Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Tutorial:

Warehouse automation by logistic robotic networks: a cyber-physical control approach*

Kai CAI

Department of Electrical and Information Engineering, Osaka City University, Osaka 558-8585, Japan

E-mail: kai.cai@eng.osaka-cu.ac.jp

Received Apr. 7, 2020; Revision accepted Apr. 19, 2020; Crosschecked Apr. 27, 2020

Abstract: In this paper we provide a tutorial on the background of warehouse automation using robotic networks and survey relevant work in the literature. We present a new cyber-physical control method that achieves safe, deadlock-free, efficient, and adaptive behavior of multiple robots serving the goods-to-man logistic operations. A central piece of this method is the incremental supervisory control design algorithm, which is computationally scalable with respect to the number of robots. Finally, we provide a case study on 30 robots with changing conditions to demonstrate the effectiveness of the proposed method.

Key words: Discrete-event systems; Cyber-physical systems; Robotic networks; Warehouse automation; Logistics
https://doi.org/10.1631/FITEE.2000156 **CLC number:** TP27

1 Introduction

The rapid growth of e-commerce, constantly rising labor cost, and increasingly demanding expectation of customers have prompted warehouses worldwide to be equipped with newer and more advanced automation technologies. This has been manifested, over the past decade, by such emergent trends as omni-channel retailing, frequent/complex ordering, same-day shipping, and last-mile delivery. To meet the demanding goals of these trends, it is critical that warehouse automation technologies be unprecedentedly efficient, adaptive, scalable, and fault-tolerant.

Among many recent warehouse automation technologies, the goods-to-man approach has received significant interest from the logistics industry. According to Westernacher Knowledge Series (2017), as of 2016 more than 10% of warehouses in the U.S. have implemented this technology. The goods-to-man technology has game-changed several key operations in warehouses, replacing traditional manual picking and transporting items from storage locations by automatic operations using self-driving robots. A landmark of this technology is the Kiva system (Wurman et al., 2008), which dispatches a large number of mobile robots to serve the logistic operations in Amazon's distribution centers. This goods-to-man approach has drastically improved operational efficiency, reduced labor cost, mitigated errors, as well as created ergonomic working environment. Given these benefits, the trend of employing this technology is currently accelerating, as the number of robots in warehouses is predicted to reach 620 000 by the end of 2021, 15 times the number in 2016 (Tractica, 2017).

With more mobile robots being operated in warehouses, a systematic control framework is indispensable to ensure that their operations are not only efficient and adaptive, but also safe and fault-tolerant. A warehouse is typically cluttered with storage shelves, which extensively constrain the space in which robots can maneuver. When

* Project supported by JSPS KAKENHI (No. JP16K18122)
ORCID: Kai CAI, https://orcid.org/0000-0002-8784-0728

there are a large number of robots simultaneously moving in such constrained space, it is particularly challenging but important to guarantee that there is no collision among the robots, nor deadlock where two or more robots block one another in, e.g., narrow passages created by storage shelves. Once these safety properties are ensured, the control framework should also optimize the total traveling time of all robots such that the overall logistic operations are efficient. Moreover, the warehouse environment is dynamic: there are orders made by customers at unpredictable times, and these orders come as new tasks for which new robots need to be dispatched. In addition, robots during operation may malfunction, or boxes may fall from storage shelves that block a path. The control framework is expected to be adaptive to deal with these changing contingencies.

In our previous work (Tatsumoto et al., 2018b), we applied a formal method, supervisory control theory of discrete-event systems (Cai and Wonham, 2016, 2020; Wonham et al., 2018; Wonham and Cai, 2019), to design supervisory controllers that by construction ensure collision- and deadlock-free behavior for multi-robot maneuvering in constrained space. Supervisory control theory was first proposed by Ramadge and Wonham (1987, 1989) and Wonham and Ramadge (1987), with the aim to formalize general (high-level) control strategies for a wide range of application domains. In this theory, discrete-event systems are modeled as finite-state automata, and their behaviors represented by regular languages. The control feature is that certain events (or state transitions) can be disabled by an external supervisor to enforce a desired behavior. This feature leads to the fundamental concept of language controllability, which determines the existence of a supervisor that suitably disables a series of events in order to satisfy an imposed control specification. Although the controllers designed by the supervisory control theory are correct by construction, there is a well-known computational bottleneck (Gohari and Wonham, 2000); that is, the complexity of computing a supervisory controller is exponential in the number of robots. Hence, the supervisory controller design is not scalable.

To tackle the issue of computational complexity, in our more recent work (Tatsumoto et al., 2018a), we proposed an online supervisory control method. This method generates at the current state of each robot a limited-step-ahead projection of its behavior, and determines based on the joint projected behavior of all robots the next control action to satisfy an imposed control specification. In this way for each computation of an online controller only (typically small) part of the system behavior in a limited-step-ahead window needs to be considered, thereby mitigating the computational effort. A new projection is generated afresh after each execution of a control action (i.e., occurrence of an event), so as a side benefit, time-varying changes in the system can be taken into account in this scheme to enable adaptive behaviors. Online supervisory control based on limited lookahead policy was extensively studied in the 1990s (Chung et al., 1992, 1993, 1994; Hadj-Alouane et al., 1996; Kumar et al., 1998). A key difference in Tatsumoto et al. (2018a) is that a limited-step-ahead projection is produced for each robot which can be efficiently computed, rather than a projection on the joint behavior of all robots (which is often infeasible to compute due again to the above-mentioned exponential complexity). In this way the approach in Tatsumoto et al. (2018a) gains computational efficiency. However, there are also a few drawbacks. First, since the online approach considers only a (small) part of the entire system behavior, it may fail to avoid deadlocks that lie further "down the road." Second, the recomputation of an online controller after each occurrence of the event may not be necessary if no change has occurred, which results in frequent waste of computational resources. Third, although the online method was experimentally demonstrated with real mobile robots, it was not explained in Tatsumoto et al. (2018a) how the discrete-event supervisory control decisions were implemented through real-time continuous controllers for the individual agents. Finally, in Tatsumoto et al. (2018a, 2018b), only the shortest paths of individual robots were taken into account. However, shortest paths may be very different from shortest time that the team of all robots spends to finish their tasks; thus, the issue of time efficiency was not addressed.

