**Modeling and automatic calculation of restart states for an industrial windscreen mounting station**

# Modeling and automatic calculation of restart states for an industrial windscreen mounting station ⋆

**Patrik Bergagård** * **Petter Falkman** * **Martin Fabian** *

* *Department of Signals and Systems, Chalmers University of Technology (e-mail: {patrikm, falkman, fabian}@ chalmers.se)*

**Abstract:** The production in an automated manufacturing system will not always progress as intended. A wide variety of possible faults may cause errors that lead to an unsynchronization between the control system and the physical system that consequently lead to production stoppages. The common industrial practice to deal with such non-intended progress is to extend the control system with tailor-made solutions to account for errors. This extension is both time consuming and there is no guarantee that all relevant errors are handled.
This paper models the control system to enable automatic derivation of restart states applied to an existing station for automatic mounting of windscreens onto car bodies. These restart states are states in the control system where it is correct to resynchronize the control and the physical systems so that the automated production can be resumed. This aids the preparation phase by letting the developer focus on modeling the nominal production and on specifying (un-)desired behavior during the restarted production, and then automatically retrieve the restart states for all control states. The online restart process is then reduced to a semi-automatic process where an operator can be supported with instructions for how to correctly resynchronize the control and the physical systems in a selected restart state.

*Keywords:* Restart, Error recovery, Unforeseen errors, Robot cell.

## 1. INTRODUCTION

To assure high utilization of manufacturing systems, much time is typically spent on development to handle various types of non-nominal behavior, such as errors. However, the common industrial practice to implement an error recovery mechanism typically lacks a systematic approach to take the global system behavior into consideration. The common practice is to extend the control system with tailor-made solutions to account for foreseen errors and known non-nominal processes. Considerable effort is spent during the development to enable that error messages and error codes from the resources in the manufacturing system will be displayed in the human machine interface for the operators. Restart points are typically added in robot programs to enable robots to reexecute when the production must be restarted.

Altogether, these solutions demand a lot from the operators to perform the restart online. The operators has to manage the restart without a global system knowledge only supported by the error messages and the existing restart points. This type of operator handled restart is sufficient after foreseen errors but can result in long production stoppages after unforeseen errors.

In a station for automatic mounting of windscreens onto car bodies at Volvo Car Corporation (VCC) Torslanda, Sweden, see Figure 1, the development time for the control system related to the non-nominal behavior was roughly double compared to the time related to the nominal production (Lagergren, 2014). Despite this effort, or perhaps as a consequence of the common practice, the average time that production is undesirably stopped due to errors, has been above the five minutes target time during its first months of usage (Augustsson, 2014). In many cases, the stoppages have been caused by unforeseen errors from which neither the control system nor the operators know how to efficiently restart the automated production. This station is now used in a project where formal methods are applied, aiming at supporting VCC's offline and online work with restart after unforeseen errors.

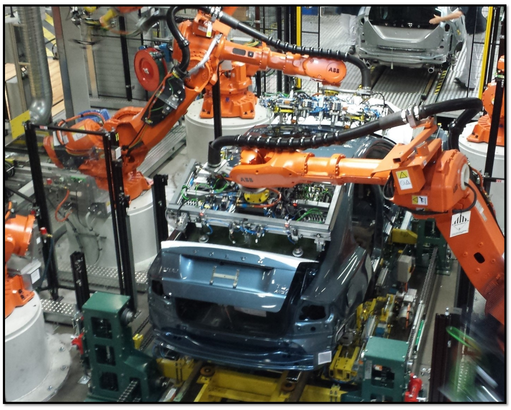

Fig. 1. The windscreen mounting station.

### 1.1 *The contribution of this paper*

In earlier work, Bergagård and Fabian (2013, 2014) proposed a method that enables restart after *unforeseen errors* in automated manufacturing systems, like the windscreen mounting station.

⋆ This work has been carried out at the Wingquist Laboratory VINN Excellence Centre within the Production Area of Advance at Chalmers. It has been supported by VIRTCOM-Virtual preparation and commissioning of production systems including PLC logics, reference number 2014-01408, Vinnova, FFI within Sustainable production technology. The support is gratefully acknowledged.

By exploiting supervisory control theory synthesis (Ramadge and Wonham, 1987), the *restart states* for an automata representation of the control system are calculated. Restart states are states from where it is valid to restart the production after a, possibly unforeseen, error such that the production can be completed. Both *nominal requirements* and possible *reexecution requirements* are respected in the restarted production. Based on this set of restart states, an operator can be supported to correctly restart the production without any reduced start-up pace.

