Why is my supervisor empty?

Finding causes for the unreachability of states in synthesized supervisors

L. Swartjes, M.A. Reniers, D.A. van Beek, and W.J. Fokkink
Control Systems Technology
Department of Mechanical Engineering
Eindhoven University of Technology

***Abstract*—Although supervisory control synthesis has been around for many years, adoption is still low. A weak point of synthesis is the absence of a reporting mechanism. When an empty or unexpected supervisor is returned, it is very difficult to explain why this is the case. It is desired to return an explanation for a question, like, "Why is my supervisor empty?". In general, the information needed to provide such an explanation is not present in the synthesized result.**

In this paper, causes (explanations) are generated for questions regarding the absence of behavior in the synthesized system. To this end, it is first investigated what information is needed and how it should be stored. Based on these findings, information of the influence of each requirement is encoded in the supervisor. This is done by annotating colors. The resulting so-called colored predicates can be used after synthesis to derive a cause for a given question.

I. Introduction

The concept of supervisory control synthesis was introduced in the 1980's in [1]. Based on a model of the uncontrolled behavior of the system, i.e., the plant, and models of the specification, i.e., the requirements, a controller is synthesized. This controller abides by the requirements and several other properties, notably, the absence of deadlocks.

Many real-life examples have been presented that show the applicability of supervisory control synthesis, e.g., [2], [3], [4], [5]. Moreover, work has been done on the correct implementation of synthesized supervisors on control platforms, e.g., [6]. In addition, new algorithms have been introduced that reduce computation time and introduce new features, e.g, [7].

A weak point of synthesis is a lack of a reporting mechanism. During the design phase of the system, or especially the controller of the system, the synthesis process often yields an empty or unsatisfactory supervisor. Typically, an empty supervisor does not provide any information on the cause of emptiness. For example, none of the commonly used tools for synthesis, e.g., [8], [9], provide reasons when synthesis fails. A designer, faced with an undesirable supervisor, would like to know why this supervisor is as it is (with respect to both the given plant and the requirements); an explanation or *cause* should be returned, based on a *query*, i.e., a specific question regarding why the supervisor is unsatisfactory or empty. Commonly asked questions as "Why is state s unreachable?", or "Why is event e blocked in location l?" are related to the concept of a query. Then, given a query, a cause provides those requirements and states of the system responsible for the unreachability of the query state. In other words, a cause points to requirements and/or states that, when relaxed, allow the query state to be reached. Currently such causes cannot be provided, due to the lack of sufficient information during synthesis.

The contributions of this paper are threefold. Our first contribution is the *formal definition of a cause*, which allows us to provide a reporting mechanism for synthesized controllers. Our second contribution is a *method to generate a cause*, based on the synthesized result. To this end, more information is needed than provided from the synthesized result. Hence, our third contribution is the *addition of information to the synthesized system* such that causes can be provided. Combining these contributions, this paper provides a method to provide causes for those questions that cannot be answered at the moment, like "Why is my supervisor empty?".

To the authors' knowledge, no verification methods are presently available that provide causes regarding synthesis. To aid the verification of the synthesis result, model-based verification tools can provide some insights. For instance, [10], [11], and [12] provide methods to verify specific properties after the supervisor has been created. With these methods, safety and liveness properties can be verified. Nevertheless, generated counterexamples do not necessarily provide useful information in the scope of synthesis. The methods of [13] and [14], can provide more elaborate results by combining the counterexamples with reachability information. This allows for more elaborate results. However, a question like "Why is my supervisor empty?" cannot be answered by the mentioned methods.

II. Preliminaries

In this section, the preliminary concepts like an automaton, a path, and supervisory control synthesis are introduced. Let t and f denote the boolean values true and false, respectively.

A. Automata

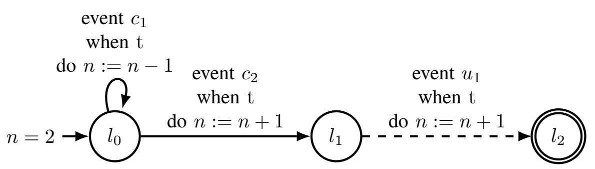


Fig. 1. Example of an automaton. Location l_0 is the initial location, with an initial condition $n = 2$. Location l_2 is a marked location. Variable n is an integer variable contained in the domain $[0, 4]$.

Automata are finite state machines, used to model different kinds of systems or processes. The automata used in this paper are based on Extended Finite Automata (EFA), as introduced in [15]. An automaton consists of a number of locations that model the different modes of a system. Data is modeled by means of variables. At each moment in time, each variable has a value. The mapping between variables and their values is called a valuation.

An automaton has an initial location and an initial valuation. In addition, an automaton has marked locations to denote acceptance; an automaton may reside indefinitely long in a marked location.

The transitions between locations are modeled by edges. An edge contains an event (or action), a guard, and an update. An event is used to distinguish the different transitions. An event can either be controllable or uncontrollable. A guard specifies the condition which must be satisfied before the transition can occur. An update specifies the change of the value of variables.

In Figure 1, an automaton is depicted that has three locations (l_0, l_1, and l_2), three edges, one variable (n), and one marked location (l_2). The initial location is denoted by the short arrow pointing inwards, and the marked location is denoted by means of a double enclosed circle. Edges containing uncontrollable events are denoted by a dashed line. All other edges are denoted by solid lines.

