On test case reduction for testing safety properties of manufacturing systems

On test case reduction for testing safety properties of manufacturing systems

Adnan Khan, Sahar Mohajerani, Martin Fabian

Chalmers University of Technology, Department of Electrical Engineering, 41296 Göteborg, Sweden

Abstract

This paper presents an approach to reduce the number of test cases, and hence testing time for the safe input-output conformance simulation relation (safe-IOCOS). The safe-IOCOS relation requires the implementation to be trace equivalent with respect to the specification only for traces composed of safety behaviors, which makes safe-IOCOS a suitable relation to test safety properties in practical settings. However, in typical manufacturing systems, multiple safety behaviors are typically associated with each nominal operation in the implementation. Thus, if safe-IOCOS is used industrially then testing for safety related faults becomes time consuming as the traces composed of same safety behaviors gets tested multiple times. This is possible either if the target states reached after the execution of traces have the same past behavior or the same future behavior. To remedy this, two reduction methods are proposed in this paper, *subset construction* and *bisimulation equivalence*. Both reduction methods preserve the traces of the system. Using both subset construction and bisimulation, a given specification can be maximally reduced and then used to implement the manufacturing system. The implementation based on a maximally reduced bisimilar specification allows the test engineer to omit test cases if the same safety behavior has already been tested. Furthermore, faults related to missing safety behaviors that are associated with multiple traces can be uncovered more efficiently compared to if the non-reduced specification is used for testing. To summarize, testing is a laborious problem, which can benefit from methods that enable reduction in testing time and makes the testing procedure efficient in terms of uncovering errors.

Keywords:
Labelled transition systems, Input-output conformance testing, Bisimulation, Safety, Discrete event system, Subset construction method

1. Introduction

Currently, many machines in the industrial sector are controlled via programmable logic controllers (PLCs). These PLCs are usually programmed such that the installed machines are coordinated to carry out specified tasks. The controlled behavior of such machines is mainly discrete in nature, therefore these can be formally modelled as *discrete event systems* [1].

Discrete event systems evolve with respect to occurring *events*, while at each time instant occupying a specific *state* where certain conditions are valid. Formally, the interaction of such systems can be described by different variants of synchronous composition, see [2].

To ensure safety, a specialized controller called a *safety PLC* is typically used. The job of the safety PLC is to keep both machines and humans safe in critical situations.

Email addresses: adnan.khan@chalmers.se (Adnan Khan), mohajera@chalmers.se (Sahar Mohajerani), fabian@chalmers.se (Martin Fabian)

When it comes to control code this can be generated either automatically or manually. For automatic generation of controllers the *supervisory control theory* [3] framework is often used in the formal domain. However, industrially, engineers mostly carry out this task manually. After the controller generation, the physical controller (the PLC) is coupled with the real plant in a closed-loop setting referred to as the *implementation*.

To increase the confidence that the implementation is fault free, testing of the implementation is carried out with respect to some specification. Frequently, this task is done using a formal approach like *model-based testing* (MBT) [4].

In model-based testing, the implementation, which is regarded as a black-box, is exposed to various inputs in accordance to the specification and the emitted outputs are observed. There are several variations of such an approach, and in this paper, we focus on two approaches i.e. *input-output conformance simulation relation* (IOCOS) [5] and *safe input-output conformance simulation relation* (safe-IOCOS) [6].

IOCOS is a more fine-grained variation of IOCO [7, 8], with an extra requirement on the inputs. According to the IOCOS testing framework presented in [5, 9], an implementation fails the IOCOS relation if the implementation is missing mandatory inputs or if it generates unexpected outputs. However, to uncover faults associated with safety in practical settings, IOCOS is found to be inadequate due to the subset requirement on outputs and the superset requirement on inputs [6].

Due to this, the *safe input-output conformance simulation relation* (safe-IOCOS) [6] is introduced. The safe-IOCOS relation requires equality for the inputs and outputs related to safety. And this requirement makes safe-IOCOS suitable to find faults related to safety. However, in practical settings, there are many safety behaviors implemented with each nominal control sequence of a production system that must be tested per the definition of safe-IOCOS. This obligation to test all repetitive safety combinations unnecessarily increases the time spent for testing.

For example, if a floor scanner is activated by a human entering a certain zone, then all machines in that zone must pause their activity until the human leaves. Hence, the safety of the floor scanner input is common for all the machines in that particular zone.

Though testing can only reveal the presence of faults, never their absence, it can raise the confidence in the system to be free of obvious and frequently occurring faults, *if the system can be subjected to enough test cases*. Thus, there is a need to reduce the testing time for this confidence to be raised. Reducing the testing time can be achieved if the specification, and hence the implementation, is "well-behaved" in the sense of not containing unnecessary branching and non-determinism.

There are various approaches that deal with reduction of test cases. For example [10], proposes an approach based on integer linear programming and on the properties of the control flow graph. Similarly, the approach proposed in [11] exploits neural networks to reduce test cases. However these approaches does not deal with the problem of unnecessary branching and non-determinism in relation to safety.

In an attempt to achieve this, a testing approach based on *bisimulation* [12] equivalence and the *subset construction method* [13] is studied in this paper.

Bisimulation is an equivalence relation that considers two states equivalent if they have the same future behavior. This property can be exploited to reduce the given specification such that traces with same future safety behaviors need not be tested multiple times.

The subset construction method, on the other hand, removes non-determinism by merging states with the same past behavior. This method can help to avoid test cases with non-deterministic behavior, which as a consequence allow the test engineer to avoid retesting traces that have already been tested.

1.1. Contribution

In this paper we show how reduction methods called *bisimulation* and *subset construction* can be exploited to reduce the complexity of the specification before implementation to make testing effi-

cient. It is shown that by employing bisimulation and subset construction, the number of test cases can be reduced for the safe-IOCOS relation, while the behavior of the system is preserved. Moreover, as non-determinism is removed from the system the confidence in the test procedure will increase. Furthermore, an example is presented to show how reducing the given specification helps in reducing test cases and consequently the time spent during testing of an implementation.

1.2. Outline

This paper is structured in seven sections. In Section 2, the formal definitions required to describe safe-IOCOS are detailed. Section 3 gives an overview of the safe-IOCOS testing relation. Section 4 introduces the bisimulation relation. In Section 5 the subset construction method and the problem associated with non-determinism are introduced. Section 6 introduces the proposed approach and formal proofs along with some examples modelled in the tool *Supremica* [14]. Finally, Section 7 concludes the paper and presents some future work directions.

2. Preliminaries

In this section, some formalism and definitions that are used to represent labelled transition systems (LTS), IOCOS, and safe-IOCOS are detailed.

For an LTS, consider two disjoint sets of output actions O, and input actions I. The output actions consists of nominal actions O_n, semi-nominal actions O_{sn}, and safety actions O_x, such that it holds that $O = O_n \cup O_x$, $O_{sn} \subseteq O_n$, and, $O_x \cap O_n = \emptyset$. These output actions are initiated by the system under test and are expressed with an exclamation mark, such as $!x \in O$. Similar to output actions, input actions also consist of nominal actions I_n, semi-nominal actions I_{sn}, and safety actions I_x such that it holds that $I = I_n \cup I_x$, $I_{sn} \subseteq I_n$, and, $I_x \cap I_n = \emptyset$. The input actions are signals to the system such as $a \in I$.

