AutoConf: New Algorithm for Reconfiguration of Cyber-Physical Production Systems

Kaja Balzereit, *Graduate Student Member, IEEE*, and Oliver Niggemann, *Member, IEEE*

***Abstract*—The increasing size and complexity of cyber-physical production systems (CPPS) lead to an increasing number of faults, such as broken components or interrupted connections. Nowadays, faults are handled manually, which is time-consuming because for most operators mapping from symptoms (i.e., warnings) to repair instructions is rather difficult. To enable CPPS to adapt to faults autonomously, reconfiguration, i.e., the identification of a new configuration that allows either reestablishing production or a safe shutdown, is necessary. This article addresses the reconfiguration problem of CPPS and presents a novel algorithm called *AutoConf*. *AutoConf* operates on a hybrid automaton that models the CPPS and a specification of the controller to construct a QSM. This QSM is based on propositional logic and represents the CPPS in the reconfiguration context. Evaluations on an industrial use case and simulations from process engineering illustrate the effectiveness and examine the scalability of *AutoConf*.**

***Index Terms*—Automated reconfiguration, cyber-physical production systems (CPPS), intelligent fault handling.**

I. Introduction

CYBER-PHYSICAL production systems (CPPS) are systems consisting of collaborating computational units that are strongly interconnected with physical components, such as actuators and sensors, collecting and processing data [1]. CPPS are expected to be capable of self-organization, self-maintenance, and self-repair. *Plug and produce* is a term frequently used in the context of CPPS, aiming at minimizing configuration efforts in production when faults occur, components are exchanged, or a new system layout is established [2]. However, while the possibilities of CPPS to enable a holistic selfhealing nowadays are limited due to missing redundancies and selfhealing capabilities of materials [3], there are attempts toward designing CPPS in a way that allows for fail-safe behavior and maintaining production even in the presence of faults [4]. These designs integrate redundancies, spare parts, and the possibility to adapt the system layout to offer various production paths. Reconfiguration aims at recovering a system from a fault by automatically adapting the system configuration, such that production, which was interrupted due to the fault, can be maintained, possibly by an adapted control [5]. In other words, the goal of reconfiguration is to transfer the system to a valid configuration, i.e., a configuration that allows normal system operation according to [6]. Hence, effects of faults are minimized and production outages are reduced, accepting a degradation of the system performance, e.g., a reduction of speed, if necessary [5]. However, the adaptations that enable continuous production need to be identified from a large space: the possibilities usually are binary choices that lead to the search space rising exponentially with every possibility. Nowadays, approaches are based on naive heuristics that establish cooperative control for local subsystems [7]. To the best of our knowledge, there is just little research on identifying a suitable, system-wide adaptation using efficient algorithms [8]. As a consequence, reconfiguration for CPPS faces some unsolved research questions *(RQ1–RQ3)*, which result in requirements on an efficient solution *(R1.1–4, R3.1–3)*:

RQ1: Which properties of the CPPS are relevant in the context of reconfiguration? Usually, CPPS have continuous and discrete characteristics: velocities, temperatures, or water levels are typical continuous properties, positions of valves (opened/closed) and of switches (ON/OFF) are typical discrete properties; formally, CPPS can be described as hybrid systems [8]. Hence, a model of CPPS in the context of reconfiguration needs to *(R1.1)* model the current system behavior and the possibilities of the CPPS to adapt to a fault including the effects of the adaptation using available system information, *(R1.2)* allow for an integration of the capabilities of the controller such that a state allowing for reaching the goal can be identified, *(R1.3)* contain an estimation of how the CPPS evolves, and *(R1.4)* be based on a formalism that allows for using known solution algorithms. However, AI approaches on reconfiguration often assume a system model to be given[3], [9], while approaches from fault-tolerant control (FTC) require great manual efforts. In addition, faults may require structural adaptations, e.g., the integration of redundant hardware, which usually is not covered by FTC [5]. There are numerous modeling formalisms for CPPS [10], [11]. Here, the

Manuscript received 23 July 2021; revised 5 November 2021 and 5 January 2022; accepted 18 January 2022. Date of publication 31 January 2022; date of current version 8 November 2022. This work was supported in part by the Fraunhofer Cluster of Excellence "Cognitive Internet Technologies" and in part by the German Federal Ministry for Education and Research Grant 01IS19082. Paper no. TII-21-3135. *(Corresponding author: Kaja Balzereit.)*

Kaja Balzereit is with Fraunhofer IOSB, Industrial Automation Branch, 32657 Lemgo, Germany (e-mail: kaja.balzereit@iosb-ina.fraunhofer.de).

Oliver Niggemann is with the Institute for Automation Technology, Helmut Schmidt University, 22043 Hamburg, Germany (e-mail: oliver.niggemann@hsu-hh.de).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3146940.

Digital Object Identifier 10.1109/TII.2022.3146940

goal is not to find another modeling formalism but to focus on the AI-based reconfiguration algorithm and extract the information relevant from established formalisms.

RQ2: How can the system dynamics be handled by static reconfiguration? CPPS in general are dynamic, i.e., the system evolves over time, usually described using differential equations. However, fault handling in general is static: the task of reconfiguration is a one-time adaptation while the dynamic handling is task of the controller [5]. In addition, static models are easier to create and more robust [12]. Hence, for reconfiguration, the system dynamics need to be integrated into a static model, i.e., information about dynamics needs to be estimated and formalized to be compatible with a static model. FTC uses estimations based on simulations or state space representation [13]. However, these representations need to model the system, including the faulty behavior in detail, which in general is not possible since the required amount of expert knowledge is not available [14]. Thus, the estimation of the dynamic behavior needs to be done using available resources, such as topology descriptions or historic system data.

RQ3: Is propositional logic well suited to identify a reconfiguration? An algorithm solving the reconfiguration problem of CPPS should *(R3.1)* take restrictions on the solution space coming from the CPPS into account since enumerating all possibilities to adapt to a fault is not possible due to combinatorial explosion [5], *(R3.2)* be compatible with static models containing qualitative system information, i.e., information about causal dependencies in the system, and a binary validity of configurations [6], and *(R3.3)* enable a direct integration of expert knowledge and intuitive modeling. Propositional logic is used widely for diagnosis [12] and planning [9] since it mimics human reasoning [15]. If it is well suited for reconfiguration of CPPS has not been examined yet.

The contribution of this article is threefold. First, we present a novel approach enabling automatic reconfiguration, which is especially important for frequently changing systems, such as CPPS; otherwise, reconfiguration would need to be implemented for each new system layout manually. Given a fault leading to deviations in the system, the approach identifies an input mask that recovers a configuration that allows for maintaining production. The approach is based on a qualitative system model (QSM), i.e., a model representing the properties relevant for reconfiguration, which omits the need for quantitative descriptions, such as differential-algebraic equations (DAE). The QSM is created using a description of the causal dependencies between the system variables. In contrast to existing approaches, our approach requires less expert knowledge and can be built from available information. However, the approach is compatible with more detailed system models. Second, an AI-based solution approach that is capable of reasoning about the consequences of intervening into control and identifying reconfiguration is presented. As far as we know, this is the first approach that is based on implementing reconfiguration in propositional logic, allowing for using established solvers and intuitive modeling. Last, the effectiveness *(R1.1–3, RQ2, R3.2–3)* and scalability *(R1.4, R3.1)* of the approach is illustrated using an industrial use case and representative simulations from process engineering. A comparison to an established FTC approach is given and the runtime is compared to established search techniques, which are random search and a black-box approach.

The rest of this article is organized as follows. In Section II, the related work from is discussed and the delimitations of our approach are shown. After that, in Section III the problem is defined and formalized. In Section IV, the solution approach, which is based on the creation of a QSM, is presented. Then in Section V, the evaluation, consisting of the application to an industrial use case, a comparison to an existing approach and the examination of scalability, is presented. Finally, Section VI concludes this article.

II. Related Work

In what follows we have included representative publications from highly ranked journals, conferences, and workshops with more attention on reconfiguration during the last decade to make the recent developments of this field quickly accessible. We will introduce each of the primary research directions, FTC, diagnosis and planning, multiagent systems (MAS), Industry 4.0 (I4.0), and ontology-based approaches.