In this study, we propose a new cyber-physical control method that handles all the drawbacks mentioned above. On the cyber level, we design a learning approach that incrementally constructs a controller that is guaranteed to achieve collision- and

deadlock-free behavior. This learning approach at each iteration selects (whenever possible) an event from each robot in a random order, and tests if the trajectory consisting of the selected events satisfies an imposed specification. If so, the trajectory will be added to form a new part of the controller; otherwise, a different trajectory will be selected and tested alike. This process is repeated until the goal state where all robots accomplish their tasks is reached. Since the computational complexity at each iteration is linear with respect to the number of robots, this learning approach is scalable. Moreover, since the approach selects as many robots as possible to move at each iteration, it addresses the issue of time efficiency in combination with shortest paths of individual robots.

On the physical level, at each iteration the cyber-level control decisions, or "command signals" (given in terms of the events of individual robots), are converted to a stabilization problem for each robot to move from the current location to the next waypoint location. To solve this stabilization problem, standard state-feedback control by pole placement is employed. Once each robot reaches (the proximity of) its next waypoint, a "report signal" is sent back to the cyber level for the high-level controller to update its state accordingly. This feedback of command and report between the cyber and physical levels integrates the discrete and continuous controls.

Finally, to achieve adaptive behaviors in response to environment changes, we employ an online reconfiguration-recomputation approach. Specifically, whenever a change occurs (and is detected), models of the relevant robots will be updated according to the change. For example, if a fallen box blocks a path, the robots scheduled to use that path will update their models by recomputing shortest paths while treating the path as an obstacle. Or, if a robot is added (resp. removed due to malfunction), simply add (resp. remove) the robot model. Once the robot models are updated, the learning approach will be used to recompute a new online controller that adapts to the new situation. Different from the online approach in Tatsumoto et al. (2018a), recomputation is done only when a change occurs, thereby avoiding possibly unnecessary use of computational resources.

2 Related work

The goods-to-man warehouse automation based on robotic networks is broadly a multi-robot path planning and motion scheduling problem. Such a problem has been extensively studied in the literature. In this section, we review the work most related to the method we propose in this study. The listed references are by no means complete, but organized in the following three categories.

2.1 Symbolic motion planning and formal methods

Symbolic motion planning for multi-robot systems is the problem of automatic synthesis of motion control algorithms for multiple robots from a given high-level specification. Formal methods address this problem by constructing a high-level supervisor (modeled as finite-state automata) that satisfies the given specification (described using temporal logic formulas). Subsequently, the high-level supervisor is implemented using continuous-time algorithms on the physical level of the individual robots. Relying on a key concept of bisimulation, the physical-level implementation maintains the achieved high-level correctness. Commonly used temporal logic includes linear temporal logic, computation tree logic, and more recently interval temporal logic, real-time temporal logic (Manna and Pnueli, 1992; Kroger and Merz, 2010; Goranko and Galton, 2015). Finite-state automata used in this context are typically Büchi and Rabin automata, which accept infinite-length inputs (Grädel et al., 2002).

Much recent work has been done for multi-robot symbolic motion planning using formal methods (Belta et al., 2007, 2017; Kloetzer and Belta, 2008; Kress-Gazit et al., 2009; Chen et al., 2012). The strength of formal methods lies in guaranteeing correctness (thus safety) by construction, on both the cyber level and the physical level owing to bisimulation. As a result, continuous-time dynamics of robots can be dealt with, and robustness issues addressed.

The same as the supervisory control theory, however, the computational complexity using formal methods (involving finite-state automata) is exponential in the number of robots. Consequently, the computation of a high-level supervisor is not scalable. Moreover, the issue of ensuring deadlock-free

behaviors is usually not studied using formal methods, but it is critical in the goods-to-man warehouse automation.

2.2 Dynamic vehicle routing

Dynamic vehicle routing deals with the problem of assigning dynamically appearing tasks to unmanned vehicles or mobile robots, and planning corresponding routes for these vehicles/robots. The problem is typically formulated under a stochastic framework, which assumes that tasks arrive with a certain probability distribution. Performance of designed routing algorithms is also evaluated using suitable stochastic criteria.

Extensive studies have been conducted on the dynamic vehicle routing problem (Bertsimas and van Ryzin, 1991, 1993; Arsie et al., 2009; Smith et al., 2010; Bullo et al., 2011). Various interesting issues have been addressed, including time constraints of task completion, task queueing and stability, tasks with different priorities, tasks requiring different types of vehicles (providing different services), vehicle team forming, and continuous-time dynamics of vehicles. With ingenious heuristics, a number of routing algorithms have been reported that achieve highly efficient and adaptive behaviors.

However, most work on dynamic vehicle routing focused on free space, and consequently the issues of collision and deadlock were typically not considered. These issues, on the other hand, cannot be ignored and are among the most important in warehouse automation since the space in warehouses is severely constrained.

2.3 Path planning and motion scheduling

In robotics and operations research, path planning and motion scheduling for multi-robot systems have been widely studied (LaValle, 2006; Standley, 2010; Karaman et al., 2011; Pinedo, 2012; Čáp et al., 2015). Specifically, path planning is concerned with assigning non-interacting paths to multiple robots such that no collision occurs among robots and the total length of all paths is minimum. Motion scheduling, on the other hand, brings in time and tackles the problem of designing paths along which multiple robots can complete their maneuvers in minimum time.

Both problems are combinatorial optimization problems, and in their generality suffer from intractable computational complexity as the number of robots increases (just as formal methods and supervisory control theory). Inventing smart heuristics, a number of well-known algorithms have been proposed, including A* and its many variants, independence detection, operator decomposition, and rapidly-exploring random tree (Hart et al., 1968; LaValle and Kuffner, 2001; Standley and Korf, 2011).

However, the algorithms developed for path planning and motion scheduling often focus on high-level path design and do not take continuous-time dynamics or continuous-time control design into account.

In the next section, we present a new cyber-physical control approach that complements the above reviewed literature by addressing issues of continuous-time control, collision avoidance, deadlock free, computational scalability, as well as adaptivity to changes. Owing to the tutorial nature of this paper, we focus on providing an intuitive account of the proposed approach, and leave out technical details including proofs.

