Automated Reconfiguration of Cyber-Physical Production Systems using Satisfiability Modulo Theories

Kaja Balzereit
Fraunhofer IOSB-INA,
Lemgo, Germany
kaja.balzereit@iosb-ina.fraunhofer.de

Oliver Niggemann
Helmut-Schmidt-University
Hamburg, Germany
oliver.niggemann@hsu-hh.de

Abstract

Today, Cyber-Physical Production Systems (CPPS) are controlled by manually written software, therefore the software is not able to adapt to unforeseen faults or external system changes. So even if a fault is diagnosed correctly, the system normally needs to be repaired manually by a human operator. To implement the vision of an autonomous system, besides self-diagnosis a self-reconfiguration or self-repair step is needed. Here reconfiguration is the task of restoring valid system behavior after an invalid system behavior occurred. For complex CPPS, finding such a new valid configuration always requires a system model covering all potential new configurations—only for rather simple systems the possible reconfigurations for a fault can be modeled explicitly. Unfortunately, such models are hardly available for such systems.

To solve this challenge, in this paper, a novel approach for the automated reconfiguration of CPPS is presented. It is based on Satisfiability Modulo Theories and operates on observed system data as well as on information about the system topology. By doing this, the modeling efforts are reduced. To evaluate this new approach, a simulation of such CPPS is used.

1 Introduction

In the last years, there has been a strong trend towards more autonomy in factories [1; 2; 3], e.g. towards self-diagnosis [4] or towards self-optimization [5]. All these use cases have one common denominator: the main bottleneck is always the automation software which can handle only situations which the software developer has foreseen. In this paper, this problem is tackled for one specific use case, reconfiguration or self-repair.

Reconfiguration here denotes the automatic reaction to a system breakdown or other instantaneous problem, i.e. it is the follow-up reaction after an automatic diagnosis and aims at restoring a valid system functioning. Today, after a diagnosis, the software is changed manually which is time consuming and expensive [6]. To the best of our knowledge, there exists no solution for the reconfiguration problem of CPPS that is able to meet the requirements of autonomous factories. The existing approaches mainly handle the parametrization of continuous signals and are at most adaptable to a predefined set of production goals.

This lack of existing solutions is mainly due to some unsolved research questions (RQ):

RQ 1: Weak Fault Models: Of course, automatic reconfiguration can be implemented easily by listing all possible faults and also corresponding reconfiguration steps. But listing all possible faults is not feasible since plants comprise too many and also independently developed modules and faults are often due to unforeseen module interactions. So a solution is needed which only requires weak fault models [7], i.e. models which capture only the normal system behavior and do not model faulty behavior. Such models of the normal behavior only also allow for the integration of models learned from observations [4].

RQ 2: Hybrid Systems: Most existing reconfiguration methods only deal with the reconfiguration, i.e. reparameterization, of continuous variables. Discrete control signals, which often change system structures and the general system behavior, are not handled [8]. So a solution is needed which deals with both discrete and continuous signals, i.e. which is applicable to hybrid systems.

RQ 3: Formal Logic: Since the goal of any reconfiguration is to go from a non-valid system configuration to a valid one, the definition of the concept of system validity must be at the heart of each solution. And this concept of validity must also be efficiently computable. While several formalism for this concept definition are thinkable, most approaches for reconfiguration (and diagnosis) use some kind of formal logic. Since most popular logical calculi such as propositional logic can not model the behavior of modern CPPS, a logical calculus is needed which captures sufficient aspects of modern CPPS for the reconfiguration.

In this paper, a novel approach for the reconfiguration of CPPS is presented. The contribution of this paper is as follows:
(i) A concept of validity for production plant is defined using the logical calculus of Satisfiability Modulo Theories (SMT).
(ii) A new algorithm for the computation of a new valid configuration after the occurrence of a problem is presented. Thus, the reconfiguration can be done dy-

namically during the operation of the CPPS.
(iii) The new solution approach is verified using several faults of a simulated plant.

This paper is structured as follows: First, the related work concerning the reconfiguration of CPPS is discussed. Then, the problem of reconfiguration is formalized and an overview of basic logical formulations is given. Following, the solution approach is presented. and evaluated using a simulation of a CPPS. Finally, a summary and an outlook is given.

2 Related Work

Reconfiguration methods can be separated using the terms *quantitative* and *qualitative*. Quantitative approaches rely on the use of differential equations to precisely describe a system's behavior. Qualitative approaches are based on an abstraction of the system. Therefore not the numerical value of a variable itself but some discrete property is modelled [9].
Quantitative Approaches: 8 [8] differentiate quantitative approaches into multiple-model approaches and adaptive control approaches. When using multiple-model approaches numerous statistical models are used to represent normal as well as faulty system behavior. These models are generated from non-faulty system data and data containing known faults. So for the normal behavior and for a pre-defined set of faults different models are trained, that describe the system behavior for the specific case. Each of these models is linked with a controller suitable to the current system behavior. Thus, when an already known system fault occurs, the specific controller is chosen to reconfigure the system [8].

