Efficient Validation of Supervisory Controllers using Symmetry Reduction

Lars Moormann [*] **Martijn A. Goorden** [*]
Joanna M. van de Mortel-Fronczak [*] **Wan J. Fokkink** [**]
Patrick Maessen [***] **Jacobus E. Rooda** [*]

[*] *Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands (email: l.moormann@tue.nl)*
[**] *Department of Computer Sciences, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands*
[***] *Major Projects and Maintenance, Rijkswaterstaat, Utrecht, The Netherlands*

Abstract: Supervisory control synthesis is a method to automatically generate a correct-by-construction supervisory controller. Validation of the synthesized controller is an important step to guarantee correct and safe system behavior. Especially requirement validation for systems with numerous components can be a difficult and time-consuming task. This paper proposes a method that reduces the required validation time and effort of systems through symmetry reduction, and is based on the concept of isomorphism. Isomorphism of component models and requirement models means that these models are equivalent in behavior, and therefore only part of the system needs to be validated. This method is used in an industrial case study, in which a supervisory controller is synthesized for a road tunnel (the Koning Willem-Alexandertunnel, the Netherlands). In this case study, the modeling of the plant and the requirements, supervisor synthesis, simulation, and validation are described.

Keywords: Automata theory, supervisory control, control system synthesis, system reduction, validation.

1. INTRODUCTION

Over the years, cyber-physical systems (CPSs) have increased in size and complexity due to increasing market-demands for quality, efficiency, functionality, and safety. Consequently, the control systems of these CPSs have become more complex as well, while costs and time-to-market are desired to decrease. Formal methods, such as model-based engineering, can help tackling these challenges. Previous applications for which a control system was successfully designed using model-based engineering include an MRI support system in Theunissen et al. (2014), a waterway lock in Reijnen et al. (2017), transportation systems in Scott et al. (2016), and an automotive manufacturing line in Albers et al. (2015). Furthermore, by creating models of the system behavior and system requirements, formal verification and early validation become possible, as described in Baier and Katoen (2008) and Braspenning et al. (2006). This way, fewer errors are made during the design phase, resulting in a reduction in the time-to-market and in the repair costs.

The schematic overview of a CPS is shown in Fig. 1, inspired by the CPS architecture representations in Melnyk (2016) and Hu et al. (2012). Fig. 1 shows five layers. Layer 1 includes the mechanical components. Actuators and sensors in layer 2 are meant to actuate the mechanical components and monitor their current state, respectively. Often a third layer of resource controllers is present to control the low-level dynamic behavior of the mechanical components. The supervisory controller, in layer 4, is responsible for correctly coordinating all lower subsystems. Layer 5 of a CPS consists of the man-machine interface, through which an operator can monitor the CPS and send commands to it.

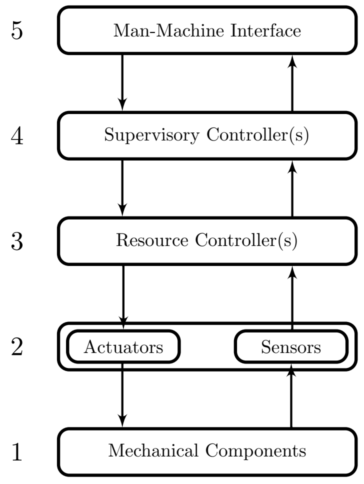

Fig. 1. Schematic overview of a CPS.

Designing a safe and correct supervisory controller can be challenging when many individual subsystems need to cooperate in a CPS. One of the biggest challenges when designing a supervisory controller is defining the controller requirements, as discussed in Kotonya and Sommerville

Manuscript 32 submitted to 15th IFAC Workshop on Discrete Event Systems.
Received December 17, 2019.

(1998), and validating the correctness of the resulting system behavior, as demonstrated in a case study in Liang et al. (2012). Defining a clear set of requirements is an important step in controller design. Using supervisory control theory can help in defining this clear set of requirements. Supervisory control theory is a formal method in which two types of models are created. First, all possible behavior of the discrete-event system (DES) is modeled using automata. These models are called the component models. Secondly, requirement models are defined to specify the desired system behavior. These models can be automata or state-based expressions. Supervisor synthesis, explained in Ramadge and Wonham (1987), can then be applied to automatically generate a supervisor, which by construction adheres to the defined requirements. Validation of the component and requirement models is still an important step as the resulting controlled system behavior can differ from the intended behavior due to errors in the component models or the requirement models. This validation can be a difficult and time-consuming task, especially for large CPSs.

This paper proposes a method to more efficiently validate supervisory controllers by reducing the number of component and requirement models that need to be validated. It is thus a step that can be taken before the validation, so the validation process itself (e.g., through simulations or hardware-in-the-loop tests) can still be chosen freely. The method, called symmetry reduction, is based on symmetry in the component models of a CPS. More formally, symmetry in component models is determined using isomorphism. Component models are isomorphic when their behavior is equivalent, thus indicating that validation of one of these component models is sufficient. Practice shows that symmetry occurs frequently in real-world systems, mainly due to the inherent desire of system engineers to decompose systems into smaller components. Modeling such components is often done using templates, described in Grigorov et al. (2011), which inherently introduces symmetry in the system. Symmetry reduction can significantly reduce the complexity of system models, as is shown in a case study in Section 4.