3 Cyber-physical control approach

Different warehouses have different configurations. In this study we adopt the grid-type layout as displayed in Fig. 1, and consider the goods-to-man scenario using a team of self-driving robots. Other warehouse layouts can in principle be handled

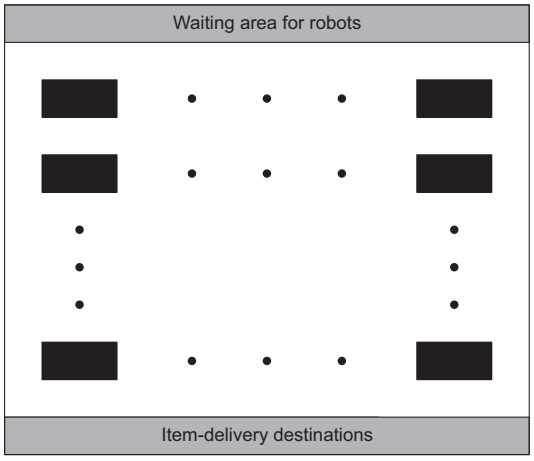

Fig. 1 Warehouse grid layout: items to be picked up are stored in black-rectangle areas. Reprinted from Tatsumoto et al. (2018a), Copyright 2018, with permission from Elsevier

similarly. The top area is where mobile robots are waiting for tasks, the black-rectangle areas are where items to be picked up are stored, and the bottom area is the item-delivery destination (where a handful of human workers are operating).

When a robot is assigned a task, three locations are given: (1) start location, which is a cell immediately below the top waiting area; (2) item location, which is a cell in one of the black-rectangle areas; (3) goal location, which is a cell immediately above the bottom item-delivery destination.

Once being assigned a task, a robot should travel first to the item location, picking up the item, and then transport the item to the goal location.

While navigating in the warehouse, each robot must always avoid collisions with the item-storage areas (black-rectangles), except when it needs to enter such an area where its assigned item is stored. It is assumed that a robot can turn left, turn right, move forward, but never move backward. The following issues are of particular interest, when multiple robots are concurrently serving the warehouse for goods-to-man logistic operations. These issues are: (1) safety—every robot should never collide with static item-storage areas nor with other moving robots; (2) deadlock-free—robots should never block one another (in aisles between item-storage areas as shown in Fig. 2) such that none can accomplish its item pickup or delivery; (3) efficiency—the total time of all robots finishing their assigned item delivery tasks should be minimum; (4) adaptivity—robots should adjust their maneuvers to cope with task and environment changes.

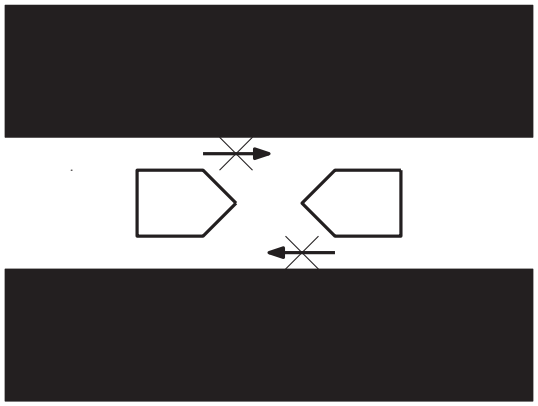

Fig. 2 Deadlock (or blocking): neither robots can move forward and no robot can finish its delivery. Reprinted from Tatsumoto et al. (2018a), Copyright 2018, with permission from Elsevier

In the sequel, we present a cyber-physical control framework that addresses all the above issues.

3.1 Cyber level

3.1.1 Modeling of mobile robots

Consider n ($n > 1$) robots concurrently serving the warehouse for goods-to-man logistics. On the cyber level, we propose the following automaton model for each robot. Sequentially, for each cell of the grid, we assign natural numbers which will be used as state numbers. The top waiting area is assigned q_0, which is the initial state for each robot. On the other hand, the bottom item-delivery destination is assigned q_∞, which is the only goal state for each robot.

In the grid a robot may move to the north, to the south, to the west, or to the east (however, based on the direction of maneuver, it can only turn left, turn right, or move forward). Thus, each robot is associated with four events, one for each direction of move. In Table 1, we designate robot $k \in \{1, 2, \ldots, n\}$ its four events correspondingly with four numbers. Write $\Sigma_k = \{k1, k3, k5, k7\}$ for the event set of robot k. For this particular application, it is reasonable to assume that all events are controllable; that is, an external supervisor may enable or disable all the robots' events.

When robot k is assigned a task, represented by a start location $q_{k,\mathrm{start}}$, an item location $q_{k,\mathrm{item}}$, and a goal location $q_{k,\mathrm{goal}}$, we calculate the following shortest paths (measured by the number of events): (1) compute all shortest paths between the start location $q_{k,\mathrm{start}}$ and the cell immediately above $q_{k,\mathrm{item}}$ (say, denoted by $q^{\uparrow}_{k,\mathrm{item}}$); (2) compute all shortest paths between the cell immediately below $q_{k,\mathrm{item}}$ (say, denoted by $q^{\downarrow}_{k,\mathrm{item}}$) and the goal location $q_{k,\mathrm{goal}}$.

These shortest paths contribute in part to the previously mentioned efficiency issue.

Denote by $Q_{k,\mathrm{sp}}$ the set of all states included in the computed shortest paths. Thus, the total state set Q_k of robot k is

$$Q_k := Q_{k,\mathrm{sp}} \cup \{q_0, q_\infty, q_{k,\mathrm{item}}\}. \tag{1}$$

Table 1 Event numbers of each robot $k \in \{1, 2, \ldots, n\}$

Event	Event numbers	Event	Event numbers
Move north	$k \times 10 + 1$	Move south	$k \times 10 + 5$
Move east	$k \times 10 + 3$	Move west	$k \times 10 + 7$

Moreover, denote by $\delta_{k,\mathrm{sp}}$ all the state transitions included in the shortest paths; thus, the total transition function of robot k is given by

$$\delta_k := \delta_{k,\mathrm{sp}} \cup \Big\{ [q_0, k5, q_{k,\mathrm{start}}], [q^{\uparrow}_{k,\mathrm{item}}, k5, q_{k,\mathrm{item}}], [q_{k,\mathrm{item}}, k5, q^{\downarrow}_{k,\mathrm{item}}], [q_{k,\mathrm{goal}}, k5, q_\infty] \Big\}. \quad (2)$$

In summary, the automaton model of an arbitrary robot $k \in \{1, 2, \ldots, n\}$ is the five-tuple:

$$\boldsymbol{G}_k = (Q_k, \Sigma_k, \delta_k, q_0, q_\infty). \quad (3)$$

Note that while Σ_k, q_0, and q_∞ are fixed, Q_k and δ_k depend on the start, item, and goal locations assigned to robot k. Also, note that in our setup, q_∞ is reachable from q_0.