Adaptive control approaches are based on controllers that can adapt to abrupt changes of system parameters [8]. The idea here is not to train numerous models with linked controllers but to use a single controller that is adapted to the current situation. Adaptive control approaches have successfully been used in multiagent systems [10] and in stochastic nonlinear systems [11].

Quantitative approaches mainly focus on the control of numeric parameters, and thus are mostly just adaptable to a pre-defined set of system states. However, when handling CPPS reconfiguration methods may not be static but need to be adaptable to unforeseeable events.
Qualitative Approaches: The problem of reconfiguration is closely related to the problem of model-based diagnosis (MBD). The task of MBD is given a description of a system and an observation of the current system behavior to find the set of components that causes the fault [12]. 13 [13] pointed out the analogy between diagnosis and reconfiguration and formulated the task of reconfiguration as an extension of the task of diagnosis. One focus of current research in MBD is to automatically diagnose CPPS by integrating data-driven methods [14] and using hybrid modeling strategies [15]. 16 [16] presented an approach on the diagnosis of hybrid systems where the non-linear behavior of the system is handled by a transformation using the max-plus algebra.

In the last years there have been first approaches to the automated reconfiguration of CPPS. 17 [17] presented an approach on the reconfiguration capabilties of autonomous systems using a combination of automated planning and diagnosis. 18 [18] presented a combination of Timed Game Automata and Satisfiability Modulo Theories to solve the problem of dynamic controllability in disjunctive temporal networks, which can be used to model dynamic systems. 19 [19] presented an approach on how to lead a discrete event system into a diagnosable state using automated planning. 20 [20] presented an approach on continuous diagnosis in real-time environments so that reconfiguration can be done in a short time.

Satisfiability theory has already been used for the solution of configuration problems, which task is to find a satisfying matching given a set of constraints [21], and has been evaluated to outperform other approaches [22].

3 Formalization of Reconfiguration Problem

Following, the problem of reconfiguration of CPPS is formalized. Therefore, a description of the current behavior of the CPPS and a definition of valid behavior is needed.

The current state of the CPPS is described by a tuple of variables (x, u, p, t) with x describing state variables, u describing input variables, p describing parameters, and t describing the time. The set of all possible operating points (x, u, p, t) is described by S. The *parametrization* of a CPPS is represented by the current values of the parameters p. A tuple (u, p) where u is an input vector and p is a parameter vector is called a *configuration*.

CPPS underlie different external and internal changes: An *event* $e : S \to S$ is a discrete change in one or more system variables. Events that can be controlled by an external control authority (e.g. a control software or a human operator) are called *actions*. They are used to manipulate the current system behavior. Another kind of events are *errors*. Errors lead to faulty system states. To define errors, first, a definition of valid and invalid system states is needed. This definition separates valid from invalid system behavior. Additionally, the system limitations like valid parameter intervals are modeled by the definition of validity: If parameters lie outside a valid interval or a topology is not valid, the state is labelled invalid. Thus, faulty system states are separated from non-faulty states.

Definition 1. A *definition of validity* is a function

$$w : S \to \{0, 1\}$$

that separates valid from invalid system behavior. Without loss of generality 1 represents valid system behavior and 0 represents invalid system behavior.

Given a definition of validity and an operating point $s \in S$ with $w(s) = 1$, an *error* is an event e that leads to an invalid configuration, so $w(e(s)) = 0$.

The task of reconfiguration is to restore valid system behavior after an error occurred. Formally, a reconfiguration problem can be described as follows:

Definition 2. A *reconfiguration problem* is a tuple (S, w, s_0) where

- S describes a set of possible operating points,
- $w : S \rightarrow \{0, 1\}$ is a definition of validity and
- $s_0 \in S$ with $w(s_0) = 0$ is a current invalid system state.

The solution to a reconfiguration problem is a set of actions $acts = \{e_1, e_2, ...e_s\}$ that changes the system to a valid state, so

$$w(e_s(...e_2(e_1(s_0)))) = 1.$$

The possible actions of a CPPS are described by the possible changes of the system inputs u and the parameters p.

The presented solution approach is based on the transformation of a reconfiguration problem into a problem of logical satisfiability. Therefore, a logical formula is created which is is satisfiable if and only if the configuration is valid. A logical formula consists of single atoms which in turn consist of terms [23]:

Definition 3. Given a set of variables X, a *term* t is either a variable $x \in X$ or has the form $f(t_1, ..t_{n_f})$ with a function symbol f, an $n_f \in \mathbb{N}_0$, and terms t_i of sort σ_i $\forall i \in \{1, 2, ...n_f\}$.