Formally, an automaton is defined as a septuple:

$$A = \langle L, X, \Sigma, \Delta, l_0, v_0, L_m \rangle$$

Herein, L is the set of locations, X the set of variables, Σ the set of events, Δ the set of edges, $l_0 \in L$ the initial location, v_0 the initial valuation, and $L_m \subseteq L$ the set of marked locations.

The edges are defined as follows:

$$\Delta \subseteq L \times G \times \Sigma \times U \times L$$

Herein, G is the universe of predicates over variables of the set X. Additionally, U is the set of assignment expressions over variables of the set X. Assignments are of the form $x := a$, where a is an arithmetic expression that may contain variables. The assignment states that the value of x in the next state becomes the current value associated to a.

B. Transition System

Given an automaton, all possible transitions are encoded in a transition system. A state of the system is expressed by the combination of a location and a valuation. A transition from one state to another is possible if there is an edge that models this transition and the valuation satisfies the guard.

Formally, a transition system associated with an automaton is defined as:

$$T = (S, \rightarrow, s_0)$$

with S the set of states, $\rightarrow \subseteq S \times \Sigma \times S$ the transition relation, and s_0 the initial state.

The transition relation is defined by means of the following rule:

$$\frac{(l, g, \sigma, u, l') \in \Delta, \quad v \models g, \quad v \cup v' \models u}{(l, v) \xrightarrow{\sigma} (l', v')}$$

Herein, $v \models g$ is used to denote that guard g is valid under valuation v, and $v \cup v' \models u$ is used to denote that v' is a correctly updated valuation w.r.t. valuation v and update u.

A path to state $s \in S$ is given as follows:

$$s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} s_2 \xrightarrow{\sigma_3} \ldots \longrightarrow s_n = s$$

Here, $(s_i, \sigma_{i+1}, s_{i+1}) \in \rightarrow$ for every $0 \leq i < n$.

C. Synthesis

Given a plant, i.e., a model of the uncontrolled behavior of the system, and a set of requirements, i.e., predicates over the states of the system that model the specifications, synthesis is the process of creating a supervised system or supervisor.

During synthesis, the guards of edges from the plant are altered in an effort to meet the specification. Only guards of edges with controllable events can be changed. The synthesis can fail, resulting in an empty supervisor, if no guards can be found that meet the requirements and the conditions of the plant. Or, synthesis fails when no marked location can be reached in the supervised system.

More details can be found in [1] and [7]. The synthesis algorithm from [7] will be sketched in more detail, while explaining our approach.

III. Problem Definition

In order to return causes, the goal is to determine which requirements are violated. Additionally, it is key to determine when these violations occur. Hence, the violation of requirements is expressed along a path using colors to distinguish the requirements. To this end, each requirement is assigned a unique color.

In a certain state, a violation of a requirement occurs when the state does not satisfy the specification. To derive a cause, all variables involved in the violated requirement are colored with the color associated with the requirement. To allow this coloring, the concept of an annotation is defined; variables are augmented with colors. As we must retrieve which variables are colored with which colors, an annotation is, for each state, a mapping from colors to a set of variables. Formally, an annotation is defined as:

Definition 1 (Annotation)**.** *Given the set of colors, C, the set of states, S, and the set of variables, X, an annotation $\alpha \in S \rightarrow (C \rightarrow 2^X)$ contains for each state a mapping from colors to a set of variables.*

The context in which we would like to retrieve causes is determined by a query. As stated in the introduction, a query is related to a question over the supervised system. A query can be seen as a verification property. Formally, a query is defined as:

Definition 2 (Query)**.** *Given a location $l \in L$ of the plant and a predicate $p \in \mathcal{P}$, a query is defined as (l, p)?.*

where $\mathcal{P}$ is the universe of predicates.

Based on a query, an annotated path of the plant can be returned. An annotated path is a path with an annotation, starting from the initial state of the plant.

Definition 3 (Annotated path)**.** *Given a path π and an annotation α, an annotated path is a tuple (π, α) containing the path and the annotation.*

For the sake of simplicity, an annotated path is depicted as follows:

$$(s_0, \alpha_0) \xrightarrow{\sigma_1} (s_1, \alpha_1) \xrightarrow{\sigma_2} (s_2, \alpha_2) \xrightarrow{\sigma_3} \dots$$

where $\alpha_i = \alpha(s_i)$ for every $i \geq 0$, and $s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \dots$ is a path.

Such an annotated path is a cause if the final state of the path is a state satisfying the query. A state satisfies the query if, given a query $q = (l, p)$ and a state $s = (l', v)$, $l = l'$ and $v \models p$. Moreover, an annotated path is a cause if the introduction of colors along the path is associated with the (future) violation of requirements. Note the fact that a variable is annotated in a certain state does not imply the direct violation of a requirement. Due to the nature of the synthesis algorithm, conditions propagate backwards. Hence, annotations also propagate backwards. Additionally, a path does not contain the complete dynamics of the system. Therefore, the introduction of a color does not necessarily imply the violation of a requirement along that path.

Note that a cause is not directly comparable to a counter-example; a cause is a path of the system that contains information for deriving "counter-examples".

