Automated Formal Verification of Routing in Material Handling Systems

Thomas Klotz, *Student Member, IEEE,* Jens Schönherr, Norman Seßler, Bernd Straube, and Karsten Turek

***Abstract*—The design of correctly implemented controls in material handling systems (MHS) is time consuming and cumbersome. The developer has to deal with an ever increasing complexity and heterogeneity of MHS on the one hand, but also with short development cycles and high demands to MHS on the other hand. For baggage handling systems (BHS) at airports, the error-free implementation of routing strategies is especially of importance, as these strategies are critical to safety. This paper proposes a compositional approach to the formal verification of routing in MHS. The approach is based on the theory of assume-guarantee reasoning, where proofs of the overall system are derived from proofs of subsystems. Moreover, the approach has been implemented in a tool that automatically carries out the verification. A real-world example is discussed in the paper, showing the benefits and scalability of the presented approach.**

***Note to Practitioners*— Routing strategies in MHS such as BHS at airports can be complex, and thus, their implementation is tedious and error-prone. Currently, these systems are simulated to validate their control strategies, and then implemented on real equipment. With this, however, only a limited number of test cases is considered, and hence, routing errors may stay undetected. This paper proposes an approach to automatically prove the correctness of routing strategies in MHS by means of formal verification. A main difficulty of the formal verification of real-world MHS is the so-called state space explosion, i.e. the model to be verified has too many states. To also apply formal verification to these systems, a compositional approach is presented which allows to verify system properties with regard to routing in a reasonable amount of time. A case study shows the application of the approach to the BHS of an international airport. A number of properties with respect to routing functionality have been proven automatically for this system, for instance, no unclear or unidentified bags reach an airplane. Furthermore, the proposed approach is not restricted to routing only and can be extended for other system properties.**

***Index Terms*—material handling systems (MHS), baggage handling systems (BHS), routing, formal verification, model checking**

Manuscript received May 16, 2013; revised July 24, 2013; accepted July 30, 2013. This paper was recommended for publication by Associate Editor T. Nishi and Editor M. C. Zhou upon evaluation of the reviewers' comments. This work was partially supported by the German Research Foundation under Grant STR 412/4-1 and Grant SCHM 2689/3-1. This paper was presented in part at the Eight IEEE International Conference on Automation Science and Engineering, Seoul, South Korea. *(Corresponding author: T. Klotz.)*

T. Klotz, N. Seßler, and B. Straube are with the Fraunhofer Institute for Integrated Circuits, Design Automation Division, Dresden 01069, Germany (e-mail: thomas.klotz@eas.iis.fraunhofer.de; norman.sessler@eas.iis.fraunhofer.de; bernd.straube@eas.iis.fraunhofer.de).

J. Schönherr is with the Dresden University of Applied Sciences, Dresden 01069, Germany (e-mail: schoenherr@htw-dresden.de).

K. Turek is with the Institute of Material Handling and Industrial Engineering, Technische Universität Dresden, Dresden 01062, Germany (e-mail: karsten.turek@tu-dresden.de).

I. Introduction

TODAY, *material handling systems* (MHS) controlling the flow of goods play an important role in most industries. Moreover, they are a crucial factor in the overall production of goods. With the rising degree of automation, especially in manufacturing and distribution, the impact of material handling will increase even further in the future. In a typical manufacturing plant, material handling accounts for 25% of all employees, 55% of all company space, 87% of the production time, and 15–70% of the total cost of a product [1]. Hence, there are strong demands on MHS: beside being cost-efficient and having a large throughput, they are expected to be free of design errors, safe, highly available, but also flexible.

A common field of application of MHS are *baggage handling systems* (BHS) at airports. There, conveyor lines span up to hundreds of kilometers, consist of tens of thousands of conveyor segments and transport more than twenty thousand pieces of luggage per hour. The ongoing globalization process and the fact that more and more people travel result in ever-increasing demands to these systems. However, due to the AEA consumer report for January–June 2012 [2], these demands have not been completely fulfilled. As a consequence, 1.7 million travelers had to wait for their delayed luggage at European airports.

In BHS, especially the implementation of complex routing strategies is cumbersome and error-prone as these strategies have to deal with load balancing, alternative paths in case of a failure, several levels of security checks, etc. In addition, routing can partially be implemented based on local strategies on programmable logic controller level and based on global strategies provided by the material flow control. In conclusion, the correct implementation of these routing strategies is crucial, non-trivial, and safety-critical.

By validating the MHS using simulation, design errors can be found depending on the considered test cases. However, the absence of such errors cannot be verified. Model checking [3] as a formal method, though, provides a general means to *prove* the correctness of the system with regard to its specification.

The authors of this paper proposed a modeling methodology for MHS [4] that allows for the formal verification of these systems. In this methodology, the MHS is modeled using so-called *MHS elements*. Each of these elements is a finite state model of a technical MHS component. By interconnecting these elements, a formal model of the MHS is built, which thereafter can be fed into a model checker to check the specified properties on the model. With this, requirements to MHS can be automatically proven. Unfortunately, the verification using this methodology has been hampered because it could

only be applied to MHS of a moderate size. On the other hand, real-world MHS may consist of thousands of interacting components and their controls, i.e. their corresponding formal models may be complex and large.

To also verify real-world MHS, a compositional approach for the verification of MHS has been developed and is presented in this paper. The approach is based on the theory of assume-guarantee reasoning (AGR) which allows to derive properties of the overall system from proving properties of its subsystems. Therefore, the MHS is partitioned into subsystems that are subsequently verified separately. A challenge in AGR is to determine appropriate assumptions about a subsystem in order succeed with a proof. The proposed approach has specifically been developed for the verification of MHS, where such assumptions can automatically be computed. This paper focuses on the verification of routing in MHS; the general idea, though, can also be considered for checking other properties.

The rest of the paper is organized as follows: Section II gives an overview on recent related work. Then, a brief background in model checking and compositional verification is given in Section III. The modeling methodology for MHS is revisited in Section IV. Next, a novel compositional approach to the verification of routing in MHS is thoroughly explained in Section V. The application of the approach to formally verify routing is shown first with an illustrative example and second by considering the BHS of an international airport, as a real-world case study, in Section VI. Finally, Section VII summarizes the work and provides pointers for future work.

II. Related work

The existing related work and the aim of this work is classified using the VDI guideline 3628 [5]. This guideline provides a definition for the abstraction layers of the control of automated MHS:

- Administration Planning
- Warehouse Management System
- Material Flow Control System
- Conveying Segment Control
- Conveying Group Control
- Conveying Element Control

Administration planning (HOST) includes the storage of central data, e.g. about incoming deliveries, orders, total stock. The *warehouse management system* (WMS) controls and monitors the movement and storage of goods in a warehouse, which includes e.g. choosing of storage and retrieval strategies. The *material flow control system* (MFCS) controls and optimizes transports, and continually monitors the availability of the existing material handling equipment. *Conveying segment control* (CSC), *conveying group control* (CGC), and *conveying element control* (CEC) hierarchically control the conveying system by reading inputs from sensors and writing outputs to actuators. The latter task is typically implemented using programmable logic controllers (PLCs).

Petri nets (PNs) have been utilized for the analysis of MHS at a high level of abstraction (HOST, WMS), e.g. for modeling supply chains and workflows, schedulability analysis, stochastic evaluation, see for instance [6]–[11]. Contrary, this paper aims at the verification of the layer MFCS and the conveying controls, i.e. considers a lower level of abstraction.

PNs have also been employed to verify routing in automated guided vehicle systems (AGVS). These systems employ a number of driver-less vehicles to transport goods or material following certain guide-paths. A central control assigns routes and tasks to the vehicles. Dispatching and routing in AGVS is more complex than in conveyor systems because all vehicles may change their initial positions. Hence, approaches to prevent deadlocks [12]–[14] and to find shortest routes [15] in AGVS have been proposed. To deal with computational complexity, decomposition [16], [17] has been applied.

The formal verification of PLC programs has been tackled by a plethora of work in the last two decades [18]. Most work has been done with regard to the formalization of PLC code according to IEC 61131-3 [19], e.g. for ladder diagrams (LD) [20]–[22], instruction lists (IL) [23]–[25], function block diagrams (FBD) [26], and sequential function charts (SFC) [27]–[30]. More recently, also model-based approaches to the verification of PLCs gained interest [31]–[33]. On the other hand, the verification of single PLCs only captures local behavior. Thus, global properties of an MHS, e.g. with regard to deadlock freedom or correctly implemented routing algorithms, cannot be analyzed at this level of abstraction.

Ljungkrantz *et al.* [34] proposed the concept of reusable automation components. Each component consists of an implementation as well as a mathematical specification. This specification is formulated using a special type of LDs that are augmented with temporal logic. This ensures that each component can be verified and maintained separately, but the overall system behavior that is given by the interaction and composition of the components is not taken into account.

Another related field is the verification of routing algorithms implemented in communication networks. Renesse and Aghvami [35] showed how to formally verify a five node network using the SPIN model checker. The state space explosion seems to hamper the application to larger networks. Camara *et al.* [36] proposed an approach to the verification of mobile ad-hoc networks. However, these approaches are not applicable to the verification of MHS, as the underlying models are too abstract and not appropriate for describing MHS.

Currently, mainly simulation methods are applied to validate the MFCS and the conveying controls of MHS. In industry, especially commercial simulators such as Automod [37] or Flexsim [38] are utilized for this purpose. Besides the existing commercial solutions, there is also current research dealing with the simulation of MHS. Seidel *et al.* [39] introduced a simulation-based validation tool for MHS. There, one simulation model is used for a material flow simulation of the plant model as well as to validate the interaction of the plant model with the material flow control and the involved PLCs. Black and Vyatkin [40] proposed a methodology to the design of BHS, where the BHS is implemented as a distributed automation system according to IEC 61499. Each technical component such as a conveyor is implemented by a function block and viewed as an agent. For these components, model

components have been created. Using these models, simulation runs can be conducted. Simulation-based methods, though, typically capture only parts of the behavior of the overall system. In contrast, formal methods are capable of proving the correctness of a system with regard to its specification.

The actual control architecture for BHS and the determination of efficient routing strategies has also been studied in literature. Tarau *et al.* [41] analyzed concepts to find an optimal routing in case of dynamic demand in BHS. They compare centralized, decentralized, and also distributed predictive methods. Johnstone *et al.* [42] also compared centralized, search-based routing strategies with decentralized, learning-based approaches to make optimal routing decisions. A more general overview on distributed control using agents can be found in [43]. Haneyah *et al.* [44] provided a comprehensive review of the current literature regarding control structures for MHS. General requirements to an MHS, e.g. regarding throughput, blocking, deadlock avoidance, etc. are discussed, and a derived concept of a control architecture for MHS is proposed. Unfortunately, the presented work deals only with how optimal routing decisions with respect to performance are made but does not provide any means to ensure the correctness of the implementation of the proposed strategies. Contrary, this paper proposes an approach to automatically prove the correctness of the implemented material flow control, in particular, whether cargos reach their correct destination.

In conclusion, the existing work aims either at a higher or at a lower level of abstraction than the work presented in this paper. There is no existing work dealing with the formal verification of MHS that takes the actual MHS layout, the involved technical components, the material flow control system, and the conveying controls into account.

III. Formal verification

This section briefly introduces model checking and compositional verification. For a more detailed presentation, the reader is referred to [3], [45]. First, Section III-A discusses mathematical models used for formal verification. Next, in Section III-B, temporal logics and model checking are presented. Finally, assume-guarantee reasoning, which is a compositional verification technique, is explained in Section III-C.

A. Models

For specifying and analyzing systems, a mathematical description (a model) of the system is needed. For this purpose, the notion of finite state machines (FSM) [46] describing a system with inputs, states, and outputs, is often considered.

Definition 1. A deterministic Finite State Machine of Mealy type is defined as the 5-tuple $M = (S, s_0, I, O, \sigma, \lambda)$ where S represents the finite set of states; $s_0 \in S$ marks the initial state; I is the finite set of input variables; O is the finite set of output variables with $I \cap O = \emptyset$; σ is the next state function $\sigma : S \times \Sigma_I \mapsto S$; λ is the output function $\lambda : S \times \Sigma_I \mapsto \Sigma_O$. Each input and output variable v has a corresponding domain D_v of possible values. The input and output alphabet are defined as $\Sigma_I = \prod_{i \in I} D_i$ and $\Sigma_O = \prod_{o \in O} D_o$, respectively.

In formal verification, systems are usually described as labeled transition systems such as *fair Kripke structures* [3].

Definition 2. A fair Kripke structure K is a 6-tuple $K = (S, S_0, A, L, R, F)$ where S is a finite set of states; $S_0 \subseteq S$ is the set of initial states; A is a finite set of atomic propositions; $L : S \mapsto 2^A$ is a function that labels each state with the set of atomic propositions true in that state; $R \subseteq S \times S$ is a total transition relation; $F \subseteq 2^S$ is a set of fairness constraints.