Definition 4. An *atom* a has the form $ps(t_1, ..., t_{n_{ps}})$ with a predicate symbol ps, an $n_{ps} \in \mathbb{N}_0$, and terms t_i of sort σ_i $\forall i \in \{1, 2, ...n_{ps}\}$.

Definition 5. A *formula* φ is either an atom a or has the form $\neg\varphi_0$, $\varphi_0 \vee \varphi_1$, $\varphi_0 \wedge \varphi_1$, $\varphi_0 \Rightarrow \varphi_1$, $\varphi_0 \Leftrightarrow \varphi_1$, $\exists x : \varphi_0$ or $\forall x : \varphi_0$ with x being a variable and φ_0, φ_1 being formulas.

The task of satisfiability theory is to find an assignment to the variables/ predicates of a formula that *satisfies* the formula. An assignment R to all of the variables, function symbols, and predicate symbols of a formula φ *satisfies* the formula if the formula evaluates to true under the assignment. A formula φ is *satisfiable* if at least one assignment exists that satisfies the formula.

For the solution of reconfiguration problems for CPPS, classical satisfiability theory is not expressive enough to model the complexity of CPPS. In classical satisfiability theory only boolean variables, which can be assigned true or false, are allowed. When modeling the reconfiguration problem for CPPS, however, not only boolean variables but also expressions over integer or real variables are needed. Hence, classical satisfiability is expanded by a theory [23].

Definition 6. A *theory* T is a set of of formulas without free variables, i.e. with only variables that are bounded by a quantifier $\exists, \forall$. Given a theory T and a formula φ, φ is *satisfiable modulo* T if $T \cup \{\varphi\}$ is satisfiable.

In this solution approach the extension of Satifiability Modulo the Theory of Linear Arithmetic is used. This theory is based on the arithmetical operations plus $(+)$ and minus $(-)$ applied to numerical variables and the multiplication $(\cdot)$ of a numerical variable with a constant value. The atomic predicates are formed using inequations $(\leq, \geq)$ and equations $(=)$.

4 Solution Approach

The presented solution approach consists of two parts, that will be described in the following sections. First, in section 4.1, the reconfiguration problem is transformed into a problem of Satisfiability Modulo Theories (SMT). Therefore, a logical formula representing the reconfiguration problem is created that is satisfiable if and only if the current system state is valid. This formula is based on the definition of validity, that separates faulty from non-faulty system behavior, and the current system data represented by variables, that are continuously updated. Additionally, information about the system topology, which is used to describe the structure of the CPPS, is inserted into the formula. Therefore, the system topology is defined as follows:

Definition 7. The *topology* of a CPPS is described by a directed multigraph $G = (C, I)$, where

- $C = \{c_1, c_2, ..., c_n\}$ where $n \in \mathbb{N}$ is a set of components and
- $I = \{(c_1, c_2), (c_2, c_1), ...\}$ where each interconnection (c_k, c_l) with $k, l \in \{1, ..., n\}$ describes a connection between two variables within the components $c_k, c_l \in C$. In the following, v_{kl} is used to denote the connection (c_k, c_l).

Second, in section 4.2, the current configuration of the system is checked. To do this, the variables in the formula representing the current system data are updated based on the actual values and the satisfiability of the formula is checked. If the formula is satisfiable, the current configuration is valid and no further actions are needed. Otherwise, the current configuration is invalid and a new, satisfiable variable assignment is searched. If such an assignment exists, the system is reconfigured by updating the current configuration based on the found assignment. Otherwise the system cannot be reconfigured and an error is returned. Whilst the first step only needs to be executed once, the second step is repeated continuously while the system is operating. The variables which are based on the system data are continuously updated and the validity of the current configuration is checked. Thus, a reconfiguration is determined directly when it is needed.

Figure 1 shows the functionality of the presented solution approach for CPPS.

4.1 Creation of Logical Formula

Based on the definition of validity and a description of the system topology, a logical formula is created that represents the reconfiguration problem. For the creation of the logical formula, first atoms which depend on the material flow, the current component states, and the system goal are generated. These atoms contain variables that are updated based on the current system state in the second step. The formula representing the complete reconfiguration problem is created by the logical conjunction of these atoms. Given a current system state, this formula is only satisfiable if the current system state is defined valid.

Material Flow

Components are connected by connecting either their material flow, information flow or energy flow. Here, we