Definition 4 (Cause)**.** *Given a query q and a state s_q that satisfies the query, a cause is an annotated path (π, α) leading to s_q with sound annotations. An annotation is sound if and only if the coloring of each variable in every state of that path is sound.*

The coloring of a variable x with a color c in a state s is sound, if and only if:

- *The requirement r associated with c is a state expression over x, and*
- *One of the following statements holds:*
  1) *The current state s violates the requirement r, i.e., $s \not\models r$.*
  2) *The succeeding state s' of state s in the path π has a sound coloring for x with color c.*
  3) *There exists a path π' going through state s, for which the succeeding state s'' has a sound coloring for x with color c in state s''.*

The formal problem is defined as follows: Given a plant, p, a set of requirements, $R = \{r_1, \dots r_n\}$, and a query, q, give at least one cause if the query is unsatisfiable.

IV. Approach

This section highlights the approach of creating the information for reporting causes. In Figure 2 the overall approach is depicted. The synthesis is done according to [7]. Refinement is also a process from [7], though we will introduce colors during this process.

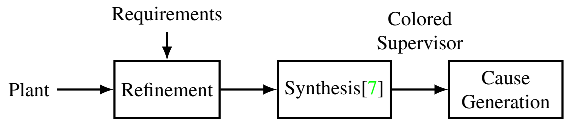


Fig. 2. Synthesis process.

Colors are introduced during refinement; refinement encodes in the plant the forbidden behavior w.r.t. the requirements. In this encoding, the color of each requirement is added to the predicates. The actual refinement process is not influenced. More details are given in Section V.

Due to the backward propagating nature of the synthesis algorithm, information of blocking requirements is propagated towards the initial states. Hence, if colors are associated with predicates, the needed information for the generation of causes is created during synthesis. This is shown in Section VI.

As the synthesis process now yields a colored supervisor, causes are explicitly encoded in the supervisor. Hence, Section VII will explain how to retrieve the conditions on variables causing the unsatisfiability of a specific requirement based on the current state and the synthesized predicates.

V. Refinement

In order to apply the synthesis process of [7], the plant automaton must be refined with the requirements.

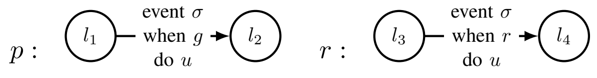


Fig. 3. Part of a plant p and a requirement r on the event σ.

In Figure 3, a plant p and a requirement automaton r are depicted. In the refinement process, as defined in [7], transitions that do not comply to the requirement lead to a twin-state of the original location, that models the violation of a requirement. These twin-locations are called forbidden locations. Note that the twin-state of a marked location is not marked. In the case of Figure 3, the requirement is satisfied if the transition labeled σ only occurs if g and r are satisfied. Between two forbidden locations, the original plant behavior is duplicated.

The result of refinement is depicted in Figure 4. Herein, the locations labeled with ϕ are the forbidden locations. The forbidden transition, when g or r is not satisfied, leads to a forbidden location (the twin-state of location l_2). The gray edge, i.e., the edge at the bottom, shows the duplication of the plant behavior between forbidden locations.

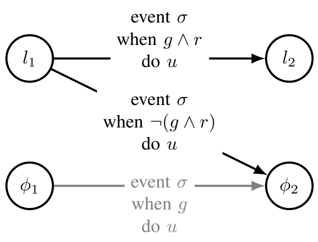


Fig. 4. Resulting refinement of Figure 3.

Refinement strengthens the guard of the plant with the guard of the requirement. This can be observed in Figure 4, as the resulting guard becomes $g \wedge r$. In our version of refinement, the guard of the requirement, r, is colored with the color associated with the requirement. An example is: $g \wedge r^{\blacktriangle}$. For the transition leading to the undesired behavior, i.e., the transition with guard $\neg(g \wedge r)$ in Figure 4, the color is also added to r. This results in: $\neg(g \wedge r^{\blacktriangle})$.

VI. Generating Causes

In this section, the approach of generating causes is discussed. To generate causes, the information on the requirements is encoded explicitly in the synthesis result. To this end, the concept of an annotated automaton is introduced in this section. Next, based on this annotated automaton the causes are generated; based on the information present in the annotated automaton, an annotation must be found for a path given a query.

It is clear from the problem definition that the individual influence of each requirement needs to be determined. As stated before, each requirement is assigned a unique color. Therefore, to store information about the requirements, it is possible to color the parts of the plant influenced by the synthesis.

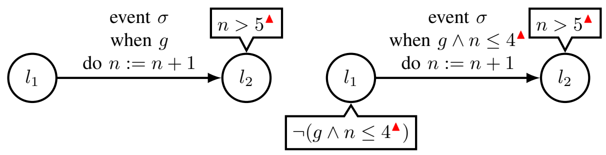


Fig. 5. The concept of storing information in the supervised system. The callouts represent the predicates describing the blocking situation.

As an example, consider Figure 5, where the same part of an automaton is depicted twice. In the left part, the original plant is depicted. In this figure, the balloon contains the predicate that specifies the blocking condition. This condition is implied by a requirement that specifies that in location l_2 the condition $n \leq 5$ must hold. The color of the requirement is annotated to the predicate; the ▲ requirement is responsible for the blocking condition. In the right part, the envisioned supervised system is depicted. Here, the guard of the plant is altered to prevent the blocking situation. To denote that this alteration is due to the ▲ requirement, the guard is annotated. As the guard is strengthened such that the blocking condition does not occur in location l_2, a new blocking situation arises for location l_1. This is reflected in the predicate describing the blocking situation of l_1.