There are two main contributions in this paper. Firstly, to model the control system using *variables*, describing important and distinguishable aspects of the products and the resources in the physical system, and *operations*, that by updating the variables describe the sub-processes in the production. Secondly, to represent the variables and the operations by automata such that the calculation part of the method proposed in Bergagård and Fabian (2013, 2014) can be applied. The restart states can then be retrieve automatically. The modeling approach used in the earlier work requires explicitly given sequential relations between each pair of operations. In this way, this paper contributes to shorten the development time for the control system related to restart and to provide an operator with support based on the global system behavior during the restart phase.

### *1.2 The windscreen mounting station*

The modeling approach proposed in this paper and the succeeding calculation of restart states will be illustrated on the mounting of the rear windscreen onto the tailgate. The windscreen mounting station is shown in Figure 1.

The studied part of the station contains two robots, one glue dispatcher, and one exchange table where windscreens enter and may exit the station. The station is crosscuted by a conveyer carrying the cars.

The windscreen is mounted onto the tailgate after the windscreen has had glue applied to it and the tailgate has been measured. To meet tolerance requirements, the mounting of the windscreen is adjusted to the exact position of the tailgate, given by the measurement. One robot carries the windscreen and the other robot measures the tailgate.

## 2. RELATED WORK

It is quite common in graph based restart methods to include additional error states and/or augmentations for recovery procedures, see for example Zhou and Dicesare (1989); Tittus et al. (2000); Odrey (2008); Lee and Tilbury (2008). Augmentations will most likely increase the state-space of the models. The models used throughout this paper have no explicit error states. The purpose of the models for the offline analysis is only to capture how the control states can be updated and not why, thus no additional states are necessary.

Some methods assume foreseen faults, like machine breakdowns, that can be included into the control system. In contrast to the systems considered in this paper, these methods are typically targeting material handling systems where it is reasonable to include specifications for the faulty behaviors to reroute the unaffected products to avoid the temporarily faulty machine(s), see for example Wen et al. (2008); Sülek and Schmidt (2014); Shu (2014). Specifications related to the faulty behavior resemble the reexecution requirements used in this paper.

Restart related to robot stations in the automotive industry, like the windscreen mounting station, are among others presented in Tittus et al. (2000); Andersson et al. (2010). Similarly to the method used in this paper, the production is restarted from restart states in the control system. The methods are however limited to specific control architectures only enabling straight operation sequences. This is in contrast to the modeling with variables and operations proposed in this paper that enables alternative operation sequences.

## 3. ONLINE RESTART OF A MANUFACTURING SYSTEM

During nominal production the control system and the physical system will operate in synchrony, with the actual state of the physical system at all times corresponding to the intended state of the control system. This nominal production, from an initial control state $q_i$ to a complete control state $q_c$, is illustrated in the lower plane in Figure 2. The plane represents all states in the control system. The striped areas represent states where the nominal requirements are broken and must be avoided. The thin and the thick solid and dashed lines show the intended and the actual progress in the control and the physical systems, respectively. The active state in the control system is updated when an operation is started or completes.

When a fault occurs, the intended progress deviates from the actual progress, so that the control and the physical systems become unsynchronized. This is illustrated by the thick dashed line in the lower plane. The control state $q_e$ that is active when the error is detected is referred to as the *error state*. The physical state $p_e$ at the time has no corresponding state in the control system and is thus located outside of the plane (outside of the model). The error is thus unforeseen.

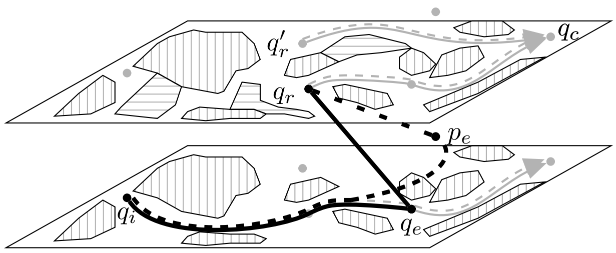


Fig. 2. The online restart process. The control and the physical systems are resynchronized in the restart state $q_r$. The nominal and the restarted productions are pictured in the lower and the upper planes, respectively.

The error induced unsynchronization between the control and the physical systems bring on that there are operations that have been executed in the control system but where the actual execution in the physical system has been performed incorrectly or not at all. To compensate for the unperformed execution, the overall idea in Bergagård and Fabian (2013, 2014) is therefore to restart the production from a control state, a so called *restart state*, from where it is correct to resume the production such that it can be completed, respecting both the nominal and the reexecution requirements.