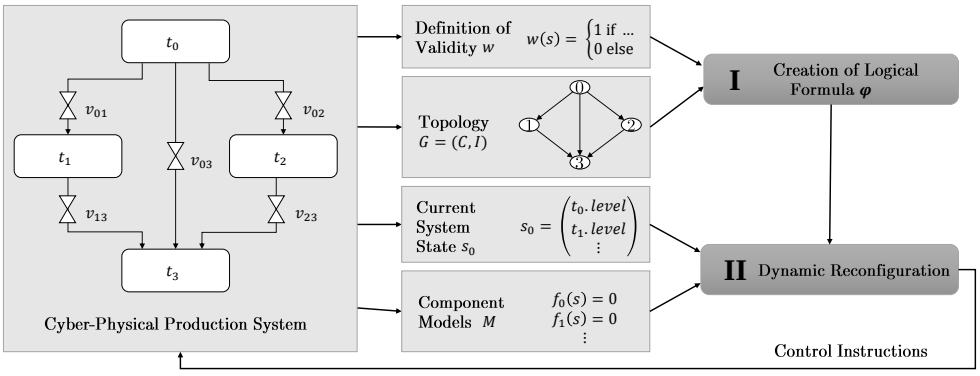

Figure 1: Used Reconfiguration Method. Step I is described in section 4.1, step II is described in section 4.2

focus on the material flow aspect. The material flow in every connection of the system, i.e. in every edge in I, is monitored.

If an interconnection is opened, the flow i_{kl}^{out} out of component $k \in C$ must be equal to the inflow i_{kl}^{in} into component $l \in C$. If this is not satisfied, the connection is faulty. Using SMT this is expressed by

$$b_{kl}^{\text{ok}} \Leftrightarrow (i_{kl}^{\text{in}} = i_{kl}^{\text{out}}) \ \forall v_{kl} \in I \tag{1}$$

where the boolean variable b_{kl}^{ok} indicates whether the connection is non-faulty or faulty. In practice, sensor uncertainties may occur leading to the right side of (1) being false even though the flows are the same. So the equation is relaxed to an expression being false if the deviation of i_{kl}^{k} and i_{kl}^{l} is too large. This is realized by

$$b_{kl}^{\text{ok}} \Leftrightarrow (|i_{kl}^{\text{in}} - i_{kl}^{\text{out}}| \leq \tau) \ \forall v_{kl} \in I \tag{2}$$

with a $\tau > 0$. A connection v_{kl} may only be used if it is non-faulty, i.e. if the variable b_{kl}^{ok} is true. This is expressed by

$$b_{kl}^{\text{use}} \Rightarrow b_{kl}^{\text{ok}} \tag{3}$$

where b_{kl}^{use} is true if the connection is used and false otherwise.

Valid Component Behavior

The correctness of the behavior of every component of the system, so for every $c \in C$, is modelled as follows:

Definition 8. Given a component $c \in C$ a *component model* is an equation system

$$f_c(s) = 0 \tag{4}$$

with $f_c : S \to \mathbb{R}$ and $s \in S$.

Traditionally, such behavior models are modelled manually. Nowadays, such models are learned from observations [4]. Component models in general describe how a component behaves [24], i.e. for every $s \in S$ a component behavior is described by equation system (4). The set of component models for every component in C is described by M. Deviations in a component behavior are called *symptom*. So if $f_c(s_o) = \tau$ and $|\tau| \geq \epsilon$ for a given state $s_o \in S$ with an $\epsilon > 0$, the deviation τ is called a symptom. With the use of component models non-faulty component behavior can be separated from faulty behavior. Therefore, the current system state $s_0 \in S$ is checked. If $f_c(s)$ equals zero, the component behavior is non-faulty, otherwise, i.e. when a symptom occurs, the component behavior is faulty.

In the logical formula this information is represented by boolean variables cb_c that are true when the component's behavior is non-faulty and otherwise are false, so

$$cb_c \Leftrightarrow f_c(s_0) = 0.$$

A connection between two components may only be used if the behavior of the connected components is non-faulty. This is expressed by the atom

$$b_{kl}^{\text{use}} \Rightarrow cb_k \wedge cb_l \ \forall v_{kl} \in I. \tag{5}$$

System Goal

The system goal depends on the application case, and thus needs to be defined by an expert. The goal makes sure that the system's behavior is as required and is not leading to undesired products. This goal specification will be transformed into a logical expression so that it can be added to the logical formula.

For example, a possible goal specification for the CPPS in figure 2 would be, that the inflow into the bottom tank t_3 must be greater than a threshold $\epsilon_{\min}$. Using logical expression this is expressed by

$$inflow_3 > \epsilon_{\min}.$$

Definition 9. The logical formula φ representing the reconfiguration problem is generated by the logical conjunction of the material flow atoms (2), (3), the component atoms (5), and the goal atoms. With this conjunction valid system behavior is separated from invalid behavior; the logical formula is satisfiable if and only if the given system state is valid, i.e.

$$\exists R : R \models \varphi(s_0) \Leftrightarrow w(s_0) = 1.$$

Please note that no assumptions about faulty behavior is made and the model is a weak fault model applicable in realistic scenarios.

4.2 Dynamic Reconfiguration

In the second step the validity of the current configuration is checked. Therefore, the logical formula φ as in Definition 9, the current system state s_0, and the component models are used as input to the reconfiguration algorithm. The current configuration of the system is represented by the values of the input vector and the

parameter vector, which are part of s_0. For the sake of readability this tuple is denoted with *con* and listed as an additional input. The reconfiguration algorithm checks the current configuration and returns a set of actions to restore valid system behavior if necessary.