Suppose the following query is given: $q = (l_2, \mathrm{t})?$. This query questions whether location l_2 is reachable for any valuation. A possible path, leading to a state satisfying the query is the following, assuming $(l_1, \{n \mapsto 8\})$ is the initial state and $g = \mathrm{t}$:

$$(l_1, \{n \mapsto 8\}) \xrightarrow{\sigma} (l_2, \{n \mapsto 9\})$$

To give a cause for this query, an annotated path must be created that is sound. Without much effort, the following cause can be determined:

$$((l_1, \{n \mapsto 8\}), \{\blacktriangle \mapsto \{n\}\}) \xrightarrow{\sigma} ((l_2, \{n \mapsto 9\}), \{\blacktriangle \mapsto \{n\}\})$$

It is easy to see that this is a cause for the query; the value of n violates the ▲ requirement in l_2. Moreover, due to propagation, the ▲ requirement also arises in l_1, as with the current violation the requirement can never be satisfied. This can be determined easily, by considering the predicates describing the blocking situation. Note that this underlines the fact that the presence of a color and the actual violation of the requirement can occur at different states.

In Figure 6, an uncontrollable event is introduced. For the sake of simplicity, it is assumed that the plant has no guards. The requirement now implies that the condition $n \leq 5$ must hold in location l_3. The upper part is the

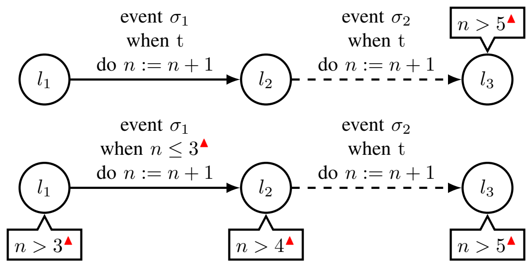


Fig. 6. The concept of propagation of information over uncontrollable events.

original plant, while the lower part is the envisioned supervised system. As the guard of an uncontrollable edge cannot be altered, the blocking condition is propagated until a controllable edge is encountered.

If the query $q = (l_3, \mathrm{t})?$ is considered, the following cause can be constructed:

$$((l_1, \{n \mapsto 8\}), \{\blacktriangle \mapsto \{n\}\}) \xrightarrow{\sigma_1} ((l_2, \{n \mapsto 9\}), \{\blacktriangle \mapsto \{n\}\}) \xrightarrow{\sigma_2} ((l_3, \{n \mapsto 10\}), \{\blacktriangle \mapsto \{n\}\})$$

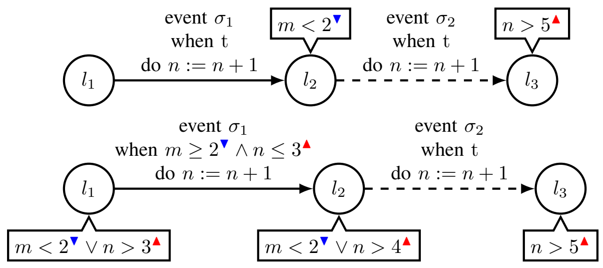


Fig. 7. The concept of merging information of multiple requirements.

In Figure 7, a new requirement is introduced; the $\blacktriangledown$ requirement states that condition $m \geq 2$ must hold in location l_2. Due to propagation over uncontrollable edges, the blocking condition of l_2 is the disjunction of both requirements.

Again, the query $q = (l_3, \mathrm{t})?$ is considered. It is clear, that the valuation of both m and n has to be considered. However, the valuations can be considered separately; if either the value m or the value of n violates the guard, the transition is blocked. To this end, let us only consider the transition labeled σ_1. If both m and n comply, no coloring is applied. If m does not comply, e.g., $m = 1$, while n does comply, only variable m must be colored; only the $\blacktriangledown$ requirement causes a blocking situation. Vice versa, if only n does not comply, e.g., $n = 5$, only n must be colored. However, if both do not comply, both must be colored:

$$\ldots \longrightarrow ((l_1, \{m \mapsto 1, n \mapsto 5\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m\}\}) \xrightarrow{\sigma_1} ((l_2, \{m \mapsto 1, n \mapsto 6\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m\}\}) \xrightarrow{\sigma_2} \ldots$$

If a predicate is a function of multiple variables, e.g., $m < n$, then all variables comprising this predicate will