The upper plane in Figure 2 shows the restarted production. Horizontally striped areas illustrate control states that break reexecution requirements and are thus to be avoided, together with the vertically striped areas. Under the assumption that the restart states for each control state have been precalculated, the online restart is reduced to a semi-automatic process. At first, the operator selects a restart state from the set of restart states for the error state $q_e$. In Figure 2, this set of restart states is $\{q_r, q'_r\}$, from which $q_r$ is selected. Based on the selection, the operator places the physical system in the physical state corresponding to this restart state, illustrated by the dashed line from $p_e$ to $q_r$. In the succeeding automatic part of the restart, the active control state is updated to the selected restart state $q_r$, illustrated by the line from $q_e$ to $q_r$. After this automatic part, the control and the physical systems have now been resynchronized and the production can be resumed.

## 4. MODELING NOMINAL PRODUCTION

A robot is modeled by a single variable capturing its crucial poses. There are two types of poses; *waiting poses* and *motion poses*. As the names signal, a robot is static waiting in a waiting pose and is in motion between two waiting poses in a motion pose. A product is modeled by one variable capturing its *form* and, in some cases, one variable capturing its *carrier*. For other systems, variables capturing other aspects might be more appropriate.

One of the values for each variable is set as *initial*. This initial value captures the *active* value of the variable when the production starts. At least one of the values for each value is set as *marked*. The production may complete when the active value for all variables are marked.

Processes and tasks that are executed in the system are modeled by operations. In this paper, each operation is *realized* by one robot. For other systems, more than one resource may be required for realization. An operation comprises one *guard*, and two *actions* referred to as *pre-* and *post-actions*. The guard is a boolean function defined over a subset of the variables. An operation can start to execute when its guard evaluates to *true*.

The pre- and post-actions specify how the variables are updated when an operation starts and completes, respectively. Each action specifies a value to a non-empty subset of the variables from their respective domains. The variables specified in the pre-action keep the specified values throughout the execution of the operation. Throughout this paper, the pre-action must only specify values to variables in the scope of the guard and the post-actions must only specify values to variables in the scope of the pre-action.

Table 1 illustrates how an operation **op** affects the variables **v0**, **v1**, and **v2**. Their domains, initial values, and marked values are irrelevant for this demonstration and are thus undefined. The symbols ?, $\Uparrow$, and $\Downarrow$ denote a sub-guard, a sub-pre-action, and a sub-post-action, respectively.

Table 1. Variables **v0**, **v1**, and **v2** affected by operation **op**.

| | |
|---|---|
| **op** | **v0** ?a $\Uparrow$c $\Downarrow$a |
| **op** | **v1** ?e $\Uparrow$n |
| **op** | **v2** ?o |

The final guard for an operation is the conjunction of all its sub-guards. The operation **op** can thus start if **v0**==a$\wedge$**v1**==e$\wedge$**v2**==o is satisfied. The pre- and the post-actions for an operation are built up from the sub-pre- and the sub-post-actions, respectively. Any two sub-pre-actions or any two sub-post-actions added for the same operation are assumed to not specify different values for the same variable. When **op** starts, **v0** and **v1** are assigned the values c and n, respectively. When **op** completes, **v0** is assigned the value a.

The actions for a general operation will update the variable values for the resource (robot) realizing the operation and the products being refined by the operation. To keep the product variable domains distinct, no values describe that a product is refined from one form to another or from one carrier to another. All product values are specified in the pre-actions.

The guards for the operations are typically related to local nominal requirements, like robot in correct pose carrying product in correct form. Global nominal requirements, like two robots must not be in two of their poses simultaneously to avoid collisions, are typically easier to specify with forbidden state combinations. Two or more variable value combinations can be specified as forbidden and should thus never be reachable.

### 4.1 Automata representation

The variables, the operations, and the forbidden state combinations can be represented by *automata* (Ramadge and Wonham, 1987). Each variable corresponds to an automaton where the values in the domain become the states. Initial and marked values become initial and marked states, respectively. The translation of the operations is less straightforward.

Since the guard for an operation is a boolean function over a subset of variables, each guard is satisfied by a finite set of assignments, called *guard assignments*, for these variables. Since each variable takes a single value in each guard assignment, this value corresponds to a state in the automaton representing the variable. When the operation starts, the pre-action updates the values for a subset of the variables in the scope of the guard. In the automata representation of the control system, this start of an operation is modeled by transitions.

For the case that the guard is satisfied by a single guard assignment, a transition is added in each automaton corresponding to a variable in the scope of the guard assignment. The transition is added differently depending on if the variable is in the scope of the pre-action or not. For a variable in the scope of the pre-action, the transition is added from the state corresponding to the value in the guard assignment to the state corresponding to the value specified by the pre-action. For a variable not in the pre-action, the transition is self-looped in the state corresponding to the value in the guard assignment. For the case that the guard is satisfied by more than one guard assignment, the presented translation is repeated for each guard assignment satisfying the guard.