Algorithm 1 Reconfiguration Algorithm

```
1: procedure SMTReconf(φ, s_0, con, M)
2:     // generate symptoms
3:     for f_c ∈ M do
4:         if |f_c(s_0)| ≥ ε then
5:             cb_c ← ⊥
6:         else
7:             cb_c ← ⊤
8:         end if
9:     end for
10:    // check current configuration
11:    if checkSAT(φ((s_0, cb_c, con))) then
12:        // no reconfiguration needed
13:        return ok
14:    else
15:        // check satisfiability without configuration
16:        if checkSAT(φ((s_0, cb_c))) then
17:            // get new variable assignment
18:            con_new = getModel(φ((s_0, cb_c)))
19:            // get actions needed
20:            acts ← {}
21:            for i ∈ {1, 2, ...length(con)} do
22:                if con[i] ≠ con_new[i] then
23:                    act ← getAction(i, con[i], con_new[i])
24:                    acts ← acts ∪ {act}
25:                end if
26:            end for
27:            return acts
28:        else
29:            // no reconfiguration possible
30:            return error
31:        end if
32:    end if
33: end procedure
```

Algorithm 1 shows how the dynamic reconfiguration of CPPS works. In the description of the algorithm $\varphi(d)$ is used to denote the formula φ where the values for some variables are fixed according to the given values d. The function checkSAT(φ) checks the satisfiability of the formula φ; it returns true if the formula is satisfiable and false otherwise. The function getModel(φ) also is part of the used SMT solver and returns a satisfying assignment to the variables of the formula φ. The function $getAction(i, val_{old}, val_{new})$ returns the action that changes the i-th control variable from val_{old} to val_{new}. $\top$ denotes the truth using the binary truth function, $\bot$ denotes the falsehood.

First (lines 2-9), based on the current system state s_0, the boolean variables cb_c describing the current components behavior are updated using the corresponding component model. If the component model returns a value differing from zero (line 4), the corresponding boolean variable is set false (line 5), otherwise (line 6) it is set true (line 7). Next (lines 10-13), the current configuration represented by the current value of the control variables *con* is checked. Therefore, the corresponding variables in the formula φ are assigned with the current system state, the current values of the component variables, and the control values. Then

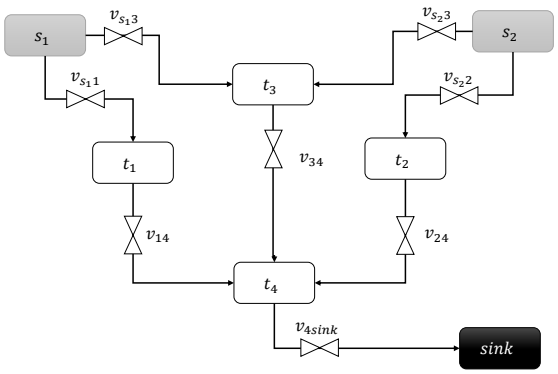

Figure 2: Structure of the Tank-System

the satisfiability of the formula is checked (line 11). If the formula is satisfiable, the current configuration is valid and the reconfiguration algorithm returns ok (line 13). Otherwise (line 14-31), if the current configuration does not lead to a satisfiable formula, the current control instructions are removed and the satisfiability is checked again to find a valid configuration (line 16). If the formula is satisfiable now, a satisfying assignment for the control variables is got using the function getModel() (line 18). To determine the actions needed to change the current configuration into the new configuration, the control variables, that need to be changed, are identified. Thus, for every component of the control variables (line 21) the current value is compared to the new value (line 22). If these values are differing, the corresponding action is identified using the function getAction() (line 23) and added to the set of necessary actions (line 24). After checking all control variables the set of necessary actions is returned (line 27). If the formula is not satisfiable (line 28), no assignment to the control variables can be found that restores valid system behavior. Then the reconfiguration algorithm returns an error (line 30).

5 Evaluation

The presented solution approach is evaluated using an example from process engineering. First, the application case is described. After that, some example cases and their solutions are presented. Finally, the functionality of the SMT-based reconfiguration method is compared to a method that is based on classical constraint satisfaction.

5.1 Description of Application Case

A model of a tanksystem as shown in figure 2 serves as example. The model consists of four tanks ($t_1 - t_4$), two sources (s_1, s_2), and a sink *sink* that are connected by pipes. Every connection can be opened and closed using discrete valves ($v_{s_1 1}, v_{s_1 3}, v_{14}, v_{s_2 2}, v_{s_2 3}, v_{24}, v_{34}, v_{4sink}$). Source s_1 delivers material m_1, source s_2 delivers material m_2. The sources s_1 and s_2 are physically located above the tanks t_1, t_2, t_3, which are as well physically located above tank t_4. Hence, if for example valve v_{14} is opened, the medium flows from tank t_1 to tank t_4. Tank t_4 is connected to a sink.