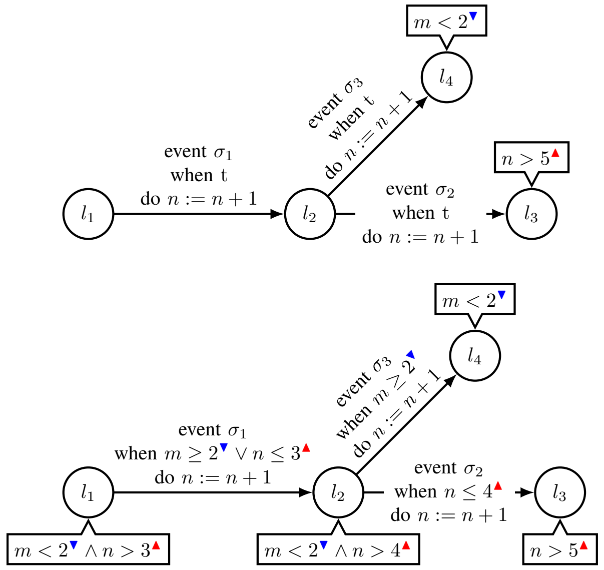

$m < 2^{\blacktriangledown}$
l_4
event σ_3
when t
do $n := n+1$
$n > 5^{\blacktriangle}$
event σ_1
when t
do $n := n+1$
l_1
l_2
event σ_2
when t
do $n := n+1$
l_3
$m < 2^{\blacktriangledown}$
l_4
event σ_3
when $m \geq 2^{\blacktriangledown}$
do $n := n+1$
event σ_1
when $m \geq 2^{\blacktriangledown} \vee n \leq 3^{\blacktriangle}$
do $n := n+1$
event σ_2
when $n \leq 4^{\blacktriangle}$
do $n := n+1$
l_1
l_2
l_3
$m < 2^{\blacktriangledown} \wedge n > 3^{\blacktriangle}$
$m < 2^{\blacktriangledown} \wedge n > 4^{\blacktriangle}$
$n > 5^{\blacktriangle}$

Fig. 8. The concept of merging information of multiple requirements along different paths.

be colored accordingly. Suppose the predicate of the blocking situation in l_2 is $m < n^{\blacktriangledown} \vee n > 4^{\blacktriangle}$. For $m = 1$ and $n = 5$, the following cause is obtained:

$$\ldots \longrightarrow ((l_1, \{m \mapsto 1, n \mapsto 5\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m, n\}\}) \xrightarrow{\sigma_1} ((l_2, \{m \mapsto 1, n \mapsto 6\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m, n\}\}) \xrightarrow{\sigma_2} \ldots$$

It is clear that n is bicolored; the value of n fails to satisfy the $\blacktriangle$ requirement and (together with the value of m) fails to satisfy the $\blacktriangledown$ requirement.

Finally, the case with a conjunction in the predicate describing the blocking situation is depicted in Figure 8. If at least one part of the predicate is satisfiable, no blocking situation occurs. Only if m and n both violate the $\blacktriangle$ and $\blacktriangledown$ requirements, then all transitions are blocked. Hence, annotation of m and n only occurs if both requirements cannot be satisfied. Note that in this system a choice between transitions is possible. Therefore, the actual violation of a requirement does not have to occur along that path. For instance, consider a cause comprising event σ_2 for the query $q = (l_3, \mathrm{t})?$:

$$\ldots \longrightarrow ((l_1, \{m \mapsto 1, n \mapsto 5\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m\}\}) \xrightarrow{\sigma_1} ((l_2, \{m \mapsto 1, n \mapsto 6\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m\}\}) \xrightarrow{\sigma_2} ((l_3, \{m \mapsto 1, n \mapsto 7\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \varnothing\}) \longrightarrow \ldots$$

In the cause it is indicated that the $\blacktriangledown$ requirement will be violated. However, the actual violation does not occur along this path, as the transition labeled σ_3 is not considered. Nevertheless, this path is a cause, as another cause can be found in which the $\blacktriangledown$ requirement is blocked:

$$\ldots \longrightarrow ((l_1, \{m \mapsto 1, n \mapsto 5\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m\}\}) \xrightarrow{\sigma_1} ((l_2, \{m \mapsto 1, n \mapsto 6\}), \{\blacktriangle \mapsto \{n\}, \blacktriangledown \mapsto \{m\}\}) \xrightarrow{\sigma_3} ((l_4, \{m \mapsto 1, n \mapsto 7\}), \{\blacktriangle \mapsto \varnothing, \blacktriangledown \mapsto \{m\}\}) \longrightarrow \ldots$$

$$\chi(B,s)(c) = \chi(m=3^{\blacktriangledown},s)(c) \cup \begin{cases} \chi(n=3^{\blacktriangledown},s)(c) \cup \chi(m=5^{\blacktriangle},s)(c) & \text{if } s \models (n=3 \wedge m=5) \\ \varnothing & \text{otherwise} \end{cases}$$

Fig. 9. The resulting coloring function for the blocking condition B.

The synthesis process does not have to altered to incorporate these colorings: solely the data structures have to be altered to keep track of the colors. Therefore, colored predicates are introduced.

VII. Colored Predicates

In Section VI it became clear that the blocking predicates hold the information on why a certain state is not reachable. With help of these predicates, the coloring function can be defined. This function maps a predicate and state to the colors of blocking requirements for that state. This coloring will therefore give rise to causes.

In Section VI, the coloring of predicates was introduced. The grammar of these colored predicates is defined in BNF, as follows:

$$\mathcal{CP} ::= 2^{\mathcal{C}} \times \mathcal{P} \mid \neg\mathcal{CP} \mid \mathcal{CP} \vee \mathcal{CP} \mid \mathcal{CP} \wedge \mathcal{CP}$$

Note that the first term, i.e., $2^{\mathcal{C}} \times \mathcal{P}$, will be referred to as a basic colored predicate.