The idea to use symmetry reduction for verification and validation is not new. In Ip and Dill (1996), a state-space reduction method based on symmetry is introduced for the purpose of verification. Huber et al. (1985) and Schmidt (2000) use symmetry in the reachability analysis of Petri nets, and in Miller et al. (2006) symmetry reduction is used in temporal model checking.

This paper is structured as follows. In Section 2 the concepts and notations of DESs, isomorphism, and supervisor synthesis are explained. Section 3 focuses on describing the proposed method of symmetry reduction. The application of this method to a case study is discussed in Section 4, including the case description, modeling steps taken, and results. Finally, Section 5 concludes this paper.

2. PRELIMINARIES

2.1 Discrete-event systems

Discrete-event systems (DESs), explained in Cassandras and Lafortune (2009), are systems of which the state space is described by a discrete set and in which state transitions purely depend on events. One way to model DESs is using finite-state automata. An automaton P is denoted as a five-tuple:

$$P = (Q, E, f, q_0, Q_{\mathrm{m}}) \tag{1}$$

with the finite state set Q, the finite event set E, the partial transition function $f : Q \times E \to Q$, the initial state $q_0 \in Q$, and the marked state set $Q_{\mathrm{m}} \subseteq Q$. For the partial transition function, $f(x, e) = y$ means that in state x there is an eligible transition labeled with event e to state y. The event set E can be partitioned into the set of controllable events E_{c} and the set of uncontrollable events E_{u}. Controllable events are events that can be enabled by the supervisor, e.g., to turn an actuator on or off, whereas uncontrollable events cannot be influenced but only observed, e.g., a sensor turning on or off.

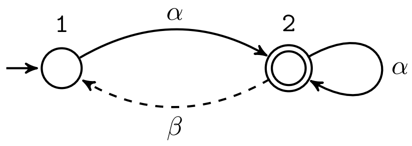

Fig. 2. Graphical representation of an automaton.

The graphical representation of an automaton is shown in Fig. 2. Circles in this automaton represent states, and transitions are represented by arrows that are labeled with an event. Solid arrows indicate controllable events and dashed arrows indicate uncontrollable events. The inward arrow on the left of state 1 denotes the initial state and the double circle at state 2 indicates a marked state.

In the context of supervisory control theory, introduced in Ramadge and Wonham (1987), two sets of models are created to describe the behavior of a system. The first set contains the component models and forms the model of the plant. The plant describes the possible behavior of the system and is modeled using automata. The second set contains the models of the control specifications, called the requirements. Requirements can be modeled either using automata or using state-based expressions, which are described in Markovski et al. (2010). There are two ways to model a state-based expression, being e **`needs`** q and q **`disables`** e. The first describes that event e is only allowed to occur in state q. The second means the opposite, thus that e is allowed to occur in all states but state q.

2.2 Isomorphism

There are multiple concepts of equivalence of finite-state automata. One of these concepts is the concept of isomorphism, described in Glushkov (1961). Two automata are said to be isomorphic if there exists a bijective mapping from the first to the second automaton, where the second automaton preserves the transition function, the output function, and the initial state. A bijective mapping between two sets pairs each element of one set with exactly one element of the other set, and pairs each element of the other set with exactly one element of the first set.

In Glushkov (1961), isomorphism is formally defined for Mealy machines. The main difference between Mealy machines, explained in Mealy (1955), and finite state automata, as defined in Section 2.1, is how the output is determined. In a Mealy machine, the output is determined based on the input and the current state using a certain output function. In a finite-state automaton, the only notion of an output is the set of marker states Q_m. In this paper, the notion of isomorphism for Mealy machines is transposed to finite-state automata. This involves the omission of the mapping between the output alphabets of the two Mealy automata. Furthermore, an additional condition is added for isomorphism of finite state automata, as is given in equation (6), that defines that marked states of the first automaton must be mapped to marked states of the second automaton. The definition of isomorphism for a finite-state automaton is described below. In the sequel, a finite state automaton is called an automaton.

An isomorphism h between plant automata $P_1 = (Q_1, E_1, f_1, q_{0,1}, Q_{\mathrm{m},1})$ and $P_2 = (Q_2, E_2, f_2, q_{0,2}, Q_{\mathrm{m},2})$, with controllable event sets $E_{\mathrm{c},1}$ and $E_{\mathrm{c},2}$, consists of bijective mappings

$$h_1 : E_1 \to E_2, \quad h_2 : Q_1 \to Q_2 \tag{2}$$

such that, for every $q \in Q_1$, $e \in E_1$:

$$h_2(f_1(q, e)) = f_2(h_2(q), h_1(e)) \tag{3}$$

$$h_2(q_{0,1}) = q_{0,2} \tag{4}$$

$$e \in E_{c,1} \Leftrightarrow h_1(e) \in E_{c,2} \tag{5}$$

$$q \in Q_{m,1} \Leftrightarrow h_2(q) \in Q_{m,2} \tag{6}$$

Furthermore, if P_1 and P_2 share events, meaning $E_1 \cap E_2 \neq \emptyset$, condition (7) must also be met for every $e \in E_1 \cap E_2$:

$$h_1(e) = e \tag{7}$$

The definition of isomorphism between plant automata P_1 and P_2 can be extended to isomorphism between sets of automata. Isomorphism between sets of automata is defined using the following lines.