A. FTC

FTC is concerned with the accommodation of faults using methods from control theory [16]. Today, the faulty system behavior is often assumed to be known, which usually is not the case in practical applications [13]. Hence, in recent years, several techniques have been developed to estimate unknown fault behavior, e.g., using adaptive feedback gains, that successively adapt the control values [13], or by integrating an observer that estimates the consequences of an actuator fault [17]. Thus, the maximal impact of faults no longer needs to be known but is determined during fault handling. Ma *et al.* [18] showed an approach based on the combination of FTC and fuzzy logic. A fuzzy observer determines unknown state variables and approximates their behavior using nonlinear functions. Then, an optimal control that minimizes the errors is calculated. FTC uses ordinary differential equations (ODEs) to describe the timed system behavior. ODEs require high manual efforts for modeling and parameter adjustment. Major system disturbances, such as failing system components would lead to massive changes in these quantitative system models, which cannot be covered by estimation techniques [5]. Hence, a new system model is needed. Furthermore, FTC mainly focuses on continuous variables. When CPPS contain multiple discrete operation modes, for each mode one control is needed.

B. Diagnosis and Planning

Model-based diagnosis (MBD) is the task of identifying root causes. Automated planning identifies, given an initial and a goal state and a model of the environment, a sequence of actions reaching the goal state [9]. Travé-Massuyès [12] emphasized the importance of such qualitative approaches in the context of control: they enable reasoning about causal dependencies while quantitative control approaches can ensure optimal control and

stability. For diagnosis, hybrid systems are modeled using hybrid automata and qualitative fault models [11] or Bayesian networks, that model the dependencies between sensors and faults qualitatively [19]. For solving, Feldman *et al.* [15] presented an enhanced SAT-based algorithm to MBD, for that the system and observations are modeled in propositional logic. Borgo *et al.* [9] used automated planning for the optimal control of manufacturing systems. The system components are modeled as symbols, the domain model describes the dependencies between these. Then, automated planning is used to identify a sequence of actions that enable optimal control. In qualitative simulation (QS), physical systems are described using qualitative differential equations where instead of quantitative values, functions are described by symbols expressing qualitative properties, such as monotony information [10]. Thus, the need for expert knowledge is reduced. However, QS is limited by its prediction correctness since it suffers from a combinatorial blowup.

C. Multiagent Systems

MAS are characterized by multiple agents that each are specialized on a task and are connected to achieve a common goal [20]. For reconfiguration, multiple agents are combined using either automated planning such that a system-wide control is established [20], or heuristics for redistribution of control [7]. Other MAS identify reconfiguration dynamically and use agents to overcome time-delays coming from manual reconfiguration [21]. An important constraint on all MAS discussed is the need for highly specialized agents, which are technically complex to train [22].

D. Approaches From Industry 4.0

Part of the vision of I4.0 are selfadapting CPPS capable of reacting autonomously to faults [23]. Strasser *et al.* [24] proposed a formalism based on the creation of a reference model according to the IEC 61499 standard [25] in the fields of manufacturing, power systems, and robotics that enable systems to adapt to changed circumstances, such as faults. The work also outlines the need for a system-wide control infrastructure that enables interoperability of the system components. Ladiges *et al.* [26] presented an approach that combines selfdescriptions of components in order to achieve a system-wide synchronized control and a minimization of configuration efforts. For this purpose, each component is assigned a formal description of its functionalities and its input and output variables. However, both approaches [24], [26] rely on the creation of detailed models, which are costly to build. In addition, they do not focus on solving the reconfiguration task but on providing reference models containing the information necessary for system-wide control and generating cooperative control strategies based on predefined heuristics [7]. These strategies enable automatic handling of minor deviations, but faults requiring more complex adaptations, e.g., the integration of redundant hardware into the control loop, push the approaches to their limitations. Also, the expert knowledge needed for creating the models may not be available [14]. Sinha *et al.* [27] presented an approach on modeling CPPS in first-order logic and identifying valid configurations using a satisfiability solver. This approach, however, focuses on adapting the system to handle changes in the system architectures or available resources and not on handling faults in the system. Otto *et al.* [28] used optimization to identify optimal parameter combinations for reusable software components. However, the scope of this work is on optimal control and not on recovering from faults. We build upon this work by providing a sophisticated solution algorithm, which identifies even complex adaptations. It works on causal graphs but it is also compatible with more detailed models [24], [26] since these provide the information needed for reconfiguration, too.

TABLE I
COMPARISON OF EXISTING APPROACHES

Requirement	FTC	MBD	MAS	I4.0	Ontologies
(R1.1)	-	+	-	-	-
(R1.2)	+	-	0	-	+
(R1.3)	-	-	+	+	+
(R1.4)	+	+	-	-	+
(R3.1)	-	+	-	-	+
(R3.2)	-	+	0	0	+
(R3.3)	-	+	+	+	+

Note: "+" ("-") stands for approaches (not) satisfying the requirement, "0" stands for no proposition concerning the requirement possible.

E. Ontology-Based Approaches

In automation technology, ontologies are used to model CPPS including their capabilities in terms of concepts and relations [29], [30]. Combined with observations, ontologies allow for reasoning about alternative control instructions. For reconfiguration, resources, functionalities, and requirements of CPPS are described, and an optimal mapping between those is searched [30]. Engel *et al.* [29] presented how to formalize modules and orchestration instructions of CPPS in a knowledge base using ontologies, and thus, reducing configuration efforts. The main drawbacks of ontologies are the high modeling efforts and expert knowledge required for creating the system models [14].

F. Novelty of Our Approach

Table I summarizes the capabilities and limitations of existing approaches from FTC, e.g., [13], [17], [18]; MBD, e.g., [9], [11], [15]; MAS, e.g., [7], [20], [21]; I4.0 [24], [26], [31]; and ontologies, e.g., [29], [30] concerning *(R1.1–4)* and *(R3.1–3)*. Our approach builds upon the related work as follows. As in MBD [11], [12], we use a QSM abstracting relevant CPPS properties using symbols. Compared to FTC or I4.0, our approach requires less expert knowledge to create the model; fault models describing the system under faults are not necessary. However, our approach is compatible with more detailed models since it relies on logic that is able to also reason about these models in a short time. An important difference compared to QS [10] is that qualitative information is not used for simulating the CPPS in detail but only to estimate how the CPPS evolves. Our model is drawn from causal graphs describing the dependencies between system variables *(R1.1)*. Similar to approaches from FTC (e.g., Wang *et al.* [13]), we use observations in form of sensor readings

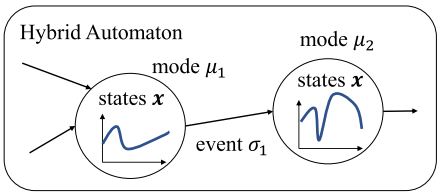

Fig. 1. Hybrid automaton.

to reason about the current system status and estimate future behavior. But in contrast, we create the system model from causal graphs and not from sampled system data, reducing the impact of noise and false sensor readings. Also, no quantitative calculation of future system states but qualitative information is used as in ontology-based approaches *(R1.2)*. FTC models the system including the capabilities of the controller in a quantitative way, which usually is time-consuming. In I4.0 and ontology-based approaches, the capabilities are modeled qualitatively, i.e., using available process specifications. To minimize modeling efforts, our approach also uses a qualitative control model *(R1.3)*. MBD and ontologies rely on logical reasoning enabling intuitive modeling and the use of established solvers [15], [29]. To build on these advantages, our approach uses propositional logic *(R1.4)*. Logic also permits restricting the solution space intuitively by constraints such that efficient solving is enabled *(R3.1)*. And as usual in MBD and in ontologies, dynamic system information is modeled statically *(R3.2)*. In addition, expert knowledge can be integrated directly *(R3.3)*.

III. Problem Formulation

CPPS consist of physical and virtual components that are connected to achieve a given production task [1]. Formally, a CPPS can be described by a hybrid automaton $\mathcal{H}$ (see Fig. 1).