An execution of a Kripke structure, called a path, is an infinite sequence of states where possible successors of a state are defined by the transition relation. A *fair path* must contain an element of each fairness constraint infinitely often. A *trace* is the sequence of observable atomic propositions on a path. The *diameter* of a Kripke structure is the minimum number of transitions that is required to reach all states of the structure from an initial state.

Systems to be modeled are normally divided into a number of modules, each describing a specific part of the functionality of the overall system. If each of these modules is modeled using a Kripke structure, the composition of Kripke structures needs to be defined.

Definition 3. The composition K'' of two fair Kripke structures K and K', denoted as $K \parallel K'$, is defined as

- $S'' = \{(s, s') \mid L(s) \cap A' = L'(s') \cap A\}$,
- $S_0'' = (S_0 \times S_0') \cap S''$,
- $A'' = A \cup A'$,
- $L''((s, s')) = L(s) \cup L'(s')$,
- $R''((s, s'), (t, t'))$ iff $R(s, t)$ and $R'(s', t')$, and
- $F'' = \{(F_i \times S') \cap S'' \mid F_i \in F\} \cup \{(S \times F_i') \cap S'' \mid F_i' \in F'\}$.

For modeling purposes Mealy machines are often preferred to Kripke structures. A Mealy machine can be converted into a Kripke structure by adding the input and output values of the FSM to the state information of the Kripke structure [3].

B. Model Checking

For the specification of properties describing the desired behavior of a Kripke structure, temporal logics such as *computation tree logic* (CTL) [47] are considered. By restricting CTL to allow universal quantifiers only, a subset of the logic called A-computation tree logic (ACTL) [45] is derived. ACTL formulas make propositions about *all* computation paths of a system. They consist of atomic propositions, boolean operators, and the universal path quantifier $\mathbf{A}$. Negation occurs only before atomic formulas. The four basic temporal operators are $\mathbf{G}$ ("globally"), $\mathbf{F}$ ("future"), $\mathbf{X}$ ("next time") and $\mathbf{U}$ ("until").

Given a fair Kripke structure K and an ACTL formula φ, the model checking algorithm [3] determines if K satisfies φ, denoted as $K \models \varphi$. A major problem of model checking is the *state space explosion problem*. This refers to the fact that the number of states of a model is exponential in the size of the system.

C. Assume-Guarantee Reasoning

One method to tackle the state space explosion problem is compositional verification [3]. It follows the principle of divide

and conquer: a proof of a property of the whole system is derived from proofs of properties of its modules. With this, the model checking is carried out on the single modules separately, and thus, the number of involved states is reduced.

However, modules are usually designed in such a way that they only fulfill certain properties in a given environment (context). Based on this idea, *assume-guarantee reasoning* (AGR) [48] has been proposed. In AGR, tuples of the form $\langle\varphi\rangle\, K\, \langle\psi\rangle$ are checked, which state that if the module K is part of a model that satisfies the assumption φ, then it is guaranteed that the property ψ holds.

To prove a property ψ for the composition of two modules K_1 and K_2, the basic assume-guarantee rule [45] can be used:

$$\frac{\begin{array}{ccc}\langle\varphi\rangle & K_1 & \langle\psi\rangle \\ \langle true\rangle & K_2 & \langle\varphi\rangle\end{array}}{\langle true\rangle\, K_1 \parallel K_2\, \langle\psi\rangle} \qquad \text{(AGR-2)}$$

The rule denotes that if K_1 satisfies ψ under the assumption φ, and if K_2 satisfies assumption φ, then the composition of K_1 and K_2 satisfies ψ. The advantage of this approach is that only the state spaces of K_1 and K_2 have to be examined separately, but not the (possibly huge) state space of the composition $K_1 \parallel K_2$. In order to derive a sound conclusion, the base assumption φ itself must be proven without any further assumptions.

By repeatedly applying the basic AGR rule, it can be generalized to n modules:

$$\frac{\begin{array}{ccc}\langle\varphi_1\rangle & K_1 & \langle\psi\rangle \\ \langle\varphi_2\rangle & K_2 & \langle\varphi_1\rangle \\ & \vdots & \\ \langle\varphi_{n-1}\rangle & K_{n-1} & \langle\varphi_{n-2}\rangle \\ \langle true\rangle & K_n & \langle\varphi_{n-1}\rangle\end{array}}{\langle true\rangle\, K_1 \parallel K_2 \parallel \ldots \parallel K_n\, \langle\psi\rangle} \qquad \text{(AGR-}n\text{)}$$

With this rule, starting without any further assumptions ($\langle true\rangle\, K_n\, \langle\varphi_{n-1}\rangle$), a chain of conclusions is derived in terms of φ_i until finally the property ψ is proven ($\langle\varphi_1\rangle\, K_1\, \langle\psi\rangle$). In each step, the derived conclusion φ_i is taken into account as an assumption for the next step.

Grumberg and Long [45] identified that the framework consisting of the logic ACTL, the notion of fair Kripke structures K and their composition $\parallel$, together with the fair simulation preorder $\preceq$ allows to implement AGR using a standard model checker. The simulation preorder $K \preceq K'$ can be viewed as "K is smaller than K'", "K refines K'", or "K' abstracts K". In their framework, the following theorem holds: $K \preceq K'$ and $K' \models \varphi$ implies $K \models \varphi$, i.e. if the property φ holds for a model, then it also holds for any model that is smaller in the preorder.

A major challenge in AGR is the determination of sufficient assumptions in order to prove a certain property. In most cases, these assumptions have to be manually identified and specified. Current research [49] focuses on automatically determining such assumptions by learning algorithms. Nevertheless, these learning approaches are computationally expensive, and thus not suitable yet for applying them to industrial examples.

IV. Modeling Methodology for MHS

This section introduces the modeling methodology for MHS as described in [4]. First, necessary preliminaries are provided in Section IV-A. Next, the notion of MHS elements is revisited in Section IV-B, and then, MHS elements with material flow control are studied in Section IV-C. Finally, the interconnection of MHS elements to networks is discussed in Section IV-D.

A. Preliminaries

In general, processes in a technical MHS are continuous, i.e. goods are either continuously in motion or stopped, e.g. to wait at a divert. In practice, goods are mostly combined to *unit loads* [50] by using loading aids such as pallets, containers, trays, bins, and so on.

However, for applying the formal verification method model checking to MHS, a discrete and finite model of this continuous system is mandatory. Hence, space and time of the technical MHS have to be discretized.

The *space* is discretized in terms of so-called *places*. It is further assumed that each cargo has the same size, which is also the size of a place. As a consequence, on each place there can be either one cargo or no cargo.

For discretizing *time*, it is assumed that a cargo can move from one place to another place in one time step.

Moreover, a finite number of *types of cargos* is considered. A type of a cargo may describe the destination of the cargo, e.g. to mark all bags in a BHS that have to be routed to a certain terminal. There, the set $\mathcal{C}$ defines the types of cargos that have to be distinguished. A cargo type is specified by a letter such as $a, b, c, \ldots$. Furthermore, the special cargo type $L \in \mathcal{C}$ represents the absence of a cargo at a place.

The assumptions have particularly been chosen to model the behavior of MHS transporting general cargo, i.e. individually packaged goods. Currently, the models are restricted in such a way that all cargos have them same size. However, this is especially true for BHS where bags are usually transported using loading aids (trays), which all have the same size. This ensures safe transport as all trays behave the same, i.e. have the same grip, there are no rollovers, two cargos cannot end up on top of each other, the sequence of the cargos stays the same, etc. Allowing different cargo sizes can be realized by extending the modeling methodology, in particular by changing the modeled MHS elements (cf. Section IV-B). The assumptions do not suffice to model all kinds of MHS, e.g. are not appropriate to model bulk cargo, i.e. loose material.

Based on the assumptions, an *MHS element* specifies a number of places and when cargos move from one place to another. As an illustrative example, a simple MHS element modeling a conveyor of length l could consist of l neighboring places, and a control logic determining when cargos move from one place to the next place.

A technical MHS is composed of a quantity of commonly used *MHS components* such as conveyors, working stations, pushers, etc. For each of these technical MHS components, a behavioral model in terms of an MHS element is created. The model of the whole technical MHS is obtained

by interconnecting instances of the single MHS elements to an *MHS network*. This MHS network is utilized for formal verification, simulation, analysis, etc.

B. MHS Elements

An MHS element is a finite state behavioral model of a technical MHS component with a defined interface in terms of input and output variables. More specifically, an MHS element refers to a number of so-called input and output *connection ports*, which are used to interconnect MHS elements. Through input connection ports, connections to preceding MHS elements are set up, whereas output connection ports allow for connections to succeeding MHS elements. The connection ports are considered to model the actual flow of cargos and the control flow between MHS elements. In addition, an MHS element has a number of input and output *signals* to model control flow between MHS elements that are neither a direct predecessor nor a direct successor of each other.

Definition 4. For the interconnection of MHS elements, two types of connection ports are distinguished:

- input connection port CP^i_{in} is the finite set of variables: $\mathrm{CP}^i_{in} = \{IN^i, Pre^i, Take^i, Clear^i\}$, and
- output connection port CP^j_{out} is the finite set of variables: $\mathrm{CP}^j_{out} = \{OUT^j, Give^j, Suc^j, Ack^j\}$.

Considering the notion of connection ports, abstract MHS elements are defined in the sequel.

Definition 5. An abstract MHS element $\hat{\mathrm{E}}$ is defined as $\hat{\mathrm{E}} = (\hat{\mathcal{CP}}^{\mathrm{E}}_{in}, \hat{\mathcal{CP}}^{\mathrm{E}}_{out}, \hat{\mathcal{S}}^{\mathrm{E}}_{in}, \hat{\mathcal{S}}^{\mathrm{E}}_{out}, \hat{M}^{\mathrm{E}})$, where

- $\hat{\mathcal{CP}}^{\mathrm{E}}_{in}$ is the finite set of input connection ports $\hat{\mathcal{CP}}^{\mathrm{E}}_{in} = \{\mathrm{CP}^1_{in}, \mathrm{CP}^2_{in}, \ldots, \mathrm{CP}^p_{in}\}$ with $p = |\hat{\mathcal{CP}}^{\mathrm{E}}_{in}|$ ($p \in \mathbb{N}$),
- $\hat{\mathcal{CP}}^{\mathrm{E}}_{out}$ is the finite set of output connection ports $\hat{\mathcal{CP}}^{\mathrm{E}}_{out} = \{\mathrm{CP}^1_{out}, \mathrm{CP}^2_{out}, \ldots, \mathrm{CP}^s_{out}\}$ with $s = |\hat{\mathcal{CP}}^{\mathrm{E}}_{out}|$ ($s \in \mathbb{N}$),
- $\hat{\mathcal{S}}^{\mathrm{E}}_{in}$ is the finite set of input signals, $\hat{\mathcal{S}}^{\mathrm{E}}_{in} = \{\mathrm{S}^1_{in}, \mathrm{S}^2_{in}, \ldots, \mathrm{S}^k_{in}\}$ with $k = |\hat{\mathcal{S}}^{\mathrm{E}}_{in}|$ ($k \in \mathbb{N}$),
- $\hat{\mathcal{S}}^{\mathrm{E}}_{out}$ is the finite set of output signals, $\hat{\mathcal{S}}^{\mathrm{E}}_{out} = \{\mathrm{S}^1_{out}, \mathrm{S}^2_{out}, \ldots, \mathrm{S}^m_{out}\}$ with $m = |\hat{\mathcal{S}}^{\mathrm{E}}_{out}|$ ($m \in \mathbb{N}$), and
- $\hat{M}^{\mathrm{E}}$ is a deterministic FSM $\hat{M}^{\mathrm{E}} = (S, s_0, I, O, \sigma, \lambda)$ with
  - $I = \{IN^1, \ldots, IN^p, Pre^1, \ldots, Pre^p, Stop, Ack^1, \ldots, Ack^s, Suc^1, \ldots, Suc^s, \mathrm{S}^1_{in}, \mathrm{S}^2_{in}, \ldots, \mathrm{S}^k_{in}\}$, and
  - $O = \{OUT^1, \ldots, OUT^s, Give^1, \ldots, Give^s, Error, Clear^1, \ldots, Clear^p, Take^1, \ldots, Take^p, \mathrm{S}^1_{out}, \mathrm{S}^2_{out}, \ldots, \mathrm{S}^m_{out}\}$,

where p is the number of predecessors, s is the number of successors, k is the number of input signals, and m is the number of output signals of the abstract MHS element $\hat{\mathrm{E}}$. The input and output variables of $\hat{M}^{\mathrm{E}}$ are identified with the corresponding variables in the connection ports $\mathrm{CP}^i_{in} \in \hat{\mathcal{CP}}^{\mathrm{E}}_{in}$ and $\mathrm{CP}^j_{out} \in \hat{\mathcal{CP}}^{\mathrm{E}}_{out}$, and the signals $\mathrm{S}_{in} \in \hat{\mathcal{S}}^{\mathrm{E}}_{in}$ and $\mathrm{S}_{out} \in \hat{\mathcal{S}}^{\mathrm{E}}_{out}$.