(1) Let $\mathcal{P}^1 = \{P_{1,1}, P_{1,2}, .., P_{1,n}\}$ and $\mathcal{P}^2 = \{P_{2,1}, P_{2,2}, .., P_{2,n}\}$ be two sets of automata.
(2) Let $\Sigma_{\mathcal{P}^1} = \bigcup_{i=1}^{n} P_{1,i}$ and $\Sigma_{\mathcal{P}^2} = \bigcup_{i=1}^{n} P_{2,i}$ be the event sets of $\mathcal{P}^1$ and $\mathcal{P}^2$.
(3) $\mathcal{P}^1$ and $\mathcal{P}^2$ are isomorphic if
  (a) There exists a 1-to-1 correspondence between $\mathcal{P}^1$ and $\mathcal{P}^2$, such that for each automaton in $\mathcal{P}^1$ there exists an isomorphism $h = \{h_1, h_2\}$ with the corresponding automaton in $\mathcal{P}^2$.
  (b) For every $h \in \mathcal{H}, e \in \Sigma_{\mathcal{P}^1} \cap \Sigma_{\mathcal{P}^2} : h_1(e) = e$, where $\mathcal{H}$ is the set of isomorphisms between $\mathcal{P}^1$ and $\mathcal{P}^2$.

Isomorphism can also be defined for requirement models. In the case of automaton requirements, the same formulations can be used as above.

For state-based requirements, a similar reasoning can be applied. Isomorphism between requirements $R_1 : e_1$ **needs** q_1 and $R_2 : e_2$ **needs** q_2 can be determined using the following steps:

(1) Determine P_1, being the automaton containing state q_1, and P_2, being the automaton containing state q_2.
(2) Determine $\mathcal{P}^1$, being the set of automata containing a transition labeled by event e_1, and $\mathcal{P}^2$, being the set of automata containing a transition labeled by event e_2.
(3) Determine if $\mathcal{P}^1$ and $\mathcal{P}^2$ are isomorphic, with $\mathcal{H}$ being the set of isomorphisms between $\mathcal{P}^1$ and $\mathcal{P}^2$.
(4) If P_1 or P_2 is part of $\mathcal{P}^1$ or $\mathcal{P}^2$, the additional requirement is that in the 1-to-1 correspondence between $\mathcal{P}^1$ and $\mathcal{P}^2$, P_1 must be mapped to P_2.
(5) Requirements R_1 and R_2 are isomorphic, if for every $h \in \mathcal{H}$ it holds that $h_1(e_1) = e_2$ and $h_2(q_1) = q_2$.

The statements above are based on requirements where q_1 and q_2 consist of one state. They can, however, easily be extended for requirements that refer to multiple states, such as e **needs** q_1 `and` q_2. The additional condition is that the logical operators (e.g., `and`, `or`, `not`) are the same for both requirements.

Isomorphism of requirements of the form q_1 **disables** e_1 can be defined following the same lines.

2.3 Supervisor synthesis

Supervisor synthesis is a method to generate a supervisor from a set of component models and a set of requirement models. Supervisor synthesis guarantees that the supervisor by construction satisfies the following properties:

- Safe: the supervisor prevents all behavior that conflicts with the specified requirements.
- Non-blocking: from every reachable state, there exists a path to reach a marked state.
- Controllable: the supervisor never disables uncontrollable events.
- Maximally permissive: the supervisor disables as few events as possible, while guaranteeing the three previously mentioned properties.

A synthesis algorithm that generates supervisory controllers that satisfy these properties is explained in Ouedraogo et al. (2011). In the sequel, this algorithm is referred to as monolithic synthesis.

The main challenge of supervisor synthesis is dealing with the state-space size that grows exponentially with the number of automata. This exponential growth is called state-space explosion and is described in Lin et al. (1987). A possible method to synthesize supervisors for larger state-spaces than monolithic synthesis can handle is multilevel synthesis, described in Komenda et al. (2016). In this method, the uncontrolled plant is divided into multiple subsystems. This uncontrolled plant has a tree-like structure, where each node of the tree-structure consists of a subset of component and a subset of requirement models. A supervisor can then be synthesized for each node in this tree. Multilevel synthesis has been used in case studies that are presented in Komenda et al. (2016); Goorden et al. (2017); Reijnen et al. (2018), and has shown to significantly reduce computation time and controlled state-space sizes.

3. METHOD

A synthesized supervisor is guaranteed to be compliant to the requirements, so verification of these requirements is not needed. Validation is still required, as incorrectly defined requirements may result in incorrect controlled behavior. Requirements can be defined too relaxed or too strict, resulting in undesired or over-restricted behavior, respectively. Furthermore, errors can be made in the component models, leading to undesired functionality.

Requirements can be validated at an early design stage by performing simulations of the supervisory controller. In these simulations, a hybrid plant can be used instead of the discrete plant. The hybrid plant contains the same components as the discrete plant, though extended with continuous-time behavior to allow for easier and more intuitive simulations.

Even though simulations are a useful tool to validate requirements at an early stage, the validation process can still be an extensive and cumbersome task if it concerns a large set of component models and requirement models. Here, a method is proposed that can be used to reduce the number of components and requirement models that need to be validated.