The transitions added for the different guard assignments are labeled by unique events, called *start events*. For an operation *op*, the start events are denoted $\uparrow_{op}$ or suffixed by an enumeration index if the guard is satisfied by more than one guard assignment.

As for the start of an operation, the completion of an operation is modeled by transitions in the automata representation. The construction of these transitions follows the case with a single satisfying guard assignment presented above, however, using the post-action instead of the pre-action. From the requirement that all variables in the scope of the pre-action keep their specified values throughout the execution of the operation, these specified values are used as the single satisfying guard assignment. The added transitions are labeled by an unique event, called *complete event*. For an operation *op*, the complete event is denoted $\downarrow_{op}$.

The automata representation of the example operation *op* specified in Table 1 is shown in Figure 3. Only the states and the transitions related to *op* are shown for each automaton. The guard for *op* is satisfied by a single guard assignment, thus there is no index on the start event.

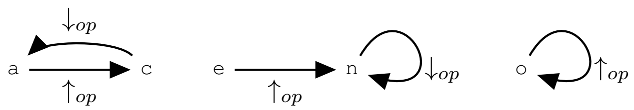


Fig. 3. From left to right, automata corresponding to the variables **v0, v1**, and **v2**.

A method for specifying *forbidden state combinations* locally for a set of automata such that the combination of these states are never reached in a supervised system is among others presented in Magnusson et al. (2010). Throughout this paper, this technique is used to specify the global nominal requirements that undesirable product and resource state combinations are avoided.

### *4.2 Supervised model*

The *supervisory control theory* (SCT, Ramadge and Wonham, 1987) is a model-based framework for automatic calculation of discrete event controllers. Given a set of automata that interact through full synchronous composition, a *supervisor* can be *synthesized* such that the composition of the automata and the supervisor is both *non-blocking* and *controllable*. Non-blocking assures that at least one marked state can be reached from any state in the supervised system. In SCT, a subset of the events are *uncontrollable*. By being controllable, the supervisor is never allowed to disable an uncontrollable event that might be generated by the automata.

To simplify the modeling task and to assure that all nominal requirements are satisfied by the model, the model is preferably retrieved as the solution of a synthesis problem. A supervisor is therefore synthesized for the automata representation of the control system. This supervised system satisfies all nominal requirements.

The synthesis problem is general, thus any synthesis algorithm can be used. In this paper, the supervisor is represented by one boolean function for each controllable event as presented in Miremadi et al. (2012). The boolean functions are defined over the variables in the model. To force the supervisor to always let an executing operation complete, all complete events are uncontrollable. The start events are thus the only controllable events in the model. The final form for each operation guard is then a conjunction of the initially defined guard with the synthesized boolean function for the start event of the operation. These conjuncted guards are referred to as *extended guards*.

## 5. MODELING NOMINAL PRODUCTION FOR THE WINDSCREEN MOUNTING STATION

The windscreen mounting station presented in Section 1.2 contains two products: the (rear) *windscreen* and the *tailgate*. The windscreen is modeled by two variables. The first variable models the carrier of the windscreen, denoted **wsCarr**, and the second variable models its form, denoted **wsForm**. The tailgate on the car body remains stationary on the conveyer throughout the work cycle and is therefore modeled by a single variable for its form, denoted **tgForm**. The domains for the three variables are given below. Initial and marked values are $\overline{\text{over}}$- and underlined, respectively.

$$\begin{aligned} \textbf{wsCarr} &:= \big\{\overline{\texttt{byInFeed}}, \texttt{byRobot}, \underline{\texttt{byTailgate}}, \\ &\quad\ \underline{\texttt{byOutFeed}}\big\} \\ \textbf{wsForm} &:= \big\{\overline{\texttt{blank}}, \underline{\texttt{glued}}\big\} \\ \textbf{tgForm} &:= \big\{\underline{\overline{\texttt{empty}}}, \underline{\texttt{measured}}, \underline{\texttt{hasWindscreen}}\big\} \end{aligned}$$

Two of the eight variable combinations for the windscreen are undesirable. The windscreen must not have the form `glued` and be carried `byInFeed` nor have the form `blank` and be carried `byTailgate`. In the automata representation, the corresponding states are specified as two forbidden state combinations.