$\mathcal{H}$ is a tuple $(I, X, \boldsymbol{x}^0, \mathcal{F}, \Sigma, \Phi)$ with I being a set of input variables. I consists of discrete I_D and continuous I_C variables $I = I_D \cup I_C$. $\boldsymbol{i}(t), \boldsymbol{i}_D(t)$, and $\boldsymbol{i}_C(t)$ refer to an (discrete/continuous) input at time t. $m = |I|$ denotes the dimension of I. Every combination of discrete inputs $\boldsymbol{i}_D$ defines a *mode* $\mu \in \mathcal{M}$. $X \subset \mathbb{R}^n$ with $n \in \mathbb{N}$ denotes state variables. $\boldsymbol{x}(t)$ refers to a state describing the continuous behavior at time t. $\boldsymbol{x}(t=0) = \boldsymbol{x}^0$ are initial states, $\mathcal{F}$ is a finite set of functions $\{\boldsymbol{f}_{\mu_1}, \boldsymbol{f}_{\mu_2}, \ldots\}$ with $\boldsymbol{f}_{\mu_i} : X \times \mathbb{R} \times I_C \to X$ describing the continuous behavior in mode μ_i over time $t \in \mathbb{R}$, Σ is a finite set of discrete events $\{\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots\}$ with $\boldsymbol{\sigma}_i : I_D \to I_D$ that transits the system between modes, $\Phi : \Sigma \times \mathcal{M} \times X \to \mathcal{M} \times X$ is a transition function mapping an action, a mode, and a state into a new mode and initial state. If all functions $\boldsymbol{f}_{\mu_i}$ are independent from t, so that $\boldsymbol{f}_{\mu_i} : X \times I_C \to X \ \forall \mu_i \in M$, the automaton is called *time-invariant*. Otherwise, it is called *time-variant*. For readability, variables depending on time t are shortened to the variable itself, e.g., $\boldsymbol{i}$ instead of $\boldsymbol{i}(t)$. Given a state $\boldsymbol{x}$ and an input $\boldsymbol{i}$, the tuple $(\boldsymbol{x}, \boldsymbol{i})$ is called a *configuration* of the hybrid system.

A. Running Example

The two-tank system in Fig. 2, which is an adaptation of a system used for FTC and reconfiguration [5], serves as the running example. The system consists of two tanks T_1, T_2, two heating elements H_1, H_2, two cooling elements C_1, C_2, and two connecting pumps p_{12}, p_{21}. Every tank has an inflow with either hot (90°C) or cold (10°C) water and an outflow controlled by valves v_{01}, v_{10}, v_{02}, and v_{20}. State variables are given by water levels $\boldsymbol{l}_1, \boldsymbol{l}_2$ and temperatures $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$. These are measured in each tank. The ten input variables are given by the opening states of the valves and pumps. Hence, the hybrid automaton contains 2^{10} modes.

Fig. 3 shows how reconfiguration interacts with existing control loops. The input variables I usually are controlled by a program P, e.g., a control or a production plan. To define the system goal, reference values on output variables are specified by human operators. The program controls the system by a sequence of input changes, which establish the wanted reference values [5]. In case of a fault, this program may become invalid such that the system goal is no longer reached. Nowadays, most known faults are handled by P while often no fault handling is done at all [8]. Hence, when a fault occurs, reconfiguration becomes necessary to alter the system to a new configuration, from that production can be maintained by control. For this purpose, reconfiguration changes the system to a new configuration by masking inputs, i.e., applying a function f_R on a subset of the inputs $I_R \subset I$ that may no longer be affected by control. The new configuration shall enable that a possibly adapted production program P' maintains system operation [5]. Second, P' is identified, which works on the remaining, unmasked inputs $I \setminus I_R$ and controls the system as specified by human operators. The reconfiguration algorithm operates on a model of the CPPS and a description of the controller, i.e., information when the system goal can be reached by at least one program P. The identification of an adapted program P' can be done using well-known control methods. However, the identification of an adequate reconfiguration is still an open research topic [5]. Given a system goal, a configuration $(\boldsymbol{x}, \boldsymbol{i})$ is *valid* iff the system goal can be reached by at least one program P.

Definition 1 (Reconfiguration): Given an invalid configuration $(\boldsymbol{x}^0, \boldsymbol{i}^0)$ and a set of production programs $\mathcal{P} = \{P_1, P_2, \ldots\}$, reconfiguration is a function $f^R : I_R \to I_R$ with $I_R \subset I$ such that $(\boldsymbol{x}, f_R(\boldsymbol{i}))$ is valid. Thus, a valid configuration is recovered and the system goal can be reached by a program $P' \in \mathcal{P}$, which operates on the unmasked inputs $I \setminus I_R$.

B. Running Example

Let the valve v_{01} be blocked, such that the water level in tank T_1 becomes too low. A possible reconfiguration is to open pump p_{21}. The corresponding reconfiguration function is $f^R(\boldsymbol{b}_{p_{21}}) = 1$ with $\boldsymbol{b}_{p_{21}}$ being the binary input defining the opening state of p_{21}.

IV. Solution Approach

The solution approach *AutoConf* consists of two steps (see Fig. 4). First (see Section IV-A), a QSM in propositional logic representing the CPPS in the context of reconfiguration is created. For this purpose, the algorithm *AutoConf_GenerateSM* is created, which operates on the input and state variables of the CPPS and causal graphs describing the dependencies between these. The causal graphs can be created automatically from a

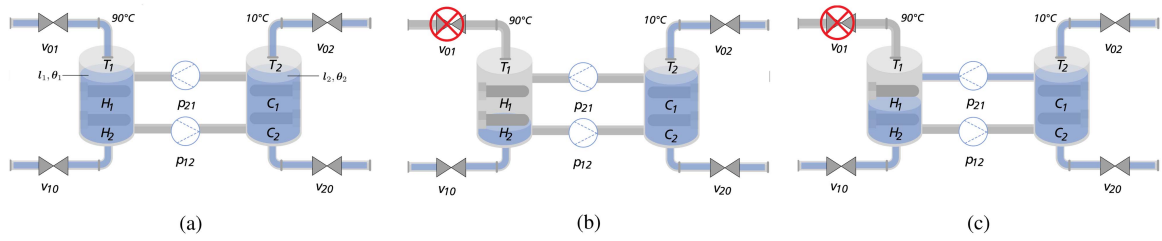

Fig. 2. Running example. (a) Nonfaulty operation. (b) Valve v_{01} is blocked. (c) Reconfiguration via pump p_{21}.

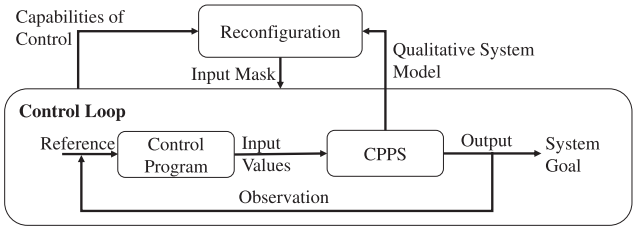

Fig. 3. Interaction between control and reconfiguration.

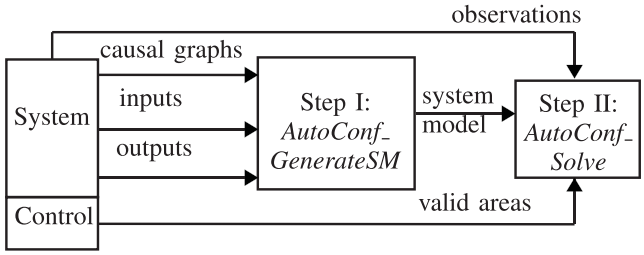

Fig. 4. Steps of the solution approach.

topology description, e.g., a piping and instrumentation diagram, selfdescriptions of system components and their interconnection [26], or using learning techniques [32]. The resulting system model is given to the second step, the algorithm *AutoConf_Solve* (see Section IV-B), which also operates on observations of state and input variables, and a specification of valid areas of the controller. The result of *AutoConf_Solve* is a mask that assigns fixed values to the inputs of the hybrid automaton and ensures recovery of a valid configuration. *AutoConf* is the concatenation of *AutoConf_GenerateSM* and *AutoConf_Solve*.

Hence, given causal graphs and observations, reconfiguration works completely automatically. Propositional logic is used for following several reasons. it: (1) enables the direct integration of expert knowledge in the form of process specifications and domain knowledge due to intuitive modeling [15]; (2) reduces the underlying search space to the relevant configurations, since the logical formula models constraints coming from the CPPS; and (3) allows for fast solving, sacrificing the optimal solution for a quick identification of an arbitrary solution. Since reconfiguration is a search problem, it could also be directly implemented in search. But this would come with higher manual efforts due to less intuitive modeling. The high-dimensional search space would need to be modeled explicitly. In addition, it would be necessary to evaluate the influence of constraints coming from expert knowledge on the search space manually. Here, the masking of inputs is done by assigning inputs fixed values, e.g., fixing a valve to open or close position. This simplification is valid since reconfiguration aims at recovering a valid configuration and not at optimally controlling the system [5]. Hence, continuous inputs $I = \{\boldsymbol{i}_1, \ldots\}$ are discretized to binary variables $B = \{\boldsymbol{b}_1, \ldots\}$. Setting a binary input variable to true represents the opening of the corresponding valve. We are concerned with time-invariant systems only, by modeling past system behavior in state variables.

