $X_0^+ = \{x^\delta\}$ is a valid model delta for the automaton that is input when Algorithm 5 is initiated.
- From Section 3, we know that $X_0^- \subseteq X_0$. At the time Algorithm 6 (`TSSRIP`) is initiated with $x^\delta \in X_0^-$, this is with an automaton with initial state set $X'_0 \subseteq X \cup X^+$. x^δ is in X'_0 when Algorithm 6 is called with x^δ, as x^δ is only removed from X'_0 after this call. We can conclude that $X_0^- = \{x^\delta\}$ is a valid model delta for the automaton that is input when Algorithm 6 is initiated.
- From Section 3, we know that $X_m^+ \subseteq (X \cup X^+) \setminus X^-$, and $X_m^+ \cap X_m = \emptyset$. At the time Algorithm 7 (`TSSAMP`) is initiated with $x^\delta \in X_m^+$, this is with an automaton with state set $X' = X \cup X^+$ and marked state set X'_m. It holds that $\{x^\delta\} \subseteq X'$. It holds that $\{x^\delta\} \cap X'_m = \emptyset$, as x^δ is only added to X'_m after the call to Algorithm 7 with x^δ as added marked state. We can conclude that $X_m^+ = \{x^\delta\}$ is a valid model delta for the automaton that is input when Algorithm 7 is initiated.
- From Section 3, we know that $X_m^- \subseteq X_m$. At the time Algorithm 8 (`TSSRMP`) is initiated with $x^\delta \in X_m^-$, this is with an automaton with initial state set $X'_m \subseteq X \cup X^+$. x^δ is in X'_m when Algorithm 8 is called with x^δ, as x^δ is only removed from X'_m after this call. We can conclude that $X_m^- = \{x^\delta\}$ is a valid model delta for the automaton that is input when Algorithm 8 is initiated.
- From Section 3, we know that: $\longrightarrow \cup [--] \longrightarrow^+ \subseteq X' \times \Sigma' \times X'$, and $\longrightarrow^+ \cap \longrightarrow = \emptyset$. At the time Algorithm 9 (`TSSAT`) is initiated with $(x_{or}, \sigma, x_{tar}) \in \longrightarrow^+$, this is with an automaton with state set $X' = X \cup X^+$ and event set $\Sigma' = \Sigma \cup \Sigma^+$. It holds that $\{(x_{or}, \sigma, x_{tar})\} \subseteq X' \times \Sigma' \times X'$. It holds that $\{(x_{or}, \sigma, x_{tar})\} \cap \longrightarrow' = \emptyset$, as $\{(x_{or}, \sigma, x_{tar})\}$ is only added to $\longrightarrow'$ after the call to Algorithm 9 with $(x_{or}, \sigma, x_{tar})$ as added transition.

We can conclude that $\longrightarrow^+ = \{(x_{or}, \sigma, x_{tar})\}$ is a valid model delta for the automaton that is input when Algorithm 9 is initiated.

– From Section 3, we know that $\longrightarrow^- \subseteq \longrightarrow$. At the time Algorithm 10 (TSSRT) is initiated with $(x_{or}, \sigma, x_{tar}) \in \longrightarrow^-$, this is with an automaton with state set $X' = X \cup X^+$ and event set $\Sigma' = \Sigma \cup \Sigma^+$. $(x_{or}, \sigma, x_{tar})$ is in $\longrightarrow'$ when Algorithm 10 is called with $(x_{or}, \sigma, x_{tar})$, as $(x_{or}, \sigma, x_{tar})$ is only removed from $\longrightarrow'$ after this call. We can conclude that $\longrightarrow^- = \{(x_{or}, \sigma, x_{tar})\}$ is a valid model delta for the automaton that is input when Algorithm 10 is initiated.