Given these variables describing the products, the value adding part of a work cycle is modeled by four *process operations*. Three of these refine the windscreen:

| | |
|---|---|
| **gripWindscreen** | **wsCarr** ?byInFeed ⇑atRobot |
| **glueWindscreen** | **wsForm** ?blank ⇑glued |
| **mountWindscreen** | **wsCarr** ?byRobot ⇑byTailgate |

and two refine the tailgate:

| | |
|---|---|
| **measureTailgate** | **tgForm** ?empty ⇑measured |
| **mountWindscreen** | **tgForm** ?measured ⇑hasWindscreen |

The **mountWindscreen** operation thus affects both products. The three operations that affect the windscreen are realized by a robot R3 and the **measureTailgate** operation is realized by a robot R8.

### *5.1 Showing and scrapping of the windscreen*

Gluing of the windscreen is an operation with high tolerance requirements. In addition to an automatic inspection, performed continuously throughout the operation execution, an operator must be able to manually inspect a windscreen with glue applied to it. To assure a high quality, if either of the inspections fail, the windscreen must be scrapped and the car will leave the station without a windscreen mounted onto the tailgate. If this happens, a new windscreen is mounted onto the tailgate in a later station, outside the main production line.

To account for manual inspection and/or scrapping of the windscreen, the two process operations **showWindscreen** and **scrapWindscreen** are added to the model. Both operations are realized by R3. From a control point of view, both scrapping and mounting of the windscreen are correct ways to complete the work cycle, thus the `byOutFeed` value is marked. This gives alternative production processes in the control system.

| | |
|---|---|
| **showWindscreen** | **wsForm** ?glued |
| **showWindscreen** | **wsCarr** ?byRobot |
| **scrapWindscreen** | **wsCarr** ?byRobot ⇑byOutFeed |

### *5.2 Robot poses*

To start and end the realization of a process operation, the realizing robot must be in an operation specific waiting pose. During the realization, the robot is in an operation specific motion pose. The domains for the variables modeling the poses for the two robots are given below. A value suffixed by a *W* and a *M* refers to a waiting and a motion pose, respectively. In addition to the poses modeled for the operations, each robot has per default a home waiting pose where it must be when the work cycle starts and ends. This home pose is thus both initial and marked. An abbreviation like `grip{W,M}` is short for `gripW`, `gripM`.

$$\textbf{R3Pose} := \left\{\overline{\underline{\texttt{homeW}}}, \texttt{grip\{W,M\}}, \texttt{glue\{W,M\}}, \texttt{mount\{W,M\}}, \texttt{show\{W,M\}}, \texttt{scrap\{W,M\}}\right\}$$

$$\textbf{R8Pose} := \left\{\overline{\underline{\texttt{homeW}}}, \texttt{tailgate\{W,M\}}\right\}$$

The values modeling the poses for R3 and R8 during realization of the six operations are given below. In this station, the same waiting pose is used both before and after the realization, this is not required in general.

| | |
|---|---|
| **gripWindscreen** | **R3Pose** ?gripW ⇑gripM ⇓gripW |
| **glueWindscreen** | **R3Pose** ?glueW ⇑glueM ⇓glueW |
| **mountWindscreen** | **R3Pose** ?mountW ⇑mountM ⇓mountW |
| **showWindscreen** | **R3Pose** ?showW ⇑showM ⇓showW |
| **scrapWindscreen** | **R3Pose** ?scrapW ⇑scrapM ⇓scrapW |
| **measureTailgate** | **R8Pose** ?tailgateW ⇑tailgateM ⇓tailgateW |

### 5.3 Transport operations

A *transport operation* is an operation which upon execution reconfigures a robot from one waiting pose to another waiting pose, without affecting a carried product. Each robot has its own set of transport operations. By exploiting the succeeding supervisor synthesis, the transport operations can be added automatically to the model, based on the waiting poses defined for each robot. A single transport operation is created for each ordered pair of waiting poses for each robot. For example, **R3Pose** contains six waiting poses so 6*(6-1)=30 transport operations are thus created for R3. At the start and completion of a transport operation, the robot must be in the first and the second waiting pose in the pair, respectively. During the realization of the operation, the robot is in an operation specific motion pose. The motion pose for the operation that models that R3 reconfigures from `homeW` to `gripW` is, as an example, denoted `homeGripM`.

To avoid unjustified transport operations, there must be an alternation in the execution of process and transport operations for each robot. This is modeled by a variable, denoted altExR, for each robot R plus sub-guards and sub-pre-actions in the operations realized by R, as shown below. Let $\textbf{op}_R^P$ and $\textbf{op}_R^T$ denote general process and transport operations that are realized by robot R, respectively. At the start and end of each work cycle a robot is in its home waiting pose. The first and last operation must therefore be a transport operation, thus `tNext` is initial and `pNext` is marked.

$$\textbf{altExR} := \left\{\overline{\texttt{tNext}}, \underline{\texttt{pNext}}\right\}$$

| | |
|---|---|
| $\textbf{op}_R^P$ | **altExR** ?pNext ⇑tNext |
| $\textbf{op}_R^T$ | **altExR** ?tNext ⇑pNext |

### 5.4 Global nominal requirements

The robots will collide if they work close to the tailgate simultaneously. Four forbidden state combinations are created to specify that R3 cannot be in the pose `mountM` simultaneously with R8 in any of the poses `tailgate{W,M,Home}` or the motion pose `homeTailgateM`.