A. Step I: Algorithm AutoConf_GenerateSM to Generate System Model

The following sections describe how the logical formula representing the system in context of reconfiguration is created and how the requirements *(R1.1–4)* are met, i.e., to model the current system status and the possibilities of adaptation including their effects *(R1.1)*, the logical formula needs to contain variables representing the state and input variables, including the causal dependencies between those to enable reasoning about the consequences of masking inputs (Section IV-A1). The capabilities of the controller *(R1.2)* are modeled in terms of valid and invalid configurations meaning, a configuration is valid if it allows for maintaining production (Section IV-A2). The handling of how the system evolves *(R1.3)* is achieved by integrating a qualitative prediction of the dynamics (Section IV-A3). And logic permits using established solving algorithms, e.g., SAT solvers, which are in focus of research for many years now [33] *(R1.4)*. The overall algorithm describing how the logical formula is created is shown in Section IV-A4.

1) Causal Dependencies Between Input and State Variables: To describe the causal dependencies between binary input variables B and state variables X, causal graphs are used. In causal graphs nodes represent variables and edges represent causal dependencies between the variables [34]. For reconfiguration, we use specific causal graphs $G^+ = (V, E^+)$ and $G^- = (V, E^-)$. V is given by the binary input variables and the state variables, so $V = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_n, \boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_m\}$. The edges in E^+, E^- indicate if setting a binary input variable to true (i.e., fixing the corresponding input variable to a completely opened position) leads to a significant increase/decrease of a state variable $E^+ = \{(\boldsymbol{b}_j, \boldsymbol{x}_i) | j \in \{1, \ldots m\}, i \in \{1, \ldots n\}$ and $\boldsymbol{b}_j$ leads to significant increase of $\boldsymbol{x}_i\}$. An increase/decrease is significant, if it enables recovery of a valid interval within the reconfiguration time Δt. These causal graphs can either be

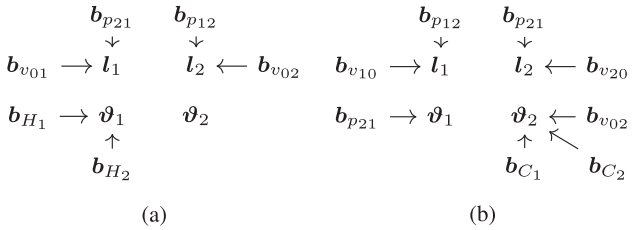

Fig. 5. Causal graphs for dependencies of running example. An arrow from a variable to another indicates that activating the first one leads to an increase/decrease of the second. (a) Increasing dependencies. (b) Decreasing dependencies.

generated using expert knowledge [34] or learning techniques [32]. Throughout the article, we assume the properties of the causal graphs to not change over time. This poses the requirement of monotonic dependencies between input and states, e.g., a valve always leads to an inflow in a tank, on CPPS. However, if this requirement is not satisfied, further restrictions can be added to ensure validity of the QSM.

Running Example: Fig. 5 shows the causal graphs for the running example. For example, when v_{01} or p_{21} are opened, represented by Boolean variables $\boldsymbol{b}_{v_{01}}, \boldsymbol{b}_{p_{21}}, \boldsymbol{l}_1$ increases.

2) Capabilities of Controller: Since the goal of reconfiguration is to recover a configuration that enables the controller to maintain production, the capabilities of the control define valid and invalid configurations. Iff the control is capable of working properly, given a configuration, the configuration is valid. Usually, the areas from that a control can maintain production are described by intervals on states, i.e., the control enables the system to reach the goal if all state variables $\boldsymbol{x}$ are in specified intervals $[\boldsymbol{lb}, \boldsymbol{ub}]$ with $\boldsymbol{lb}, \boldsymbol{ub} \in \mathbb{R}^n$ being bound values [35]. The Boolean vectors $\boldsymbol{low}, \boldsymbol{high} \in \{0,1\}^n$ model if the states $\boldsymbol{x}$ are too low/high, so

$$\boldsymbol{low}_i = BinaryValue(\boldsymbol{x}_i < \boldsymbol{lb}_i) \tag{1}$$

$$\boldsymbol{high}_i = BinaryValue(\boldsymbol{x}_i > \boldsymbol{ub}_i) \tag{2}$$

with $BinaryValue$ returning the truth value of the statement.

Running Example: The valves v_{10}, v_{20} are controlled by a program ensuring constant outflow when the water levels are between 0.3 and 0.4 m. $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ shall be between 65°C and 75°C, and 10°C and 20 °C, respectively. These values define $\boldsymbol{lb}$ and $\boldsymbol{ub}$.

3) System Dynamics: The system dynamics are integrated by state variables, which store past system behavior, and by integrating a qualitative estimation of how the system evolves given a specific input into the QSM. This comes with the advantage of a simpler system model than those used in FTC. In addition, this approach mimics human fault handling, i.e., in case of a fault, a human would also adapt the system inputs once by estimating from expert knowledge how the system evolves. Hence, an estimation from causal graphs is drawn if a state will increase or decrease in a specified reconfiguration time Δt: first, for each state variable $\boldsymbol{x}_i$ the sets POS, NEG containing those binary input variables that increase/decrease $\boldsymbol{x}_i$ are created. A variable that is too low is estimated to recover to $[\boldsymbol{lb}_i, \boldsymbol{ub}_i]$ if at least one additional binary input leading to an increase is activated or an additional binary input leading to a decrease is deactivated (analogously for variables that are too high). Please note, that this restriction reduces the solution space, i.e., further combinations of Boolean inputs might exist, which also lead to recovery of a valid configuration, but not satisfy the restriction of not activating decreasing inputs for variables that are too low and vice versa. The integration of these combinations can be done manually for each CPPS.

Here, we use a fixed reconfiguration time Δt. However, the approach is also compatible with flexible times, e.g., approximated using historical data or expert knowledge.

Running Example: The system dynamics are represented by the state variables $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ for temperature and $\boldsymbol{l}_1, \boldsymbol{l}_2$ for water levels. Their values at time t are defined by their past values and evolve based on the laws of thermo- and fluid dynamics.

4) Creation of Logical Formula: If at least one state is not in its valid area, one of two following cases occurs. Either the states that are not in their valid areas will return to their valid areas within reconfiguration time Δt due to a suitable input, or not. Given the second case, reconfiguration identifies the input mask to achieve the first case, a configuration that recovers validity. These are the configurations, in which for the state variables, that are not too low or too high, inputs are activated that enable recovery of a valid interval.

Algorithm 1 shows how the logical formula SM is created. Inputs are the sets X, B and the graphs G^+, G^-. SM is initialized as empty set in Algorithm l. 1. Then, for each state variable $\boldsymbol{x}_i \in X$, the sets POS and NEG are identified, which contain those binary input variables that lead to an increase/decrease of $\boldsymbol{x}_i$ using the information from the causal graphs G^+, G^- (see Algorithm l. 4, 5). Then, SM is extended by constraints that ensure that if a state is too low, at least one additional binary input leading to an increase is activated or an additional binary input leading to a decrease is deactivated (see Algorithm l. 6) (analogously for states that are too high in Algorithm l. 7). For this purpose, the binary variables $\boldsymbol{b}^0$ are added to the model, which represent the initial value of the binary inputs.

Please note that the system model requires no fault models, i.e., data about how the system behaves in the presence of faults.

Algorithm 1: *AutoConf_GenerateSM.*

Input: $X, B, G^+ = (V, E^+), G^- = (V, E^-)$
Output: SM
1 $SM := \{\}$
2 **for** $i \in \{1, 2, ...n\}$ // for every state variable
3 **do**
4 $POS_i := \{\boldsymbol{b}_j \in B | \exists (\boldsymbol{b}_j, \boldsymbol{x}_i) \in E^+\}$ // binary inputs with pos. impact
5 $NEG_i := \{\boldsymbol{b}_j \in B | \exists (\boldsymbol{b}_j, \boldsymbol{x}_i) \in E^-\}$ // binary inputs with neg. impact
6 $SM := SM \cup (\boldsymbol{low}_i \Rightarrow \bigvee_{\boldsymbol{b}_j \in POS_i} (\neg \boldsymbol{b}_j^0 \wedge \boldsymbol{b}_j) \vee \bigwedge_{\boldsymbol{b}_j \in NEG_i} (\boldsymbol{b}_j^0 \wedge \neg \boldsymbol{b}_j))$ // constraint for states below $\boldsymbol{lb}$
7 $SM := SM \cup (\boldsymbol{high}_i \Rightarrow \bigvee_{\boldsymbol{b}_j \in NEG_i} (\neg \boldsymbol{b}_j^0 \wedge \boldsymbol{b}_j) \wedge \bigwedge_{\boldsymbol{b}_j \in POS_i} (\boldsymbol{b}_j^0 \wedge \neg \boldsymbol{b}_j))$ // constraint for states above $\boldsymbol{ub}$
8 **return** SM