Both, input and output connection ports refer to input and output variables; all input variables from connection ports and input signals are input variables to the FSM $\hat{M}^{\mathrm{E}}$. Contrary, the output variables of $\hat{M}^{\mathrm{E}}$ refer to either output variables in the connection ports or to output signals. In addition to the variables of the connection ports, each abstract MHS element $\hat{\mathrm{E}}$ has one input variable called $Stop$, to halt the element, and one output variable named $Error$, to signalize error states. The FSM $\hat{M}^{\mathrm{E}}$ specifies the behavior of the MHS element. To ease modeling, $\hat{M}^{\mathrm{E}}$ may be the composition of several FSMs again, each describing a part of the behavior of the overall behavior of the MHS element.

The input and output variables of the connection ports have a given semantics [4], e.g. the variable $Give$ may only become $true$ if the MHS element requests to transfer a cargo to the successor. The variables $Clear$ and Ack are only necessary for segmented MHS elements, where the receipt of each cargo has to be acknowledged before the next cargo can be transported. For MHS elements transporting more than one cargo at a time, these variables can be omitted.

To achieve a *closed* MHS network, the two special abstract MHS elements *MHS source* and *MHS sink* are introduced.

Definition 6. An MHS source, denoted as SRC, is an abstract MHS element with one output connection port and without input connection ports, i.e. $|\hat{\mathcal{CP}}^{\mathrm{E}}_{in}| = 0$ and $|\hat{\mathcal{CP}}^{\mathrm{E}}_{out}| = 1$. In contrast, an MHS sink, denoted as SNK, is an MHS element with one input connection port and without output connection ports, i.e. $|\hat{\mathcal{CP}}^{\mathrm{E}}_{in}| = 1$ and $|\hat{\mathcal{CP}}^{\mathrm{E}}_{out}| = 0$.

For differentiating between MHS sources, MHS sinks, and abstract MHS elements that are neither a source nor a sink, the definition of the abstract MHS element $\hat{\mathrm{E}}$ is restricted to define the concrete MHS element E.

Definition 7. A concrete MHS element $\mathrm{E} = (\mathcal{CP}^{\mathrm{E}}_{in}, \mathcal{CP}^{\mathrm{E}}_{out}, \mathcal{S}^{\mathrm{E}}_{in}, \mathcal{S}^{\mathrm{E}}_{out}, M^{\mathrm{E}})$ refers to an abstract MHS element $\hat{\mathrm{E}}$ with at least one input and at least one output connection port, i.e. $|\mathcal{CP}^{\mathrm{E}}_{in}| \geq 1$ and $|\mathcal{CP}^{\mathrm{E}}_{out}| \geq 1$.

Among the MHS elements that have been modeled are conveyors, merger, pushers, accumulative conveyors, lifts, working stations, turntables, etc. In [4], some examples of common MHS elements are explained more in detail.

C. MHS Elements with Material Flow Control

The material flow control in an MHS is normally implemented in the MFCS (cf. Section II). The MFCS acts as a global instance in the MHS and receives status messages from the technical MHS components, e.g. about the fact that a cargo has been received at a component. Through these status messages, the MFCS updates an image of the MHS, i.e. it stores where each cargo is located in the MHS. Moreover, the MFCS sends control signals to the MHS components, for instance, to set global routing decisions.

In this work, the global MFCS functionality is split up into local functions in such a way that it is moved from the MFCS to the MHS elements. Hence, MHS elements with material flow control receive not only inputs from their neighboring MHS elements (through connection ports) but also additional inputs from other MHS elements (through signals). There, the actual material flow control is implemented as part of the FSM M^{E} of an MHS element (cf. Section IV-B). The split-up does not change the behavior of the modeled MHS but allows for encapsulating the material flow control in the MHS elements. This has advantages when considering compositional verification (cf. Section V).

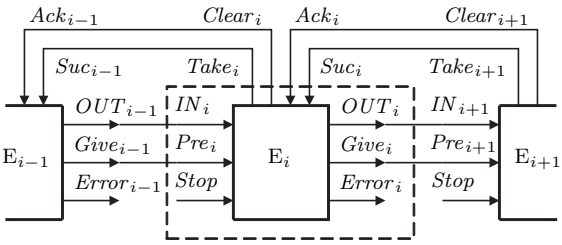

Figure 1. Example of interconnected MHS elements

Material flow control may either appear in MHS elements with *divert* function (more than one successor) or in MHS elements with *merge* function (more than one predecessor). While the former is called routing, the latter is often called intersection control.

It is typically the case in MHS that routing is based on a static shortest path computation, i.e. the shortest path to the destination is always selected. However, this can lead to crowding or even blocking of cargos in sections of the MHS, and thus, the overall throughput of the MHS would strongly suffer from this. For avoiding these blockings, a number of checks are performed in a certain order before the actual routing decision is made, for instance

1) velocity check: successive MHS elements are able to move, i.e. are not stopped,
2) content check: successive MHS elements are able to receive goods, i.e. provide empty places,
3) counter check: restricts the total number of cargos in a certain area, e.g. to avoid deadlocks in loops, and
4) others, e.g. to implement load balancing.

In the same manner also intersection control at MHS elements with merge function has been modeled. At an intersection, not the succeeding MHS element to route a cargo to is chosen, but the preceding element from which a cargo is transferred. Typical intersection control strategies comprise, among others, first-in-first-out (FIFO) and static prioritizing.

D. MHS Networks

After the notion of MHS elements has been introduced in Section IV-B, the interconnection of these elements needs to be defined in order to describe networks of MHS elements.

Definition 8. A *connection* CN is defined as the pair $\mathrm{CN} = (\mathrm{CP}^j_{out}, \mathrm{CP}^i_{in})$ and refers to linking and identifying the variables $OUT^j \mapsto IN^i$, $Give^j \mapsto Pre^i$, $Suc^j \mapsto Take^i$, and $Ack^j \mapsto Clear^i$ of the two connection ports.

Figure 1 shows a simple example of a series connection of three MHS elements. The connection ports are omitted in the figure to improve readability.

Using the notion of MHS elements and their connection, MHS networks are defined in the following.

Definition 9. An MHS network N is defined as the tuple $\mathrm{N} = (\mathcal{C}^{\mathrm{N}}, \mathcal{SRC}^{\mathrm{N}}, \mathcal{SNK}^{\mathrm{N}}, \mathcal{E}^{\mathrm{N}}, \mathcal{S}^{\mathrm{N}}_{in}, \mathcal{S}^{\mathrm{N}}_{out}, \mathcal{CN}^{\mathrm{N}}, \eta^{\mathrm{N}})$ such that

- $\mathcal{C}^{\mathrm{N}}$ is the set of cargo types available in the network,
- $\mathcal{SRC}^{\mathrm{N}}$ is a finite set of MHS sources,
- $\mathcal{SNK}^{\mathrm{N}}$ is a finite set of MHS sinks,
- $\mathcal{E}^{\mathrm{N}}$ is a finite set of concrete MHS elements,
- $\mathcal{S}^{\mathrm{N}}_{in}$ is a finite set of input signals of the network,
- $\mathcal{S}^{\mathrm{N}}_{out}$ is a finite set of output signals of the network,
- $\mathcal{CN}^{\mathrm{N}}$ is a finite set of connections, and
- η^{N} is a total function mapping signals, $\eta^{\mathrm{N}} : \bigcup_{\mathrm{E}} \mathcal{S}^{\mathrm{E}}_{in} \to \mathcal{S}^{\mathrm{N}}_{in} \cup \bigcup_{\mathrm{E}} \mathcal{S}^{\mathrm{E}}_{out}$,

where the relation provided through $\mathcal{CN}^{\mathrm{N}}$ is a bijective mapping, i.e. for each output connection port $\mathrm{CP}^j_{out} \in \mathcal{CP}^{\mathrm{E}}_{out}$ of an MHS element E or an MHS source SRC there exists exactly one input connection port $\mathrm{CP}^i_{in} \in \mathcal{CP}^{\mathrm{E}}_{in}$ of another MHS element E or a sink SNK such that $(\mathrm{CP}^j_{out}, \mathrm{CP}^i_{in}) \in \mathcal{CN}^{\mathrm{N}}$.

η^{N} connects and identifies output signals of MHS elements and input signals of the network to input signals of elements. As a consequence, the behavior of each MHS element is defined because all input signals of an element are input variables of its FSM M^{E}. Moreover, all output signals of MHS elements of an MHS subnetwork are viewed as output signals of the network, i.e. $\mathcal{S}^{\mathrm{N}}_{out} = \bigcup_{\mathrm{E}} \mathcal{S}^{\mathrm{E}}_{out}$.

Through input connection ports $\mathcal{S}^{\mathrm{N}}_{in}$ of the network, free inputs to MHS elements can be created, e.g. to model non-deterministic choices. With this, for instance, non-deterministic delays at working stations are modeled.

V. Compositional Verification Approach

For applying compositional verification, an MHS network is partitioned into subnetworks. The partitioning of networks is presented in Section V-A and Section V-B. Next, Section V-C introduces a compact description of partitioned MHS networks. Section V-D proposes the compositional approach for the verification of routing in MHS [51]. The algorithms are explained in Section V-E, and the verification results are discussed in Section V-F. Finally, the implementation of the algorithms is briefly summarized in Section V-G.

A. Partitioning of MHS Networks

First, the designer partitions the original MHS network into subnetworks. More specifically, an MHS network is partitioned into a number of MHS subnetworks by placing all concrete MHS elements into MHS subnetworks. MHS sources and sinks are considered to be always outside of subnetworks. Hence, each subnetwork has at least one input and at least one output connection port that allows for connecting the MHS subnetwork to other subnetworks and/or sources and sinks. Additionally, each subnetwork may have input and output signals.

Definition 10. An MHS subnetwork SN is defined as the tuple $\mathrm{SN} = (\mathcal{E}^{\mathrm{SN}}, \mathcal{CP}^{\mathrm{SN}}_{in}, \mathcal{CP}^{\mathrm{SN}}_{out}, \mathcal{S}^{\mathrm{SN}}_{in}, \mathcal{S}^{\mathrm{SN}}_{out}, \mathcal{CN}^{\mathrm{SN}}, \eta^{\mathrm{SN}})$ such that

- $\mathcal{E}^{\mathrm{SN}}$ is a finite set of concrete MHS elements,
- $\mathcal{CP}^{\mathrm{SN}}_{in}$ is the finite set of input connection ports of the subnetwork, $\mathcal{CP}^{\mathrm{SN}}_{in} = \{\mathrm{CP}^1_{in}, \mathrm{CP}^2_{in}, \ldots, \mathrm{CP}^k_{in}\}$ with $k = |\mathcal{CP}^{\mathrm{SN}}_{in}|$ $(k \in \mathbb{N}, k \geq 1)$,
- $\mathcal{CP}^{\mathrm{SN}}_{out}$ is the finite set of output connection ports of the subnetwork, $\mathcal{CP}^{\mathrm{SN}}_{out} = \{\mathrm{CP}^1_{out}, \mathrm{CP}^2_{out}, \ldots, \mathrm{CP}^m_{out}\}$ with $m = |\mathcal{CP}^{\mathrm{SN}}_{out}|$ $(m \in \mathbb{N}, m \geq 1)$,
- $\mathcal{S}^{\mathrm{SN}}_{in}$ is a finite set of input signals of the subnetwork,
- $\mathcal{S}^{\mathrm{SN}}_{out}$ is a finite set of output signals of the subnetwork,

- $\mathcal{CN}^{\mathrm{SN}}$ is a finite set of connections, and
- η^{SN} is a total function mapping signals, $\eta^{\mathrm{SN}} : \bigcup_{\mathrm{E}} \mathcal{S}^{\mathrm{E}}_{in} \to \mathcal{S}^{\mathrm{SN}}_{in} \cup \bigcup_{\mathrm{E}} \mathcal{S}^{\mathrm{E}}_{out}$,

where the relation provided through $\mathcal{CN}^{\mathrm{SN}}$ is an injective mapping, i.e. for each output connection port CP^{j}_{out} of a concrete MHS element E there exists exactly one or no input connection port CP^{i}_{in} of another MHS element such that $(\mathrm{CP}^{j}_{out}, \mathrm{CP}^{i}_{in}) \in \mathcal{CN}^{\mathrm{SN}}$. The input and output connection ports of the subnetwork correspond to all connection ports of MHS elements that are not connected to any other elements, i.e. all free connection ports of elements become connection ports of the subnetwork.

η^{SN} connects and identifies output signals of MHS elements and input signals of the subnetwork to input signals of elements. Furthermore, all output signals of MHS elements of an MHS subnetwork are viewed such that they are also output signals of the subnetwork, i.e. $\mathcal{S}^{\mathrm{SN}}_{out} = \bigcup_{\mathrm{E}} \mathcal{S}^{\mathrm{E}}_{out}$.