Definition 1. *An* I/O labelled transition system *(LTS) is a 4-tuple* $\langle Q, q_0, \Sigma, \rightarrow \rangle$[1] *where:*

- Q *is a non-empty set of states;*
- $q_0 \in Q$ *is the initial state;*
- Σ *is a finite set of actions. These represent the observable actions of a system, i.e.* $\Sigma = I \cup O$*, where* I *and* O *are as defined previously.*
- $\rightarrow \subseteq Q \times \Sigma \times Q$ *is the transition relation defining from which state, which action leads to which next state.*

We write transitions in in-fix form, $q \xrightarrow{a} q'$ meaning that $\langle q, a, q' \rangle \in \rightarrow$, and $q \xrightarrow{a}$ for $a \in \Sigma$, if there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. Furthermore, a *trace* t is a finite sequence of symbols of Σ, i.e. $t \in \Sigma^*$, including the empty trace ϵ. The transition relation is extended to traces in Σ^* by letting $q \xrightarrow{\varepsilon} q$ for all $q \in Q$, and $q \xrightarrow{t\sigma} p$ if $q \xrightarrow{t} y$ and $y \xrightarrow{\sigma} p$ for some $y \in Q$. Furthermore, $q \xrightarrow{t}$ means that $q \xrightarrow{t} p$ for some $p \in Q$.

An LTS is said to be *non-deterministic* if for some $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ it holds that $q \neq q'$.

Definition 2. *The set of traces from a state* q *in an LTS is:*

$$\boldsymbol{traces}(q) = \{t \in \Sigma^* \mid q \xrightarrow{t}\}. \tag{1}$$

For an LTS $S = \langle Q_S, q_0, \Sigma, \rightarrow_S \rangle$*, its set of traces are the ones defined from its initial state:*

$$\boldsymbol{traces}(S) = \{t \in \Sigma^* \mid q_o \xrightarrow{t}\}. \tag{2}$$

Definition 3. *The set of states reached* after *a trace* t *from a state* q *is:*

$$\boldsymbol{after}(q, t) = \{q' \in Q \mid q \xrightarrow{t} q'\}. \tag{3}$$

For an LTS $S = \langle Q_S, q_0, \Sigma, \rightarrow_S \rangle$ *the set of states reached* after *a trace* t *is:*

$$\boldsymbol{after}(S, t) = \{q' \in Q_S \mid q_0 \xrightarrow{t} q'\}. \tag{4}$$

[1] This paper does not treat marked states, so this property is not formally defined.

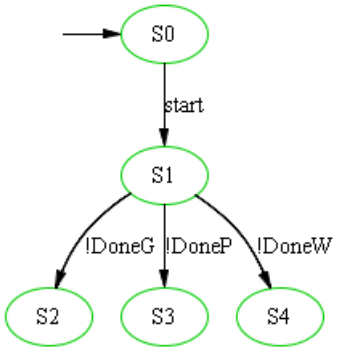

Figure 1: LTS model of a simple manufacturing system

Definition 4. *The set of all outgoing actions enabled at a state q is:*

$$\boldsymbol{act}(q) = \{\sigma \in \Sigma \mid q \xrightarrow{\sigma}\}. \tag{5}$$

The set of all outputs and inputs enabled at a state q is then given by $\mathbf{outs}(q) = \mathbf{act}(q) \cap O$ and $\mathbf{ins}(q) = \mathbf{act}(q) \cap I$, respectively.

2.1. Example

To further elaborate the notation, a simple manufacturing system consisting of a machine and three buffers is used. The LTS model S of the system is shown in Fig. 1. The action *start* represents that the system receives a workpiece and starts working on it. Based on the outcome of this work, the piece will be put in one of three different buffers, actions *DoneG*, *DoneP*, and *DoneW*. In S, *start* is an input action, and *!DoneG*, *!DoneP*, and *!DoneW* are the output actions. S_0 is the initial state, denoted by the arrow pointing to it.

The traces of S are:

$$\mathbf{traces}(S) = \{\epsilon, start, start.DoneG, start.DoneP, start.DoneW\}.$$

The possible traces from state S_1 are:

$$\mathbf{traces}(S_1) = \{!DoneG, !DoneP, !DoneW\}.$$

The states reached from the initial state S_0 after the trace *start.!DoneP* is S_3, thus $\mathbf{after}(S_0, start.!DoneP) = \{S_3\}$. Moreover, the outgoing actions from state S_1 are $\mathbf{act}(S_1) = \{!DoneG, !DoneP, !DoneW\}$.

Though, for brevity, the example only has a finite set of traces, in general $\mathbf{traces}(\cdot)$ is an infinite set, see for instance Fig. 5. So, even though each trace is finite in length, testing them all is impossible. The set of states is finite, however, thus the testing can stop when the global state space of the given LTS is exhausted.

Two or more LTSs can interact with each other by simultaneously executing shared actions. This is modeled by *synchronous composition.*

Definition 5. *For two LTSs A and B with the same set of actions, their* synchronous composition *is $A||B = \langle Q_A \times Q_B, \Sigma, \langle q_{A0}, q_{B0}\rangle, \rightarrow_{A||B}\rangle$, where*

$$\langle p, q\rangle \xrightarrow{\sigma}_{A||B} \langle p \xrightarrow{\sigma}, q \xrightarrow{\sigma}\rangle \tag{6}$$

if both $p \xrightarrow{\sigma}$ and $q \xrightarrow{\sigma}$ are defined, else undefined.

Since synchronous composition requires simultaneous execution of shared actions, such actions are blocked if they cannot occur in one of the LTSs from its current state.

Reduction of LTSs essentially merges *equivalent* states. Thus, there needs to be rigorously defined the notion of equivalence classes over states.

Definition 6. *Let Q be a set of states. A relation $\sim \subseteq Q \times Q$ is called an* equivalence relation *on Q if it is reflexive, symmetric, and transitive. Given an equivalence relation $\sim$ on q, the* equivalence class *of $q \in Q$ is $[q] = \{\, q' \in Q \mid q \sim q' \,\}$, and $\widetilde{Q} = \{\, [q] \mid q \in Q \,\}$ is the set of all equivalence classes modulo $\sim$.*

Def. 6 defines an equivalence relation on states Q, i.e. if two states are equivalent then they belong to the same class of states.

Definition 7. *Let $S = \langle Q_S, q_0, \Sigma, \rightarrow_S\rangle$ be an LTS and let $\sim \subseteq Q_S \times Q_S$ be an equivalence relation. The* quotient LTS *of S modulo $\sim$ is $\widetilde{S} = \langle \widetilde{Q}_S, \tilde{q}_0, \Sigma, \rightarrow/\sim\rangle$, where $\rightarrow/\sim = \{\langle [q], \sigma, [q_1]\rangle \mid \exists q' \in [q], q'_1 \in [q_1] : q' \xrightarrow{\sigma} q'_1\}$, $\tilde{q}_0 = \{[q_0] \mid q_0 \in q_0\}$.*

Def. 7 defines an quotient modulo operation, which is an equivalence relation on the state set.