Based on the synthesis process, colors are added to the predicates based on the blocking behavior of requirements. Given a valuation, the colors of requirements can be returned for which the current state satisfies the basic colored predicate, with $c \in C$:

$$\chi((C,p),s)(c) = \begin{cases} X(p) & \text{if } s \models p \\ \varnothing & \text{otherwise} \end{cases}$$

Herein, $X(p)$ is the set of variables used in the predicate and $s \models p$ is used to denote the situation where the state s satisfies the predicate p. Note that colors do not play a role when checking satisfiability of the predicate.

For the negation of a basic colored predicate, the negation can be moved under the color:

$$\chi(\neg(C,p),s)(c) = \chi((C,\neg p),s)(c)$$

For the disjunction of two colored predicates cp_1 and cp_2, the intuition was given in Figure 7; the state must violate at least cp_1 or cp_2 to introduce a color. It was shown that the individual influences can be considered. Hence, for all $cp_1, cp_2 \in \mathcal{CP}$ and $c \in \mathcal{C}$, the coloring is defined as:

$$\chi(cp_1 \vee cp_2, s)(c) = \chi(cp_1,s)(c) \cup \chi(cp_2,s)(c)$$

For the conjunction of colored predicates, the intuition was given in Figure 8; the state must violate both cp_1 and cp_2 to introduce a color. The individual influences can be considered, after both cp_1 and cp_2 are satisfied. For all $cp_1, cp_2 \in \mathcal{CP}$ and $c \in \mathcal{C}$, the coloring for the conjunction is defined as:

$$\chi(cp_1 \wedge cp_2, s)(c) = \begin{cases} \chi(cp_1,s)(c) \cup \chi(cp_2,s)(c) & \text{if } s \models cp_1 \wedge cp_2 \\ \varnothing & \text{otherwise} \end{cases}$$

Finally, the negations are defined, as follows:

$$\begin{aligned} \chi(\neg(\neg cp),s)(c) &= \chi(cp,s)(c) \\ \chi(\neg(cp_1 \vee cp_2),s)(c) &= \chi(\neg cp_1 \wedge \neg cp_2, s)(c) \\ \chi(\neg(cp_1 \wedge cp_2),s)(c) &= \chi(\neg cp_1 \vee \neg cp_2, s)(c) \end{aligned}$$

As an example, consider the following predicate describing a blocking condition:

$$B = (n = 3^{\blacktriangledown} \wedge m = 5^{\blacktriangle}) \vee m = 3^{\blacktriangledown}$$

The resulting coloring is depicted in Figure 9. It is easy to see that the coloring function for B yields the correct annotation for all possible valuations of the state. Namely, colors are only present if $n = 3 \wedge m = 5$ or $m = 3$ is satisfiable. If a predicate is satisfiable, each part of the predicate is considered separately, only introducing colors if each individual part introduces a color.

As expected, for the valuation $\{m \mapsto 3, n \mapsto 3\}$ a $\blacktriangledown$ color is returned for m. For the valuation $\{m \mapsto 2, n \mapsto 3\}$ no colors are returned. Finally, for the valuation $\{m \mapsto 5, n \mapsto 3\}$ a $\blacktriangle$ color is returned for m and a $\blacktriangledown$ color is returned for n.

The correctness of the coloring is captured in the following theorem.

Theorem 1. *Given a colored supervisor and a query, the coloring function returns a sound coloring that explains the unreachability of the query state.*

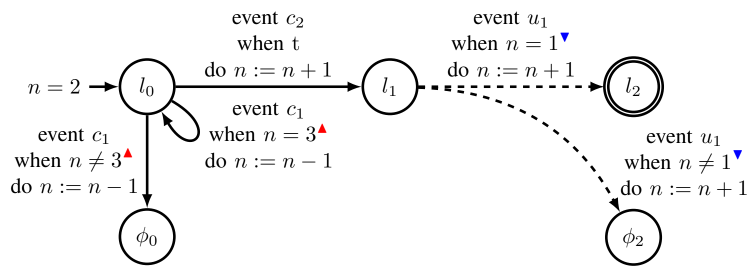
Although not formally proven, colors are introduced based on the blocking conditions of the supervisor. Hence, colors are only introduced when a blocking situation has occurred or will occur in the future. Therefore, the colors will explain the (possible) unreachability of the query state.

VIII. Example

In this section, the system of Figure 1 is examined. For synthesis, the following requirements are considered:

$$\begin{aligned} \blacktriangle &= l_0 \Rightarrow n = 2 \\ \blacktriangledown &= l_2 \Rightarrow n = 2 \end{aligned} \qquad (1)$$

These requirements state that at both location l_0 and l_2 the condition $n = 2$ must hold.



$n = 2$
l_0
l_1
l_2
ϕ_0
ϕ_2
event c_2 when t do $n := n + 1$
event c_1 when $n = 3^{▲}$ do $n := n - 1$
event c_1 when $n \neq 3^{▲}$ do $n := n - 1$
event u_1 when $n = 1^{▼}$ do $n := n + 1$
event u_1 when $n \neq 1^{▼}$ do $n := n + 1$

Fig. 10. Refinement of the plant of Figure 1 with respect to the requirements of Equation 1.