The proposed method is as follows. When two systems are isomorphic, only the behavior of one system needs to be validated. Let us define a control problem as a two-tuple, $\mathcal{S} = (\mathcal{P}, \mathcal{R})$, where $\mathcal{P} = \{P_1, P_2, .., P_m\}$ is a set of automata, and $\mathcal{R} = \{R_1, R_2, .., R_n\}$ is a set of requirements. Furthermore, let us define the event set, also called the alphabet, of control problem $\mathcal{S}$ as

$$\Sigma_{\mathcal{S}} = \bigcup_{i=1}^{m} E_i \tag{8}$$

and the state set of control problem $\mathcal{S}$ as

$$\Phi_{\mathcal{S}} = \bigcup_{i=1}^{m} Q_i. \tag{9}$$

Two control problems $\mathcal{S}^1$ and $\mathcal{S}^2$ are isomorphic if

(1) the set of automata $\mathcal{P}^1$ is isomorphic to the set of automata $\mathcal{P}^2$ with isomorphism set $\mathcal{H}$, and
(2) there exists a 1-to-1 correspondence between $\mathcal{R}^1$ and $\mathcal{R}^2$, such that for every requirement in $\mathcal{R}^1$ there exists an isomorphic requirement in $\mathcal{R}^2$ using $\mathcal{H}$.

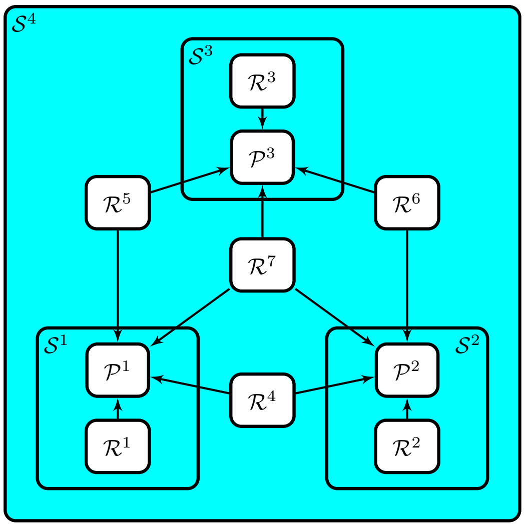

Fig. 3. Schematic overview of two isomorphic control problems $\mathcal{S}^1$ and $\mathcal{S}^2$ and a third control problem $\mathcal{S}^3$.

Fig. 3 shows a schematic overview of the general case, where two isomorphic control problems $\mathcal{S}^1$ and $\mathcal{S}^2$ are identified that are connected by requirement set $\mathcal{R}^4$. A third control problem $\mathcal{S}^3$ is connected to $\mathcal{S}^1$ and $\mathcal{S}^2$ via requirement sets $\mathcal{R}^5$, $\mathcal{R}^6$, and $\mathcal{R}^7$. The arrows indicate to which plant sets a certain requirement set refers. Five principles are now defined that can be used to reduce the number of component and requirement models that need to be validated.

(1) If the components in $\mathcal{P}^1$ are validated, the components in $\mathcal{P}^2$ do not need to be validated.
(2) If the requirements in $\mathcal{R}^1$ are validated, the requirements in $\mathcal{R}^2$ do not need to be validated.
(3) If $\mathcal{R}^4$ contains the requirements

$$e_1 \ \texttt{needs} \ q_2 \tag{10}$$
$$e_2 \ \texttt{needs} \ q_1 \tag{11}$$

with $e_1 \in \Sigma_{\mathcal{S}^1}, e_2 \in \Sigma_{\mathcal{S}^2}, q_1 \in \Phi_{\mathcal{S}^1}, q_2 \in \Phi_{\mathcal{S}^2}$, and the automata containing e_1 and e_2 are isomorphic, with $h_1(e_1) = e_2$, and the automata containing q_1 and q_2 are isomorphic, with $h_2(q_1) = q_2$, then only one of the two requirements needs to be validated.
(4) If $\mathcal{R}^5$ and $\mathcal{R}^6$ contain the requirements

$$e_1 \ \texttt{needs} \ q_3 \tag{12}$$
$$e_2 \ \texttt{needs} \ q_3 \tag{13}$$

respectively, with $e_1 \in \Sigma_{\mathcal{S}^1}, e_2 \in \Sigma_{\mathcal{S}^2}, q_3 \in \Phi_{\mathcal{S}^3}$, and the automata containing e_1 and e_2 are isomorphic, with $h_1(e_1) = e_2$, then only one of the two requirements needs to be validated.
(5) If $\mathcal{R}^7$ contains the requirements

$$e_1 \ \texttt{needs} \ q_2 \wedge q_3 \tag{14}$$
$$e_2 \ \texttt{needs} \ q_1 \wedge q_3 \tag{15}$$

with $e_1 \in \Sigma_{\mathcal{S}^1}, e_2 \in \Sigma_{\mathcal{S}^2}, q_1 \in \Phi_{\mathcal{S}^1}, q_2 \in \Phi_{\mathcal{S}^2}, q_3 \in \Phi_{\mathcal{S}^3}$, and the automata containing e_1 and e_2 are isomorphic, and the automata containing q_1 and q_2 are isomorphic, then only one of the two requirements needs to be validated.