3.1.2 Learning-based incremental supervisory control design

With individual robots' automaton models $\boldsymbol{G}_k$, in the standard supervisory control theory (Wonham and Cai, 2019) the "synchronous product" of $\boldsymbol{G}_k$ is first computed, and then all "bad" states (those where collision and/or deadlock occur) are algorithmically removed. A brief summary of the standard supervisory control design is provided in the appendix. However, the computation of synchronous product is known to have complexity exponential in the number of robots; thus, the computation quickly becomes infeasible as the number of robots increases (Tatsumoto et al., 2018b). In Tatsumoto et al. (2018a), though the presented online method was more efficient in that it computed the synchronous product of projected versions of $\boldsymbol{G}_k$, the issue of deadlock-free was not guaranteed (as a tradeoff to achieved efficiency).

In this subsection we propose a new learning-based incremental supervisory control design method, which avoids computing any synchronous product. Instead, this method tries one event per robot (based on $\boldsymbol{G}_k$), and tests after each event whether or not collision occurs. If collision occurs, discard the selected event and choose a different one (whenever possible) from the same robot. Otherwise (i.e., there is no collision), record the event and the next state reached by the one-step transition as incremental expansion of the supervisor being built. Repeat the operations on the next new state (by way of recursion). In this manner, collision-free behavior is ensured. If every robot is at its goal state q_∞, then the algorithm terminates.

If, at some state, all possible events of all robots result in collision, then no event can occur at this state; i.e., the state is a deadlock. In this case, the operations at this state are all done, and by the recursive mechanism computation backtracks to the previous state and continues to search other trajectories (potentially leading to the goal). Finally, when the algorithm terminates, it first trims the obtained automaton by removing all deadlock states as well as those states that can transit only to a deadlock, and then outputs the trimmed automaton as the resulting supervisor. In this way, deadlock-free behavior is also guaranteed.

Moreover, the proposed method tackles the time-efficiency issue that was not addressed in Tatsumoto et al. (2018a, 2018b). This issue is tackled by means of incorporating a mechanism of choosing as many robots as possible in each round, such that the corresponding events can be executed simultaneously on multiple robots on the physical level (see Section 3.2). Since all routes of all robots are shortest paths, and as many robots as possible simultaneously execute their maneuvers, the time for the team to reach the goal is minimized.

In the following, we present the learning-based incremental supervisory control algorithm. In Algorithm 1, trim$(\cdot)$ computes an automaton by removing all deadlock states and those states that can transit only to a deadlock, and $\delta_k(q_k, \sigma)!$ means that the transition δ_k by event σ is defined at state q_k.

In the incremental supervisory control (iSupCon) algorithm, lines 1–5 initiate the supervisor to be incrementally built. Line 6 initiates a set $\mathcal{R}$ to be used for choosing as many robots as possible in one round. Line 7 initiates an integer D to be used for detecting deadlock states. Lines 8 and 9 call the function isupcon with $\bar{q}_0$ as the argument.

In the isupcon function, lines 10–15 choose a robot and remove it from the set $\mathcal{R}$ for the next selection (unless it is the last robot, in which case $\mathcal{R}$ is reset to the full set); in this way, as many robots as possible may be selected in a round. Lines 17–33 try to choose an event from the selected robot that does not cause collision. If a collision is confirmed by the chosen event, then choose a different possible event from the same robot (lines 17–21). If no collision

Algorithm 1 Incremental supervisory control (iSupCon) algorithm

Input: $\boldsymbol{G}_k = (Q_k, \Sigma_k, \delta_k, q_0, q_\infty)$, $k = 1, 2, \ldots, n$
Output: **SUP** $= (Q, \Sigma, \delta, \bar{q}_0, \bar{q}_\infty)$

1: $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_n$
2: $\bar{q}_0 = (q_0, q_0, \ldots, q_0)$
3: $\bar{q}_\infty = (q_\infty, q_\infty, \ldots, q_\infty)$
4: $Q = \{\bar{q}_0, \bar{q}_\infty\}$
5: $\delta = \varnothing$
6: $\mathcal{R} = \{1, 2, \ldots, n\}$
7: $D = 0$
8: $q = \bar{q}_0$
9: isupcon$(q = (q_1, q_2, \ldots, q_n))$
10: Choose a robot $k \in \mathcal{R}$
11: **if** $\mathcal{R} \setminus \{k\} = \varnothing$ **then**
12: $\quad \mathcal{R} = \{1, 2, \ldots, n\}$
13: **else**
14: $\quad \mathcal{R} = \mathcal{R} \setminus \{k\}$
15: **end if**
16: $D = D + 1$
17: **while** $\Sigma_k(q_k) := \{\sigma \in \Sigma_k \mid \delta_k(q_k, \sigma)!\} \neq \varnothing$ **do**
18: $\quad$ Choose an event $\sigma_k \in \Sigma_k(q_k)$
19: $\quad \Sigma_k(q_k) = \Sigma_k(q_k) \setminus \{\sigma_k\}$
20: $\quad$ **if** $\delta_k(q_k, \sigma_k) =: q'_k = q_i$ for some $i \neq k$ **then**
21: $\quad\quad$ **continue**
22: $\quad$ **else**
23: $\quad\quad D = 0$
24: $\quad\quad q' = (q_1, q_2, \ldots, q'_k, \ldots, q_n)$
25: $\quad\quad Q = Q \cup \{q'\}$
26: $\quad\quad \delta = \delta \cup \{[q, \sigma_k, q']\}$
27: $\quad\quad$ **if** $q' = \bar{q}_\infty$ **then**
28: $\quad\quad\quad$ Output trim(**SUP** $= (Q, \Sigma, \delta, \bar{q}_0, \bar{q}_\infty)$)
29: $\quad\quad$ **else**
30: $\quad\quad\quad$ isupcon(q')
31: $\quad\quad$ **end if**
32: $\quad$ **end if**
33: **end while**
34: **if** $D < n$ **then**
35: $\quad$ **go to** line 10
36: **end if**

occurs, then update the supervisor **SUP** by adding the new state reached by the chosen event and the corresponding transition (lines 22–26). In case the newly reached state is the goal state $\bar{q}_\infty$, then output the trimmed **SUP** as the final result (lines 27 and 28). Otherwise, call the function isupcon with the new state as the argument (lines 29 and 30); in this manner the algorithm is recursive. Finally, lines 16, 23, and 34–36 of operations on the integer D determine if a state is a deadlock. If line 16 is executed n times (without ever being reset by line 23), then no robot can execute any event without causing collision; i.e., the current state is a deadlock. In this case, line 35 will not be executed, and the isupcon function terminates. By recursion the algorithm returns to the previous isupcon function on a previous state.