Note that these requirements are state-based expression, and not automata. Conversion from state-based expressions to automata is discussed in [16]. However, due to the simplicity of these two expression, direct encoding of the requirements in the refinement is trivial.

Based on the requirements, the refinement can be determined. In Figure 10, the refinement of the plant is depicted. Note that the duplication of the plant behavior is not depicted, for the sake of simplicity. Based on this refinement, the synthesis process of [7] has been applied. The obtained supervised system is depicted in Figure 11.

During the synthesis process, some rules are used for the simplification of predicates, with $p \in \mathcal{P}$ and $\triangle \in \mathcal{C}$:

$$\begin{aligned} {p_1}^{\triangle} \wedge {p_2}^{\triangle} &= (p_1 \wedge p_2)^{\triangle} \\ {p_1}^{\triangle} \vee {p_2}^{\triangle} &= (p_1 \vee p_2)^{\triangle} \\ \neg(p^{\triangle}) &= (\neg p)^{\triangle} \end{aligned}$$

Applying these rules does not alter the resulting coloring.

One additional rule is used for simplification of the blocking predicates: $\neg(\mathrm{t}^{\triangle} \vee p) = \mathrm{f}$, with $p \in \mathcal{CP}$. Only for blocking predicates, this rule does not change the coloring. Namely,

$$\chi(\neg(\mathrm{t}^{\triangle} \vee p), s) = \chi(\mathrm{f}^{\triangle} \wedge \neg p, s) = \chi(\mathrm{f}, s) = \varnothing$$

From the supervisor of Figure 11, it is clear that no solution can be found for the given requirements; the initial predicate is f, i.e., an empty supervisor is returned. So, why is our supervisor empty? This can be determined by computing the cause for the unreachability of the initial state, given by the query: $q = (l_0, n = 2)?$. This query is based on the initial location and initial valuation.

From the supervisor, the following coloring is obtained for $n = 2$ in location l_0: $\{▲ \mapsto \{n\}, ▼ \mapsto \{n\}\}$. The reason is deduced, as follows: n is colored ▼, as there is a path to location l_2 that does not result in $n = 2$ at l_2. This is the path in which c_2 is executed. Moreover, n is colored ▲, because in order to reach $n = 2$ at l_2, event c_1 must occur, which will violate a requirement (in the future). Hence, the supervisor is empty due to the interaction of both requirements. It can be seen that if one of the requirements is relaxed (or removed), a supervisor can be synthesized.

The query $q = (l_2, n = 2)?$ is considered to verify the emptiness of the supervisor w.r.t. the ▼ requirement; in location l_2 a valuation of $n = 2$ is required. This query gives rise to the following cause:

$$\begin{aligned} &((l_0, \{n \mapsto 2\}), \{▲ \mapsto \{n\}, ▼ \mapsto \{n\}\}) \xrightarrow{c_1} \\ &\quad ((l_0, \{n \mapsto 1\}), \{▲ \mapsto \{n\}, ▼ \mapsto \{n\}\}) \xrightarrow{c_1} \\ &\quad ((l_0, \{n \mapsto 0\}), \{▲ \mapsto \{n\}, ▼ \mapsto \varnothing\}) \xrightarrow{c_2} \\ &\quad ((l_1, \{n \mapsto 1\}), \{▲ \mapsto \varnothing, ▼ \mapsto \varnothing\}) \xrightarrow{u_1} \\ &\qquad ((l_2, \{n \mapsto 2\}), \{▲ \mapsto \varnothing, ▼ \mapsto \varnothing\}) \end{aligned}$$

For every state, the coloring is investigated and it will be shown that this is indeed a cause. The five states along the path are denoted as s_i with $0 \le i \le 4$.

- s_0 As explained; the same reasoning as why the supervisor is empty.
- s_1 n is still colored ▼, as there is a path to l_2 that does not result in $n = 2$ at l_2. Moreover, n is colored ▲, as currently $n \neq 2$.
- s_2 n is colored ▲, as the ▲ requirement is still violated. However, n is no longer colored ▼, as there is now a path to l_2 with the condition $n = 2$.
- s_3 No requirement is violated anymore; no requirement concerns location l_1, and a path is possible to l_2 with the condition $n = 2$.
- s_4 The query state, which does not violate any requirements.

IX. Concluding Remarks

In this paper, a method for the generation of causes is given. Based on a query, a cause gives an explanation why a certain state is no longer reachable in the supervised system. A cause may indicate which requirements need to be relaxed for the query state to be reachable. To compute causes, colors and colored predicates are introduced to distinguish the influence of each requirement.