Isomorphic systems are equivalent in behavior. If, according to the systems engineer, these systems are required to have the same behavior, validation time can be reduced, since only the behavior of one of the systems needs to be validated. Isomorphic systems that, according to the systems engineer, should have different behavior may indicate modeling errors in the plant or requirements.

Determining if a system contains isomorphic subsystems can both be an intuitive task and a difficult task, depending on the state size and complexity of the component models and requirement models in these subsystems. In this paper, the component models and requirement models have a sufficiently small number of states and events so that this isomorphism can be determined manually. Furthermore, most models are created using templates so isomorphism can be verified even more easily, since component models that are instantiated from the same template tend to be isomorphic by construction.

Manuscript 32 submitted to 15th IFAC Workshop on Discrete Event Systems.
Received December 17, 2019.

4. CASE STUDY

4.1 System description

Road tunnels are integrated in road networks to overcome infrastructural challenges such as bypassing geological obstacles (e.g., rivers or mountain ranges) or improving traffic flow in dense urban areas. In Bouwmeester (2018), a tunnel is defined as "an enclosed part of the road, separated from the surrounding environment, with the aim of crossing other infrastructure, often waterways, and/or increasing the quality of life of the surrounding area." In this case study, a supervisory controller is created for the Koning Willem-Alexandertunnel (KWA-tunnel), shown in Fig. 4. The KWA-tunnel is located in Maastricht, the Netherlands. It is the first two-layered road tunnel in the Netherlands, and has two traffic tubes in each layer.

Fig. 4. Northern entrance of the KWA-tunnel.

The complete tunnel system consists of two sets of two traffic tubes, and one safe space between the two tubes in both sets. Within nominal behavior, the tunnel supervisory controller only monitors the sensors in the tunnel and does not intervene by driving the actuators. After an emergency detection a co-operation of multiple tunnel subsystems is required to correctly handle the emergency. In this phase, the supervisory controller both monitors the sensors and drives the actuators. The process of the tunnel behavior during an emergency is shown in Fig. 5.

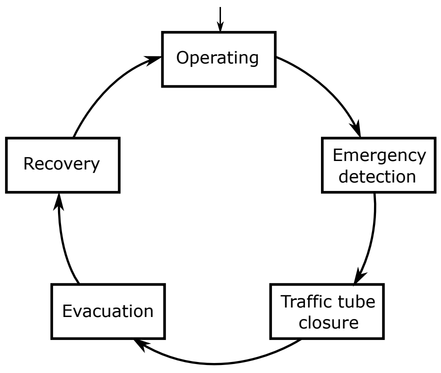

Fig. 5. Tunnel process during an emergency.

In the operating phase, the lighting and ventilation are active, and emergency detection systems survey the current traffic tube state. These emergency detection systems include sensors to detect standstills, smoke, and opening of emergency cabinets. When an emergency is detected, the traffic tube is closed using boom barriers, traffic lights, and matrix signs. During the evacuation phase, escape route indication is activated, emergency doors are opened, and the lighting and air pressure in the safe space are activated and regulated. The final phase is the recovery phase, during which all previously mentioned subsystems are manually reset to the initial settings.

The tunnel is controlled remotely by a tunnel operator. This operator monitors the current tunnel state using a man-machine-interface, as well as closed circuit television images. The tunnel is usually in automatic control mode, though the tunnel operator is always able to take action by switching the control mode of a specific system to a manual mode.

4.2 Modeling

In this section, several examples are given of component models and requirement models in the KWA-tunnel model, to show how isomorphism is determined intuitively for automata and state-based requirements. Specifically, models of the traffic tube barrier (TTB) are shown in this paper. The TTB is a subsystem of the tunnel with the function of closing a traffic tube, e.g., during an emergency. Fig. 6 shows an overview of the most relevant TTB components, which are the traffic lights and the boom barriers.

The component models described in this section are the mode of the traffic light, the movement of both boom barriers, the position of both boom barriers, and the obstacle detection sensor of both boom barriers. When modeling the components, all physically possible behavior is considered. The initial states of all automata are the states during normal operation, and the marked states are the states of a fully opened traffic tube. A traffic tube is fully opened when both boom barriers are open and the traffic light mode is off.

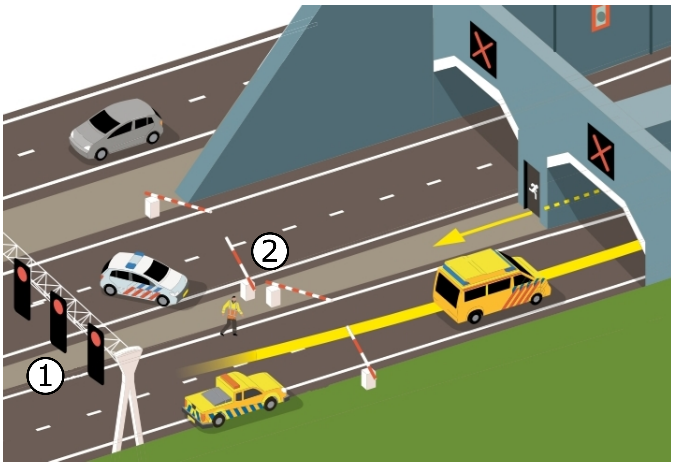

Fig. 6. Overview of traffic lights (1) and boom barriers (2).