Theorem 1 The iSupCon algorithm terminates in a finite number of steps, and the resulting supervisor **SUP** $= (Q, \Sigma, \delta, \bar{q}_0, \bar{q}_\infty)$ ensures collision- and deadlock-free behavior.

Proof Since in each robot $\boldsymbol{G}_k$ the goal state q_∞ is reachable from the initial state q_0, and all event sets Σ_k are disjoint, $\bar{q}_\infty$ is reachable from $\bar{q}_0$ in the synchronous product of all $\boldsymbol{G}_k$. Further, owing to the check of line 34, the algorithm will always explore new transitions (unless $\bar{q}_\infty$ is reached). Hence, it follows that the condition of line 27 will eventually be met; i.e., the algorithm terminates. Since there are finite numbers of states and transitions, the termination occurs in a finite number of steps.

It is left to show that the resulting supervisor **SUP** ensures collision- and deadlock-free behavior. Owing to the check of line 20, an event that causes a collision will never be added to **SUP**; thereby, collision-free behavior is guaranteed. Moreover, the trim operation on line 28 when outputting **SUP** ensures deadlock-free behavior.

3.2 Physical level

Once a supervisor **SUP** $= (Q, \Sigma, \delta, \bar{q}_0, \bar{q}_\infty)$ on the cyber level is computed by the iSupCon algorithm, it will send supervisory control signals to the physical level, in terms of which events to be executed by which robots. At each round, the supervisor **SUP** sends an event to each robot that is selected to move, and disables maneuvers of those robots that are not selected (to avoid a collision or deadlock). Specifically, suppose that the supervisor **SUP** is at state $q \in Q$ and let $\Sigma(q) := \{\sigma \in \Sigma \mid \delta(q, \sigma)!\}$ be the subset of events defined at q. Namely, the events in $\Sigma(q)$ are enabled by **SUP** (and those not in $\Sigma(q)$ are disabled). Let k be such that $\Sigma_k \cap \Sigma(x) \neq \varnothing$; i.e., there exists an event of robot k that is enabled by the supervisor. Then **SUP** selects $\sigma_k \in \Sigma_k \cap \Sigma(x)$ and sends σ_k to robot k for the corresponding maneuver.

When robot k receives event $\sigma_k \in \{k1, k3, k5, k7\}$, it needs a continuous-time controller to carry out the corresponding maneuver. For simplicity, we assume that each robot k is a

point mass moving on the two-dimensional (2D) plane, i.e.,

$$\dot{\boldsymbol{x}}_k = \boldsymbol{u}_k,\ \boldsymbol{x}_k,\ \boldsymbol{u}_k \in \mathbb{R}^2.$$

At time t, the coordinates of robot k are $\boldsymbol{x}_k(t)$ (in the global coordinate frame). Based on σ_k, one may obtain the coordinates of the next waypoint (say) $\boldsymbol{x}_k^*$. Then the maneuver is a stabilization problem of driving robot k from $\boldsymbol{x}_k(t)$ to $\boldsymbol{x}_k^*$.

Suppose that the state $\boldsymbol{x}_k(t)$ of robot k is measurable for all time t (say, by an indoor camera positioning system). Then a standard state-feedback control

$$\boldsymbol{u}_k = \boldsymbol{F}_k(\boldsymbol{x}_k^* - \boldsymbol{x}_k(t))$$

solves the stabilization problem (asymptotically), where $\boldsymbol{F}_k \in \mathbb{R}^{2\times 2}$ is a diagonal matrix with positive diagonal entries.

We point out that there are a number of issues to consider for practical applications. First, for finite convergence to the next waypoint $\boldsymbol{x}_k^*$, one needs to impose a stop condition:

$$||\boldsymbol{x}_k(t) - \boldsymbol{x}_k^*|| < d,$$

where d is a small positive number. This means that if the position of robot k is sufficiently close to $\boldsymbol{x}_k^*$, one may terminate the continuous-time controller. Second, since real robots have nonzero sizes, one needs to first obtain "configuration space" (e.g., LaValle (2006)) for robots' maneuvers, and consider control under state constraints. Third, real robots also have constraints on their inputs, which can be dealt with by, e.g., anti-windup control strategies. Finally, it might be more accurate to use double-integrator or unicycle models for mobile robots to take velocity and orientation into account. More advanced control techniques are needed for these dynamic models.

When robot k finishes the commanded maneuver, it sends a report signal back to the supervisor **SUP** to indicate its status of completion. Recall that **SUP** is at state $q \in Q$. Once **SUP** receives the report signal from robot k, it updates its state by executing the event σ_k; that is, **SUP** on the cyber level transits to the new state $q' := \delta(q, \sigma_k)$. Then the cycle of command and report repeats (until all robots reach their goal states q_∞).

3.3 Online reconfiguration-recomputation

To make the presented cyber-physical control method adapt to unpredictable changes such as new tasks, malfunctioned robots, and fallen boxes, we further propose an online reconfiguration-recomputation strategy.

Specifically, when a change is detected, we reconfigure the automaton models of the robots that are affected by the change. For example, if a fallen box blocks a path that is to be used by some robots, then we recompute the shortest paths for these robots by treating the blocked paths as obstacles. As another example, if a malfunctioned robot is detected by another robot that passes by, then a new robot needs to be dispatched to take out the failed robot. In this case, we remove the model of the failed robot, add the model of the newly dispatched robot that treats the failed robot as an item to pick up, and recompute the shortest paths of those robots whose previous paths include the location occupied by the failed robot.

Once the automaton models of the relevant robots are reconfigured, we apply the iSupCon algorithm to recompute a new supervisor. In this way, the new supervisor adapts to the new scenario, thus achieving adaptive behavior.

4 Case study

In this section we consider a case study of goods-to-man logistics using multiple robots, and apply the proposed cyber-physical control method for controller design.