### 5.5 Supervised model

Based on the automata representation of the operations, the variables, and the forbidden state combinations, defined throughout this section, a supervisor is synthesized to retrieve the supervised model. The supervised model contains 179 states. The synthesis enables eleven and two automatically added transport operations for R3 and R8, respectively.

### 5.6 The **mountWindscreen** operation as automata

The representation of the model by automata is exemplified on the **mountWindscreen** operation. The extended guard and the actions for **mountWindscreen** are given in Table 2. Due to forbidden state combinations for the tailgate, the synthesized supervisor requires that **mountWindscreen** is only enabled if R8 is in its home pose.

Table 2. Extended guard and the actions for **mountWindscreen**.

| | |
|---|---|
| **mountWindscreen** | **wsCarr** ?byRobot ⇑byTailgate |
| **mountWindscreen** | **tgForm** ?measured ⇑hasWindscreen |
| **mountWindscreen** | **R3Pose** ?mountW ⇑mountM ⇓mountW |
| **mountWindscreen** | **altExR3** ?pNext ⇑tNext |
| **mountWindscreen** | **R8Pose** ?homeW |

The operation is represented by the five automata corresponding to the variables in the scope of its guard and actions. The three generic automata in Figure 3 are used for illustration. The events $\uparrow_{op}$ and $\downarrow_{op}$ are set to $\uparrow_{mountWindscreen}$ and $\downarrow_{mountWindscreen}$, respectively. Only the states and transitions related to **mountWindscreen** are shown in each automaton.

The automaton corresponding to the variable **R3Pose** will get two transitions like the left automaton in Figure 3. The states `a` and `c` are set to `mountW` and `mountM`, respectively. The automata corresponding to the variables **wsCarr**, **tgFrom**, and **altExR3** will get one transition each like the middle automaton in Figure 3. The states `e` and `n` are set to `byRobot` and `byTailagate`, `measured` and `hasWindscreen`, and `pNext` and `tNext`, respectively. The automaton corresponding to the variable **R8Pose** will get one self-loop transition like the right automaton in Figure 3. The state `o` is set to `homeW`.

## 6. MODELING RESTART OF PRODUCTION

To analyze how the production can be restarted, the automata representation of the control system of the nominal production is extended with transitions, so called *reset transitions*, that reset operations, or more precisely, reset the variable values updated by the operations (Bergagård and Fabian, 2014). The control system then contains process and transport operations that take the production process "forward" and reset transitions that restart the production process by going "backward". Based on such an extended control system, the restart states can be derived by analyzing the sequences of reset transitions that can be taken from each control state, such that the production can be resumed and eventually be completed, respecting both nominal and reexecution requirements. By calculating all restart states for all contorl states, the set of restart states can be further analyzed to for example derive the states where the operator placement becomes simplest according to some appropriate metric or to minimize the overall number of restart states.

The modeling with variables proposed in this paper means that the reset transitions cannot be derived as in Bergagård and Fabian (2014). This section therefore describes how the reset transitions can be added automatically to the automata representation of the control system.

Each operation that is executing or just completed should be reset such that the operation can be reexecuted. An operation is executing when the variables in the scope of the pre-action take the values specified by the pre-action. Similarly, an operation is just completed when the variables in the scope of the post-action take the values specified by the post-action and the variables in the scope of the pre-action but not in the scope of the post-action take the values specified by the pre-action. Recall that the post-action must only update the values on variables in the scope of the pre-action.

The variables in the scope of the pre-action should be reset to the values they had when the guard for the operation was satisfied, such that the operation can be reexecuted. As outlined in Section 4.1, the extended guard for an operation is satisfied by a finite set of guard assignments. In each such guard assignment the variables in the scope of the pre-action take a value.

For the case the guard is satisfied by a single guard assignment, up to two reset transitions are included in each automaton corresponding to a variable in the scope of the pre-action. One reset transition is included from the state corresponding to the value when the operation is executing to the state corresponding to the value in the guard assignment. For variables updated by the post-action, an additional reset transition is included from the state corresponding to the value when the operation is just completed to the state corresponding to the value in the guard assignment. All the included transitions are labeled by the same event. For an operation *op* the event is denoted $\hookleftarrow_{op}$. For the case the guard is satisfied by more than one guard assignment, the reset transitions are included for each assignment separately and the event for each assignment is suffixed with an enumeration index. No reset transitions are included when the guard is not satisfied by any guard assignment.