3. Input-output conformance relations

Black-box conformance testing [15] is typically based on a specification model to uncover faults in an implementation. There are several conformance relations that are based on black-box testing, however their conformance principles differ from each other.

Some conformance relations require partial conformance, e.g. the *input-output conformance relation* (IOCO) [7]. In IOCO, an implementation is required to have a subset of the specified outputs after the execution of each trace for all possible traces in the specification. An important point here is that the IOCO relation does not put any requirement on the inputs of the implementation.

Definition 8. *For a given implementation G and specification K with equal sets of labels, G is said to be IOCO with respect to K if*

$$\forall t \in \boldsymbol{traces}(K) : \boldsymbol{outs}(\boldsymbol{after}(G,t)) \subseteq \boldsymbol{outs}(\boldsymbol{after}(K,t)) \tag{7}$$

The requirement on inputs is addressed by the *input-output conformance simulation relation* (IOCOS) [5]. The IOCOS relation tests an implementation for unexpected outputs, and the rejection of mandatory inputs [9]. In other words, the implementation must have all the specified inputs and must not have any unspecified outputs after the execution of each trace for all possible traces per the specification.

Definition 9. *For two LTS S and P, a relation $R \subseteq (Q_P \cup Q_S) \times (Q_P \cup Q_S)$ is an* iocos-relation *if for any $\langle p, q\rangle \in R$ it holds that [5]:*

(i) $\boldsymbol{ins}(q) \subseteq \boldsymbol{ins}(p)$

(ii) $\forall a \in \boldsymbol{ins}(q)$ *and* $\forall p' : p \xrightarrow{a} p'$, *there exist* $q \xrightarrow{a} q'$ *such that* $\langle p', q'\rangle \in R$

(iii) $\forall !x \in \boldsymbol{outs}(p)$ *and* $\forall p' : p \xrightarrow{!x} p'$, *there exists* $q \xrightarrow{!x} q'$ *such that* $\langle p', q'\rangle \in R$

Def. 9 defines the three conditions that are required to be fulfilled by a state pair in an implementation and specification to be iocos-related.

Definition 10. *IOCOS relation* $= \cup \{R \subseteq (Q_P \cup Q_S) \times (Q_P \cup Q_S) \mid R$ *is an iocos-relation*$\}$, *and we write,* $\langle p, q\rangle \in IOCOS$ *[5].*

Def. 10 defines IOCOS over *all* pairs of states, even state-pairs that are in practice unreachable. However, in a practical setting, only state-pairs actually reachable in the implementation and defined by the specification are of interest. Thus, Lemma 1, below, shows that when the initial pair $\langle p_0, q_0\rangle$ fulfills Def. 10, then for all mutual traces in $G||K$, the reached state-pairs are IOCOS.

Lemma 1. *For given non-deterministic LTSs* $K = \langle Q_K, q_0, \Sigma, \rightarrow_K\rangle$ *and* $G = \langle Q_G, p_0, \Sigma, \rightarrow_G\rangle$. $\langle p_0, q_0\rangle \in IOCOS \implies \forall t \in \boldsymbol{traces}(G||K), \forall p \in \boldsymbol{after}(G,t), \exists \langle p, q\rangle \in \boldsymbol{after}(G||K, t) : \langle p, q\rangle \in IOCOS$

Proof. This is proved by induction on a trace t.

- *Base case:* For $t = \epsilon$, $\langle p_0, q_0\rangle$ is reached, and we know that $\langle p_0, q_0\rangle \in IOCOS$.
- Inductive step: Assume for $t \in \mathbf{traces}(G||K)$ that $\exists \langle p, q\rangle \in \mathbf{after}(G||K, t) : \langle p, q\rangle \in IOCOS$. Now, for $a \in \Sigma$ assume $\forall q' \in \mathbf{after}(K, ta) : \langle p', q'\rangle \notin IOCOS$. From Def. 9, this means that $\langle p, q\rangle \notin IOCOS$, which contradicts the assumption that $\langle p, q\rangle \in IOCOS$. Therefore, $\langle p', q'\rangle \in IOCOS$.
- Conclusion: According to the principle of induction, $\forall t \in \mathbf{traces}(G||K)$, $\forall p \in \mathbf{after}(G, t)$, $\exists \langle p, q\rangle \in \mathbf{after}(G||K, t) : \langle p, q\rangle \in IOCOS$.

□

Lemma 1 shows that for any two given LTSs if the initial state-pair is IOCOS then the IOCOS relation also holds for all other reachable state-pairs. Now we give a new formal definition of IOCOS based on Lemma 1, that is more useful in practical settings compared to Def. 10.

Definition 11. *For an implementation G and a specification K with initial states $p_0 \in Q_G$ and $q_0 \in Q_K$, respectively, we say that G is IOCOS with respect to K, denoted G IOCOS K, if* $\langle p_0, q_0\rangle \in IOCOS$.

For testing safety properties, the IOCOS relation is found to be inadequate as some faults can go undetected [16]. For example, due to the subset requirement on outputs, missing outputs in the implementation that are prescribed by the specification will not be registered as faults. For safety properties these undetected faults can have dangerous consequences.

Due to this, the *safe input-output conformance simulation relation* (safe-IOCOS) [6] was introduced. safe-IOCOS requires equality for traces associated with safety behaviors, common between the implementation and specification. The equality requirement enables detection of faults related to safety outputs/inputs that are prescribed by the specification.

The safe-IOCOS relation is in [6] defined for deterministic LTSs, while here it is extended to cover non-deterministic LTSs.

Definition 12. *Given LTSs G and K, and safety actions Σ_x, a relation $R \subseteq (Q_G \cup Q_K) \times (Q_G \cup Q_K)$ is safe-IOCOS if $\forall \langle p, q \rangle \in R$:*

(i) $\boldsymbol{act}(p) \cap \Sigma_x = \boldsymbol{act}(q) \cap \Sigma_x$;

(ii) $\forall \sigma \in \boldsymbol{act}(p), \forall p' : p \xrightarrow{\sigma} p', \forall q' : q \xrightarrow{\sigma} q' \wedge \langle p', q' \rangle \in R$

The union over all such R relations is said to be the safe-IOCOS relation between G and K.

4. Bisimulation

Typically, multiple safety behaviors are associated with each nominal activity of a production system. Depending on the severity of the safety action, corrective safety actions affect the associated nominal activity of the production system. Hence, during testing, such safety behaviors must be tested in regard to each nominal activity as their absence makes that specific nominal activity unsafe. This means that the test engineer cannot skip a test of a safety scenario that has already been tested for another nominal operation.

The safe-IOCOS testing relation requires the implementation to be tested for all possible traces in the specification that are composed of safety behaviors, i.e. of actions from Σ_x. And to be safe-IOCOS, for each tested trace, the implementation must exhibit the exact specified safety behavior (both inputs and outputs). This repetitive testing of common safety behaviors adds unnecessary time to the testing process. Especially, if the system under test has a large number of traces.

The root of the problem lies in the very nature of how behaviors are prescribed in the specification used for implementation. If this is don in such a way that skipping certain traces compromises test coverage, then the approach used to prescribe the repetitive behaviors in the specification needs to be modified. And to remedy this the bisimulation relation [12, 17] can help.