Modeling the traffic light The traffic light that is used to close the traffic tube has a yellow and a red aspect. It has four possible modes being `off`, `flashing`, `yellow`, and `red`. The automaton for this traffic light is shown in Fig. 7. The possible transitions between the four modes are predefined and incorporated as such in this automaton.

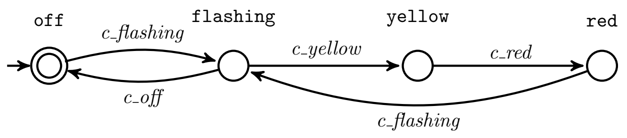

Fig. 7. The mode of the traffic light.

Modeling the boom barriers The boom barrier model consists of two actuator models, regulating the movement of the boom barrier, and two sensors, detecting the position of the boom barrier. The physical relation of both the movement (the barrier cannot move up and down simultaneously) and the position (the barrier cannot be open and closed simultaneously) are incorporated in these models. Each event name in the component models of a boom barrier is appended with *_i*, with $i \in \{1, 2\}$ to distinguish the events of the first and the second boom barrier. The automaton displayed in Fig. 8 shows the movement model of a boom barrier.

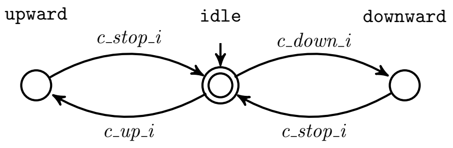

Fig. 8. The movement of a boom barrier.

Fig. 9 shows the automaton for indicating the position of a boom barrier, which is the combination of the lower and upper position sensor. The transitions in this automaton are uncontrollable, since it concerns sensor detections.

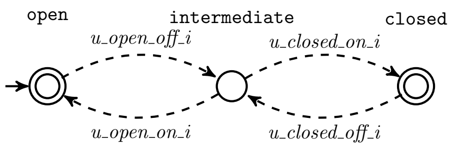

Fig. 9. The position of a boom barrier.

A third sensor is also part of the boom barrier, which detects if an obstacle is present below the boom barrier. The automaton for this sensor is shown in Fig. 10.

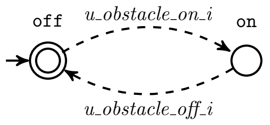

Fig. 10. The obstacle detection sensor at a boom barrier.

When looking at the component models of both boom barriers, one can see that the component model of the movement of boom barrier 1 is isomorphic to the component model of the movement of boom barrier 2. Furthermore, the component models of the position of the boom barriers are isomorphic, as well as the component models of the obstacle detection sensors. It is therefore determined that boom barrier 1 is isomorphic to boom barrier 2. The possible behavior of the second boom barrier is thus equivalent to the possible behavior of the first boom barrier. Following Principle 1 of Section 3, one can establish that the component models of only one of the boom barriers need to be validated.

Modeling the requirements Several requirements related to the traffic light and the two boom barriers are described here, and the principles from Section 3 are used to determine which requirements do not need to be validated.

Following Principle 2 in Section 3, only one requirement of the requirement pairs listed below needs to be validated:

(1) A boom barrier may only move up if the boom barrier position is not `open`.

c_up_1 **needs** ¬`boombarrier_1_position.open`

c_up_2 **needs** ¬`boombarrier_2_position.open`

(2) A boom barrier may only move down if the boom barrier position is not `closed`.

c_down_1 **needs** ¬`boombarrier_1_position.closed`

c_down_2 **needs** ¬`boombarrier_2_position.closed`

(3) A boom barrier may only move down if the obstacle detection is `off`.

c_down_1 **needs** `obstacle_detection_1.off`

c_down_2 **needs** `obstacle_detection_2.off`

Following Principle 4 in Section 3, only one requirement of the requirement pairs listed below needs to be validated:

(1) A boom barrier may not move down if the traffic light mode is not `red`.

¬`traffic_light.red` **disables** *c_down_1*

¬`traffic_light.red` **disables** *c_down_2*

(2) The traffic light may not turn off if a boom barrier position is not `open`.

¬`boombarrier_1_position.open` **disables** *c_off*

¬`boombarrier_2_position.open` **disables** *c_off*

(3) A boom barrier may only stop if the boom barrier movement is `downward` and either the obstacle detection is `on` or the traffic light mode is not `red`.

c_stop_1 **needs** `boombarrier_1_movement.downward` $\wedge($`obstacle_detection_1.on` $\vee$ ¬`trafficlight.red`$)$

c_stop_2 **needs** `boombarrier_2_movement.downward` $\wedge($`obstacle_detection_2.on` $\vee$ ¬`trafficlight.red`$)$

4.3 Results

A supervisory controller has been synthesized for three different models of the KWA-tunnel: for one, two, and four traffic tubes. Monolithic synthesis proved to be impossible for these models due to memory issues, so multilevel synthesis has been used. Table 1 shows results of the synthesis for all three models, including plant models, requirement models, number of supervisors, and state-space sizes.

Table 1. Results for the one, two, and four tube KWA-tunnel model.