### 6.1 *Reset transitions for the* **mountWindscreen** *operation*

The retrieving of reset transitions is exemplified on the **mountWindscreen** operation, specified in Section 5.6. The reset transitions for **mountWindscreen** are given in Figure 4. Only the states and the reset transitions related to *op* are shown for each automaton.

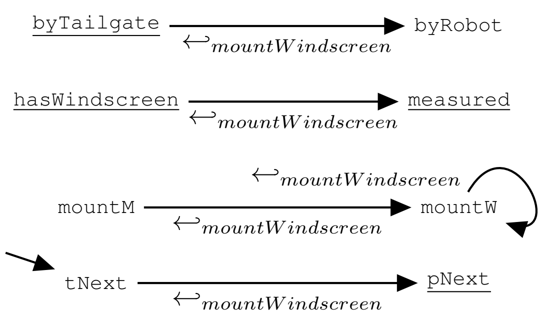


Fig. 4. From top to down, reset transitions for **mountWindscreen** in automata for the variables **wsCarr**, **tgForm**, **R3Pose**, and **altExR3**.

## 7. CALCULATION OF RESTART STATES

Given the set of automata modeling the nominal production, the reset transitions, and the reexecution requirements, a supervisor is synthesized to derive the enabled reset transitions. As presented in Bergagård and Fabian (2014), the set of restart states for each control state are the set of states that are reachable using a sequence of reset transitions in the supervised system. By constructing the transitive closure (Pemmaraju and Skiena, 2003) for the supervised system using only reset transitions, the set of restart states for each control state correlates to the set of adjacent states.

### 7.1 *Reexecution requirements for the windscreen mounting station*

The windscreen mounting station contains three reexecution requirements. To guarantee a high product quality, the **glueWindscreen** operation is not reexecutable. This operation can start at most once during each mounting work cycle. This is captured by the automaton **glueOnce** shown in Figure 5. The state `zero` is initial. Both states are marked.

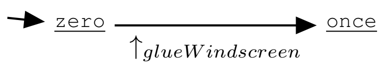


Fig. 5. Automaton to model that **glueWindscreen** is not reexecutable.

The **mountWindscreen** operation can only be reexecuted if the quality of the windscreen is approved by an operator and the tailgate is re-measured. These reexecution requirements are modeled by two instances of the automaton shown in Figure 6. To capture the operator approval, the windscreen must be shown to the operator before **mountWindscreen** is reexecuted. In the first automaton instance, called **mustShow**, the start event $\uparrow_{op}$ and the state `toK` are therefore set to $\uparrow_{showWindscreen}$ and `toShow`, respectively. To capture the re-measurement, $\uparrow_{op}$ and `toK` are set to $\uparrow_{measureTailgate}$ and `toMeasure`, respectively, in the second automaton instance, called **mustMeasure**. In both instances, the state `normal` is initial and all states are marked.

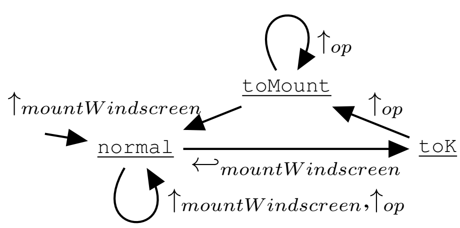


Fig. 6. Automaton to model that an operation **op** must execute between the resetting and the reexecution of **mountWindscreen**.

### 7.2 *Restart states for the* **mountWindscreen** *operation*

The synthesized supervisor contains 574 states. All control states except those related to gluing the windscreen have at least one restart state.

The restart states calculation is exemplified for the case when the control state modeling execution of **mountWindscreen** is the error state. The results are presented in Table 3.

Table 3. An error state with four restart states.

| Variables/ Automata | Error state | Restart states |
|---|---|---|
| **wsForm** | glued | glued |
| **wsCarr** | byTailgate | byRobot |
| **tgForm** | hasWindscreen | empty |
| **R3Pose** | mountM | glueWVshowW |
| **R8Pose** | homeW | homeWVtailgateW |
| **altExR3** | tNext | _ |
| **altExR8** | pNext | _ |
| **glueOnce** | once | once |
| **mustShow** | normal | toShow |
| **mustMeasure** | normal | toMeasure |

Recall that the product variables take their refined values in the pre-action. Thus, during a substantial part of the **mountWindscreen** execution, the windscreen is carried by R3, despite that the automata corresponding to **wsCarr** and **tgForm** are in the states `byTailgate` and `hasWindscreen`, respectively. The placement action to reset these variables to `byRobot` and `empty` is thus straightforward if the error occurs while the windscreen is still carried by R3. Also note, the **glueWindscreen** operation always precedes **mountWindscreen**, thus the automaton corresponding to **glueOnce** is in the state `once`.