A bisimulation relation is an equivalence relation that considers as equivalent two states with the same future behaviour. Such states are said to be *bisimilar*.

Definition 13. *Let $S = \langle Q_S, q_0, \Sigma, \rightarrow_S \rangle$ and $G = \langle Q_G, p_0, \Sigma, \rightarrow_G \rangle$ be two LTSs. A relation $\approx \subseteq (Q_S \cup Q_G) \times (Q_S \cup Q_G)$ is said to be a* bisimulation *between S and G if, $\forall \langle p, q \rangle \in \approx$ and $\forall \sigma \in \Sigma$:*

(i) $\forall \sigma \in \boldsymbol{act}(p), \forall p' : p \xrightarrow{\sigma} p', \exists q \xrightarrow{\sigma} q' : \langle p', q' \rangle \in \approx$

(ii) $\forall \sigma \in \boldsymbol{act}(q), \forall q' : q \xrightarrow{\sigma} q', \exists p \xrightarrow{\sigma} p' : \langle p', q' \rangle \in \approx$

S and G are bisimilar if there exists a bisimulation $\approx$ between S and G such that $\langle p_0, q_0 \rangle \in \approx$.

A *bisimulation reduction* is a method to reduce the state-space of an LTS by merging bisimilar states. The traces of the original LTS are preserved by the reduced one. Thus, for safe-IOCOS, traces with the same safety behaviors can converge to a single state. Now, this reduced specification can be used to implement the system. If the systems are implemented using a reduced specification, the common safety behaviors need to be tested only for a single trace to confirm their presence per specification. And if the test passes, then for the rest of the traces these safety behaviors can be omitted.

Systems that are implemented according to the reduced specification not only enable test case reduction but also preserves the safe-IOCOS relation, as shown by the following lemma.

Lemma 2. *Given non-deterministic implementation* $G = \langle Q_G, p_0, \Sigma, \rightarrow_G \rangle$ *that is constructed using non-deterministic specification* $K = \langle Q_K, q_0, \Sigma, \rightarrow_K \rangle$. *Let* $\widetilde{K} = \langle \widetilde{Q}_K, \tilde{q}_0, \Sigma, \rightarrow_{\widetilde{K}} \rangle$ *be the specification reduced via bisimulation on* K *such that* $K \approx \widetilde{K}$ *and* $\widetilde{G} = \langle \widetilde{Q}_G, \tilde{p}_0, \Sigma, \rightarrow_{\widetilde{G}} \rangle$ *be the implementation based on* $\widetilde{K}$ *such that* $G \approx \widetilde{G}$ *then the following condition hold:*

- G *safe-IOCOS* $K \implies \widetilde{G}$ *safe-IOCOS* $\widetilde{K}$

Proof. Assume that G safe-IOCOS K. Then for all $\langle p, q \rangle \in R \subseteq (Q_G \cup Q_K) \times (Q_G \cup Q_K)$ there $\exists a \in \mathbf{act}(q)$ such that $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$, and $\langle p', q' \rangle \in R$. From Def. 12 this means that $\mathbf{act}(p) \cap \Sigma_x = \mathbf{act}(q) \cap \Sigma_x$.

Assume now that $\widetilde{G}$ not safe-IOCOS $\widetilde{K}$. Then there $\exists \langle \tilde{p}, \tilde{q} \rangle \in \widetilde{R} \subseteq (\widetilde{Q}_G \cup \widetilde{Q}_K) \times (\widetilde{Q}_G \cup \widetilde{Q}_K)$ such that for some $a \in \mathbf{act}(\tilde{q})$, $\tilde{p} \xrightarrow{a} \tilde{p}'$ and $\tilde{q} \xrightarrow{a} \tilde{q}'$ but $\langle \tilde{p}', \tilde{q}' \rangle \notin \widetilde{R}$. From Def. 12 this means that $\mathbf{act}(\tilde{p}') \cap \Sigma_x \neq \mathbf{act}(\tilde{q}') \cap \Sigma_x$.

However, from Def. 7 we have that since $\widetilde{G}$ and $\widetilde{K}$ are quotient LTSs of G and K, respectively, $\mathbf{act}(\tilde{p}') = \mathbf{act}(p')$ and $\mathbf{act}(\tilde{q}') = \mathbf{act}(q')$ which contradicts the assumption. □

Lemma 2 proves that if a non-deterministic G is safe-IOCOS with respect to non-deterministic K, then $\widetilde{G}$, which is constructed by using bisimilulation reduction $\widetilde{K}$, is also safe-IOCOS with respect to $\widetilde{K}$. Thus, since the the bisimilar $\widetilde{K}$ has usually less states compared to K, then the implementation based on $\widetilde{K}$ has less states compared to G, which results in reducing the test cases. The following example further illustrates how bisimulation preserves the safe-IOCOS relation.

4.1. Example

Consider the implementation G that is based on the specification K in Fig. 2 with safety actions prefixed with *!*. The states in the implementation G are enumerated using the prefix p, while the states in specification K, are enumerated using the prefix q. This same way of enumerating states is followed throughout the paper.

First, G is tested with respect to K for the safe-IOCOS relation for the prescribed traces $a.!x.!y$

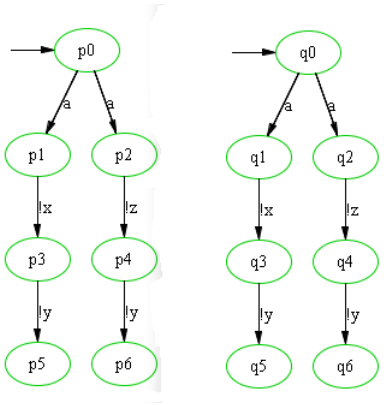

Figure 2: Implementation G (left), specification K (right).

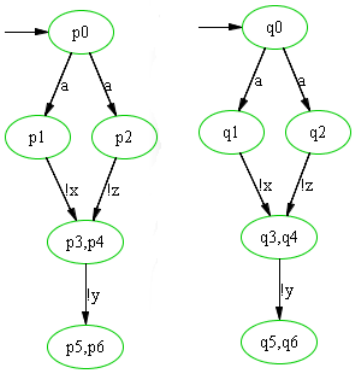

Figure 3: Implementation $\widetilde{G}$ (left), specification $\widetilde{K}$ (right).

and $a.!z.!y$ in K. As can be seen, after executing the action a in G both the safety behaviors, i.e. $!x.!y$ and $!z.!y$, follow. Thus, the safe-IOCOS relation is validated as all the traces composed of prescribed safety behaviors $!x.!y$ are present in K.

Fig. 3 shows $\widetilde{K}$ that is a bisimulation reduction of K, and the implementation $\widetilde{G}$, which is constructed based on $\widetilde{K}$. Testing $\widetilde{G}$ for safe-IOCOS with respect to the specification $\widetilde{K}$ reveals that the safe-IOCOS relation is valid also in this case. This is because after executing the action a, the presence of the safety behaviors $!x.!y$ and $!z.!y$ is confirmed. And due to bisimulation reduction, the same behavior $!y$ need to be tested only for one trace and can be omitted for the other.