The definition of an MHS subnetwork excludes subnetworks without input or output connection ports. However, this is not a restriction to the generality of the presented verification method (cf. Section V-D). Such subnetworks do not influence the routing properties of the overall MHS network (except for the assignment of signals). If the partitioning results in such subnetworks, then these subnetworks should be analyzed first as they reveal structural deficiencies of the MHS network because they are not reachable for any cargo.

Considering MHS subnetworks, the notion of partitioned MHS networks, which are networks consisting of sources, sinks, and subnetworks, is introduced.

Definition 11. A partitioned MHS network $\mathrm{N}^{\star}$ is defined as the tuple $\mathrm{N}^{\star} = (\mathcal{C}^{\mathrm{N}^{\star}}, \mathcal{SRC}^{\mathrm{N}^{\star}}, \mathcal{SNK}^{\mathrm{N}^{\star}}, \mathcal{SN}^{\mathrm{N}^{\star}}, \mathcal{S}^{\mathrm{N}^{\star}}_{in}, \mathcal{S}^{\mathrm{N}^{\star}}_{out}, \mathcal{CN}^{\mathrm{N}^{\star}}, \eta^{\mathrm{N}^{\star}})$ such that

- $\mathcal{C}^{\mathrm{N}^{\star}}$ is the set of cargo types available in the network,
- $\mathcal{SRC}^{\mathrm{N}^{\star}}$ is a finite set of MHS sources,
- $\mathcal{SNK}^{\mathrm{N}^{\star}}$ is a finite set of MHS sinks,
- $\mathcal{SN}^{\mathrm{N}^{\star}}$ is a finite set of subnetworks,
- $\mathcal{S}^{\mathrm{N}^{\star}}_{in}$ is a finite set of input signals of the network,
- $\mathcal{S}^{\mathrm{N}^{\star}}_{out}$ is a finite set of output signals of the network,
- $\mathcal{CN}^{\mathrm{N}^{\star}}$ is a finite set of connections, and
- $\eta^{\mathrm{N}^{\star}}$ is a total function mapping signals, $\eta^{\mathrm{N}^{\star}} : \bigcup_{\mathrm{SN}} \mathcal{S}^{\mathrm{SN}}_{in} \to \mathcal{S}^{\mathrm{N}^{\star}}_{in} \cup \bigcup_{\mathrm{SN}} \mathcal{S}^{\mathrm{SN}}_{out}$,

where the relation provided through $\mathcal{CN}^{\mathrm{N}^{\star}}$ is a bijective mapping, i.e. for each output connection port $\mathrm{CP}^{j}_{out} \in \mathcal{CP}^{\mathrm{SN}}_{out}$ of a subnetwork SN or a source SRC there exists exactly one input connection port $\mathrm{CP}^{i}_{in} \in \mathcal{CP}^{\mathrm{SN}}_{in}$ of another subnetwork SN or a sink SNK such that $(\mathrm{CP}^{j}_{out}, \mathrm{CP}^{i}_{in}) \in \mathcal{CN}^{\mathrm{N}^{\star}}$.

$\eta^{\mathrm{N}^{\star}}$ connects and identifies output signals of MHS subnetworks and input signals of the network to input signals of subnetworks. Moreover, all output signals of MHS subnetworks, and hence, all output signals of MHS elements of an MHS network are viewed as output signals of the network as well, i.e. $\mathcal{S}^{\mathrm{N}^{\star}}_{out} = \bigcup_{\mathrm{SN}} \mathcal{S}^{\mathrm{SN}}_{out}$.

B. Finding of an Appropriate Partitioning

In the following, a simple procedure to find a suitable partitioning of an MHS network into subnetworks is provided.

MHS network	$\mathrm{N}^{\star}$	$:= \langle \mathcal{C}_{global}, \mathcal{SRC}, \mathcal{SNK}, \mathcal{SN}, \mathcal{S}_{in}, \mathcal{S}_{out}, \mathcal{CN}, \eta \rangle$
MHS subnetwork	SN	$:= \langle \mathcal{E}, \mathcal{CP}_{in}, \mathcal{CP}_{out}, \mathcal{S}_{in}, \mathcal{S}_{out}, \mathcal{CN}, \eta \rangle$
MHS element	E	$:= \langle \mathcal{CP}_{in}, \mathcal{CP}_{out}, \mathcal{S}_{in}, \mathcal{S}_{out}, M \rangle$
Source	SRC	$:= \langle \mathrm{CP}_{out}, \mathcal{S}_{in}, \mathcal{S}_{out}, \mathcal{C}_{out} \rangle$
Sink	SNK	$:= \langle \mathrm{CP}_{in}, \mathcal{S}_{in}, \mathcal{S}_{out}, \mathcal{C}_{spec} \rangle$
Connection port	CP	$:= \langle \mathcal{C}_{port} \rangle$
Connection	CN	$:= \langle \mathrm{CP}_{ogn}, \mathrm{CP}_{tgt} \rangle$

Figure 2. Compact description of the partitioned MHS network

1) *Partition* the MHS network into subnetworks according to the functionality of the MHS design. For instance, an MHS of a warehouse distribution center may consist of the parts high bay warehouse, order picking area, dispatch area, etc. The MHS elements that belong to one part are placed into the same subnetwork.
2) *Run compositional verification* approach to prove a property on the MHS network.
   a) *Computation succeeded*: Examine the verification results as described in Section V-F to figure out whether the given properties hold or not.
   b) *Computation timed out/ran out of memory*: *Refine the partitioning and rerun verification*. The compositional verification revealed the specific subnetwork which was too large to be handled. Split up this subnetwork according to the functionality of the MHS. For the example of the warehouse distribution center, for instance, the high bay warehouse may be subdivided into several separate aisles.

A similar procedure could also be automated in a software tool. However, the algorithm cannot automatically determine the functional parts of the MHS as the designer views them. Hence, the automated partitioning can only take the control flow dependencies between MHS elements into account to decide which elements are put in which subnetwork. Additionally, by manually choosing subnetworks the designer can decide at which positions (ingoing and outgoing connections of the subnetworks) in the MHS network the cargo types are proven, and through this, are observable for the designer.

C. Compact Description of Partitioned MHS Networks

This section introduces a compact description of partitioned MHS networks (cf. Figure 2). The description is the basis for the algorithms presented in Section V-E. Furthermore, a specification is added to the description, which is necessary to automatically compare the obtained verification results with the expected ones. The following notation is used: regular uppercase letters refer to single components and calligraphic uppercase letters to sets of components, e.g. $\mathcal{SN}$ is a set of components with the type SN. Moreover, a function to return a subcomponent is defined by the subcomponent name followed by round brackets, e.g. the expression $\mathcal{C}_{out}(\mathrm{SRC})$ represents a function to return the cargo types $\mathcal{C}_{out}$ of an MHS source SRC.

The partitioned network $\mathrm{N}^{\star}$ consists of sources $\mathcal{SRC}$, sinks $\mathcal{SNK}$, and subnetworks $\mathcal{SN}$. They are interconnected using connections $\mathcal{CN}$. Furthermore, signals are connected according to η (cf. Section V-A).

An MHS subnetwork is composed of MHS elements $\mathcal{E}$. Connections $\mathcal{CN}$ of a subnetwork link MHS elements to other elements by their input and output connection ports $\mathcal{CP}_{in}$ and $\mathcal{CP}_{out}$, respectively. Connection ports of a subnetwork allow for connections with sources, sinks, and other subnetworks. The partitioned MHS network is required to be closed, i.e. all existing connection ports in the network are connected to other connection ports. The set of available cargo types in the network is given by $\mathcal{C}_{global}$.

A concrete MHS element E is modeled by an FSM M and interconnected through connection ports and signals. An MHS source has one output connection port CP_{out} and a set of cargo types $\mathcal{C}_{out}$ it can generate.

In order to verify routing, information about the cargo types in the MHS network is needed. Therefore, a sink refers to the set $\mathcal{C}_{spec}$ defining the cargo types that are required to reach this sink. This set is derived from the specification of the MHS. Additionally, the set $\mathcal{C}_{port}$ characterizes the cargo types at the connection ports that are determined during verification (cf. Section V-D).

A connection relates an origin (output) connection port CP_{ogn} to a target (input) one CP_{tgt}. Through a connection port, the single input or output variables of an MHS element, such as $Give$, can be accessed.

Finally, the function $\mathcal{CN}(\mathrm{CP})$ returns for a given input connection port the corresponding output connection port, and vice versa, as defined through the connections $\mathcal{CN}$. Here, it is not differentiated whether the corresponding connection links subnetworks or elements.

D. Verification Method

The verification goal is to prove the correctness of the routing in the given MHS network, i.e. whether the set of determined cargo types $\mathcal{C}_{port}$ at a sink is the same as its specified cargo types $\mathcal{C}_{spec}$.

To apply compositional verification, the MHS network is partitioned into subnetworks as described in Section V-A. The goal of the partitioning is to achieve a partitioned MHS network without feedback loops between subnetworks. However, if this is not possible, feedback loops have to be resolved as discussed later in this section.

The method to formally verify routing works as follows: the partitioned MHS network $\mathrm{N}^\star$ without feedback loops between MHS subnetworks spans a directed acyclic graph where the nodes are the subnetworks, sources, and sinks, and the direction is given by the flow of the cargos, i.e. from sources $\mathcal{SRC}$ through subnetworks $\mathcal{SN}$ to sinks $\mathcal{SNK}$. At each of the sources, the possible cargo types being generated $\mathcal{C}_{out}(\mathrm{SRC})$ are defined. Thus, from the given cargo types at the sources, without any further assumptions, the possible cargo types $\mathcal{CP}_{out}(\mathrm{SN})$ at the output connection ports of the subnetworks directly connected to these sources can formally be verified. This first step refers to $\langle true\rangle\, K_n\, \langle\varphi_{n-1}\rangle$ in AGR-N (cf. Section III-C). Based on the verified assumptions, i.e. the proven cargo types at the input connection ports, the possible cargo types at the output connection ports of subnetworks directly connected to the subnetworks from the first step can now be verified ($\langle\varphi_{n-1}\rangle\, K_{n-1}\, \langle\varphi_{n-2}\rangle$). This procedure is repeated for the remaining subnetworks. Finally, whenever these steps lead to the verification of the output connection ports of subnetworks directly connected to sinks, the algorithm terminates, as the properties of interest (possible cargo types at sinks) have been proven ($\langle\varphi_1\rangle\, K_1\, \langle\psi\rangle$).

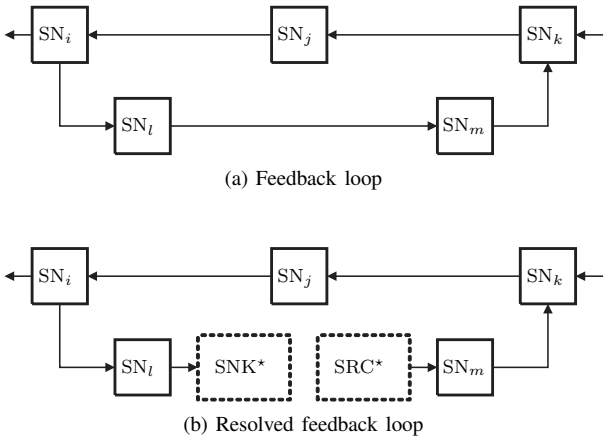

Figure 3. Resolving of a feedback loop between two MHS subnetworks in a partitioned MHS network

The number of reachable states that needs to be traversed during verification grows exponentially with the size of the system (cf. Section III-B), i.e. for MHS networks, it grows with the number of included MHS elements, the count of different cargo types, etc. Hence, the number of the reachable states of each subnetwork is much smaller than the state space of the whole MHS network. Consequently, the compositional approach may succeed in cases where the whole MHS network cannot be verified at once because of the state space explosion. This is illustrated in the case study in Section VI-B.

For MHS networks with feedback loops between MHS subnetworks, a preprocessing is performed. While feedback loops *inside* MHS subnetworks stay untouched, feedback loops *connecting* MHS subnetworks are resolved: One arbitrary connection of one feedback loop is replaced by a source $\mathrm{SRC}^\star$ creating the cargo types $\mathcal{C}_{out}$ and an appropriate sink $\mathrm{SNK}^\star$ consuming them ($\mathcal{C}_{spec}(\mathrm{SNK}^\star) = \mathcal{C}_{out}(\mathrm{SRC}^\star)$) (cf. Figure 3). If the partitioned MHS network still contains feedback loops after this resolution, this procedure will be repeated until all loops have been resolved.

For the verification of routing, *only* the cargo types $\mathcal{C}_{out}$ generated by $\mathrm{SRC}^\star$ are of interest. All outgoing control flow variables of $\mathrm{SRC}^\star$ and $\mathrm{SNK}^\star$ may take arbitrary values. $\mathrm{SRC}^\star$ and $\mathrm{SNK}^\star$ are only needed to receive a closed MHS network, but they do not restrict the behavior of the rest of the network in any terms, with the exception of the provided cargo types $\mathcal{C}_{out}$. Thus, only the cargo types $\mathcal{C}_{out}$ have to be computed.