Together, the above cases proof Lemma 7. □

Lemma 8 *Each time states are added or removed, or events are added or removed in* Algorithm 11*, this adaptation is a valid model delta for the automaton it is performed on.*

Proof From Section 3, we know that $X^+ \cap X = \emptyset$ and $\Sigma^+ \cap \Sigma = \emptyset$ The added states and added events are only added once to state set X in Algorithm 11, at the point they are added, this is a valid model delta. The removed states and removed events are removed from automaton $A' = (X', \Sigma', \longrightarrow', X_0', X_m')$ in line 28. At this point, all added transitions are added to $\longrightarrow'$ and removed transitions are removed from $\longrightarrow$. Also all states with added marked property are added to X_m', all states with removed marked property are removed from X_m', all states with added initial property are added to X_0', and all states with removed initial property are removed from X_0'. So in automaton A', $X^- \cap X_m' = \emptyset$, $X^- \cap X_0' = \emptyset$, and $\longrightarrow' \subseteq ((X \cup X^+) \setminus X^-) \times ((\Sigma \cup \Sigma^+) \setminus \Sigma^-) \times ((X \cup X^+) \setminus X^-)$. In other words, the removed states are not initial or marked, there are no transitions to-or from removed states, and there are no transitions over removed events. Thus, Lemma 8 is proven for Algorithm 11. □

Lemma 9 *If final fixpoint result Y in* SS *is equal to fixpoint result Y' in* ITSS*, then supervisor S is also the same.*

Proof From Lemma 6 it follows that the final fixpoint result Y' in ITSS is equal to the final fixpoint result Y in SS. By following the steps of Algorithm 11, it follows that at line 27; $\Sigma' = (\Sigma \cup \Sigma^+)$. Line 27 of Algorithm 11 is equal to line 2 of Algorithm 1 when $Y' = Y$ and $\Sigma' = (\Sigma \cup \Sigma^+) \setminus \Sigma^-$, thus computing the same automaton S. □

From Lemmas 6-8 it follows that each intermediate result (Y', G') is correctly constructed in Algorithm 11, so that the final result (Y', G') is equal to computeFixpoint(A'). From Lemma 9 it follows that the supervisor automaton computed by Algorithm 11 is the same as the supervisor automaton computed by Algorithm 1, for the same supervisor states in Y. Together, the lemmas show that Theorem 3 holds. □

Appendix C

First we proof the application of a set $\Delta^{\sim}$ atomic model adaptations in $\Delta^{\times}$, defined in Section 5.2, in the same manner as Appendix A. Following, we provide the proof for Theorem 4 for Algorithm 12.

Lemma 10 *For a set $\Delta^{\sim}$, the same statements associated with a case statements of Algorithms 5 - 10 can be applied, as long as $\forall \delta \in \Delta^{\sim}$ the same case condition holds.*

Proof We structure our proof the same way as Appendix A, for each atomic TSS algorithm the possible $\Delta^{\sim}$ in $\Delta^{\times}$ is discussed. We denote $(Y, G) = \texttt{computeFixpoint}(A)$, $(Y', G') = \texttt{computeFixpoint}(A')$, and $(\hat{Y}, \hat{G})$ is calculated by the TSS algorithm.

- We consider $\Delta^{\sim} = X_0^{+} \cap (G \setminus Y)$.
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0 \cup \Delta^{\sim}, X_m)$.
  - For any $(X_0, X_0') \in X \times X$ it holds that $G = G'$ when computing $(Y, G) = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0, X_m)$, $(Y', G') = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0', X_m)$, as the initial state does not influence the computation of G. We also observe that for all switchcases in Algorithm 5, $\hat{G} = G$ is computed. It follows that $\hat{G} = G'$.
  - As $\Delta^{\sim} \subseteq G \setminus Y$, for the supervisor synthesis of A', the reachable part is determined by $Y = \texttt{FRS}(G', \Sigma, \longrightarrow, X_0')$, where we know that $G' = G$ and $X_0' = X_0 \cup \Delta^{\sim}$. $\hat{Y}$ is calculated by $\texttt{FRS}(G, \Sigma, \longrightarrow, Y \cup \Delta^{\sim})$. The result of these FRSs is the same, as we know that all states in Y are reachable in G from X_0 from the base synthesis. So $\hat{Y} = Y'$.