$\widetilde{K}$ and $\widetilde{G}$ are non-deterministic, hence, when the action a is executed with the intention to test $!x$ it may never happen. This is because, the $\widetilde{G}$ may always choose its right non-deterministic branch and the same is true if the intention is to test $!z$ after a. Thus, it is better to remove non-determinism as it strengthens the testing procedure by giving the test engineer more control on what to expect. This can be achieved by the subset construction method.

5. The subset construction method

In addition, to repetitive testing of the same behaviors, the presence of non-determinism in the implementation can further lengthen the whole testing process. Since the implementation is a black-box, the exact state reached after the execution of a trace is unknown. Thus, though testing can never guarantee a fault-free system, the added mystery of non-determinism does not help in raising the confidence of a fault-free system.

Furthermore, non-determinism can cause the same trace to pass on some test runs, while it fails on other, seemingly identical test runs. This is because different states may be reached after each execution. The faults uncovered due to this sporadic behavior could be false positives, i.e. the implementation is not actually faulty but the test results say otherwise. And the same is true the other way around, that is, for false negatives. This further amplifies the uncertainty associated with the testing procedure and the implementation cannot be trusted to be safe-IOCOS.

Both problems mentioned above can be present in an implementation if the specification that was used by the engineers to implement the system is non-deterministic. The implementation as discussed above is a closed-loop system of a controller with a real physical plant.

Also, depending on the chosen traces, faults may get uncovered on the first few test runs or it can take forever especially in the presence of non-determinism. This so since the test engineer or a randomized test generator can pick or choose any random trace based on the specification.

The subset construction method [13] transforms a non-deterministic LTS to a deterministic LTS by merging states that have the same past behavior and this transformation preserves the traces of the original system.

Definition 14. *Let $N = \langle Q_N, n_0, \Sigma_N, \rightarrow_N \rangle$ be a non-deterministic LTS and $N' = \langle Q_{N'}, n_0, \Sigma_N, \rightarrow_{N'} \rangle$ be a deterministic LTS. The states of N', i.e. $Q_{N'}$, are computed via the subset construction method, resulting in:*

- *$Q_{N'}$ is a non-empty set of states such that for all $q \in Q_{N'}$, we have $q \subseteq Q_N$*
- *$\rightarrow_{N'}$ the transition function of N' that maps a state $q_{N'} \in Q_{N'}$ and an action $a \in \Sigma_N$ to the set, **after**$(q_{N'}, a) = q'_{N'}$ where $\forall q \in q_{N'} \; \exists q \xrightarrow{a} q'$ with $q' \in q'_{N'}$*

As the subset construction method converts the non-deterministic LTS to a deterministic one, it can be used to reduce a given specification. This reduced specification is then used for implementation, as well as for testing.

Using the reduced specification will remove uncertainty attached to the testing procedure. This so since a specific state is reached after each trace. Thus, the number of test executions are decreased and consequently it can help to reduce testing time. Moreover, the safe-IOCOS relation can be validated with confidence due to the absence of non-determinism. The following lemma proves that subset construction preserves safe-IOCOS.

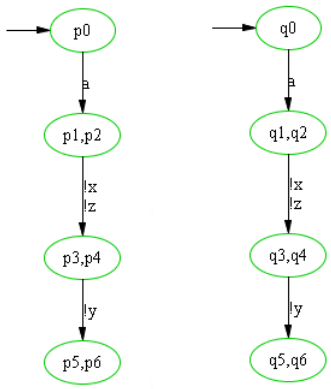

Figure 4: Implementation G_S (left), specification K_S (right)

Lemma 3. *Given non-deterministic implementation* $G = \langle Q_G, p_0, \Sigma, \rightarrow_G \rangle$ *that is constructed according to non-deterministic specification* $K = \langle Q_K, q_0, \Sigma, \rightarrow_K \rangle$. *Let* $K_S = \langle Q_{K_S}, q_{S_0}, \Sigma, \rightarrow_{K_S} \rangle$ *be the deterministic specification that is constructed via subset construction on* K *and* $G_S = \langle Q_{G_S}, p_{S_0}, \Sigma, \rightarrow_{G_S} \rangle$ *be the implementation constructed according to* K_S. *Then the following condition holds:*

$$G \text{ safe-IOCOS } K \implies G_S \text{ safe-IOCOS } K_S$$

Proof. Assume G safe-IOCOS K but G_S is not safe-IOCOS with respect to K_S. Then there is an action $a \in \Sigma$ such that a trace $ta \in \mathbf{traces}(K_S)$ but $ta \notin \mathbf{traces}(K)$ or vice versa. This implies that $\mathbf{traces}(K) \neq \mathbf{traces}(K_S)$. But the subset construction method preserves the traces, i.e. $\mathbf{traces}(K) = \mathbf{traces}(K_S)$. This contradicts the assumption, hence G_S is safe-IOCOS K_S. □

The following example will illustrate Lemma 3.

5.1. Example

Consider Fig. 2, which shows the implementation G that is safe-IOCOS according to specification K. Fig. 4 on the other hand shows K_S that is constructed via subset reduction on K, and the implementation G_S, which is based on K_S.

When G_S is tested for safe-IOCOS, the presence of the safety actions, i.e. *!x*, *!z*, and *!y* after the action a, are validated per K_S. Hence G_S safe-IOCOS K_S.

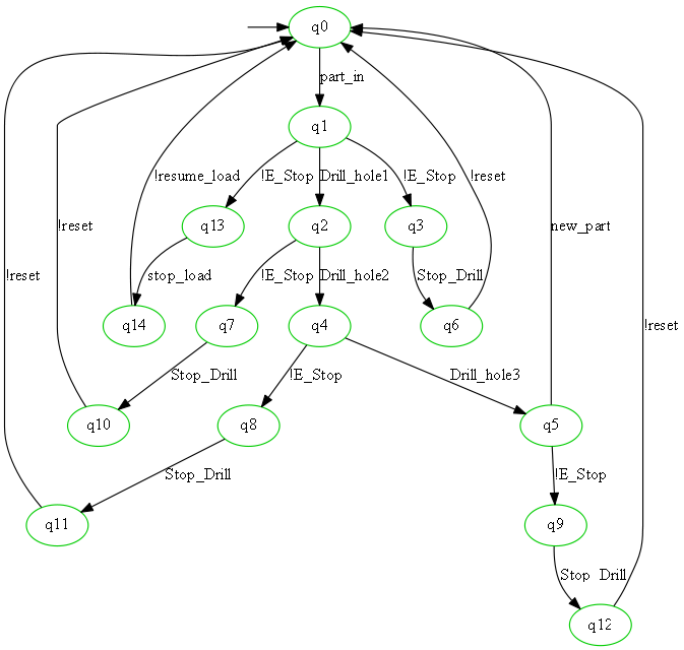

Figure 5: Original specification

The example above shows how subset construction method helps in strengthening the testing procedure by removing non-determinism. Hence, the test engineer knows that after the execution of a trace a specific state is reached, thus the output generated by the implementation can be trusted.

The use of bisimulation on the other hand as shown in Sec. 4 helps in reducing test cases for traces with repetitive behaviors.