Specifically, consider the warehouse layout displayed in Fig. 3. Assign natural numbers to each cell

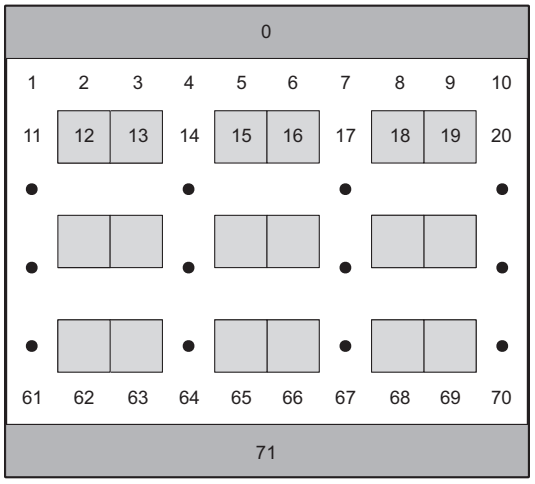

Fig. 3 Case study: warehouse layout. Reprinted from Tatsumoto et al. (2018a), Copyright 2018, with permission from Elsevier

of the grid; these will be used as state numbers. The waiting area for robots at the top is assigned "0," which is the initial state for all robots, i.e., $q_0 = 0$. On the other hand, the item-delivery destination at the bottom, assigned in this case "71," is the goal state for each robot, i.e., $q_\infty = 71$.

We consider 30 robots serving the warehouse, and 30 tasks assigned in three batches (10 tasks per batch). The following sequence of scenarios is considered:

1. Initially 10 robots are dispatched to serve the first batch of 10 tasks.

2. Next 10 more robots are dispatched to serve the second batch of 10 tasks.

3. Subsequently 10 more robots are dispatched to serve the third batch of 10 tasks.

4. A while later, a fallen box occupies the cell numbered "54," thus blocking the corresponding aisle.

Each robot is assigned a task, given by three locations (start, item, and goal). We obtain for each robot an automaton model by computing the corresponding shortest paths. As an example, in Fig. 4, a robot is assigned a start location "7," item location "32," and goal location "70." As shown in Fig. 5, first find all shortest paths from "7" (start) to "22" (location just above item), and then from "42" (location just below item) to "70" (goal).

To deal with the sequence of four scenarios, the online reconfiguration-recomputation strategy is employed. Initially, for the first 10 robots with their automaton models, we apply the iSupCon algorithm to compute a supervisor that achieves collision-free, deadlock-free, and time-efficient behavior.

To physically implement the computed supervisor, we consider the 10 robots each being a point mass moving on the 2D plane:

$$\dot{\boldsymbol{x}}_k = \boldsymbol{u}_k,\ \boldsymbol{x}_k,\ \boldsymbol{u}_k \in \mathbb{R}^2, k = 1, 2, \ldots, 10.$$

At each round, the supervisor sends one event to each robot that is selected to move; those robots that do not receive an event from the supervisor will stay put. The robots that receive an event simultaneously solve a stabilization problem from the current coordinates to the next waypoint coordinates. Let $\boldsymbol{x}_k(t)$ denote the coordinates of robot k at time t, and $\boldsymbol{x}_k^*$ the coordinates of the next waypoint for robot k. Then the following state-feedback controller

$$\boldsymbol{u}_k = \boldsymbol{x}_k^* - \boldsymbol{x}_k(t)$$

drives $\boldsymbol{x}_k(t)$ to $\boldsymbol{x}_k^*$ exponentially fast. Set a stop condition:

$$||\boldsymbol{x}_k(t) - \boldsymbol{x}_k^*|| < 0.1.$$

When the condition is satisfied, robot k sends a report back to the supervisor. When all robots that receive events report back to the supervisor, the supervisor makes the corresponding state transitions on the cyber level.

When the second batch of 10 tasks arrives, 10 more robots are dispatched. To address this change,

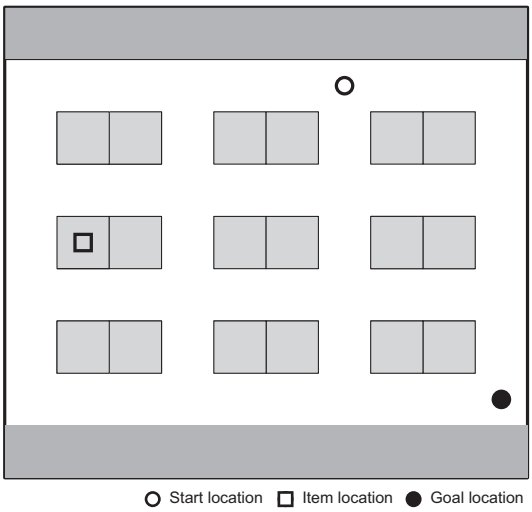

Fig. 4 Case study: start, item, and goal locations assigned to a particular robot. Reprinted from Tatsumoto et al. (2018a), Copyright 2018, with permission from Elsevier

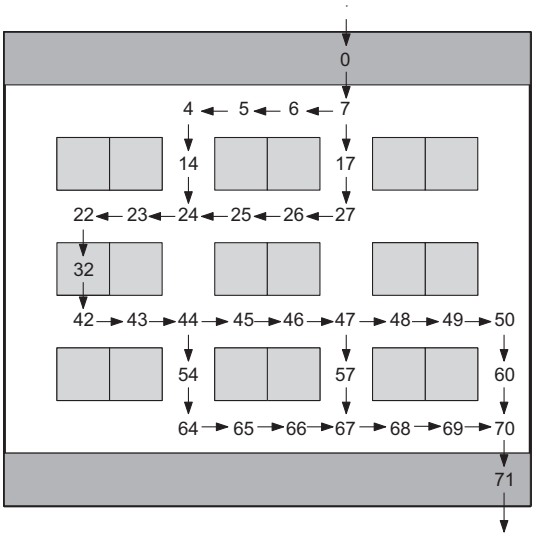

Fig. 5 Case study: automaton model of the robot in Fig. 4. Reprinted from Tatsumoto et al. (2018a), Copyright 2018, with permission from Elsevier

we first obtain the automaton models of the 10 new robots, while keeping the first 10 invariant, and then apply the iSupCon algorithm with all 20 robot models as input to compute a new supervisor for the changed situation. The implementation of the new supervisor on the physical level is similar to the above.

When the third batch of 10 tasks arrives, the final 10 robots are dispatched. To address this change, similar to the above we first obtain the automaton models of the 10 new robots, while keeping the existing 20 invariant, and then apply the iSupCon algorithm with all 30 robot models as input to compute a new supervisor for the changed situation. The implementation of the new supervisor on the physical level is the same as before.