- We consider $\Delta^{\sim} = X_0^{-} \cap Y$.
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0 \setminus \Delta^{\sim}, X_m)$.
  - For any $(X_0, X_0') \in X \times X$ it holds that $G = G'$ when computing $(Y, G) = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0, X_m)$, $(Y', G') = \texttt{computeFixpoint}(X, \Sigma, \longrightarrow, X_0', X_m)$, as the initial state does not influence the computation of G. We also observe that for all switchcases in Algorithm 5, $\hat{G} = G$ is computed. It follows that $\hat{G} = G'$.
  - In case that $\Delta^{\sim} \subseteq Y$, for the supervisor synthesis of A', the reachable part is determined by $Y = \texttt{FRS}(G', \Sigma, \longrightarrow, X_0')$, where we know that $G' = G$ and $X_0' = X_0 \setminus \Delta^{\sim}$. $\hat{Y}$ is calculated by $\texttt{FRS}(Y, \Sigma, \longrightarrow, X_0 \setminus \Delta^{\sim})$. The result of these FRSs is the same, as we know that all states in $G \setminus Y$ are not reachable from the base synthesis. So $\hat{Y} = Y'$.

- We consider $\Delta^{\sim} = X_m^{+} \cap X \setminus G$.
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0, X_m \cup \Delta^{\sim})$.
  - In case that $\Delta^{\sim} \subseteq X \setminus G$, states in Y remain reachable, coreachable, and controllable. So, $\texttt{computeFixpoint}((X, \Sigma, \longrightarrow, Y \cup X_0, Y \cup X_m \cup \Delta^{\sim})) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow, X_0, X_m \cup \Delta^{\sim}))$.

- We consider $\Delta^{\sim} = X_m^{-} \cap Y$.
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0, X_m \setminus \Delta^{\sim})$.
  - We know that $\Delta^{\sim} \subseteq X_m$ (model delta is valid). And all states in X_m are coreachable by definition. G was the maximal controllable coreachable set to X_m. After removing $\Delta^{\sim}$ as marked states; $G' \subseteq G$ and $Y' \subseteq Y$. So all states in $X \setminus G \subseteq X \setminus G'$. Therefore $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow, X_0, X_m \setminus \Delta^{\sim})) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow, X_0, X_m \setminus \Delta^{\sim}))$. So in case $\Delta^{\sim} \subseteq Y$, $\hat{Y} = Y'$ and $\hat{G} = G'$.