When we combine bisimulation and subset construction to maximally reduce the given specification, the resultant specification will be free of both repetitive behaviors as well as non-determinism. This reduction preserves the traces of the system and consequently preserves safe-IOCOS, as shown by Theorem 1 presented in the next section.

6. Approach

The problems related to repetitive testing of same behavior and non-determinism mentioned above can be addressed by reducing the given specification using bisimulation reduction and the subset construction method before implementing the system. The same reduced specification is then used by the test engineer for safe-IOCOS val-

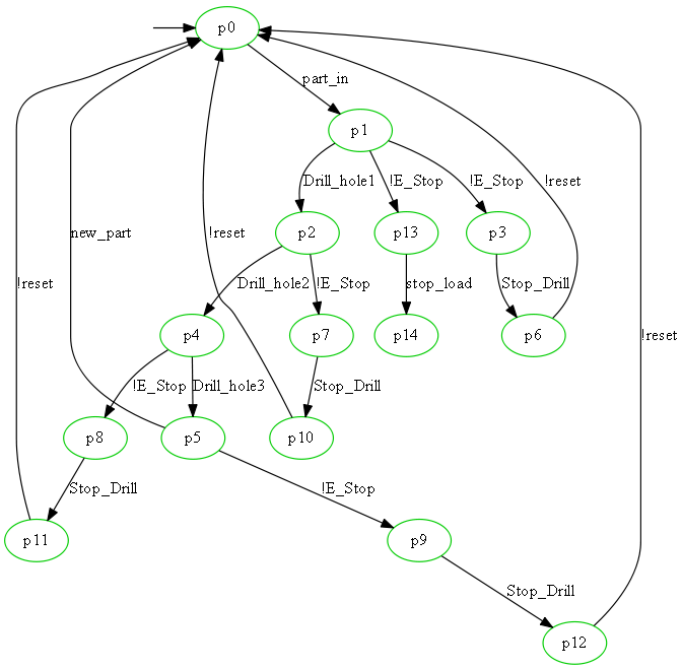

Figure 6: Implementation based on original specification

idation, which enables the engineer to execute less test cases to uncover faults (if any).

Lemma 2 and Lemma 3 above show how applying bisimulation and subset construction preserves safe-IOCOS individually. Now, the following theorem proves that applying both bisimulation and subset construction together on a given specification preserves safe-IOCOS.

Theorem 1. *Let $K = \langle Q_K, q_0, \Sigma, \rightarrow_K \rangle$ be a non-deterministic specification and let $G = \langle Q_G, p_0, \Sigma, \rightarrow_G \rangle$ be the implementation based on K. Then the constructed G_{BS} after reducing K to K_{BS} via bisimulation and subset construction method preserves the safe-IOCOS relation.*

$$G \text{ safe-IOCOS } K \implies G \text{ safe-IOCOS } K_{BS}$$

Proof. Assume G safe-IOCOS K. Now we show that G_{BS} safe-IOCOS K_{BS}. This is shown by showing that the $\mathbf{traces}(K_{BS}) = \mathbf{traces}(K)$.

From Lemma 2, we know that bisimulation preserves the traces of the system i.e. $\mathbf{traces}(K) = \mathbf{traces}(K_B)$. Similarly, from Lemma 3, we know that subset construction preserved traces as well this means, $\mathbf{traces}(K_S) = \mathbf{traces}(K)$. This implies $\mathbf{traces}(K_B) = \mathbf{traces}(K_S)$, which means that $\mathbf{traces}(K_{BS}) = \mathbf{traces}(K)$. So we can write, G safe-IOCOS K_{BS}. □

6.1. Reducing the specification before implementation

Now, to reduce the given specification, the approach is as follows:

- The first step is to apply bisimulation reduction on the given specification. The reason for applying bisimulation first is that the computational complexity of bisimulation is polynomial [17], while the subset construction has exponential (in the number of states) complexity [13]. Bisimulation reduction merges states, which results in the *reduced specification* denoted by K_B. This reduction not only reduces the number of states but also preserves the safe-IOCOS relation as shown in Lemma 2.
- In the second step, K_B is further reduced by applying the subset construction method, resulting in the *maximally reduced* specification K_{BS}. Again, K_{BS} preserves safe-IOCOS as shown in Lemma 3. The transformation of non-deterministic LTS to deterministic LTS is done to avoid uncertainty attached to test cases that are plagued with non-determinism. This so since the safe-IOCOS relation requires equality for safety actions, which is harder to identify if multiple states can be reached on the execution of a single trace. For example, if a test engineer is testing the left branch in the LTS and expecting a certain safety behavior, while the implementation chooses the right branch in the LTS that has other prescribed safety behavior. Then this would register as a fault. However, in reality this is not an actual fault because the correct behavior is present in the left branch. Thus, the validity of safe-IOCOS may never be confirmed in the presence of non-determinism. In contrast, with deterministic behavior, the test engineer would know that after executing a certain trace, a specific state is reached and if the expected safety behavior is different from the prescribed one then it is an actual fault.

After reduction of the given specification, the maximally reduced specification, K_{BS}, is given to

the engineer who is responsible to implement the system. Furthermore, the resultant specification K_{BS} does not affect the safe-IOCOS properties in general, as shown by Theorem 1.

6.2. Testing based on maximally reduced specification

Now, the system that has been implemented with respect to K_{BS} is tested by another engineer who is typically not involved in the implementation. During testing, if the test engineer applies various traces using K_{BS} randomly, the whole purpose of reducing the specification goes in vain and the test cases cannot be reduced.

Thus, to exploit the full advantage of the maximally reduced specification, an example is presented that shows how testing should be done based on K_{BS}. The example first shows how implementation and testing is done using the original specification, and then with the maximally reduced specification.

6.3. Example

Based on the given specification in Fig. 5, an implementation of the system is carried out, which is shown in Fig. 6. In terms of faults, the implementation has in state p_{14} a missing safety action *!resume_load* that is prescribed by the specification in state q_{14}.

Both the implementation from the non-reduced and the maximally reduced specifications are used, and tested according to the respective specifications. Showing both approaches helps in comparing them, and also highlights the advantages of using the maximally reduced specification.

Note that the implementation model (Fig. 6) is presented only for illustration of the issues, it is not relied upon neither during reduction nor testing. The innards of the implementation are unknown to the tester, as this is a black-box testing approach.

6.4. Implementation based on original specification

The original specification is shown in Fig. 5. The nominal operations are related to drilling, and consist of the actions *part_in*, *Drill_hole1*, *Drill_hole2*, and *Drill_hole3*. In terms of safety, each state in the specification has an emergency stop sequence associated with it. This safety sequence gets triggered via the *!E_Stop* action, which as a result should stop the drilling operation via the *Stop_Drill* action. To restore the nominal drilling operation, the *!reset* action needs to be executed from the states that are reached after the *Stop_Drill* action.

Furthermore, the action *!E_Stop* in state q_1 should also stop the part loading operation via the action *stop_load*. After the *stop_load* action, the nominal loading operation can be restored via the *!resume_ load* action. This safety sequence related to the loading operation adds non-determinism in the specification.