The computation of the corresponding cargo types $\mathcal{C}_{out}$ is done fully automatic as discussed in Section V-E. The proof that the altered MHS network is a valid abstraction of the original network is sketched in the Appendix.

E. Description of the Algorithms

The algorithm to verify routing in partitioned MHS networks is shown in Algorithm 1. In lines 1–4, the possible

input : Partitioned MHS network $\mathrm{N}^\star$ consisting of subnetworks $\mathcal{SN}$
output: Cargo types at all sinks $\mathcal{SNK}$ of $\mathrm{N}^\star$

```
// Initialize
1  foreach CN ∈ 𝒞𝒩(N*) do
2      𝒞_port(CP_ogn(CN)) := ∅
3      𝒞_port(CP_tgt(CN)) := ∅
4  end
// Propagate cargo types from sources
5  foreach SRC ∈ 𝒮ℛ𝒞(N*) do
6      𝒞_port(𝒞𝒩(CP_out(SRC))) := 𝒞_out(SRC)
7  end
8  𝒮𝒩_verify := 𝒮𝒩(N*)
// Main loop
9  while true do
       // Analyze subnetworks
10     foreach SN ∈ 𝒮𝒩_verify do
11         if ∀CP ∈ 𝒞𝒫_in(SN). 𝒞_port ≠ ∅ then
               // Compute cargo types of current
                  subnetwork
12             ComputeCargoTypes(N*, SN)
13             𝒮𝒩_verify := 𝒮𝒩_verify \ SN
14         end
15     end
       // Cargo types at all sinks determined?
16     if ∀SNK ∈ 𝒮𝒩𝒦(N*). 𝒞_port(CP_in(SNK)) ≠ ∅ then
17         foreach SNK ∈ 𝒮𝒩𝒦(N*) do
18             if 𝒞_spec(SNK) = 𝒞_port(CP_in(SNK)) then
                   // No routing error found
19             else if 𝒞_spec(SNK) ⊂ 𝒞_port(CP_in(SNK)) then
                   // Routing error candidate
20             else
                   // Routing error found
21             end
22         end
23         return
24     end
25 end
```

Algorithm 1: Verify routing in partitioned MHS network

cargo types at each connection port are initialized to the empty set. Next, the available cargo types at the sources are propagated through connections to the connection ports of the MHS subnetworks directly connected to these sources (lines 5–7).

Afterwards, the set $\mathcal{SN}_{verify}$ of subnetworks to be analyzed is initialized to all subnetworks of the MHS network (line 8). Each subnetwork is then handled as follows (lines 10–15): If all possible cargo types at input connection ports of a subnetwork have already been determined (line 11), then the routing in this subnetwork SN will be checked by calling `ComputeCargoTypes` (line 12, cf. Algorithm 2 and explanation below). After this, the currently handled MHS subnetwork is removed from the set $\mathcal{SN}_{verify}$ (line 13).

Whenever all possible cargo types at all sinks have been determined (line 16), the verified routing results are compared to the specification (lines 17–22). Since the special cargo type L is never blocked, eventually, L is in each $\mathcal{C}_{port}(\mathrm{CP}_{out})$, i.e. $\mathcal{C}_{port}(\mathrm{CP}_{out}) \neq \emptyset$. Thus, the algorithm terminates. The possible verification results are discussed in Section V-F.

input : Partitioned MHS network $\mathrm{N}^\star$ and subnetwork SN
output: Possible cargo types at all output connection ports CP of subnetwork SN

```
1  foreach CP_out ∈ 𝒞𝒫_out(SN) do
2      𝒞_block := ∅
3      foreach C ∈ 𝒞_global(N*) do
4          if BlockType(CP_out, C) then
5              𝒞_block := 𝒞_block ∪ C
6          end
7      end
8      𝒞_port(CP_out) := 𝒞_global(N*) \ 𝒞_block
       // Propagate cargo types
9      𝒞_port(𝒞𝒩(CP_out)) := 𝒞_port(CP_out)
10 end
```

Algorithm 2: `ComputeCargoTypes`: Computes cargo types at outgoing connections ports of a subnetwork

If the network contained feedback loops that were resolved as described in Section V-D, Algorithm 1 runs several times in a modified manner to compute the cargo types of the introduced sinks $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star))$. These cargo types are the same as the cargo types generated by the corresponding sources $\mathrm{SRC}^\star$. In the first run of the algorithm ($i = 0$), it is assumed that $\mathcal{C}_{out}(\mathrm{SRC}^\star) := \{L\}$. In lines 10–15, new values for $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star))$ are computed. In contrast to Algorithm 1, additional computations are carried out between line 17 and 18: If $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star)) \neq \mathcal{C}_{port}(\mathrm{CP}_{out}(\mathrm{SRC}^\star))$ for at least one sink, then $\mathcal{C}_{out}(\mathrm{SRC}^\star) := \mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star))$. In this case, the checks in lines 18–22 are not performed but the algorithm is started again with iteration $(i+1)$. If $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star)) = \mathcal{C}_{out}(\mathrm{SRC}^\star)$, then all cargo types on feedback loops have been determined and the final checks (lines 17–22) are executed.

During one iteration of Algorithm 1, the set $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star))$ of each sink $\mathrm{SNK}^\star$ is either increased or stays unchanged. Since $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star)) \subseteq \mathcal{C}_{global}$ and since $\mathcal{C}_{global}$ is a finite set, the $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}^\star))$ cannot be increased infinitely often. Hence, the algorithm terminates after a finite number of iterations.

The function `ComputeCargoTypes` (cf. Algorithm 2) computes all the cargo types $\mathcal{C}_{port}$ that can reach the output connection ports CP_{out} of the given subnetwork. Therefore the assumptions about the incoming connection ports of the subnetwork, which have been proven in previous steps, are used as environment of the subnetwork.

Each cargo type is processed separately (lines 3–7): The function `BlockType`$(\mathrm{CP}_{out}, \mathrm{C})$ (line 4) calls the model checker to prove that the given cargo type C cannot be transferred at the output connection port CP_{out}. This is realized by trying to prove the ACTL formula (cf. Section III-B)

$$\mathbf{AG}\left(\neg\left(\mathrm{CP}_{out}.OUT = \mathrm{C}\right)\right) \tag{1}$$

stating that there is no state where C reaches CP_{out}. If the outcome is true, the given cargo type C is added to the set of blocked cargo types $\mathcal{C}_{block}$ of this connection port (line 5).

After all cargo types have been examined, the set of cargo types $\mathcal{C}_{port}$ that can reach CP_{out} is determined (line 8). This

set contains all the cargo types that are not blocked at CP_{out}. From this follows the correctness of the ACTL formula

$$\mathbf{AG}\left(\bigvee_{\forall c \in \mathcal{C}_{port}(\mathrm{CP}_{out})} (\mathrm{CP}_{out}.OUT = c)\right) \quad (2)$$

which specifies the cargo types at CP_{out}. This formula is used as an assumption ("source") for the succeeding subnetworks directly connected to this subnetwork.

Finally, this result is propagated from the current connection port CP_{out} to the succeeding subnetwork (line 9).

F. Verification Results

For each sink SNK, the verified set of cargo types $\mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}))$ is compared to the given set of cargo types from the specification $\mathcal{C}_{spec}(\mathrm{SNK})$, and it results in:

- $\mathcal{C}_{spec}(\mathrm{SNK}) = \mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}))$: no routing errors found as both sets are the same.
- $\mathcal{C}_{spec}(\mathrm{SNK}) \subset \mathcal{C}_{port}(\mathrm{CP}_{in}(\mathrm{SNK}))$: routing error candidate found as there is at least one not expected cargo type reaching this sink.
- else: routing error found because at least one of the required cargo types cannot reach this sink.

The reason for the second case can be either a real routing error or a false negative caused by too weak assumptions (cf. Section III-C). Hence, the designer has to review the intermediate results $\mathcal{C}_{port}$ by further analysis.

False negatives may only occur if the routing strategy in a subnetwork depends on MHS elements outside of this subnetwork or if the sequence of cargos provided by the MHS source $\mathrm{SRC}^{\star}$ is a too coarse abstraction of the environment of the subnetwork (cf. Section V-D). Thus, if possible, all MHS elements affecting the routing in a subnetwork should be in this subnetwork.

From a practical perspective, routing based on expected sequences of cargos of a certain type is not feasible. Secondly, false negatives may only appear if a routing strategy for a cargo type at an MHS element has been implemented but is never chosen, i.e. the conditions to choose this specific strategy are never fulfilled. Again, this kind of implementation does not make sense in practice.

As a technical extension of the approach, a specification $\mathcal{C}_{port}$ could also be added to each connection CN, to also verify the cargo types at connections in the MHS network.

G. Implementation

The proposed approach has been automated and implemented in a software tool called *MHSVer* (cf. Figure 4). *MHSVer* utilizes the symbolic model checker NuSMV [52] as a backend for proving properties.

From a description of the partitioned MHS network, *MHSVer* step-wise executes the presented verification algorithms (cf. Section V-E). For each subnetwork where the incoming cargo types have already been determined, the corresponding SMV code is generated using the MHS library. This library contains the behavioral models of all existing MHS elements. When generating SMV code, the already proven cargo types have to be considered as environment of the subnetwork under verification. The generated SMV code and the appropriate properties for proving the routing are passed to the NuSMV model checker. The results from NuSMV are subsequently parsed, and the computed cargo types are extracted and stored in the MHS network. After all subnetworks have been processed, the verification results are presented to the user.

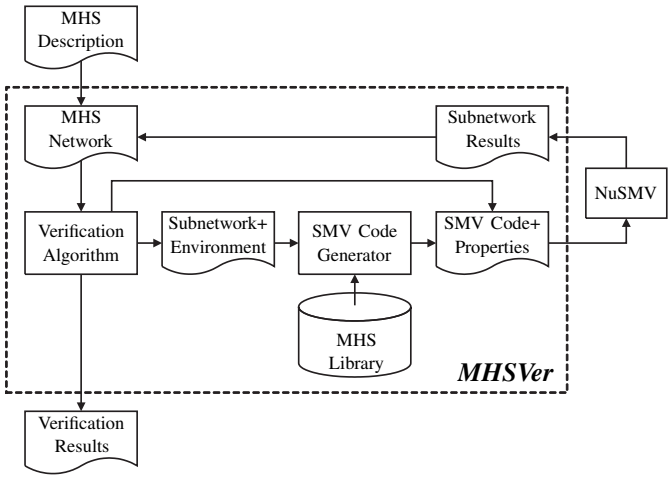

Figure 4. MHSVer verification flow

VI. Evaluation

The proposed modeling methodology (cf. Section IV) allows to create models of MHS by interconnecting MHS elements to a network. This section explains how formal verification in terms of model checking can be applied to these MHS networks with an illustrative example in Section VI-A and with a real-world case study in Section VI-B.

A. Illustrative Example

As an example, Figure 5 depicts the layout of a structure that often occurs in technical MHS, especially in warehouse distribution centers. The MHS has been modeled using the following elements:

- src: MHS source generating cargos of the types $\{p, L\}$
- $c1$–$c18$: early clear conveyors with length $l = 1$, except for $l_{5,13} = 2$ and $l_{8,18} = 3$
- $mg1$, $mg2$: turntable with merge function
- $tt1$–$tt6$: turntables
- $dv1$, $dv2$: turntables with divert function
- ws: working station, has a non-deterministic delay between 3–5 time steps
- snk: MHS sink

The example consists of a high bay warehouse (HBW), an order picking area, and a conveying system connecting the former two. The global set of cargo types is $\mathcal{C} = \{p, s, L\}$. The considered HBW is equipped with one aisle and has been modeled using a source, a sink, and a number of conveyors. Cargos from the HBW (type p) are transported to the working station ws in the order picking area.

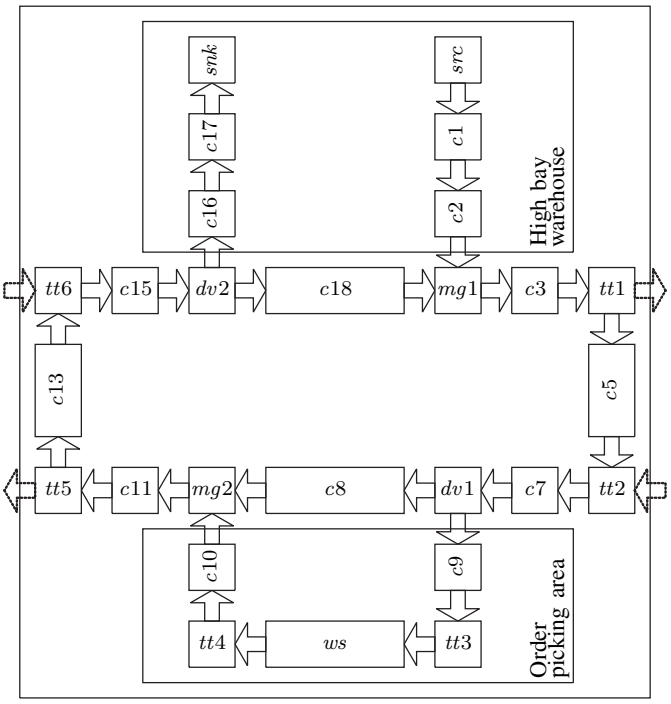

Figure 5. Example of an MHS network

The working station models the order picking. Incoming cargos of type p are handled, and subsequently, their type is changed to s, i.e. the cargos are marked as processed. All cargos of type s are transported back to the HBW. In case $c9$ ($c16$) is ready to receive a cargo, all incoming cargos of type p (s) are routed at $dv1$ ($dv2$) to ws (snk), respectively. All the cargos that could not immediately be transported to one of these two targets move on in the loop. There, the controls of the turntables with merge function, $mg1$ and $mg2$, are implemented in such a way that the cargos in the loop are preferred over the merging ones.