- We consider $\Delta^{\sim} = X_m^{-} \cap G \setminus Y$.
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow, X_0, X_m \setminus \Delta^{\sim})$.
  - In case that $\Delta^{\sim} \subseteq G \setminus Y$, all states in Y are coreachable and controllable for $X_m \setminus x^{\delta}$, as states in $\Delta^{\sim}$ are not reachable from Y, otherwise they would be contained in Y. Therefore $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow, X_0, (Y \cup X_m) \setminus \Delta^{\sim})) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow, X_0, X_m \setminus \Delta^{\sim}))$. So in case $\Delta^{\sim} \subseteq G \setminus Y$, $\hat{Y} = Y'$ and $\hat{G} = G'$.
- We consider $\Delta^{\sim} = \longrightarrow^{+} \cap Y \times \Sigma \times Y$.
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \cup \Delta^{\sim}, X_0, X_m)$.
  - The coreachability an controllability of all states remains unchanged. So $\hat{G} = G = G'$. All states in Y remain reachable, so $\texttt{FRS}(G, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, Y) = \texttt{FRS}(G, \Sigma, \longrightarrow \cup \Delta^{\sim}, X_0)$.
- We consider $\Delta^{\sim} = \longrightarrow^{+} \cap G \times \Sigma_u \times X \setminus G$.
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \cup \Delta^{\sim}, X_0, X_m)$.
  - The states in $X \setminus G$ remain non-coreachable or non-controllable. The supervisor can not disable the added transitions. All states in X_{or} are non-controllable, where $X_{or} = \{x_{or} | (x_{or}, \sigma, x_{tar}) \in \Delta^{\sim}, \sigma \in \Sigma_u, x_{tar} \in X\}$. Because also states in $X \setminus G$ remain non-coreachable or non-controllable, $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow \cap ((G \setminus X_{or}) \times \Sigma \times G), X_0 \setminus X_{or}, X_m \setminus X_{or})) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, X_0, X_m))$.
- We consider $\Delta^{\sim} = \{(x_{or}, \sigma, x_{tar}) \in \longrightarrow^{+} | x_{or} \in X \setminus G \wedge x_{or} \neq x_{tar}\}$
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \cup \Delta^{\sim}, X_0, X_m)$.
  - All states in Y remain (co-)reachable and controllable. Thus we conclude, $\texttt{computeFixpoint}((X, \Sigma, \longrightarrow \cup \{(x_{or}, \sigma, x_{tar})\}, Y \cup X_0, Y \cup X_m)) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \cup \Delta^{\sim}, X_0, X_m))$.
- We consider $\Delta^{\sim} = \{(x_{or}, \sigma, x_{tar}) \in \longrightarrow^{-} | x_{or} \in Y \wedge x_{tar} \in Y \wedge x_{or} \neq x_{tar}\}$
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \setminus \Delta^{\sim}, X_0, X_m)$.
  - States in $X \setminus G$ remain non-coreachable or non-controllable. Therefore $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow \setminus \Delta^{\sim}, X_0, X_m)) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \setminus \Delta^{\sim}, X_0, X_m))$.
- We consider $\Delta^{\sim} = \{(x_{or}, \sigma, x_{tar}) \in \longrightarrow^{-} | x_{or} \in G \setminus Y \wedge x_{tar} \in G \wedge x_{or} \neq x_{tar}\}$
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \setminus \Delta^{\sim}, X_0, X_m)$.
  - States in $X \setminus G$ remain non-coreachable or non-controllable. Also, states in Y remain (co-)reachable and controllable. It follows that $\texttt{computeFixpoint}((G, \Sigma, \longrightarrow \setminus \Delta^{\sim}, Y \cup X_0, Y \cup X_m)) = \texttt{computeFixpoint}((X, \Sigma, \longrightarrow \setminus \Delta^{\sim}, X_0, X_m))$.
- We consider $\Delta^{\sim} = \{(x_{or}, \sigma, x_{tar}) \in \longrightarrow^{-} | x_{or} \in X \setminus G \wedge x_{tar} \in X \setminus G \wedge \sigma \in \Sigma_u \wedge x_{or} \neq x_{tar}\}$
  Automaton $A = (X, \Sigma, \longrightarrow, X_0, X_m)$, and $A' = (X, \Sigma, \longrightarrow \setminus \Delta^{\sim}, X_0, X_m)$.
  - States in Y remain (co-)reachable and controllable. States in G remain coreachable and controllable. The origin states of the removed transition may have been non-controllable, but may be controllable in the variant

model. Because states in Y remain (co-)reachable and controllable, and states in G remain coreachable and controllable; $Y \subseteq Y'$ and $G \subseteq G'$. Therefore `computeFixpoint`$((X, \Sigma, \longrightarrow \setminus \Delta^{\sim}, Y \cup X_0, G \cup X_m)) =$ `computeFixpoint`$((X, \Sigma, \longrightarrow \setminus \Delta^{\sim}, X_0, X_m))$.