6.5. Testing based on original specification

To test the implementation, the test engineer is given the same specification, and can select inputs to the system to execute different traces to validate the safe-IOCOS relation.

Since the given specification has multiple repetitive safety behaviors in the form of various inputs and outputs related to the emergency stop sequence, the test engineer must execute each trace that contains *Stop_Drill.!reset* to uncover faults. This is due to the fact that the implementation is a black-box and the test engineer does not know which trace is faulty and which is not.

Also, only after testing each trace containing the *Stop_Drill.!reset* trace, can the test engineer validate the safe-IOCOS relation for these traces. Thus, the test engineer cannot skip any trace containing the *Stop_Drill.!reset* trace even though these may have been tested before for another trace. This so since if the *Stop_Drill.!reset* trace goes missing in any trace that is not tested then this fault cannot be uncovered.

Furthermore, in state q_1 of the specification, the execution of the action *!E_Stop* might transit to either q_3 or q_{13}. This means that if the test engineer chooses to execute the trace *part_in.!E_Stop*, this takes the implementation from state p_0 to p_1, and then to p_3 or p_{13}. Due to this non-determinism, the fault related to the missing *!resume_load* in state p_{14} may never get uncovered, because it is

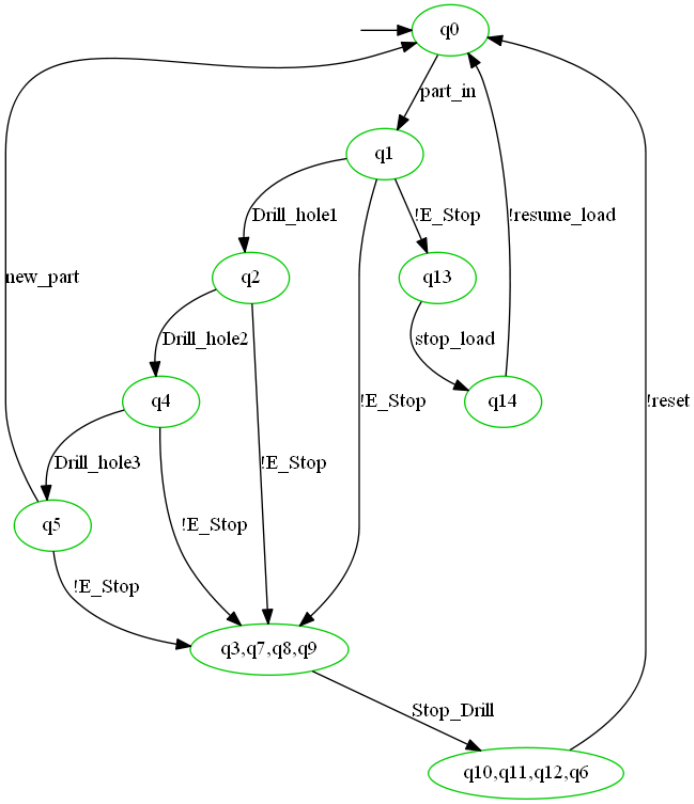

Figure 7: Reduced specification using bisimulation K_B

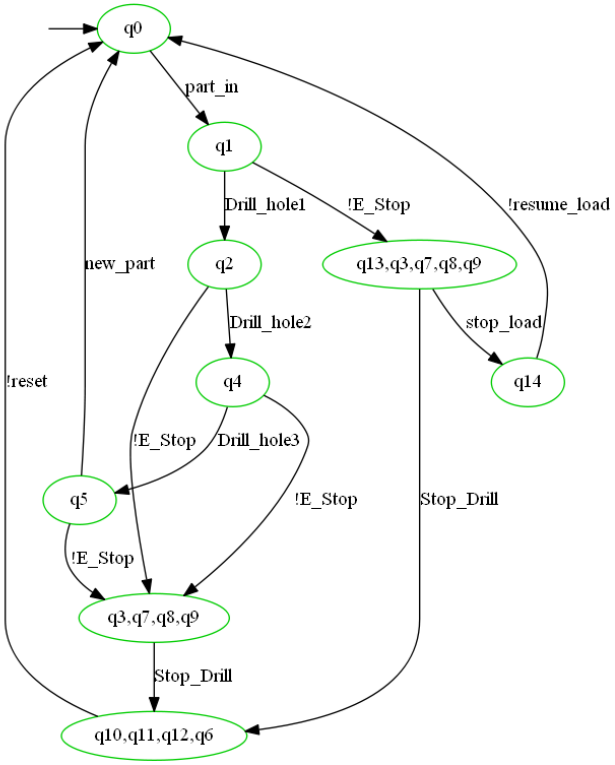

Figure 8: Maximally reduced specification K_{BS}

quite possible that each time the trace *part_in. !E_Stop* is executed, state p_3 is reached.

6.6. Implementation based on maximally reduced specification

To show the viability of the presented approach, the specification illustrated in Fig. 5 is reconsidered. According to the presented approach, the given specification is first reduced using bisimulation reduction. The resultant specification denoted K_B is depicted in Fig. 7. The bisimulation reduction merges states (q_3, q_7, q_8, q_9), and $(q_{10}, q_{11}, q_{12}, q_6)$, as these states have the same future behavior in terms of the actions *Stop_Drill* and *!reset*, respectively.

After the application of bisimulation reduction, the reduced specification K_B is further reduced using subset construction, which gives the maximally reduced specification K_{BS}, illustrated in Fig. 8. The application of subset construction removes the non-determinism from state q_1, which was due to the safety action *!E_Stop*.

Now, the maximally reduced specification K_{BS} leads to the implementation depicted in Fig. 9. This implementation has a fault in terms of a missing safety action *!resume_load* in state p_8, which will in the following section be uncovered using the proposed testing steps.

6.7. Testing with respect to maximally reduced specification

After reducing the given specification using bisimulation and subset construction, i.e. K_{BS}, the implementation illustrated in Fig. 9 is now tested with respect to K_{BS} using the steps described above.

The first step is to identify the safety related traces and as can be seen in Fig. 8, except for the nominal trace *part_in.Drill_hole1.Drill_hole2.Drill_hole3.new_part*, each trace in K_{BS} contains the safety action *!E_Stop*.

Then in the second step, the trace *part_in.!E_Stop.Stop_Drill.!reset* is selected, and the safe-IOCOS relation is validated for this trace.

The rest of the traces containing *!E_Stop* that

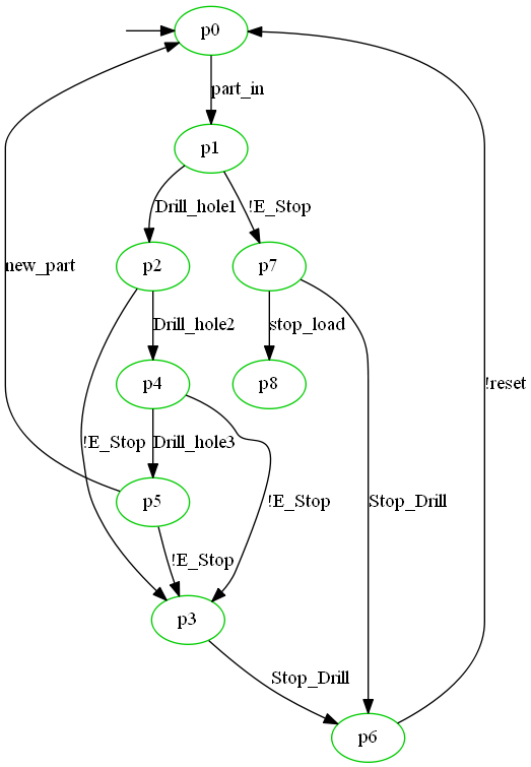

Figure 9: Implementation based on maximally reduced specification

converge to the state (q_3, q_7, q_8, q_9) are then tested. After reaching this state, the remaining traces composed of *Stop_Drill.!reset* can be omitted from testing as this is already tested for the trace *part_in. !E_Stop.Stop_Drill.!reset*, hence time is saved.