In total, the error state has four restart states that respect both the nominal and the reexecution requirements. As indicated by the or-signs, R3 can be placed in one of the poses `glueW` and `showW` and R8 can either remain in its home pose or be moved to the pose `tailgateW`. The state for the automaton modeling the alternation in the execution between process and transport operations are connected to the pose selected for each robot. These states are therefore left blank in the table. From the reexecution requirements, both the operations **showWindscreen** and **measureTailgate** must be reexecuted. The automata **mustShow** and **mustMeasure** are therefore in the states `toShow` and `toMeasure`, respectively.

The table shows how the proposed modeling approach with variables aids the operator during the restart. Both the intended state (the error state) and the restart states in the control system are described on the selected abstraction level.

## 8. CONCLUSION

This paper has shown how to model the control system for an industrial windscreen mounting station by variables, describing distinguishable aspects of products and resources, and operations, describing sub-processes in the production. Given this model, the restart states from where it is valid to restart the station after error related stoppages such that the requirements on both the nominal and the restarted production are satisfied, can be derived automatically using formal methods. This enables the control system developer to focus on modeling the nominal production and specify (un-)desirable behavior for the restarted production, and thereafter automatically retrieve the allowed restart alternatives. Based on these automatically derived restart states, an operator can be given support for how to correctly restart the production when an error has been detected, diagnosed, and corrected in the system. By studying historical production stoppages for the windscreen mounting station, it can be concluded that if the proposed restart method had been used, using restart states calculated from the developed model, the restart after unforeseen errors that caused long production stoppages could have been improved.

Future work is concerned with a demonstration implementation to study how the proposed restart method can be merged with current industrial practice. It is worth to mention that parts of the method has already been implemented and tested in a lab station, see Parsaeian (2014).

## REFERENCES


Andersson, K., Lennartson, B., and Fabian, M. (2010). Restarting Manufacturing Systems; Restart States and Restartability. *IEEE Transactions on Automation Science and Engineering*, 7(3), 486–499.

Augustsson, C. (2014). *Senior Project Leader at Volvo Car Corporation*. Personal communication.

Bergagård, P. and Fabian, M. (2013). Calculating Restart States for Systems Modeled by Operations Using Supervisory Control Theory. *Machines*, 1(3), 116–141.

Bergagård, P. and Fabian, M. (2014). Calculating restart states using reset transitions. In *IEEE International Conference on Robotics and Automation*.

Lagergren, N. (2014). *Control system developer at Teamster*. Personal communication.

Lee, S. and Tilbury, D.M. (2008). A modular control design method for a flexible manufacturing cell including error handling. *International Journal of Flexible Manufacturing Systems*, 19(3), 308–330.

Magnusson, P., Fabian, M., and Åkesson, K. (2010). Modular specification of forbidden states for supervisory control. In *Workshop on Discrete Event Systems*, 412–417.

Miremadi, S., Lennartson, B., and Åkesson, K. (2012). A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata. *IEEE Transactions on Control Systems Technology*, 20(6), 1421–1435.

Odrey, N.G. (2008). Error Recovery in Production Systems: A Petri Net Based Intelligent System Approach. In *Petri Net. Theory and applications*, chapter 2. InTech.

Parsaeian, S. (2014). *Implementation of a Framework for Restart after Unforeseen Errors in Manufacturing Systems*. Master Thesis, Chalmers University of Technology, Signals and Systems.

Pemmaraju, S. and Skiena, S. (2003). *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press.

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

Shu, S. (2014). Recoverability of Discrete-Event Systems With Faults. *IEEE Transactions on Automation Science and Engineering*, 11(3), 930–935.

Sülek, A.N. and Schmidt, K. (2014). Computation of Supervisors for Fault-Recovery and Repair for Discrete Event Systems. In *Workshop on Discrete Event Systems*, 428–433.

Tittus, M., Andréasson, S.A., Adlemo, A., and Frey, J.E. (2000). Fast restart of manufacturing cells using restart points. In *World Automation Congress*.

Wen, Q., Kumar, R., Huang, J., and Liu, H. (2008). A framework for fault-tolerant control of discrete event systems. *IEEE Transactions on Automatic Control*, 53(8), 1839–1849.

Zhou, M. and Dicesare, F. (1989). Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5), 963–973.