In our problem definition, it was stated that at least one cause is returned for a given query if a cause exists. Alternatively one could search for the complete set of

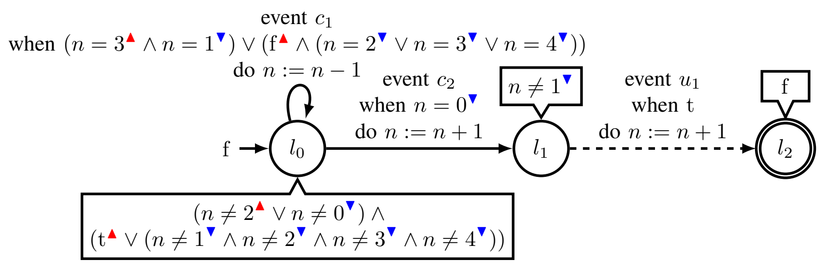

event c_1
when $(n = 3^{\blacktriangle} \wedge n = 1^{\blacktriangledown}) \vee (\mathrm{f}^{\blacktriangle} \wedge (n = 2^{\blacktriangledown} \vee n = 3^{\blacktriangledown} \vee n = 4^{\blacktriangledown}))$
do $n := n - 1$
event c_2
when $n = 0^{\blacktriangledown}$
do $n := n + 1$
$n \neq 1^{\blacktriangledown}$
event u_1
when t
do $n := n + 1$
f
f
l_0
l_1
l_2
$(n \neq 2^{\blacktriangle} \vee n \neq 0^{\blacktriangledown}) \wedge$
$(\mathrm{t}^{\blacktriangle} \vee (n \neq 1^{\blacktriangledown} \wedge n \neq 2^{\blacktriangledown} \wedge n \neq 3^{\blacktriangledown} \wedge n \neq 4^{\blacktriangledown}))$

Fig. 11. The resulting supervised system, based on the plant of Figure 1 and the requirements of Equation 1.

causes. Although it would be undesired, from an end-user's point of view, to return the complete set causes at once, this set could be used to report better information.

With respect to the generation of causes, possible future work is to investigate whether there is something like a minimal or maximal cause. These concepts would be beneficial with respect to reporting causes to the end-user.

A possible extension of this method is to give more insight on the actual propagation of requirements. In our approach, a color is introduced if a requirement is violated currently or in the future. Introducing, for instance, shades of colors, one can distinguish between a state in which the actual violation occurs, or a state which leads to a violation state.

A natural extension would be the consideration of other modeling formalisms. In this paper, automata with data are considered, though in principle it could be applied to a wide range of modeling formalisms, e.g., colored Petri nets.

The implementation of this method is ongoing. Currently, most efficient synthesis methods incorporate BDDs (Binary Decision Diagrams). In order to allow the generation of causes during synthesis, the coloring of predicates should be encoded in BDDs. How a possible implementation of colored logic in a BDD looks like or whether it can even be (efficiently) implemented is an open question.

Another possibility is the creation of causes in a process separate from the synthesis, in a second step. The information for causes is namely only necessary if an empty or unexpected result is obtained. Creating the information in a separate step, the efficiency of the actual synthesis is not influenced. Moreover, this process can be optimized for the given query.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[2] S. Balemi, G. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Transactions on Automatic Control*, vol. 38, no. 7, pp. 1040–1059, 1993.

[3] K. T. Seow and M. Pasquier, "Supervising passenger land-transport systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 3, pp. 165–176, 2004.

[4] S. T. Forschelen, J. M. van de Mortel-Fronczak, R. Su, and J. E. Rooda, "Application of supervisory control theory to theme park vehicles," *Discrete Event Dynamic Systems*, vol. 22, no. 4, pp. 511–540, 2012.

[5] R. Theunissen, M. Petreczky, R. Schiffelers, D. van Beek, and J. Rooda, "Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 20–32, 2014.

[6] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *Proc. 37th Conference on Decision and Control*. IEEE, 1998, pp. 3305–3310.

[7] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Symbolic approach to nonblocking and safe control of extended finite automata," in *IEEE Conference on Automation Science and Engineering*, 2010, pp. 471–476.

[8] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers, "CIF 3: Model-based engineering of supervisory controllers," in *Tools and Algorithms for the Construction and Analysis of Systems, ser. LNCS.*, vol. 8413, 2014, pp. 575–580.

[9] K. Akesson, M. Fabian, H. Flordal, and A. Vahidi, "Supremica – A tool for verification and synthesis of discrete event supervisors," in *Proceedings of the 11th Mediterranean Conference on Control and Automation, Rhodos, Greece*, 2003.

[10] S. Engell, S. Lohmann, and O. Stursberg, "Verification of embedded supervisory controllers considering hybrid plant dynamics," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 02, pp. 307–312, 2005.

[11] F. Lerda, J. Kapinski, E. M. Clarke, and B. H. Krogh, "Verification of supervisory control software using state proximity and merging," in *Proc. 11th Workshop on Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 4981. Springer, 2008, pp. 344–357.

[12] J. Markovski, D. van Beek, R. Theunissen, K. Jacobs, and J. Rooda, "A state-based framework for supervisory control synthesis and verification," in *Proc. 49th Conference on Decision and Control*. IEEE, 2010, pp. 3481–3486.

[13] T. Ball, M. Naik, and S. K. Rajamani, "From symptom to cause: Localizing errors in counterexample traces," in *Proc. 30th Symposium on Principles of Programming Languages*. ACM, 2003, pp. 97–105.

[14] A. Groce and W. Visser, "What went wrong: Explaining counterexamples," in *Proc. 10th SPIN Workshop on Model Checking of Software*, ser. Lecture Notes in Computer Science, vol. 2648. Springer, 2003, pp. 121–135.

[15] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *46th IEEE Conference on Decision and Control*, dec 2007, pp. 3387–3392.

[16] M. Starke, "Supervisory control using extended finite automata for ASML wafer scanners," Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, Mar. 2013.