The task of this system is to mix the materials m_1 and m_2 in a given ratio. In the absence of errors the tanks t_1, t_2 are used to deliver the material, t_4 serves as mixing tank. Tank t_3 serves as a backup solution: If one of the tanks t_1 or t_2 or the corresponding connections are faulty, tank t_3 can be used. Note that in tank t_3 no mixing is allowed.

The topology of the system is represented by a directed graph G consisting of a set of nodes

$$C = \{s_1, s_2, t_1, t_2, t_3, t_4, sink\} \tag{6}$$

and a set of edges

$$I = \{(s_1, t_1), (s_1, t_3), (t_1, t_4), (s_2, t_2), (s_2, t_3), (t_2, t_4), (t_3, t_4), (t_4, sink)\}.$$

v_{kl} with $k, l \in C$ is used to denote the edge $(k, l) \in I$. It is assumed that there are flow sensors at every inflow and at every outflow of each interconnection. With this data the atoms (2) and (3) can be created.

The opening state of every valve can be controlled; thus, the set of possible actions A is described by

$$A = \{(b_{kl}^{\text{use}} := \neg b_{kl}^{\text{use}}) \mid v_{kl} \in I\}. \tag{7}$$

Every component's behavior (in this case every tank's behavior) is modelled by a component model $f_c(s) = 0$. The information from these component models (faulty, non-faulty) is inserted into the atoms (5).

The system's goal is defined as follows: The ratio of material m_1 to material m_2 flowing to tank t_4 shall satify the equation

$$lb \leq \frac{f_1}{f_2} \leq ub, \quad f_2 > 0 \tag{8}$$

with the lower bound $lb \in \mathbb{R}$, the upper bound $ub \in \mathbb{R}$ and f_1, f_2 representing the material flows. These flows depend on which material flows out of tank t_3, hence can be expressed by

$$f_1 = \begin{cases} f_{14} + f_{34} & \text{if } b_{s_1 3}^{\text{use}}, \\ f_{14} & \text{else} \end{cases} \tag{9}$$

with f_{14} and f_{34} representing the flows through the connections (t_1, t_4) and (t_3, t_4). Note that these flows are zero if the corresponding valve is closed. Analogously, f_2 is calculated. Since the division of two variables is not allowed in the theory of Linear Artihmetic, the inequalitions (8) are transformed to

$$l \cdot f_2 \leq f_1 \leq u \cdot f_2. \tag{10}$$

In order to ensure that not every valve is closed (which would be valid behavior at the moment), the conditions

$$f_1 > 0, f_2 > 0 \tag{11}$$

are added to the logical formula. To prevent mixing of the materials in tank t_3, the predicate

$$\neg(b_{s_1 3}^{\text{use}} \wedge b_{s_2 3}^{\text{use}}) \tag{12}$$

is needed.

So, the reconfiguration method is given as input

- a graph $G = (C, I)$ representing the system topology,
- the current system state s_0 including the flow measurements i_{kl}^{in} and i_{kl}^{out} for all $v_{kl} \in I$,
- the component models M,
- the goal specification given by the expressions (9) - (12).

The logical formula, created by the conjunction of the atoms (2), (3), (5), (9), (10), (11), and (12), is solved by the Z3 SMT solver [25]. The system data is created by a simulation written in Modelica [26].

5.2 Solution of Test Cases

Four different error cases are used to evaluate the performance of the presented solution approach.

Non-faulty behavior: If the system behavior is non-faulty, the connections $(s_1, t_1), (t_1, t_4), (s_2, t_2), (t_2, t_4)$, and $(t_4, sink)$ are used. The remaining connections are not used and the corresponding valves are closed.

Error Case 1: Leaking Pipe Between t_1 and t_4: A leaking connection causes a deviation in the values of i_{14}^1 and i_{14}^4. If this deviation is larger than τ, the right side of the atom (2) becomes false. To satisfy the atom b_{14}^{ok} also has to be false. This leads to atom (3) no longer being satisfied so b_{14}^{use} needs to be set false. The valve is closed and the medium no longer flows through the connection ($f_1 = f_{14} = 0$). To satisfy the atoms (10) and (11) an alternative path for material m_1 needs to be found. Setting $b_{s_1 3}$ to true changes the calculation formula for f_1 to $f_{14} + f_{34}$ leading to $f_1 > 0$ if valve v_{34} is opened as well. Tank t_3 is integrated into the process while tank t_1 is no longer used. The connection from tank t_1 is closed.

Error Case 2: Tank t_1 Running Empty: Tank t_1 running empty (e.g. due to a jammed delivery connection) leads to a wrong mixing ratio (10). In order to keep this atom satisfied the inflow f_1 needs to be changed. This can be done by switching $b_{s_1 3}$ from false to true (as in error case 1) so that tank t_3 is used to deliver enough material m_1.