Algorithm 2: *AutoConf_Solve.*

Input: $SM, (\boldsymbol{x}^0, \boldsymbol{b}^0), \boldsymbol{lb}, \boldsymbol{ub}$
Output: $\boldsymbol{b}$ or *shutdown*
1 $n^{\mathrm{ub}} := 1$
2 $\boldsymbol{low}, \boldsymbol{high} = \mathit{InitLowHigh}(\boldsymbol{x}^0, \boldsymbol{lb}, \boldsymbol{ub})$ // initialize symbols for low, high
3 **while** $n^{\mathrm{ub}} \leq |\mathrm{B}|$ // while max. number of binary changes not reached
4 **do**
5 $\quad \beta := (\sum_k (\boldsymbol{b}^0_k \oplus \boldsymbol{b}_k) \leq n^{\mathrm{ub}})$ // set upper bound
6 $\quad$ **if** $\mathrm{SAT}(SM \wedge \boldsymbol{low} \wedge \boldsymbol{high} \wedge \beta)$ // check sat.
7 $\quad$ **then**
8 $\quad\quad \boldsymbol{b}^* := \mathit{Assign}(SM \wedge \boldsymbol{low} \wedge \boldsymbol{high} \wedge \beta)$ // get solution
9 $\quad\quad$ **return** $\boldsymbol{b}$ // return solution
10 $\quad$ **else**
11 $\quad\quad n^{\mathrm{ub}} := n^{\mathrm{ub}} + 1$ // increase upper bound
12 **return** *shutdown*

Whilst the approach is compatible with fault models, data about faulty behavior in general are not available since it is nearly impossible to foresee every fault at design time. Hence, the approach is applicable to unknown faults and situations.

CPPS may offer the possibility to replace faulty components or adapt the system layout [8]. To integrate these, binary variables indicating the exchange of components or the use of redundant hardware are used. The effects of these changes are then modeled by causal graphs. Further, the system model can be extended by capabilities extracted from a selfdescription of the system, such as digital twins [20], [36].

Running Example: The system model for the running example consists of eight constraints, two for each state variable. For example, the constraint $\boldsymbol{low}_{\theta_1} \Rightarrow ((\neg \boldsymbol{b}^0_{H_1} \wedge \boldsymbol{b}^0_{H_1}) \wedge (\neg \boldsymbol{b}^0_{H_2} \wedge \boldsymbol{b}^0_{H_2})) \wedge (\boldsymbol{b}^0_{p_{21}} \wedge \neg \boldsymbol{b}_{p_{21}})$ ensures that in case the temperature in T_1 is too low, it is increased by activating a heating element or closing pump p_{21} since it delivers cold water.

B. Step II: Solution Algorithm AutoConf_Solve

The solution algorithm meets the criteria *(R3.1–3)* as follows. Constraints directly reduce the solution space and the number of nodes that need to be examined *(R3.1)*. While there are different types of logic, also temporal ones, propositional logic operates on a static system model *(R3.2)*. Expert knowledge can be integrated directly since logic is selfexplanatory and enables intuitive modeling *(R3.3)* [15].

The formula created in the previous section is input to a SAT solver, which identifies an assignment to free variables, i.e., the Boolean input variables. A satisfying assignment leads to the recovery of a valid configuration. But a SAT solver does not take cost-functions or other valuations of different valid solutions into account. However, the space of valid solutions can be reduced further using regularization, i.e., constraints ensuring that user-specified criteria (e.g., costs under a given bound) are satisfied. In this approach, we apply regularization minimizing the cardinality of the input mask, i.e., the number of input variables affected by the reconfiguration. Hence, intervention into control is minimized as much as possible.

Algorithm 2 shows the algorithm *AutoConf_Solve*. It takes the system model SM, an initial configuration $(\boldsymbol{x}^0, \boldsymbol{b}^0)$, and the bound values $\boldsymbol{lb}$, $\boldsymbol{ub}$ as inputs. The output is a new assignment to the binary input variables, denoted by $\boldsymbol{b}$. To minimize the cardinality of the input mask, i.e., the number of input variables that are fixed, an upper bound for binary changes is defined and initialized with 1 (see Algorithm l. 1). The symbols $\boldsymbol{low}, \boldsymbol{high}$ are initialized by the function *InitLowHigh*, which returns the binary truth values of (1) and (2) (see Algorithm l. 2). Then, the solving loop starts (see Algorithm l. 3) and the constraint for the upper bound is initialized by creating a cardinality constraint for the number of variables in $\boldsymbol{b}$ that differ from $\boldsymbol{b}^0$ (see Algorithm l. 5). For this purpose, an XOR is applied component-wise to $\boldsymbol{b}$ and $\boldsymbol{b}^0$. If the logical formula $SM \wedge \boldsymbol{low} \wedge \boldsymbol{high} \wedge \beta$ is satisfiable (see Algorithm l. 6), the assignment to the binary input variables is identified using the function *Assign* (see Algorithm l. 8) and returned (see Algorithm l. 9). Otherwise (see Algorithm l. 10), the upper bound is increased (see Algorithm l. 11) and a new iteration is started. In case the upper bound reaches its maximum, given by $|B|$, and no solution is found, *shutdown* is returned and the instructions ensuring a safe shutdown are applied to the CPPS.

V. Results

An industrial use case from process engineering (see Section V-A), established simulations of tank systems (see Section V-B), and a comparison to an FTC approach [5] (see Section V-C) are used to show effectiveness *(R1.1–3, RQ2, R3.2–3)* of *AutoConf*. The systems are realistic examples of CPPS [5], [37] and contain continuous and discrete variables, they are consequently hybrid. In addition, the systems are controlled, but not fault-tolerant. Hence, the systems pose realistic reconfiguration problems. Causal graphs needed for creating system models are generated from available system specifications. To show scalability (see Section V-D) *(R1.4, R3.1)*, systems with varying numbers of input and state variables are used and a comparison to random search and a black-box approach is given. Thus, the advantages of limiting the search space using logical constraints are quantified.

For reconfiguration, faults are not separated according to the affected component but to their effect on the state variables. Continuous faults lead to a continuous change of state variables, e.g., a leak in a tank leads to a continuous decrease of the water level. Discrete faults lead to an abrupt change of state variables, e.g., an addition of water to a tank leads to an abrupt increase of the water level. Temporal effects, such as a creeping loss of temperature are handled as soon as a state variable reaches an interval area. Faults that have no effects on state variables are not within the scope of this article.

A. Industrial Use Case: Tennessee Eastman Process (TEP)

The TEP is a simulation of a production plant often used as a complex control problem [38]. It has been used widely as an application example in the areas of fault detection and diagnosis

TABLE II
RESULTS FOR SINGLE-FAULT CASES

Single-Fault Cases	# cases	# reconf.	in %
Tennessee Eastman Process	**90**	**75**	**83**
IDVs 1,2	15	15	100
IDVs 1,2,5	10	10	100
IDV 6, intensity 90%	10	10	100
IDV 6, intensity 100%	10	10	100
XMV 3	20	20	100
XMV 3-1h	5	0	0
XMV4	20	10	50

Note: For each kind of fault listed in column 1, the number of simulated cases is listed in column 2. **# Reconf.** denotes the number of correctly reconfigured test cases (shown in column 3, in percent in column 4).

TABLE III
RESULTS FOR MULTIFAULT CASES

Multi-Fault Cases	# cases	# reconf.	in %
Tennessee Eastman Process	**91**	**72**	**79**
3 faults	**59**	**47**	**80**
conc. + flow	28	24	86
temp. + flow	11	11	100
conc. + temp. + flow	20	12	60
4 faults	**22**	**17**	**77**
conc. + flow	5	3	60
temp. + flow	1	1	100
conc. + temp. + flow	16	13	81
5 faults	**7**	**6**	**86**
conc. + flow	2	1	50
conc. + temp. + flow	5	5	100
6 faults	**3**	**2**	**67**
conc. + flow	2	1	50
conc. + temp. + flow	1	1	100

[39], alarm management [40], control theory [38], for the evaluation of MAS [41], reconfiguration methods [42], and as an example for digital twin modeling [37]. It is representative for many real-world systems used in pharmacy, chemical engineering, and other. It consists of a reactor, a separator, a condenser, and a stripper, which are interconnected to deliver a specific combination of gas and liquid, created from four different input materials. Twelve valves in the process control the flow between the system components; 41 sensors measure system variables. In the absence of faults, the system is controlled by 17 control loops [40]. The causal graphs required for reconfiguration are extracted manually from the process description. While the valves originally are linear valves with continuous opening degrees, for reconfiguration, they are abstracted by discrete variables describing either completely opened or completely closed positions. This is valid since reconfiguration aims at recovering a valid area of operation and not optimally controlling the system. Various deviations leading to anomalous system behavior and undesired output can be injected into the system. These deviations, which manifest either locally or propagate through the system, are caused by manipulating one or more system variables. Typical deviations are variations in temperature, material concentration, or flow of input variables.