	One tube	Two tubes	Four tubes
Component models	184	404	808
Requirement models	360	756	1482
Supervisors	33	70	139
Uncontrolled state-space	$3.5 \cdot 10^{51}$	$1.4 \cdot 10^{111}$	$2.0 \cdot 10^{222}$
Controlled state-space	$3.0 \cdot 10^{21}$	$5.0 \cdot 10^{26}$	$8.9 \cdot 10^{31}$

The validation of the supervisory controllers has been performed through simulations. A hybrid model is created to incorporate the continuous time behavior, such as the movement of the boom barriers. Furthermore, a visualization of the system has been created, as shown in Fig. 11 for the two-tube model. The upper part of the visualization represents the system in its current state, while the man-machine interface can be seen in the lower part of the visualization.

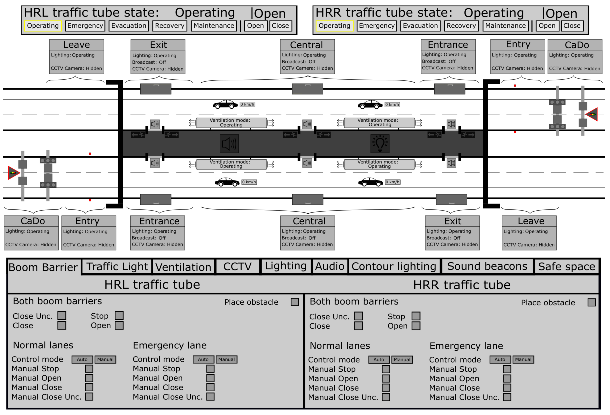

Fig. 11. Visualization of the two-tube model.

Symmetry reduction has been used to validate the models of the KWA-tunnel more efficiently. To give an indication of the effectiveness of symmetry reduction, Table 2 shows the number of component models and requirement models that did not need to be validated as well as the total number of component models and requirement models. The table also shows how many requirements did not need to be validated for each principle from Section 3.

Table 2. Number of component and requirement models that do not need to be validated per KWA-tunnel model.

	One tube	Two tubes	Four tubes
Components:	184	404	808
Principle 1	164	380	784
Requirements:	360	756	1482
Principle 2	58	416	1136
Principle 3	15	19	27
Principle 4	167	171	169
Principle 5	0	0	0
Total	240	606	1332

As can be concluded from Table 2, symmetry reduction is very effective for validating the component models of all three cases. This is mainly due to the fact that most components can be instantiated from a small set of templates, such as actuators, sensors, and buttons. The effectiveness of symmetry reduction increases with the number of component models. The highest effectiveness is seen in the four-tube model, where 784 of the 808 component models ($> 95\%$) do not need to be validated.

When looking at the results for the requirement models, symmetry reduction shows a decent effectiveness in the one-tube model, as 240 of the 360 requirements do not need to be validated. This is mostly due to Principle 4. Furthermore, the effectiveness of symmetry reduction increases for the validation of the two- and four-tube models. As can be seen for these models, almost all requirements that do not need to be validated follow Principle 2, since these requirements are isomorphic to the requirements of one traffic tube. The two-tube model introduces several new requirements, related to safe space and inter-tube behavior, that do need to be validated. The resulting difference is that 150 requirements need to be validated in the two-tube model, compared to 120 in the one-tube model. The four-tube model introduces no new requirements compared to the two-tube model. Several requirements are, however, adapted to link all four traffic tubes. For the four-tube model 150 of the 1482 requirements need to be validated.

It turns out that none of the models in this case study contain requirements that follow Principle 5. However, to show that there could exist requirements that follow Principle 5 in a such a model, two requirements from the two-tube models are adapted and shown below.

$\texttt{tube\_1}.\mathit{c\_support}$ **needs** $\texttt{tube\_2.emergency} \vee \texttt{safe\_space.emergency}$

$\texttt{tube\_2}.\mathit{c\_support}$ **needs** $\texttt{tube\_1.emergency} \vee \texttt{safe\_space.emergency}$

In these adapted requirements, an emergency can occur either in a traffic tube or in the safe space, resulting in one or two supporting traffic tubes to handle this emergency.

5. CONCLUDING REMARKS

This paper introduces the method of symmetry reduction for supervisory controller behavior that can be used to more efficiently validate models that contain isomorphic component models and requirement models. In the KWA-tunnel case study, component models and requirement models are created for one, two, and four traffic tubes. Multilevel synthesis was used to synthesize supervisory controllers. Symmetry reduction was used to efficiently validate these models, thus reducing the total validation time and effort. The effectiveness of symmetry reduction has been expressed by the number of component models and requirement models that did not need to be validated compared to the total number of models.

In this case, isomorphism has been determined manually for both the component models and the requirement models, as this was an easy task due to the relatively small component models and the use of template instantiations. However, addition of automatic detection of isomorphism can further reduce validation time, and remove the possibility of manual mistakes. Algorithms for isomorphism detection have been investigated in previous work as Mikolajczak (1991), and could be used as a basis for the implementation of an algorithm to determine isomorphism.

In this paper, symmetry reduction is used to more efficiently validate the component models and requirement models of a synthesized supervisory controller. Future work on symmetry reduction includes reducing the state-space size for the purpose of more easily synthesizing a supervisory controller. In this application, when a system is symmetrical in behavior, a supervisor can be synthesized for part of the system and translated to the other part of the system since the behavior is equivalent. Previous work on exploiting symmetry for the purpose of synthesis is shown in Eyzell and Cury (2001). In contrast to this work,

Manuscript 32 submitted to 15th IFAC Workshop on Discrete Event Systems.
Received December 17, 2019.

our notion of symmetry is based on isomorphism between automata and state-based requirements, as opposed to a language-based equivalence.