For any possible $\Delta^{\sim}$ Lemma 10 is proven. $\square$

Lemma 11 *Each time X', Σ', $\longrightarrow'$, X_0', or X_m' is constructed in Algorithm 12 (i.e., lines 1,6,8,12), it holds that* `computeFixpoint`$(X', \Sigma', \longrightarrow', X_0', X_m') = (Y', G')$*, for (Y', G') computed at that point in* Algorithm 12.

Proof We structure our proofs over the different lines where automaton A' is constructed.

- Lines 1,12: In case states are added or removed, or events are added or removed in Algorithm 12, the same proofs as for Lemmas 5 and 8 hold here.
- Line 6: This is the same iterative application as in Algorithm 11 (`ITSS`). The same proofs as for Lemmas 6 and 7 hold here.
- Line 8: The correct result of applying sets $\Delta^{\sim}$ is proven for Lemma 10 above. In conjunction with the proof for Lemma 7, this proofs Lemma 11 for line 8.

For each time X', Σ', $\longrightarrow'$, X_0', or X_m' is constructed in Algorithm 12, Lemma 11 is proven. $\square$

From Lemma 11 it follows that each intermediate result (Y', G') is correctly constructed in Algorithm 12, so that the final result (Y', G') is equal to `computeFixpoint`(A'). Using the same proof as Lemma 9, it follows that the supervisor automaton computed by Algorithm 12 is the same as the supervisor automaton computed by Algorithm 1, for automaton A'. Together, this shows that Theorem 4 holds. $\square$

References

Broadfoot GH, Hopcroft PJ (2003) Analytical software design. Technical report, Verum Consultants B.V.
Cassandras CG, Lafortune S (2008) Introduction to Discrete Event Systems, 2nd edn. Springer, Boston
Classen A, Heymans P, Schobbens PY, Legay A, Raskin JF (2010) Model checking lots of systems: Efficient verification of temporal properties in software product lines. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, p 335–344
Fei Z, Miremadi S, Åkesson K, Lennartson B (2014) Efficient symbolic supervisor synthesis for extended finite automata. IEEE Trans Control Syst Technol 22(6):2368–2375
Flordal H, Malik R, Fabian M, Åkesson K (2007) Compositional synthesis of maximally permissive supervisors using supervision equivalence. Discret Event Dyn Syst 17(4):475–504
Khan YI (2013) Optimizing verification of structurally evolving algebraic Petri nets. In: Proceedings of the 5th International Workshop Software Engineering for Resilient Systems, Lecture Notes in Computer Science, vol 8166, pp 64–78

Kleinberg J, Tardos E (2005) Algorithm Design Addison-Wesley Longman Publishing Co. Inc., Boston

Krook J, Kianfar R, Fabian M (2020) Formal synthesis of safe stop tactical planners for an automated vehicle. In: Proceedings of the 15th IFAC Workshop on Discrete Event Systems, IFAC-PapersOnLine, vol 53, pp 445–452

Larman C, Basili VR (2003) Iterative and incremental development: a brief history. Computer 36(6):47–56

Lehman MM (1996) Laws of software evolution revisited. Softw Process Technol Lect Notes Comput Sci 1149:108–124

Markovski J, van BeekD, Theunissen R, Jacobs K, Rooda J (2010) A state-based framework for supervisory control synthesis and verification. In: Proceedings of the 49th IEEE Conference on Decision and Control, pp 3481–3486

Ouedraogo L, Kumar R, Malik R, Åkesson K (2011) Nonblocking and safe control of discrete-event systems modeled as extended finite automata. IEEE Trans Autom Sci Eng 8(3):560–569

Pohl K, Böckle G, van der Linden FJ (2005) Software Product Line Engineering: Foundations Principles and Techniques. Springer-Verlag, Berlin

Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230

Ramadge PJ, Wonham WM (1989) The control of discrete event systems. Proc IEEE 77(1):81–98