Based on the model of the MHS, the correctness of a number of properties can be proven using a model checker. As an example, the correct routing in the MHS network is analyzed (here exemplarily shown for the target snk):

$$\mathbf{AG}\ \ (snk.OUT = s\ \vee\ snk.OUT = L) \tag{3}$$

This formula states that at any time at snk there can either be a cargo of type s or no cargo (L), and consequently, no cargos of type p will ever reach snk.

The corresponding Kripke structure of the MHS network modeling the example has 11.1 million states and has a diameter (cf. Section III-A) of 86. The overall verification time for proving all the above property is 47 s (Intel Xeon® X5672@3.2 GHz, 4GB RAM, Linux, NuSMV 2.5.3 64bit).

So far, the example only consisted of an MHS network with a simple conveying system spanning one loop. However, a typical warehouse distribution center consists of an HBW with multiple aisles on the one hand side, and includes an order picking area with multiple working stations on the other hand side. Therefore, also the involved conveying system needs to be adapted.

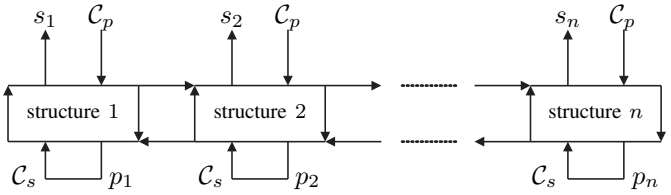

Figure 6. Extended example of an MHS network

More specifically, the basic structure sketched in Figure 5 occurs not just once but multiple times (cf. Figure 6). For interconnecting multiple copies of this structure, the turntables $tt2$ and $tt6$ are exchanged against turntables with merge function, while turntables $tt1$ and $tt5$ are exchanged against ones with divert function. Moreover, four additional conveyors, namely $c4$, $c6$, $c12$, and $c14$, are added to the example and connected to $tt2$, $tt6$, $tt1$, and $tt5$, respectively. Through these conveyors, neighboring structures are interconnected to receive a closed MHS network.

In order to model the involved routing functionality, further cargo types have to be added. Hence, for n structures, the global set of cargo types is given by $\mathcal{C} = \{s_1, s_2, \ldots, s_n, p_1, p_2, \ldots, p_n, L\}$. Each source modeling an aisle of the HBW generates arbitrary cargos of the types $\mathcal{C}_p = \{p_1, p_2, \ldots, p_n\}$ where the type represents the working station they are transported to. In the ith structure ($1 \leq i \leq n$), the HBW consumes cargos of type s_i; the working station processes cargos of type p_i and changes their types to arbitrary elements of $\mathcal{C}_s = \{s_1, s_2, \ldots, s_n\}$. The routing in the turntables with divert function, $dv1$ and $dv2$, is adjusted accordingly.

For the structure 1, i.e. on the very left side, $tt5$ and $tt6$ remain turntables without routing and conveyors $c12$ and $c14$ are removed to receive a closed MHS network. The same applies to the structure n, i.e. on the very right side, with respect to turntables $tt1$ and $tt2$ and conveyors $c4$ and $c6$.

Furthermore, the routing regarding neighboring structures has to be implemented. For incoming cargos from neighboring structures, the control of the newly added turntables with merge function $tt2$ and $tt6$ is implemented such that cargos from $c6$ and $c14$ are preferred over the ones from $c5$ and $c13$, respectively. For outgoing cargos, the routing at the turntables with divert function $tt1$ and $tt5$ needs to be defined. For this purpose, the sets $keep_l$ and $keep_r$ are introduced. For structure i, these sets are given as $keep_l = \{p_1, p_2, \ldots, p_i, s_1, s_2, \ldots, s_i\}$ and $keep_r = \{p_i, p_{i+1}, \ldots, p_n, s_i, s_{i+1}, \ldots, s_n\}$. If a cargo at $tt1$ ($tt5$) is of a type that is in $keep_l$ ($keep_r$) and if conveyor $c5$ ($c13$) is able to receive a cargo, then this cargo is transported to this conveyor. If $c5$ ($c13$) cannot receive a cargo at this point or if the cargo is of a type that is not an element of $keep_l$ ($keep_r$), then the cargo is transferred to $c4$ ($c12$). If non of the conveyors is able to receive a cargo, the cargo waits at the turntable with divert function.

By increasing the number of structures and, with this, the number of involved cargo types as discussed above, also the state space and the diameter of the corresponding Kripke structure grow. While for the example consisting of only one

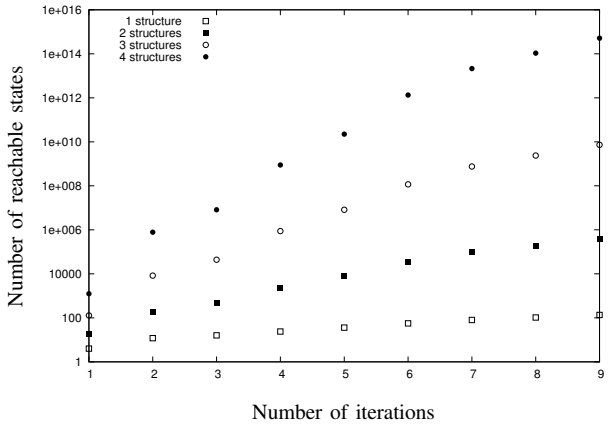

Figure 7. Reachable states for MHS network example consisting of 1, 2, 3, and 4 structures

structure the properties have been proven in 47 s, the proof of the example including two structures did not succeed in 12 h.

The considered model checker NuSMV [52] allows to compute the number of reachable states of a Kripke structure. This computation is done by traversing the reachable states using several iterations, beginning at the initial states. There, the first iteration corresponds to the number of initial states in the model; the second iterations refers to all states that are reachable in one transition from the initial states, etc.

Figure 7 depicts a plot showing the number of reachable states in relation to the number of iterations for the example consisting of 1, 2, 3, and 4 structures. The plot indicates that the state space of the underlying Kripke structures grows significantly with each added structure. While for one structure only 136 states are reachable in 9 iterations, for four structures this value increases to about 514 trillions. The number of initial states (1250) when involving four structures is already larger than the number of reachable states in 9 iterations for one structure. Consequently, the model checking of a complete model consisting of multiple structures becomes unfeasible.

Thus, to verify models of real-world MHS using the proposed modeling methodology (cf. Section IV), further means such as compositional verification (cf. Section V) are necessary. To apply the compositional approach, the MHS network needs to be partitioned into subnetworks (cf. Section V-A), e.g. as sketched in Figure 8.

The chosen partitioning contains multiple loops between subnetworks, which have to be removed as described in Section V-D. Hence, at each cut_i, an appropriate $\mathrm{SRC}_i^\star$ and $\mathrm{SNK}_i^\star$ are inserted into the partitioned MHS network and the corresponding cargo types $\mathcal{C}_{port}(\mathrm{SNK}^\star)$ are automatically computed by the proposed algorithms. After two iterations, the cargo types at the cuts have been determined as follows

- $\mathcal{C}_{port}(\mathrm{SRC}_1^\star) = \{L, p_1, p_2, p_3, p_4, s_1, s_2, s_3, s_4\}$
- $\mathcal{C}_{port}(\mathrm{SRC}_2^\star) = \{L, p_2, p_3, p_4, s_2, s_3, s_4\}$
- $\mathcal{C}_{port}(\mathrm{SRC}_3^\star) = \{L, p_3, p_4, s_3, s_4\}$
- $\mathcal{C}_{port}(\mathrm{SRC}_4^\star) = \{L, p_4, s_4\}$

This corresponds to the implemented routing strategy explained before, as for instance at cut_4 only cargos of the types p_4 and s_4 are allowed to take the shortcut to subnetwork SN_7 instead of being transferred to SN_6.

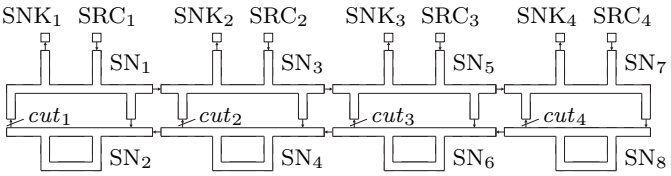

Figure 8. Illustrative example partitioned into eight subnetworks

Finally, the compositional verification of routing determines for each MHS sink SNK_i the set of cargo types as $\mathcal{C}_{port}(\mathrm{SNK}_i) = \{L, s_i\}$, i.e. the implementation is proven to be correct. The complete verification, including the resolving of the loops, took 1021 s. In summary, the illustrative example consisting of multiple structures could not be verified at once, i.e. without partitioning, but the verification succeeded by applying the proposed compositional approach.

B. BHS of an International Airport

A typical example of MHS are BHS at international airports transporting and delivering up to several thousand pieces of luggage per hour. Figure 9 shows the layout of the BHS of an existing international airport, which is discussed in the sequel.

From the 48 check-in lines Ⓐ, Ⓑ and the transfer gate Ⓕ, domestic and international bags enter the BHS. In this BHS, only international bags have to be screened, and thus, are considered uncleared. This screening is called checked-baggage-screening (CBS) and is usually carried out using X-ray machines to detect threats. The bags from the check-in block Ⓐ are identified by the automatic tag reader (ATR) Ⓒ, and then transported to the lower main line Ⓔ of the BHS, whereas bags from check-in blocks Ⓑ, and transfer bags Ⓕ are scanned at the ATR Ⓓ, and thereafter fed into the upper main line Ⓖ. In case a tag could not be identified by the ATR, these bags are routed to the manual encoding station Ⓗ, and subsequently fed back into the upper main line of the BHS.

Owing to the fact that domestic bags bypass CBS Ⓛ Ⓜ, they are routed to the lower main line by taking link Ⓘ; if this link is too populated, these bags have to take the link Ⓚ. With this, domestic bags take the outer loop and are routed to the inner make-up carousels Ⓝ. In contrast, uncleared international bags are routed to one of the two CBS stations Ⓛ Ⓜ by using the two before mentioned links. If the access to the inner CBS station Ⓛ is too populated, incoming bags are assigned to the outer CBS station Ⓜ. However, if a bag cannot be cleared automatically, this bag will be transferred to the screening station Ⓟ for manual inspection. All cleared international bags are routed to the outer two make-up carousels Ⓞ. From the make-up carousels, the bags are transported to their departing aircraft.

The BHS of this airport has been modeled considering the presented modeling methodology (cf. Section IV) and the existing MHS elements (e.g. [4]). The resulting overall MHS network consists of 358 MHS elements, including 49 MHS sources, 233 conveyors, 55 conveyors with merge function, 11 pusher, 5 working stations, and 5 MHS sinks.