Error Case 3: Jammed Pipe Between t_1 and t_4: This error case is concerned with the pipe between t_1 and t_4 becoming successively jammed more and more until no more material is flowing through it. Thus, the mixing ratio (10) no longer is satisfied. Analogously to error case 1, tank t_3 is used as a bypass.

Error Case 4: Leaking Tank t_2: When a tank is leaking the corresponding component model, which monitors the tank, returns a faulty component behavior. Thus, the right side of the corresponding atoms (5) becomes unsatisfied. To satisfy the atoms the variables b_{24}^{use} and $b_{s_2 2}^{\text{use}}$ need to be set false. So the atoms (10) and (11) are still satisfied the variables $b_{s_2 3}^{\text{use}}$ and b_{34}^{use} need to be set true. The valves $v_{s_2 3}$ and v_{34} are opened and tank t_3 is now used to bypass tank t_4.

Error Case 5: Leaking Pipe Between t_1 and t_4 and Tank t_2 Running Empty This error case is concerned with two single errors: First the pipe between tank t_1 and tank t_4 becomes leaking. Additionally, after some time tank t_2 is running empty. Thus, the leaking pipe is bypassed by tank t_3 and the valves v_{s11} and v_{14} are closed. When tank t_2 runs empty the mixing ratio (10) no longer is satisfied. Since atom (12) prevents tank t_3 from being filled with two different materials no alternative for tank t_2 can be found. Hence, no valid

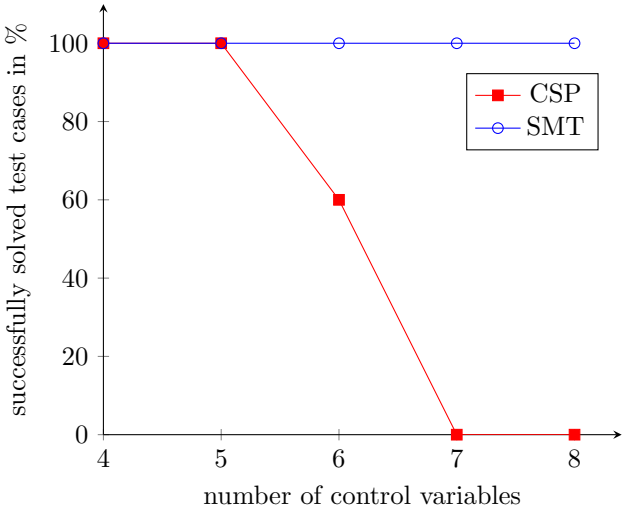

Figure 3: Amount of solved test cases for given number of control variables for constraint satisfaction-based solution approach (CSP) and logic-based solution approach (SMT).

configuration can be found given these errors. External intervention is needed to repair the faults.

5.3 Comparison of Logic-based Approach with Classical Contraint Satisfaction Approach

The performance of the presented solution approach is compared to a solution approach based on classical constraint satisfaction. A Constraint Satisfaction Problem (CSP) consists of a set of variables X, a set of corresponding domains D, and a set of constraints C (as defined by Russell et al. [27]). The reconfiguration problem is transformed into a CSP over finite domains. Thus, the atoms (2), (3), (5) as well as the goal atoms (9), (10), (11), and (12) are transformed to constraints. For the solution of the CSP the solver python-constraint is used [28].

For the comparison a number of problems with varying numbers of control variables are solved by the logic-based approach (SMT) and the CSP-based approach. The result of the comparison is illustrated in figure 3. The number of control variables varies from four to eight. The logic-based approach is able to solve 100% of the test cases whilst the CSP-based approach only solves 100% of these for a small number of variables. With a rising number of control variables the CSP-based approach solves only some of the given test cases. From seven variables the CSP-based approach is no longer able to handle the reconfiguration problems.

6 Summary and Outlook

Reconfiguration is the ability of a system to automatically react to a system breakdown or other occurring problems. In this paper, a new solution approach for the automated reconfiguration of CPPS is presented. The solution approach is based on the transformation of a reconfiguration problem into a problem of satisfiability theory expressed in SMT logic. To reduce the manual modelling efforts and to support learned models, weak fault models which describe normal system behaviour are used (research question 1 from section 1). A logical formula is created, that takes the current system structure, its parametrization, system data, and the production goal into account (research question 3 from section 1). A satisfiable variable assignment for this formula is searched. Based on this assignment the necessary actions to restore valid system behavior are determined so that the system is reconfigured.

The applicability of this approach has been evaluated using several faults of a simulated plant which comprises both continuous and discrete signals (research question 2 from section 1). It is shown that the reconfiguration algorithm can handle all faults for that a reconfiguration is possible. In addition, the solution approach is compared to a solution approach based on constraint satisfaction. It is shown that the logic-based approach outperforms the other approach.