Rawlings BC, Christenson B, Wassick JM, Ydstie BE (2014) Supervisor synthesis to satisfy safety and reachability requirements in chemical process control. In: Proceedings of the 12th IFAC Workshop on Discrete Event Systems, IFAC Proceedings Volumes, vol 47, pp 195–200

Reijnen FFH, Goorden MA, van de Mortel-Fronczak JM, Rooda JE (2020) Modeling for supervisor synthesis – a lock-bridge combination case study. Discret Event Dyna Syst 30(3):499–532

Reniers MA, Thuijsman SB (2020) Supervisory control for dynamic feature configuration in product lines. In: Proceedings of the IEEE Forum on Specification and Design Languages, pp 1–8

Rosa M, Cury JER, Baldissera FL (2020) Supervisory control in construction robotics: in the quest for scalability and permissiveness. In: Proceedings of the 15th IFAC Workshop on Discrete Event Systems, IFAC-PapersOnLine, vol 53, pp 117–122

Schaefer I, Rabiser R, Clarke D, Bettini L, Benavides D, Botterweck G, Pathak A, Trujillo S, Villela K (2012) Software diversity: state of the art and perspectives. Int J Softw Tools Technol Transfer 14(5):477–495

Theunissen RJM, Petreczky M, Schiffelers RRH, van Beek DA, Rooda JE (2014) Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. IEEE Trans Autom Sci Eng 11(1):20–32

Thuijsman SB, Reniers MA (2020) Transformational supervisor synthesis for evolving systems. In: Proceedings of the 15th IFAC Workshop on Discrete Event Systems, IFAC-PapersOnLine, vol 53, pp 309–316

Tijsse Claase RG (2020) Symbolic transformational supervisor synthesis. Master's thesis, Eindhoven University of Technology, Department of Mechanical Engineering

Wonham WM, Cai K (2019) Supervisory Control of Discrete-Event Systems. Springer, Cham

Wonham WM, Cai K, Rudie K (2018) Supervisory control of discrete-event systems: A brief history. IFAC Annu Rev Control 45:250–256

ter Beek MH, Reniers MA, de Vink EP (2016) Supervisory controller synthesis for product lines using CIF 3. In: Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques, Lecture Notes in Computer Science, vol 9952, pp 856–873

ter Beek MH, de Vink EP (2014) Towards modular verification of software product lines with mCRL2. In: Proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, Lecture Notes in Computer Science, vol 8802, pp 368–385

van Beek DA, Fokkink WJ, Hendriks D, Hofkamp A, Markovski J, van de Mortel-Fronczak JM, Reniers MA (2014) CIf 3: Model-based engineering of supervisory controllers. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol 8413, pp 575–580

van der Sanden LJ, Reniers MA, Geilen MCW, Basten AA, Jacobs J, Voeten JPM, Schiffelers RRH (2015) Modular model-based supervisory controller design for wafer logistics in lithography machines. In: Proceedings of the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, pp 416–425

van der Schriek YIC (2018) Evaluation of supervisory control theory based on requirement evolution of LOPW. Master's thesis, Eindhoven University of Technology, Deptartment of Mechanical Engineering

Sander Thuijsman has performed his Bachelor's and Master's study Mechanical Engineering at Eindhoven University of Technology. Now he is pursuing his PhD at the same department, in collaboration with ASML. His research interests are discrete event systems, supervisory controller synthesis, system configuration and evolution, and computational efficiency.

Michel Reniers (S'17) is currently an Associate Professor in model-based engineering of supervisory control at the Department of Mechanical Engineering, Eindhoven University of Technology. He has authored over 100 journal and conference papers, and is the supervisor of ten Ph.D. students. His research portfolio ranges from model-based systems engineering and model-based validation and testing to novel approaches for supervisory control synthesis. Applications of this work are mostly in the areas of high-tech systems and cyber-physical systems.