Finally, we consider the case where a fallen box occupies the cell numbered "54." Such a change may be detected by a robot traveling to cell "44," one cell above "54." In such a case, we reconfigure all those automaton models of the robots that contain "54" as a state. For these robots, we recompute their shortest paths by treating "54" as an obstacle. We keep the rest robot models invariant, and apply the iSupCon algorithm with respect to the (partially) reconfigured models of all 30 robots. The implementation of the new supervisor on the physical level is the same as before.

To conclude, our proposed cyber-physical control method successfully achieves safe, deadlock-free, time-efficient, and adaptive behavior for multiple robots serving goods-to-man logistics. We note that in each of the four computations by the iSupCon algorithm, the computation time is less than 1 s. In contrast, for the size of 30 robots, neither the standard supervisory control theory used in Tatsumoto et al. (2018b), nor the online method in Tatsumoto et al. (2018a) can feasibly compute a supervisor due to the exponential complexity of the synchronous product computation.

5 Conclusions

In this study we have considered warehouse automation, in particular the goods-to-man logistic operation using multiple self-driving robots. We have introduced the background of this technology and reviewed the literature on related problems. A new cyber-physical control method has been presented that achieves safe, deadlock-free, efficient, and adaptive behavior of the multi-robot team. Moreover, the incremental supervisory control design algorithm is computationally scalable with respect to the number of robots. Finally, a case study on 30 robots with changing conditions has been provided to demonstrate the effectiveness of the proposed method.

Our ongoing work includes experimentally implementing and testing the performance of the proposed method. In addition, exploring big logistic data to help further improve the operational efficiency is an important target of future research.

Acknowledgements

The iSupCon algorithm was originated in Yuta TATSUMOTO's Master's thesis "A Semi-Model-Free Approach for Efficient Supervisory Control Synthesis" (Osaka City University, 2019).

Compliance with ethics guidelines

Kai CAI declares that he has no conflict of interest.

References

Arsie A, Savla K, Frazzoli E, 2009. Efficient routing algorithms for multiple vehicles with no explicit communications. *IEEE Trans Autom Contr*, 54(10):2302-2317. https://doi.org/10.1109/TAC.2009.2028954

Belta C, Bicchi A, Egerstedt M, et al., 2007. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robot Autom Mag*, 14(1):61-70. https://doi.org/10.1109/MRA.2007.339624

Belta C, Yordanov B, Gol EA, 2017. Formal Methods for Discrete-Time Dynamical Systems. Springer, Cham, Switzerland.

Bertsimas DJ, van Ryzin G, 1991. A stochastic and dynamic vehicle routing problem in the Euclidean plane. *Oper Res*, 39(4):601-615. https://doi.org/10.1287/opre.39.4.601

Bertsimas DJ, van Ryzin G, 1993. Stochastic and dynamic vehicle routing in the Euclidean plane with multiple capacitated vehicles. *Oper Res*, 41(1):60-76. https://doi.org/10.1287/opre.41.1.60

Bullo F, Frazzoli E, Pavone M, et al., 2011. Dynamic vehicle routing for robotic systems. *Proc IEEE*, 99(9):1482-1504. https://doi.org/10.1109/JPROC.2011.2158181

Cai K, Wonham WM, 2016. Supervisor Localization: a Top-Down Approach to Distributed Control of Discrete-Event Systems. Springer, Cham, Switzerland.

Cai K, Wonham WM, 2020. Supervisory control of discrete-event systems. In: Baillieul J, Samad T (Eds.), Encyclopedia of Systems and Control. Springer, London, UK. https://doi.org/10.1007/978-1-4471-5102-9_54-1

Čáp M, Novák P, Kleiner A, et al., 2015. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Trans Autom Sci Eng*, 12(3):835-849. https://doi.org/10.1109/TASE.2015.2445780

Chen YS, Ding XC, Stefanescu A, et al., 2012. Formal approach to the deployment of distributed robotic teams. *IEEE Trans Robot*, 28(1):158-171. https://doi.org/10.1109/TRO.2011.2163434

Chung SL, Lafortune S, Lin F, 1992. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans Autom Contr*, 37(12):1921-1935. https://doi.org/10.1109/9.182478

Chung SL, Lafortune S, Lin F, 1993. Recursive computation of limited lookahead supervisory controls for discrete event systems. *Discr Event Dynam Syst*, 3(1):71-100. https://doi.org/10.1007/BF01439177

Chung SL, Lafortune S, Lin F, 1994. Supervisory control using variable lookahead policies. *Discr Event Dynam Syst*, 4(3):237-268. https://doi.org/10.1007/BF01438709

Gohari P, Wonham WM, 2000. On the complexity of supervisory control design in the RW framework. *IEEE Trans Syst Man Cybern*, 30(5):643-652. https://doi.org/10.1109/3477.875441

Goranko V, Galton A, 2015. Temporal Logic. Metaphysics Research Lab, Stanford University, USA.

Grädel E, Thomas W, Wilke T, 2002. Automata, Logics, and Infinite Games. Springer, Germany.

Hadj-Alouane NB, Lafortune S, Lin F, 1996. Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation. *Discr Event Dynam Syst*, 6(4):379-427. https://doi.org/10.1007/BF01797138

Hart PE, Nilsson NJ, Raphael B, 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans Syst Sci Cybern*, 4(2):100-107. https://doi.org/10.1109/TSSC.1968.300136

Karaman S, Walter MR, Perez A, et al., 2011. Anytime motion planning using the RRT. Proc IEEE Int Conf on Robotics and Automation, p.1478-1483. https://doi.org/10.1109/ICRA.2011.5980479

Kloetzer M, Belta C, 2008. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans Autom Contr*, 53(1):287-297. https://doi.org/10.1109/TAC.2007.914952

Kress-Gazit H, Fainekos GE, Pappas GJ, 2009. Temporal-logic-based reactive mission and motion planning. *IEEE Trans Robot*, 25(6):1370-1381. https://doi.org/10.1109/TRO.2009.2030225

Kroger F, Merz S, 2010. Temporal Logic and State Systems. Springer, New York, USA.

Kumar R, Cheung HM, Marcus SI, 1998. Extension based limited lookahead supervision of discrete event systems. *Automatica*, 34(11):1327-1344. https://doi.org/10.1016/S0005-1098(98)00077-6

LaValle SM, 2006. Planning Algorithms. Cambridge University Press, New York, USA.

LaValle SM, Kuffner JJJr, 2001. Randomized kinodynamic planning. *Int J Robot Res*, 20(5):378-400. https://doi.org/10.1177/02783640122067453

Manna Z, Pnueli A, 1992. The Temporal Logic of Reactive and Concurrent Systems. Springer, New York, USA.