Manca [40] published datasets comprising single-fault cases, e.g., deviations in the flow or the concentration of input products and multifault cases, i.e., combinations of single-fault cases with superposing effects. In addition, valid and invalid areas of state variables are defined by bounds, which serve as $\boldsymbol{lb}, \boldsymbol{ub}$ in the QSM. We examine 90 single-fault cases and 100 multifaults cases. The performance of *AutoConf* for the single-fault cases is shown in Table II. IDVs 1,2 are combinations of deviations in the concentration of input products, IDV 5 is a disturbance of temperature of an input product, IDV 6 is a disturbance of flow of an input product, and XMV 3 and XMV 4 are disturbances of valve opening positions. XMV 3-1 h is a disturbance of the valve for one hour. Out of 90 test cases, 75 are reconfigured correctly. Disturbances XMV 3-1 h and XMV 4 tend to fail since the deviation is not recognized by *AutoConf*.

Out of the 100 multifault cases, 91 are reconfigurable, i.e., a masking of system inputs exists that allows for reaching a valid configuration again. A total of 72 of these lead to an invalid configuration, i.e., the deviations are not mitigated by control. As soon as at least one state variable is out of its valid area, reconfiguration is initiated. Reconfiguration then calculates an input mask, which sets some of the 12 valves to open or close positions, which ensures that a valid configuration is recovered. Out of 19 faults that are not handled correctly, 15 faults did not lead to an invalid configuration. 4 faults were not handled correctly due to Δt being too small such that a valid configuration is not recovered within the reconfiguration time. Table III shows the results of the proposed approach for these multifault cases. TEP contains single and multiple continuous faults affecting the concentration or temperature of input variables or stuck valves. As can be seen, approximately 80% of faults in the industrial use case are reconfigured correctly. Otherwise, these faults would have led to an output having a reduced quality, or, in the worst case, to machine and production outages. Hence, reconfiguration can improve the performance of CPPS significantly. However, 20% of faults have not been handled correctly yet. Most of these fail due to the fault not leading to an invalid configuration. Here, a fault detection method, which is integrated into the definition of valid and invalid configurations, would enable the reconfiguration algorithm to also handle these faults. Configurations would be labeled as invalid as soon as the fault is detected and reconfiguration would be triggered. In addition, the approach is compatible with a varying time Δt, determined using expert knowledge or learning techniques.

B. Evaluation of Limitations of the Approach

We use the two-tank system from the running example and the three-tank-benchmark presented by Blanke *et al.* [5], provoke various faults, such as broken components, e.g., leaking tanks, and blocked connections, and monitor the output of *AutoConf*. Both systems have been used for evaluation of reconfiguration [43]. The ladder contains a redundant tank, which is not used during nonfaulty operation. However, if necessary, e.g., when a fault occurs, reconfiguration triggers usage of the redundant tank to maintain production. Hence, the system layout may change. For reconfiguration, redundant hardware is modeled using binary variables. The systems are simulated in Modelica. The QSMs are created using available topology information. The model of the three-tank-benchmark encounters additional constraints indicating the direction of flow in the valves depending on the water levels. The inputs of the system, i.e., the variables controlling the heating and cooling elements, the valves, and

TABLE IV
RECONFIGURATION RESULTS FOR TWO-TANK SYSTEM AND THREE-TANK BENCHMARK

Kind of Fault	# cases	# reconf.	in %
Two-Tank System	**58**	**50**	**86**
continuous	**16**	**16**	**100**
leak in one tank	2	2	
valve stuck open	4	4	
valve stuck close	4	4	
pump stuck full	2	2	
pump blocked	2	2	
continuous temperature loss/rise	2	2	
discrete	**22**	**22**	**100**
drop in wl (20%-70%)	5	5	
rise in wl (20%-70%)	5	5	
drop in temperature (7%-42%)	6	6	
rise in temperature (7%-42%)	6	6	
multiple continuous	**4**	**4**	**100**
heating failure + valve stuck	2	2	
cooler failure + valve stuck	2	2	
multiple continuous + discrete	**10**	**5**	**50**
heating failure + drop in wl	5	5	
cooler failure + rise in wl	5	0	
multiple discrete	**6**	**3**	**50**
drop in wl + in temperature	5	2	
drop in wl in 2 tanks	1	1	
Three-Tank Benchmark	**39**	**37**	**95**
continuous	**14**	**14**	**100**
leak in one tank	2	2	
valve stuck open	4	4	
valve stuck close	4	4	
pump stuck full power	2	2	
pump blocked	2	2	
discrete	**10**	**10**	**100**
drop in wl (20%-70%)	5	5	
rise in wl (20%-70%)	5	5	
multiple continuous	**4**	**4**	**100**
leak in one tank + valve stuck	4	4	
multiple continuous + discrete	**10**	**8**	**80**
valve stuck open + drop in wl	5	4	
pump blocked + rise in wl	5	4	
multiple discrete	**1**	**1**	**100**
drop in wl in 2 tanks	1	1	

Note: WL Stands for Water Level.

the pumps, are discrete; the water level and the temperature are continuous. Hence, the systems are hybrid. The two-tank system encountered 58 faults, the three-tank-benchmark 39. In addition to single faults, multiple-fault scenarios consisting of two continuous, one continuous and one discrete, or two discrete faults are simulated. In both systems, continuous faults are tank leaks (2 cases each), valves stuck in either open or closed position (eight cases each), and pumps stuck on full power or blocked (four cases each). Discrete faults are abrupt drops and rises of the water level, e.g., by manual adding cold or hot water (five cases with intensity from 20% to 70% of current level each). In the running example, which also processes temperature, further faults are injected. Temperature drops and rises (one case each) and failing heating/cooling elements (four cases) are continuous faults, discrete drops, and rises of temperature (six cases each with intensity from 7% to 42% of current temperature) are discrete faults. Also, multiple faults are injected that are combinations of two continuous faults (four cases: heating/cooler failures + stuck valves/pumps), a continuous and a discrete fault (five cases: heater/cooler failures + abrupt level drops), or a combination of two discrete faults (five cases: level and temperature drops, one case: level drops in two tanks).

The results of the evaluation are shown in Table IV. All of the continuous, discrete and multiple continuous faults are reconfigured correctly. When it comes to multiple faults with discrete faults, the method is only capable of handling 50% responsibility 80% (multiple continuous + discrete) and 50% responsibility 100% (multiple discrete) since multiple faults often lead to conflicts. Multiple states cause the invalid configuration, which may lead to contradicting constraints in the QSM. Hence, the algorithm fails even though a solution exists. Adding a prioritization to the constraints such that faults are handled successively will enable enhanced multiple fault handling.

TABLE V
COMPARISON TO FTC APPROACH BY BLANKE *ET AL.* [5]

Three-Tank Benchmark	FTC	in %	AutoConf	in %
Kind of Fault				
continuous	**3**	**100**	**3**	**100**
valve stuck	3	100	3	100
multiple continuous	**2**	**40**	**5**	**100**
valve stuck + leak in tank	2	67	3	100
two valves stuck	0	0	2	100

C. Comparison to FTC Approach by Blanke et al. [5]

The presented approach is compared to an FTC approach presented by Blanke *et al.* [5] using their three-tank-benchmark. An important difference compared to *AutoConf* is that the approach is based on a DAE representing the CPPS. The creation of the DAE requires high manual efforts and a large amount of expert knowledge while *AutoConf* operates on a hybrid automaton and causal graphs describing. Table V shows the result of the comparison. The approach by Blanke *et al.* [5] tends to fail for multiple fault cases since they lead to major deviations, which require a structural adaptation of the system that is not identified by FTC. *AutoConf* is able of handling all fault cases.