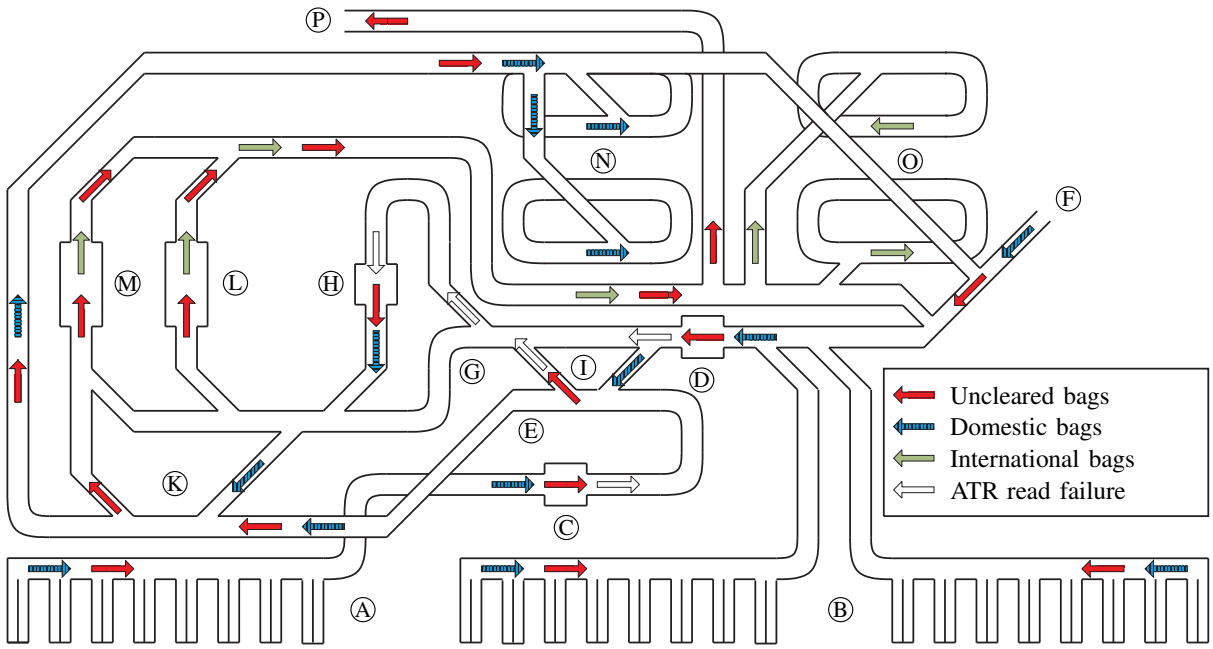

Figure 9. Layout of the BHS of an international airport

The MHS sources model the 48 check-in lines Ⓐ, Ⓑ and the transfer gate Ⓕ. Most (45) of the conveyors with merge function appear at the check-in lines, the remaining ones appear in the conveying system, e.g. at Ⓘ and Ⓚ. The pushers are needed wherever routing functionality has been implemented, i.e. at all diverts. The working stations have been introduced to model ATR Ⓒ Ⓓ, manual encoding Ⓗ, and CBS Ⓛ Ⓜ. Last, the sinks model the fact that cargos leave the conveying system at the carousels Ⓝ, Ⓞ, and at the screening station Ⓟ.

To model the different types of cargos and to implement appropriate routing strategies, four different types of cargos (bags) have been introduced (cf. Figure 9), which are named by colors as follows

- Uncleared bags: r(ed)
- Domestic bags: b(lue)
- International bags: g(reen)
- Read failure: w(hite)

The whole MHS network could not be verified without partitioning due to state space explosion. On the other hand, with the compositional approach presented in Section V, a number of properties with regard to routing functionality have been proven automatically, in particular

- no uncleared international or unidentified bags are delivered to a plane
- only domestic bags arrive at the inner make-up carousels
- only international bags reach the outer make-up carousels
- only uncleared bags arrive at the CBS stations

For the verification, the MHS network has been split up into 13 subnetworks using the procedure provided in Section V-B. There, the functionality of the BHS is taken into account, i.e. all check-in lines Ⓐ, Ⓑ are placed into a subnetwork, the outer two make-up carousels Ⓞ span a subnetwork, etc. Also, the properties to be proven are considered because the cargo types are always computed, and thus observable, at the outgoing connection ports of each subnetwork. Hence, for instance, the CBS stations Ⓛ Ⓜ and their corresponding conveyors are put into the same subnetwork.

The only feedback loop that had to be resolved is the outer conveying loop which was cut between Ⓕ and Ⓓ. Based on this partitioning, the overall verification of routing took 251.3 s plus 235.3 s needed for removing the loop and determining the cargo types at the cut, i.e. 486.6 s in total (Intel Xeon® X5672@3.2 GHz, 4GB RAM, Linux, NuSMV 2.5.3 64bit).

In the initial iteration to determine the cargo types at the loop cut (cf. Section V-E), $\mathcal{C}_{out}(\mathrm{SRC}^\star) := \{L\}$. Based on this, the cargo types reaching the cut are computed as $\mathcal{C}_{port}(\mathrm{SNK}^\star) = \{r, b, g, w, L\}$, i.e. cargos of all cargo types may reach the cut in the airport example. In the next and last iteration, the cargo types at the source $\mathrm{SRC}^\star$ are updated to $\mathcal{C}_{out}(\mathrm{SRC}^\star) := \{r, b, g, w, L\}$; the set $\mathcal{C}_{port}(\mathrm{SNK}^\star)$ does not change anymore in this iteration, because $\mathcal{C}_{port}(\mathrm{SNK}^\star) = \mathcal{C}_{global}$, i.e. no further cargo types can be added. Consequently, after two iterations the cargos types at the cut have been determined. In the second iteration, also the verification of routing is carried out, i.e. the cargo types reaching all sinks in the MHS network are computed.

To further discuss the impact of the count of subnetworks on the overall verification time, the compositional verification for the airport example has been conducted considering different partitionings. Figure 10 depicts the sum of the number of reachable states of all subnetworks in relation to the number of subnetworks for airport example. The number of reachable states is plotted in logarithmic scale.

The plot shows that the initial partitioning (13 subnetworks) is not well suited for formal verification as the number of reachable states is huge ($1.30 \cdot 10^{93}$). This is mainly due to the fact that in the initial partitioning all the check-in

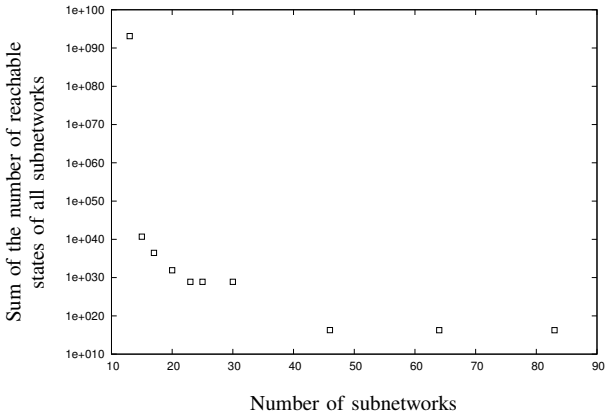

Figure 10. Reachable states in relation to the number of subnetworks

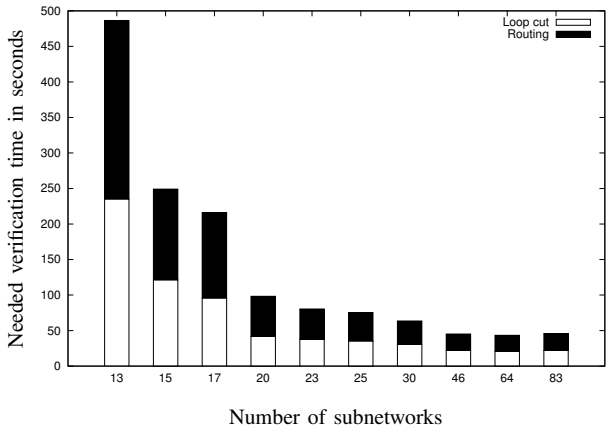

Figure 11. Overall time needed to verify routing in relation to the number of subnetworks

lines (96 MHS elements, 48 MHS sources) were put into one subnetwork, which by itself already provides $6.48 \cdot 10^{92}$, i.e. almost 50.0% of the overall reachable states. As a consequence, 245.1 s (50.4%) of the complete verification time were spent to prove outgoing cargo types on this subnetwork.

Thus, by subdividing this subnetwork into three subnetworks, which leads to a count of 15 subnetworks, the sum of reachable states of all subnetworks drops to $5.07 \cdot 10^{40}$ (cf. Figure 10). By dividing the MHS network into even more subnetworks, the overall state count further decreases. For the airport example, a partitioning in 83 subnetworks reduced the sum of the reachable states of all subnetworks to $1.83 \cdot 10^{16}$. The partitioning for the shown number of subnetworks has been chosen such that no additional loops between subnetworks were introduced.

With the smaller number of states that have to be analyzed during model checking, also the needed verification time decreases. The plot in Figure 11 presents the time needed to verify routing in relation to the chosen partitioning.

The overall time is segmented into the time needed to compute the cargo types at the loop cut and the time needed verify routing. More precisely, the latter is the amount of time for performing the last iteration of the verification algorithm (cf. Section V-E) during which the cargo types at the loop cuts do not change anymore and during which the cargo types at the MHS sinks are computed. This last iteration is shown explicitly, as it represents the actual verification time if the MHS network would not contain any loops between subnetworks. This last iteration takes slightly more time than the first iteration (cf. Figure 11), as more subnetworks have to be considered to reach not only the introduced sinks $\mathcal{SNK}^\star$ but the actual sinks $\mathcal{SNK}$ of the MHS network.

For the initial partitioning, the overall verification took 486.6 s. The fewer MHS elements are part of a subnetwork, the smaller the number of reachable states but also the smaller the diameter (cf. Section III-A) of the underlying Kripke structure. Hence, by further refining the partitioning, the verification time has been reduced to 43.3 s. Interestingly, the minimum verification time has not been not achieved when choosing the maximum number of considered subnetworks (83), but when considering 64 subnetworks. The verification time rises from 43.3 s for 64 subnetworks to 45.9 s for 83 subnetworks owing to the additional time that is needed for generating the NuSMV code and for verifying a higher number of properties on a higher number of subnetworks.

This suggests, at first sight, to split up the MHS network into a large number of subnetworks to improve the general performance. However, it has to be taken into account that false negatives (cf. Section V-F) may occur. Moreover, by introducing a large number of small subnetworks, additional cycles between subnetworks may be introduced, which have to be resolved during verification.

This case study shows that with the discussed approach, important properties of the BHS can be proven automatically in a reasonable amount of time. Another case study can be found in [53].

VII. Conclusion

This paper has proposed an approach to automatically prove the correctness of routing in MHS using compositional verification. The applicability of the approach has been shown using a real-world example. Experimental results are promising as the approach scales well with the size of the MHS network. Furthermore, the approach has advantages over simple routing graph analysis because complex routing strategies are also taken into account. Moreover, the presented approach is the basis for proving not just routing but arbitrary properties of MHS in a compositional way [53]. The general idea of the approach is not restricted to MHS but may also be applied to other applications.

Current work focuses on automatic partitioning of MHS networks into subnetworks, where the size of the subnetworks is adjusted dynamically with regard to material flow control, and the time and/or memory needed by the model checker. The overall performance of the tool can be improved by topologically sorting the subnetworks, and then, allowing for parallel proofs of subnetworks.

A major challenge for future research is the tackling of false negatives which refer to counterexamples that are reachable when verifying a subnetwork but that are not reachable when considering the complete MHS network. These false negatives have to be automatically identified and weeded out.

Another direction for the future is the design of specification templates for MHS. Expressing certain requirements in temporal logics is non-trivial and error-prone, especially for non-experts. Since some requirements to MHS occur often in practice, they can be specified in terms of templates. From these templates, temporal logic formulas are automatically generated and proven.

APPENDIX

The proof for the correctness of the proposed algorithm (cf. Section V-D) is provided in the following.

Proof: Two cases have to be addressed regarding whether the partitioned MHS network contains feedback loops between subnetworks or not. In the following, may N be the original MHS network which also stands for all traces of N which are the same traces as allowed by the partitioned version $\mathrm{N}^{\star}$. The traces assumed by the compositional verification Algorithm 1 are denoted by $Alg(\mathrm{N})$.

If the partitioned MHS network does not contain feedback loops, *only* the set of cargo types $\mathcal{C}_{port}(\mathrm{CP}_{in})$ at an input connection port of a subnetwork is assumed. All other control flow variables ($Give$ etc.) and all input signals of the subnetwork may take arbitrary values. Thus, $Alg(\mathrm{N})$ is a valid abstraction of N ($\mathrm{N} \preceq Alg(\mathrm{N})$, cf. Section III-C), i.e. the partitioning does not exclude possible traces of N. Consequently, the cargo types $\mathcal{C}_{port}(\mathrm{CP}_{out})$ reaching an output connection port of a subnetwork in $Alg(\mathrm{N})$ are an over-approximation of the cargo types that may reach this connection port in the original network N.

If the partitioned MHS networks contains feedback loops between subnetworks, additional sources $\mathrm{SRC}^{\star}$ and sinks $\mathrm{SNK}^{\star}$ are introduced at connections CN that have been resolved (cf. Figure 3). The outgoing control flow variables of $\mathrm{SRC}^{\star}$ and $\mathrm{SNK}^{\star}$ may take arbitrary values, i.e. they are not restricted. May $\mathcal{C}_{port}(\mathrm{SRC})$ denote the cargo types that are generated by a source SRC in the original MHS network N. Moreover, may $\mathcal{C}_{port}(\mathrm{SNK}^{\star})$ denote the cargo types reaching the resolved connection CN at $\mathrm{SNK}^{\star}$. In iteration 0 of Algorithm 1, each cargo that has been generated by a source and that did not pass CN yet will reach CN and its type is included in $\mathcal{C}_{port}(\mathrm{SNK}^{\star})$. In iteration i, each cargo that has been generated by a SRC and that passed CN between 0 and i times will reach CN and its type is included in $\mathcal{C}_{port}(\mathrm{SNK}^{\star})$. After infinitely many iterations, the types of all cargos that have been generated by a SRC and that arbitrary often passed CN are in $\mathcal{C}_{port}(\mathrm{SNK}^{\star})$, and thus, are also in $\mathcal{C}_{port}(\mathrm{SRC}^{\star})$. However, the algorithm terminates already after a finite number of iterations q ($q \in \mathbb{N}, q > 0$), whenever $\mathcal{C}_{port}(\mathrm{SNK}^{\star}) = \mathcal{C}_{port}(\mathrm{SRC}^{\star})$, because further iterations will not add any additional cargo types. Hence, the iterations after q can be omitted, and the algorithm correctly determines all the cargo types at the resolved connection. ■

REFERENCES

[1] J. A. Tompkins, J. A. White, Y. A. Bozer, and J. M. A. Tanchoco, *Facilities Planning*, 3rd ed. John Wiley & Sons, 2003.