ACKNOWLEDGEMENTS

The authors thank Rijkswaterstaat for funding this research. Specifically, we would like to show our gratitude to Han Vogel from Rijkswaterstaat and Pascal Etman from the University of Technology Eindhoven for their interest and assistance in this project.

REFERENCES

Albers, A., Scherer, H., Bursac, N., and Rachenkova, G. (2015). Model based systems engineering in construction kit development: Two case studies. *Procedia CIRP*, 36, 129–134.

Baier, C. and Katoen, J.P. (2008). *Principles of model checking*. MIT press.

Bouwmeester, F. (2018). Landelijke tunnelstandaard. URL `http://publicaties.minienm.nl/documenten/landelijke-tunnelstandaard`.

Braspenning, N.C.W.M., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2006). A model-based integration and testing method to reduce system development effort. *Electronic Notes in Theoretical Computer Science*, 164(4), 13–28.

Cassandras, C.G. and Lafortune, S. (2009). *Introduction to discrete event systems*. Springer Science & Business Media.

Eyzell, J.M. and Cury, J.E.R. (2001). Exploiting Symmetry in the Synthesis of Supervisors for Discrete Event Systems. *IEEE Transactions on Automatic Control*, 46(9), 1500–1505.

Glushkov, V.M. (1961). The abstract theory of automata. *Russian Mathematical Surveys*, 16(5), 1–53.

Goorden, M., van de Mortel-Fronczak, J.M., Reniers, M.A., and Rooda, J.E. (2017). Structuring multilevel discrete-event systems with dependency structure matrices. In *Proceedings of the 56th IEE Annual Conference on Decision and Control*, 558–564. IEEE.

Grigorov, L., Butler, B.E., Cury, J.E.R., and Rudie, K. (2011). Conceptual design of discrete-event systems using templates. *Discrete Event Dynamic Systems*, 21(2), 257–303.

Hu, L., Xie, N., Kuang, Z., and Zhao, K. (2012). Review of cyber-physical system architecture. In *Proceedings of the 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 25–30. IEEE.

Huber, P., Jensen, A.M., Jepsen, L.O., and Jensen, K. (1985). Towards reachability trees for high-level petri nets. In *Advances in Petri Nets 1984*, 215–233. Springer.

Ip, C.N. and Dill, D.L. (1996). Better verification through symmetry. *Formal methods in system design*, 9(1-2), 41–75.

Komenda, J., Masopust, T., and van Schuppen, J.H. (2016). Control of an engineering-structured multilevel discrete-event system. In *Proceedings of the 13th International Workshop on Discrete Event Systems*, 103–108. IEEE.

Kotonya, G. and Sommerville, I. (1998). *Requirements engineering: processes and techniques*. Wiley Publishing.

Liang, F., Schamai, W., Rogovchenko, O., Sadeghi, S., Nyberg, M., and Fritzson, P. (2012). Model-based requirement verification: A case study. In *Proceedings of the 9th International MODELICA Conference*, 385–392. Linköping University Electronic Press.

Lin, F.J., Chu, P.M., and Liu, M.T. (1987). Protocol verification using reachability analysis: The state space explosion problem and relief strategies. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology, SIGCOMM 1987*, 126–135. Association for Computing Machinery.

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

Mealy, G.H. (1955). A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5), 1045–1079.

Melnyk, A. (2016). Cyber-physical systems multilayer platform and research framework. *Advances in cyber-physical systems*, (1), 1–6.

Mikolajczak, B. (1991). Automata homomorphisms. In *Algebraic and Structural Automata Theory*, volume 44 of *Annals of Discrete Mathematics*, 155–196. Elsevier.

Miller, A., Donaldson, A., and Calder, M. (2006). Symmetry in temporal logic model checking. *ACM Computing Surveys*, 38(3), 2–36.

Ouedraogo, L., Kumar, R., Malik, R., and Åkesson, K. (2011). Nonblocking and safe control of discrete-event systems modeled as extended finite automata. *IEEE Transactions on Automation Science and Engineering*, 8(3), 560–569.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1), 206–230.

Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., Reniers, M.A., and Rooda, J.E. (2018). Application of dependency structure matrices and multilevel synthesis to a production line. In *Proceedings of the IEEE Conference on Control Technology and Applications*, 458–464. IEEE.

Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2017). Supervisory control synthesis for a waterway lock. In *Proceedings of the 1st IEEE Conference on Control Technology and Applications*, 1562–1563. IEEE.

Schmidt, K. (2000). Integrating Low Level Symmetries into Reachability Analysis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 315–330. Springer.

Scott, W., Fullalove, R., Arabian, G., and Campbell, P. (2016). Case study: A model based systems engineering (MBSE) framework for characterising transportation systems over the full life cycle. *INCOSE International Symposium*, 26(1), 916–932.

Theunissen, R.J.M., Petreczky, M., Schiffelers, R.R.H., van Beek, D.A., and Rooda, J.E. (2014). Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. *IEEE Transactions on Automation Science and Engineering*, 11(1), 20–32.

Manuscript 32 submitted to 15th IFAC Workshop on Discrete Event Systems.
Received December 17, 2019.