D. Runtime of AutoConf

The size and complexity of CPPS rise permanently [8]. Hence, a reconfiguration algorithm needs to be scalable to be of practical use for CPPS. Formally, reconfiguration is a search for an input mask recovering a valid configuration. A *search space* is a tuple (S, O) where $S = \{s_1, s_2, \ldots\}$ is a set of *search nodes* and $O = \{o_l : s_j \rightarrow s_k, l \in \mathbb{N}\}$ with $s_j, s_k \in S$ is a set of *operations*, which change the search nodes [44]. For reconfiguration, the search nodes $s_i \in S = X \times B$ represent configurations $(\boldsymbol{x}, \boldsymbol{b})$. The operations represent possible changes to the configuration, i.e., changes to the binary input variables. The dimension of the search space depends on the number of state variables n and on the dimension of the binary input variables m. In this approach, the state variables are discretized into three areas (too low, too high, ok) such that the number of search nodes is given by $3^n \times 2^m$. By default, for each node m operations exist, one for each change of a binary variable, leading to $m \times 3^n \times 2^m$ operations in total. Hence, for reconfiguration, an exponentially sized search space needs to be examined.

Simulations of artificially generated tank systems with up to 4000 input variables and 3500 tanks are used to examine the runtime of *AutoConf*. These tank systems are adaptations of the running example, where the number of tanks and their interconnections varies. Various faults are injected and reconfiguration is performed multiple times. Fig. 6 shows the average time for one execution of *AutoConf* as a function of the sum of input variables and state variables since the complexity of reconfiguration directly depends on the number of these variables. Calculation

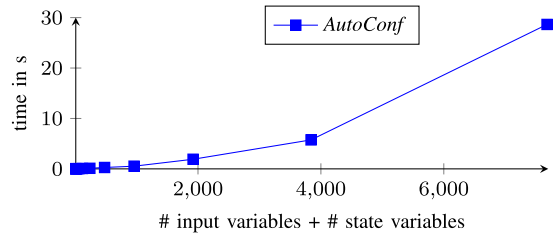

Fig. 6. Calculation time for one execution of *AutoConf*.

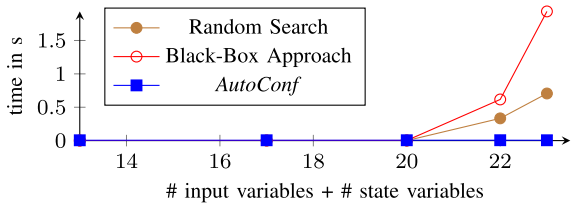

Fig. 7. Comparison to random search, a black-box approach, and *AutoConf*.

ran on a MS Windows system with Intel Core i7-8500Y, 16 GB RAM.

We compare the runtime of *AutoConf* to the established search techniques random search [45] and a black-box approach [46]. Random search enumerates all possibilities in random order; the black-box approach starts with the initial, invalid configuration and searches for the closest solution. Hence, the action of an uninformed operator, who would change inputs one-by-one, is mimicked. To examine scalability, systems with a varying number of state and input variables are used. The comparison is shown in Fig. 7. Random Search and the black-box approach suffer from combinatorial explosion even for small-scale systems while *AutoConf* solves the reconfiguration problems in short times (less than 0.05 s). The black-box approach often finishes in short times for solutions close to the initial configuration, but when it comes to larger adaptations random search is faster since the black-box algorithm contains computationally intense operations.

VI. Conclusion

Various faults, such as broken components or interrupted connections, occur in CPPS due to the increasing size and complexity of the systems. Nowadays, fault handling is mostly done manually, which usually is time-consuming. However, in many cases, the production goal still could be reached by adapting the configuration and control of the CPPS. But due to control handling only known situations, the necessary changes cannot be applied in case of faults that have not been foreseen at design time. To enable CPPS to handle faults autonomously, reconfiguration, i.e., the implementation of a new configuration that allows reestablishing production, is needed.

This article presented a novel approach on reconfiguration for CPPS, which identified an input mask counteracting deviations that occur due to faults. For this purpose, a QSM in propositional logic that describes the CPPS in the context of reconfiguration was created. The QSM integrated an estimation on how the system evolves and also took the capabilities of the controller, which defined valid and invalid configurations, into account (see *RQ1*). The system dynamics were integrated into the QSM using an estimation of how the system evolves. Here, causal graphs describing coherences between input and state variables were used. These causal graphs can either be drawn from a topology description and a minimum of expert knowledge or using learning techniques (see *RQ2*). The search for reconfiguration was implemented in propositional logic, enabling the application of a SAT solver. This comes with the benefits of intuitive modeling and human-understandable reasoning. In addition, the solution space was reduced by the constraints given by the CPPS (see *RQ3*). Experimental results were performed on an industrial use case and simulations from the field of process engineering. The algorithm was shown to handle various single faults and even multiple faults. However, some multiple faults may lead to goal conflicts that cannot be handled at the moment. Compared to an FTC method, the presented approach is shown to handle more and numerous complex faults. The runtime of *AutoConf* was examined using large-scale systems with up to 8000 variables and compared to state-of-the-art approaches, such as random search and a black-box approach. Results showed that *AutoConf* outperformed other approaches but also showed limitations when it comes to systems with more than 4000 variables. To apply *AutoConf* to real-world systems that comprise even more variables, a logic to decompose the system into subsystems will be developed in future work.

Currently, time dependency of the system was handled implicitly assuming that states store past system behavior (limitation to time-invariant systems). To integrate time variance, the definition of configurations and the algorithm will be extended.

Despite the success demonstrated, a significant limitation is that the space of possible adaptations of *AutoConf* is defined by causal graphs. Nonetheless, *AutoConf* is compatible with more complex spaces, e.g., generated from selfdescriptions of system components or digital twins. And since modeling in logic is intuitive, transferring these system models can be done easily. Besides, *AutoConf* identifies only an arbitrary valid solution, not the best one. However, each solution can be assigned a cost function, such that the SAT solver, given an objective function, identifies the best solution.

References

[1] L. Monostori, "Cyber-physical production systems: Roots, expectations and R&D challenges," *Procedia CIRP*, vol. 17, pp. 9–13, 2014.
[2] M. Colledani and A. Angius, "Integrated production and reconfiguration planning in modular plug-and-produce production systems," *CIRP Ann.*, vol. 68, no. 1, pp. 435–438, 2019.
[3] R. Frei *et al.*, "Self-healing and self-repairing technologies," *Int. J. Adv. Manuf. Technol.*, vol. 69, nos. 5–8, pp. 1033–1061, Nov. 2013.
[4] R. P. Goldman, D. Bryce, M. J. Pelican, D. J. Musliner, and K. Bae, "A hybrid architecture for correct-by-construction hybrid planning and control," in *Proc. NASA Formal Methods Symp.*, 2016, pp. 388–394.
[5] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, "Fault accommodation and reconfiguration methods," in *Proc. Diagnosis Fault-Tolerant Control*, 2016, pp. 389–466.
[6] K. Balzereit and O. Niggemann, "Gradient-based reconfiguration of cyber-physical production systems," in *Proc. IEEE Int. Conf. Ind. Cyber Phys. Syst. ICPS*, 2021, pp. 125–131.