At the moment, the reconfiguration method requires full system observability. In the future, the method will be expanded to be also applicable to partially observable systems. In addition, the presented solution approach will be evaluated using more complex and more realistic examples. Furthermore, more kind of component interactions will be integrated and the chosen reconfiguration actions will be computed in an optimized way.

References

[1] European Factories of the Future Research Association. Multi-Annual Roadmap for the Contractual PPP under HORIZONS 2020, 2013.

[2] Günter Hörcher, Markus Bressner, and Maren Röhm. MANUFUTURE-DE: Ermittlung prioritärer Forschungsthemen für die nachhaltige Ausgestatltung von europäischen Forschungsprogrammen für die produzierende Industrie bis 2030, 2018.

[3] Plattform Industrie 4.0. Aspekte der Forschungsroadmap in den Anwendungsszenarien, 2016.

[4] Oliver Niggemann and Volker Lohweg. On the Diagnosis of Cyber-Physical Production Systems - State-of-the-Art and Research Agenda. In *Twenty-Ninth Conference on Artificial Intelligence (AAAI-15)*, 2015.

[5] Jens Otto, Birgit Vogel-Heuser, and Oliver Niggemann. Automatic parameter estimation for reusable software components of modular and reconfigurable cyber physical production systems in the domain of discrete manufacturing. In *IEEE Transactions on Industrial Informatics*. IEEE, 2018.

[6] Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417, Oct 2008.

[7] Alexander Feldman, Gregory Provan, and Arjan J. C. van Gemund. Solving strong-fault diagnostic models by model relaxation. pages 785–790, 01 2009.

[8] Inseok Hwang, Sungwan Kim, Youdan Kim, and Chze Eng Seah. A survey of fault detection, isolation, and reconfiguration methods. *IEEE Transactions on Control Systems Technology*, 18(3):636–653, 2010.

[9] Jan Lunze. Qualitative modelling of dynamical systems: Motivation, methods, and prospective applications. *Mathematics and Computers in simulation*, 46(5-6):465–483, 1998.

[10] Weinan Gao, Zhong-Ping Jiang, Frank L Lewis, and Yebin Wang. Leader-to-formation stability of multiagent systems: An adaptive optimal control approach. *IEEE Transactions on Automatic Control*, 63(10):3581–3587, 2018.

[11] C. Hua, K. Li, and X. Guan. Event-based dynamic output feedback adaptive fuzzy control for stochastic nonlinear systems. *IEEE Transactions on Fuzzy Systems*, 26(5):3004–3015, Oct 2018.

[12] Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

[13] Judith Crow and John M Rushby. Model-based reconfiguration: Toward an integration with diagnosis. In *AAAI*, pages 836–841, 1991.

[14] Andreas Bunte, Benno Stein, and Oliver Niggemann. Model-based diagnosis for cyber-physical production systems based on machine learning and residual-based diagnosis models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):2727–2735, Jul. 2019.

[15] Alexander Diedrich, Alexander Maier, and Oliver Niggemann. Model-based diagnosis of hybrid systems using satisfiability modulo theory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1452–1459, Jul. 2019.

[16] Gregory Provan. An algebraic approach for diagnosing discrete-time hybrid systems. In Marina Zanella, Ingo Pill, and Alessandro Cimatti, editors, *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4 of *Kalpa Publications in Computing*, pages 37–51, 2018.

[17] Alban Grastien. Self-healing as a combination of consistency checks and conformant planning problems. In *DX@ Safeprocess*, pages 105–112, 2015.

[18] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[19] Hassan Ibrahim, Philippe Dague, Alban Grastien, Lina Ye, and Laurent Simon. Diagnosability planning for controllable discrete event systems. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[20] Alexander Felfernig, Rouven Walter, José A Galindo, David Benavides, Seda Polat Erdeniz, Müslüm Atas, and Stefan Reiterer. Anytime diagnosis for reconfiguration. *Journal of Intelligent Information Systems*, 51(1):161–182, 2018.

[21] Mikoláš Janota. *SAT solving in interactive configuration*. PhD thesis, Citeseer, 2010.

[22] Alexey Voronov, Knut Åkesson, and Fredrik Ekstedt. Enumeration of valid partial configurations. In *Proceedings of Workshop on Configuration, IJCAI 2011*, volume 755, pages 25–31, 2011.

[23] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In Marcel Vinícius Medeiros Oliveira and Jim Woodcock, editors, *Formal Methods: Foundations and Applications*, pages 23–36, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[24] Johan De Kleer and James Kurien. Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5):25–36, 2003.

[25] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[26] Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. Modelica: The new object-oriented modeling language. In *12th European Simulation Multiconference, Manchester, UK*, 1998.

[27] Stuart Russell and Peter Norvig. *Künstliche Intelligenz*. Pearson Deutschland GmbH, 2012.

[28] Gustavo Niemeyer. Python-constraint: Solving constraint satisfaction problems in python, 2017.