[2] Association of European Airlines, "Consumer Report 06/2012," http://www.aea.be/AEAWEBSITE/STATFILES/CR-12-06.pdf, 2012.

[3] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. The MIT Press, 1999.

[4] T. Klotz, B. Straube, E. Fordran, J. Haufe, F. Schulze, K. Turek, and T. Schmidt, "An approach to the verification of material handling systems," in *Proc. 16th IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2011, pp. 1–8.

[5] VDI, "VDI 3628 Draft: Automated material handling systems - Interfaces between the various function levels in the automation model," 1996.

[6] W. M. P. van der Aalst, "Logistics: A systems oriented approach," in *Proc. 3rd Int. Working Conf. Dynamic Modelling Information Syst.*, 1992, pp. 169–189.

[7] H. Chen, K. Labadi, and L. Amodeo, "Modeling, analysis, and optimization of logistics systems: Petri net based approaches," in *Proc. Int. Conf. Service Syst. Service Manage.*, 2006, pp. 575–582.

[8] M. Dotoli, M. Fanti, G. Iacobellis, and A. Mangini, "A first-order hybrid petri net model for supply chain management," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 4, pp. 744–758, 2009.

[9] D. Herrero-Perez and H. Martinez-Barbera, "Modeling distributed transportation systems composed of flexible automated guided vehicles in flexible manufacturing systems," *IEEE Trans. Ind. Inform.*, vol. 6, no. 2, pp. 166–180, 2010.

[10] F. Basile, P. Chiacchio, and J. Coppola, "A hybrid model of complex automated warehouse systems – Part I: Modeling and simulation," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 4, pp. 640–653, 2012.

[11] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on Petri nets – A literature review," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 437–462, 2012.

[12] S. A. Reveliotis, "Conflict resolution in AGV systems," *IIE Trans.*, vol. 32, no. 7, pp. 647–659, 2000.

[13] N. Wu and W. Zeng, "Deadlock avoidance in an automated guidance vehicle system using a coloured Petri net model," *Int. Jour. Prod. Res.*, vol. 40, no. 1, pp. 223–238, 2002.

[14] N. Wu and M. Zhou, "Modeling and deadlock control of automated guided vehicle systems," *IEEE/ASME Trans. Mech.*, vol. 9, no. 1, pp. 50–57, 2004.

[15] ——, "Shortest routing of bidirectional automated guided vehicles avoiding deadlock and blocking," *IEEE/ASME Trans. Mech.*, vol. 12, no. 1, pp. 63–72, 2007.

[16] T. Nishi and R. Maeno, "Petri net decomposition approach to optimization of route planning problems for AGV systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 3, pp. 523–537, 2010.

[17] T. Nishi and Y. Tanaka, "Petri net decomposition approach for dispatching and conflict-free routing of bidirectional automated guided vehicle systems," *IEEE Trans. Syst. Man Cybern. A., Syst. Humans*, vol. 42, no. 5, pp. 1230–1243, 2012.

[18] T. L. Johnson, "Improving automation software dependability: A role for formal methods?" *Control Eng. Pract.*, vol. 15, no. 11, pp. 1403–1415, 2007.

[19] International Electrotechnical Commission, "IEC Standard 61131-3: Programmable controllers - Part 3," 1993.

[20] I. Moon, "Modeling programmable logic controllers for logic verification," *IEEE Control Syst. Mag.*, vol. 14, no. 2, pp. 53–59, 1994.

[21] O. Rossi and P. Schnoebelen, "Formal modeling of timed function blocks for the automatic verification of ladder diagram programs," in *Proc. 4th Int. Conf. Automat. Mixed Processes*, 2000, pp. 177–182.

[22] B. Zoubek, J.-M. Roussel, and M. Kwiatkowska, "Towards automatic verification of ladder logic programs," in *Proc. Multiconf. Computati. Eng. Syst. Applicat.*, 2003, pp. 9–12.

[23] M. B. Younis and G. Frey, "Formalization of PLC programs to sustain reliability," in *Proc. IEEE Conf. Robotics Automat. Mechatron.*, 2004, pp. 613–618.

[24] V. Gourcuff, O. De Smet, and J.-M. Faure, "Efficient representation for formal verification of PLC programs," in *Proc. 8th Int. Workshop Discrete Event Syst.*, 2006, pp. 182–187.

[25] B. Schlich, J. Brauer, and S. Kowalewski, "Application of static analyses for state-space reduction to the microcontroller binary code," *Sci. Comp. Prog.*, vol. 76, no. 2, pp. 100–118, 2011.

[26] O. Pavlovic and H.-D. Ehrich, "Model checking plc software written in function block diagram," in *Proc. 3rd Int. Conf. Softw. Testing, Verification Validation*, 2010, pp. 439–448.

[27] S. Bornot, R. Huuck, B. Lukoschus, and Y. Lakhnech, "Verification of sequential function charts using SMV," in *Proc. Int. Conf. Parallel Distrib. Process. Tech. Applicat.*, 2000, pp. 2987–2993.

[28] N. Bauer, S. Engell, R. Huuck, S. Lohmann, B. Lukoschus, M. Remelhe, and O. Stursberg, "Verification of PLC programs given as sequential function charts," in *Integration of software specification techniques for*

applications in engineering, ser. LNCS. Springer, 2004, vol. 3147, pp. 517–540.
[29] H. Bel Mokadem, B. Beandrard, V. Gourcuff, O. De Smet, and J. Roussel, "Verification of a timed multitask system with Uppaal," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 4, pp. 921–932, 2010.
[30] N. Wightkin, U. Buy, and H. Darabi, "Formal modeling of sequential function charts with time petri nets," *IEEE Trans. Contr. Sys. Techn.*, vol. 19, no. 2, pp. 455–464, 2011.
[31] T. Klotz, E. Fordran, B. Straube, and J. Haufe, "Formal verification of UML-modeled machine controls," in *Proc. 14th IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2009, pp. 1–7.
[32] D. Witsch, M. Ricken, B. Kormann, and B. Vogel-Heuser, "PLC-statecharts: An approach to integrate UML-statecharts in open-loop control engineering," in *Proc. 8th IEEE Int. Conf. Ind. Inform.*, 2010, pp. 915–920.
[33] M. Bonfe, C. Fantuzzi, and C. Secchi, "Design patterns for model-based automation software design and implementation," *Control Eng. Pract.*, 2012, in press.
[34] O. Ljungkrantz, K. Akesson, M. Fabian, and C. Yuan, "Formal specification and verification of industrial control logic components," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 3, pp. 538–548, 2010.
[35] F. de Renesse and A. Aghvami, "Formal verification of ad-hoc routing protocols using SPIN model checker," in *Proc. IEEE Mediterranean Electrotechnical Conf.*, 2004, pp. 1177–1182.
[36] D. Camara, A. Loureiro, and F. Filali, "Methodology for formal verification of routing protocols for ad hoc wireless networks," in *Proc. IEEE Global Telecomm. Conf.*, 2007, pp. 705–709.
[37] M. Rohrer, "Automod," in *Proc. Winter Sim. Conf.*, 1994, pp. 487–492.
[38] W. B. Nordgren, "Flexsim simulation environment," in *Proc. Winter Sim. Conf.*, 2002, pp. 250–252.
[39] S. Seidel, U. Donath, and J. Haufe, "Approach to a simulation-based verification environment for material handling systems," in *Proc. 17th IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2012, pp. 1–4.
[40] G. Black and V. Vyatkin, "Intelligent component-based automation of baggage handling systems with IEC 61499," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 2, pp. 337–351, 2010.
[41] A. Tarau, B. De Schutter, and H. Hellendoorn, "Model-based control for route choice in automated baggage handling systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 3, pp. 341–351, 2010.
[42] M. Johnstone, D. Creighton, and S. Nahavandi, "Status-based routing in baggage handling systems: Searching verses learning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 2, pp. 189–200, 2010.
[43] K. Hallenborg, "Intelligent control of material handling systems," in *Environmentally Conscious Materials Handling*, M. Kutz, Ed. John Wiley & Sons, 2009, pp. 63–116.
[44] S. W. A. Haneyah, J. M. J. Schutten, P. C. Schuur, and W. H. M. Zijm, "Generic planning and control of automated material handling systems: Practical requirements versus existing theory," *Comp. Ind.*, vol. 64, no. 3, pp. 177–190, 2013.
[45] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, pp. 843–871, 1994.
[46] J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*. Addison Wesley, 1979.
[47] M. Ben-Ari, Z. Manna, and A. Pnueli, "The temporal logic of branching time," *Acta Inform.*, vol. 20, pp. 207–226, 1983.
[48] A. Pnueli, "In transition from global to modular temporal reasoning about programs," in *Logics and models of concurrent systems*. Springer, 1985, pp. 123–144.
[49] C. S. Pasareanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer, "Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning," *Formal Meth. Sys. Des.*, vol. 32, pp. 175–205, 2008.
[50] M. ten Hompel and T. Schmidt, *Warehouse Management: Automation and Organisation of Warehouse and Order Picking Systems*. Springer, 2007.
[51] T. Klotz, N. Seßler, B. Straube, E. Fordran, K. Turek, and J. Schönherr, "On the formal verification of routing in material handling systems," in *Proc. 8th IEEE Conf. Autom. Sci. Eng.*, 2012, pp. 8–13.
[52] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *Proc. 14th Int. Conf. Comput. Aided Verification*, ser. LNCS, vol. 2404, 2002, pp. 359–364.
[53] T. Klotz, N. Seßler, B. Straube, E. Fordran, K. Turek, and J. Schönherr, "Compositional verification of material handling systems," in *Proc. 17th IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2012, pp. 1–8.

Thomas Klotz (S'10) received the Dipl.-Ing. degree in information systems engineering from Technische Universität Dresden, Dresden, Germany, in 2009. Since 2010, he is working toward the Dr.-Ing. degree in computer science at Technische Universität Dresden, Dresden, Germany.

From 2007 to 2008, he was with SAP Research, Palo Alto, CA, USA, where he worked as a Software Engineer Intern. Since 2009, he has been with the Fraunhofer Institute for Integrated Circuits, Design Automation Division, Dresden. His research interests include formal verification, testing, and model-based design.

Jens Schönherr received the Dipl.-Inform. and the Dr.-Ing. degrees in computer science from Technische Universität Dresden, Dresden, Germany, in 1996 and 2002, respectively.

In 1996, he joined the Fraunhofer Institute for Integrated Circuits, Design Automation Division, Dresden, as a Researcher. From 2005 to 2007, he was with OneSpin Solutions, Munich, Germany, where he worked as a Senior Engineer. From 2007 to 2012, he worked as a Group Manager at Signalion, Dresden. Since 2012, he has been Professor of Digital System Design at the Dresden University of Applied Sciences, Germany. His current research focuses on design and verification issues of digital circuits.

Norman Seßler received the Dipl.-Ing. degree in information systems engineering from Technische Universität Dresden, Dresden, Germany, in 2012.

Since 2012, he has been with the Fraunhofer Institute for Integrated Circuits, Design Automation Division, Dresden, where he implements tools for the design and formal verification of hardware/software systems. His research interests include modeling and formal verification of circuits and systems.

Bernd Straube received the Dipl.-Ing., the Dr.-Ing., and the habil. degrees in electrical engineering, network and system theory from Technische Universität Dresden, Dresden, Germany, in 1970, 1973, and 1984, respectively.

In 1973, he joined the Central Institute of Cybernetics and Information Processes of the Academy of Sciences of the former G.D.R. From 1992 to his retirement in 2009, he worked as a group manager at the Fraunhofer Institute for Integrated Circuits, Design Automation Division, Dresden. Currently, he is a senior consultant there. Since 2002, he has also been an Adjunct Professor at the department of Computer Science at Technische Universität Dresden, Dresden, Germany. His research interests include testing, diagnosis, and formal verification.

Karsten Turek received the Dipl.-Ing. degree in electrical engineering from Technische Universität Dresden, Germany, in 1996.

Since 1997, he has been with the Institute of Material Handling and Industrial Engineering at Technische Universität Dresden, Dresden, Germany, where he worked in several research and industrial projects. His research interests focus on simulation and control design of material handling systems.