[7] K. Schenk and J. Lunze, "Fault tolerance in cooperating systems by redistribution of control tasks," *AT-AUTOM*, vol. 69, no. 6, pp. 442–456, Jun. 2021.
[8] T. Tomiyama and F. Moyen, "Resilient architecture for cyber-physical production systems," *CIRP Ann.*, vol. 67, no. 1, pp. 161–164, 2018.
[9] S. Borgo, A. Cesta, A. Orlandini, and A. Umbrico, "A planning-based architecture for a reconfigurable manufacturing system," in *Proc. Int. Conf. Autom. Plan. Sched.*, 2016, pp. 358–366.
[10] B. Kuipers, "Qualitative simulation," *Encyclopedia Phys. Sci. Technol.*, vol. 3, pp. 287–300, 2001.
[11] H. Khorasgani and G. Biswas, "Structural fault detection and isolation in hybrid systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1585–1599, Oct. 2018.
[12] L. Travé-Massuyès, "Bridging control and artificial intelligence theories for diagnosis: A survey," *Eng. Appl. Artif. Intell.*, vol. 27, pp. 1–16, Jan. 2014.
[13] B. Wang, B. Zhang, and R. Su, "Optimal tracking cooperative control for cyber-physical systems: Dynamic fault-tolerant control and resilient management," *IEEE Trans. Ind. Informat.*, vol. 17, no. 1, pp. 158–167, Jan. 2021.
[14] R. Stern and B. Juba, "Safe partial diagnosis from normal observations," in *Proc. 33rd AAAI Conf. Artif. Intell. 31st Innov. Appl. Artif. Intell. Conf. 9th AAAI Symp. Edu. Adv. Artif. Intell.*, 2019, pp. 3084–3091.
[15] A. Feldman, I. Pill, F. Wotawa, I. Matei, and J. de Kleer, "Efficient model-based diagnosis of sequential circuits," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 2814–2821.
[16] Z. Zhao, Y. Yang, S. X. Ding, and L. Li, "Fault-tolerant control for systems with model uncertainty and multiplicative faults," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 50, no. 2, pp. 514–524, Feb. 2020.
[17] B. Guo and Y. Chen, "Event-triggered robust adaptive sliding mode fault-tolerant control for nonlinear systems," *IEEE Trans. Ind. Informat.*, vol. 16, no. 11, pp. 6982–6992, Nov. 2020.
[18] H. Ma, Q. Zhou, L. Bai, and H. Liang, "Observer-based adaptive fuzzy fault-tolerant control for stochastic nonstrict-feedback nonlinear systems with input quantization," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 49, no. 2, pp. 287–298, Feb. 2019.
[19] B. Cai, L. Huang, and M. Xie, "Bayesian networks in fault diagnosis," *IEEE Trans. Ind. Informat.*, vol. 13, no. 5, pp. 2227–2240, Oct. 2017.
[20] M. Seitz, F. Gehlhoff, L. A. C. Salazar, A. Fay, and B. Vogel-Heuser, "Automation platform independent multi-agent system for robust networks of production resources in industry 4.0," *J Intell. Manuf.*, vol. 32, pp. 2023–2041, Oct. 2021.
[21] N. Rodrigues, E. Oliveira, and P. Leitão, "Decentralized and on-the-fly agent-based service reconfiguration in manufacturing systems," *Comput. Ind.*, vol. 101, pp. 81–90, Oct. 2018.
[22] S. Karnouskos, P. Leitao, L. Ribeiro, and A. W. Colombo, "Industrial agents as a key enabler for realizing industrial cyber-physical systems: Multiagent systems entering industry 4.0," *IEEE Ind. Electron. Mag.*, vol. 14, no. 3, pp. 18–32, Sep. 2020.
[23] EFFRA, "EFFRA Vision for a manufacturing partnership in Horizon Europe. 2021–2027," Eur. Factories Future Res. Assoc., Brussels, Belgium, Tech. Rep., 2019. [Online]. Available: www.effra.eu/sites/default/files/190312_effra_roadmapmanufacturingppp_eversion.pdf
[24] T. Strasser, M. Rooker, G. Ebenhofer, and A. Zoitl, "Standardized dynamic reconfiguration of control applications in industrial systems," in *Proc. Rapid Automat.: Concepts, Methodol., Tools, Appl.*, 2019, pp. 776–793.
[25] *Function Blocks for Industrial-Process Measurement and Control Systems.Part 1: Architecture*, IEC 61499, 2005.
[26] J. Ladiges *et al.*, "Integration of modular process units into process control systems," *IEEE Trans. Ind. Appl.*, vol. 54, no. 2, pp. 1870–1880, Mar. 2018.
[27] R. Sinha, K. Johnson, and R. Calinescu, "A scalable approach for reconfiguring evolving industrial control systems," in *Proc. IEEE Int. Conf. Emerg. Technol. Fact. Autom.*, 2014, pp. 1–8.
[28] J. Otto, B. Vogel-Heuser, and O. Niggemann, "Automatic parameter estimation for reusable software components of modular and reconfigurable cyber-physical production systems in the domain of discrete manufacturing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 1, pp. 275–282, Jan. 2018.
[29] G. Engel, T. Greiner, and S. Seifert, "Ontology-assisted engineering of cyber-physical production systems in the field of process technology," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2792–2802, Jun. 2018.
[30] J. Wan, B. Yin, D. Li, A. Celesti, F. Tao, and Q. Hua, "An ontology-based resource reconfiguration method for manufacturing cyber-physical systems," *IEEE/ASME Trans. Mechatronics*, vol. 23, no. 6, pp. 2537–2546, Dec. 2018.
[31] M. Schneider, D. Lucke, and T. Adolf, "A cyber-physical failure management system for smart factories," *Procedia CIRP*, vol. 81, pp. 300–305, 2019.
[32] R. Guo, L. Cheng, J. Li, P. R. Hahn, and H. Liu, "A survey of learning causality with data: Problems and methods," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–75, Jul. 2020.
[33] A. Biere, "CaDiCaL, lingeling, plingeling, treengeling and YalSat entering the SAT competition 2018," in *Proc. SAT Comp.*, 2018, pp. 13–14.
[34] J. Pearl and D. Mackenzie, *The Book of Why: The New Science of Cause and Effect*, 1st ed. New York, NY, USA: Basic books, 2018.
[35] K. Vännman and M. Albing, "Process capability indices for one-sided specification intervals and skewed distributions," *Qual. Rel. Eng. Int.*, vol. 23, no. 6, pp. 755–765, Oct. 2007.
[36] M. Schleipen, A. Lüder, O. Sauer, H. Flatt, and J. Jasperneite, "Requirements and concept for plug-and-work, adaptivity in the context of industry 4.0," *Automatisierungstechnik*, vol. 63, no. 10, pp. 801–820, Oct. 2015.
[37] R. He, G. Chen, C. Dong, S. Sun, and X. Shen, "Data-driven digital twin technology for optimized control in process systems," *ISA Trans*, vol. 95, pp. 221–234, Dec. 2019.
[38] J. Downs and E. Vogel, "A plant-wide industrial process control problem," *Comput. Chem. Eng.*, vol. 17, no. 3, pp. 245–255, 1993.
[39] H. Chen, P. Tiňo, and X. Yao, "Cognitive fault diagnosis in Tennessee Eastman process using learning in the model space," *Comput. Chem. Eng.*, vol. 67, pp. 33–42, Aug. 2014.
[40] G. Manca, "Tennessee-Eastman-Process–Alarm management dataset," Accessed: Sep. 24, 2021. [Online]. Available: https://dx.doi.org/10.21227/326k-qr90
[41] N. Rezki, O. Kazar, L. H. Mouss, L. Kahloul, and D. Rezki, "On the use of multi-agent systems for the monitoring of industrial systems," *J. Ind. Eng. Int.*, vol. 12, no. 1, pp. 111–118, Mar. 2016.
[42] J. L. de la Mata and M. RodrÍguez, "Accident prevention by control system reconfiguration," *Comput. Chem. Eng.*, vol. 34, no. 5, pp. 846–855, May 2010.
[43] K. Balzereit, A. Diedrich, J. Ginster, S. Windmann, and O. Niggemann, "An ensemble of benchmarks for the evaluation of AI methods for fault handling in CPPS," in *Proc. IEEE 19th Int. Conf. Ind. Informat.*, 2021, pp. 1–6.
[44] J. Pearl and R. E. Korf, "Search techniques," *Annu. Rev. Comput. Sci.*, vol. 2, no. 1, pp. 451–467, Jun. 1987.
[45] B. Mondal, *Artificial Intelligence: State of the Art*. Cham, Switzerland: Springer, 2020, pp. 389–425.
[46] P. M. Pardalos, V. Rasskazova, and M. N. Vrahatis, *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, 1st ed. Cham, Switzerland: Springer, 2021.

Kaja Balzereit (Graduate Student Member, IEEE) received the B.Sc. degree in applied mathematics and the M.Sc. degree in optimization and simulation from the Bielefeld University of Applied Sciences, Bielefeld, Germany, in 2016 and 2017, respectively. She is currently working toward the Ph.D. degree in artificial intelligence for industrial automation with Fraunhofer Industrial Automation Branch, Fraunhofer IOSB-INA, Lemgo, Germany.

For her Ph.D., she is researching symbolic AI methods for intelligent fault handling in hybrid systems, in order to increase the resilience of modern production facilities to external and internal disturbances and varying customer demands. Since 2017, she has been a Research Associate with the Fraunhofer Industrial Automation Branch. Her research interests include artificial intelligence and machine learning for cyber-physical production systems.

Oliver Niggemann (Member, IEEE) received the Ph.D. degree in visual data mining of graph-based data from Paderborn University, Paderborn, Germany, in 2001.

He, then, worked for eight years in leading positions in industry. From 2008 to 2019, he had a professorship with the Institute for Industrial Information Technologies, Lemgo, Germany. Until 2019, he was also the Deputy Head of the Fraunhofer IOSB-INA, which works in industrial automation. On April 1, 2019, he took over the university professorship of Computer Science in mechanical engineering with Helmut Schmidt University, Hamburg, Germany, where he is conduction research at the Institute for Automation Technology IfA in the field of artificial intelligence and machine learning for cyber-physical systems.