In the fourth step the remaining trace i.e. *part_in. !E_Stop.stop_load.!resume_load* per K_{BS} is tested. The execution of this trace uncovers the fault related to the missing *!resume_load* in state p_8. An important point here is that each time this trace is executed it will uncover the fault, as the non-determinism that was present in the original specification K has been removed by the subset construction method.

6.8. Implication of using the reduced specification

From the example above, it is clear that if the given implementation is tested using the original (non-reduced) specification (Fig. 5), then the test engineer has to execute each transition to uncover errors. And among these transitions many transitions are repeatedly tested. This repeated testing of the same transitions increases the testing time unnecessarily.

Similarly, non-determinism makes testing inefficient. This because the implementation is a black-box and the test engineer never knows which state has been reached in the implementation after the execution of an action.

However, if the test engineer uses the reduced specification (Fig. 8), then due to the application of bisimulation reduction, transitions having the same actions on them converge to a single state, which allows single execution of each transition. Again, this saves time. Reducing the original specification model using bisimulation enables polynomial reduction in complexity.

Furthermore, the application of subset construction removes non-deterministic behavior from the specification, which enables exponential reduction in complexity of the original specification. This makes the testing procedure more efficient.

7. Conclusion

This paper presents an approach to reduce test cases from a given specification to decrease testing time for the safe-IOCOS testing relation. safe-IOCOS requires equality for traces associated with safety behaviors common between the implementation and specification. And that coincides with real-life practicality, as corrective safety actions in critical situations must be implemented with respect to the design documents (nothing more, nothing less). However, there are some safety behaviors that are commonly attached to many nominal activities. And during testing the same behaviors gets tested repeatedly, which consumes time unnecessarily.

In addition to the repeated testing of some safety behaviors, the presence of non-determinism makes the test results dubious. This so since safe-IOCOS requires equality for traces associated with safety, which in a non-deterministic setting is difficult to validate, as the safety actions might get distributed to multiple branches with the same action sequence. Thus, when an engineer chooses to test one branch per specification, the implementation can choose another branch. This can

then register as a fault per safe-IOCOS definition though it is not an actual fault.

The paper highlights that if the specification is modified before implementation then the above-mentioned problems can be rectified. Hence, to address these issues, the proposed approach exploits the properties of bisimulation and subset construction to reduce a given specification.

The bisimulation reduction method is useful for removing repetitive safety behaviors by merging bisimilar states, which as a consequence helps to reduce test cases. Removing from the given specification non-determinism by the subset construction method rectifies the problem associated with fault detection.

Due to the well-known state-space explosion problem, the computation of bisimulation reduction and the subset construction are hampered by computational complexity. Of interest is therefore to investigate how existing approaches, such a compositional methods [18], can help to mitigate this problem.

Acknowledgements

This work was supported by the Swedish Research Council project SyTeC VR 2016-06204, and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) under project TESTRON 2015-04893, and was partly supported by the Wallenberg AI, Autonomous Systems and Software program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] C. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, SpringerLink Engineering, Springer US, 2009. doi:10.1007/978-0-387-68612-7.

[2] A. Hellgren, M. Fabian, B. Lennartson, Prioritised synchronous composition of inhibitor arc Petri nets, in: Discrete Event Systems, Springer, 2000, pp. 459–466.

[3] P. J. Ramadge, W. M. Wonham, Supervisory control of a class of discrete event processes, SIAM Journal on Control and Optimization 25 (1) (1987) 206–230. doi:10.1137/0325013.

[4] M. Utting, B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. doi:10.1016/B978-0-12-372501-1.X5000-5.

[5] C. Gregorio-Rodríguez, L. Llana, R. Martínez-Torres, Input-output conformance simulation (iocos) for model based testing, in: Formal Techniques for Distributed Systems, Springer, 2013, pp. 114–129. doi:10.1007/978-3-642-38592-69.

[6] A. Khan, M. Fabian, On the safe IOCOS relation for testing safety PLC code, in: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2019, pp. 1449–1452.

[7] G. Tretmans, Test generation with inputs, outputs and repetitive quiescence, 1996, https://research.utwente.nl/en/publications/test-generation-with-inputs-outputs-and-repetitive-quiescence-2 46 (1996).

[8] J. Tretmans, Model based testing with labelled transition systems, in: Formal methods and testing, Springer, 2008, pp. 1–38.

[9] C. Gregorio-Rodríguez, L. Llana, R. Martínez-Torres, Effectiveness for input output conformance simulation iocos, in: International Conference on Formal Techniques for Distributed Objects, Components, and Systems, Springer, 2014, pp. 100–116.

[10] M. Mongiovì, A. Fornaia, E. Tramontana, A network-based approach for reducing test suites while maintaining code coverage, in: International Conference on Complex Networks and Their Applications, Springer, 2019, pp. 164–176.

[11] X. Zhang, J. Chen, C. Feng, R. Li, Y. Su, B. Zhang, J. Lei, C. Tang, Reducing test cases with attention mechanism of neural networks, in: 30th {USENIX} Security Symposium ({USENIX} Security 21), 2021, pp. 2075–2092.

[12] R. Milner, Communication and concurrency, Vol. 84, Prentice hall Englewood Cliffs, 1989.

[13] J. E. Hopcroft, J. D. Ullman, Introduction to automata theory, languages and computation. adison-wesley, Reading, Mass (1979).

[14] R. Malik, K. Åkesson, H. Flordal, M. Fabian, Supremica-–an efficient tool for large-scale discrete event systems, in: IFAC World Congress, Tolouse, France, 2017.

[15] M. Krichen, S. Tripakis, Black-box conformance testing for real-time systems, in: International SPIN Workshop on Model Checking of Software, Springer, 2004, pp. 109–126.

[16] A. Khan, D. Thönnessen, M. Fabian, On-the-fly conformance testing of safety PLC code using quickcheck, in: 2019 IEEE 17th Transactions on Industrial Informatics (INDIN'19), IEEE, 2019, "in press".

[17] J.-C. Fernandez, An implementation of an efficient algorithm for bisimulation equivalence, Science of Computer Programming 13 (2-3) (1990) 219–236.

[18] S. Mohajerani, R. Malik, M. Fabian, Compositional synthesis of supervisors in the form of state machines and state maps, Automatica 76 (2017) 277–281. doi:https://doi.org/10.1016/j.automatica.2016.10.012.