Pinedo ML, 2012. Scheduling: Theory, Algorithms, and Systems (4th Ed.). Springer, New York, USA.

Ramadge PJ, Wonham WM, 1987. Supervisory control of a class of discrete event processes. *SIAM J Contr Optim*, 25(1):206-230. https://doi.org/10.1137/0325013

Ramadge PJ, Wonham WM, 1989. The control of discrete event systems. *Proc IEEE*, 77(1):81-98. https://doi.org/10.1109/5.21072

Smith SL, Pavone M, Bullo F, et al., 2010. Dynamic vehicle routing with priority classes of stochastic demands. *SIAM J Contr Optim*, 48(5):3224-3245. https://doi.org/10.1137/090749347

Standley T, 2010. Finding optimal solutions to cooperative pathfinding problems. Proc 24th AAAI Conf on Artificial Intelligence, p.173-178.

Standley T, Korf R, 2011. Complete algorithms for cooperative pathfinding problems. Proc 22nd Int Joint Conf on Artificial Intelligence, p.668-673.

Tatsumoto Y, Shiraishi M, Cai K, et al., 2018a. Application of online supervisory control of discrete-event systems to multi-robot warehouse automation. *Contr Eng Pract*, 81:97-104. https://doi.org/10.1016/j.conengprac.2018.09.003

Tatsumoto Y, Shiraishi M, Cai K, 2018b. Application of supervisory control theory with warehouse automation case study. *Syst Contr Inform*, 62(6):203-208.

Tractica, 2017. Warehousing and Logistics Robots: Global Market Analysis and Forecasts. https://www.tractica.com/research/warehousing-and-logistics-robots

Westernacher Knowledge Series, 2017. The Trend Towards Warehouse Automation. https://westernacher-consulting.com/wp-content/uploads/2017/11/Whitepaper_Trend_to_Automation_FINAL_s.pdf

Wonham WM, Cai K, 2019. Supervisory Control of Discrete-Event Systems. Springer, Cham, Switzerland.

Wonham WM, Ramadge PJ, 1987. On the supremal controllable sublanguage of a given language. *SIAM J Contr Optim*, 25(3):637-659. https://doi.org/10.1137/0325036

Wonham WM, Cai K, Rudie K, 2018. Supervisory control of discrete-event systems: a brief history. *Ann Rev Contr*, 45:250-256. https://doi.org/10.1016/j.arcontrol.2018.03.002

Wurman PR, D'Andrea R, Mountz M, 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Mag*, 29(1):9-19. https://doi.org/10.1609/aimag.v29i1.2082

Appendix: Standard supervisory control theory

Consider an automaton $\boldsymbol{G} := (Q, \Sigma, \delta, q_0, Q_m)$. The closed behavior of $\boldsymbol{G}$ is the language

$$L(\boldsymbol{G}) := \{s \in \Sigma^* \mid \delta(q_0, s)!\},$$

and the marked behavior of $\boldsymbol{G}$ is the sublanguage

$$L_{\mathrm{m}}(\boldsymbol{G}) := \{s \in L(\boldsymbol{G}) \mid \delta(q_0, s) \in Q_{\mathrm{m}}\} \subseteq L(\boldsymbol{G}).$$

$\boldsymbol{G}$ is nonblocking if $\overline{L_{\mathrm{m}}(\boldsymbol{G})} = L(\boldsymbol{G})$; namely, every string in the closed behavior may be completed to a string in the marked behavior. Since $\boldsymbol{G}$ may be

viewed as generating its closed and marked behaviors, it is also referred to as a generator.

Let $\boldsymbol{G}_k := (Q_k, \Sigma_k, \delta_k, q_{0,k}, Q_{\mathrm{m},k})$ $(k = 1, 2)$ be two generators. Their synchronous product $\boldsymbol{G}_1||\boldsymbol{G}_2 = \boldsymbol{G} = (Q, \Sigma, \delta, q_0, Q_{\mathrm{m}})$ is defined according to

$$
\begin{aligned}
&Q = Q_1 \times Q_2,\ \Sigma = \Sigma_1 \cup \Sigma_2,\\
&q_0 = (q_{0,1}, q_{0,2}),\ Q_{\mathrm{m}} = Q_{\mathrm{m},1} \times Q_{\mathrm{m},2},\\
&(\forall (q_1, q_2) \in Q_1 \times Q_2, \forall \sigma \in \Sigma_1 \cup \Sigma_2)\\
&\delta((q_1, q_2), \sigma)\\
&= \begin{cases}
(\delta_1(q_1, \sigma), q_2),\ \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2, \delta_1(q_1, \sigma)!,\\
(q_1, \delta_2(q_2, \sigma)),\ \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1, \delta_2(q_2, \sigma)!,\\
(\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)),\\
\qquad \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \delta_1(q_1, \sigma)!, \delta_2(q_2, \sigma)!,\\
\text{not defined, otherwise.}
\end{cases}
\end{aligned}
$$

The closed behavior of the synchronous product $\boldsymbol{G}$ is $L(\boldsymbol{G}) = L(\boldsymbol{G}_1)||L(\boldsymbol{G}_2)$, and the marked behavior is $L_{\mathrm{m}}(\boldsymbol{G}) = L_{\mathrm{m}}(\boldsymbol{G}_1)||L_{\mathrm{m}}(\boldsymbol{G}_2)$. For more than two generators, their synchronous product may be defined similarly.

Let $\boldsymbol{G} = ||_{k=1}^{n} \boldsymbol{G}_k$ be the plant model that comprises n components. For control purposes, let Σ be partitioned into a subset Σ_{c} of controllable events and a subset Σ_{u} of uncontrollable events, i.e., $\Sigma = \Sigma_{\mathrm{c}} \dot{\cup} \Sigma_{\mathrm{u}}$. Let Γ denote the set of all event subsets that always include Σ_{u}, i.e., $\Gamma := \{\gamma \subseteq \Sigma \mid \gamma \supseteq \Sigma_{\mathrm{u}}\}$; each $\gamma \in \Gamma$ is called a control pattern. A supervisory control V for $\boldsymbol{G}$ is any map $V : L(\boldsymbol{G}) \to \Gamma$, associating a control pattern to each string in $L(\boldsymbol{G})$. Write $V/\boldsymbol{G}$ for the closed-loop system where $\boldsymbol{G}$ is under the control of V. The closed language $L(V/\boldsymbol{G}) \subseteq L(\boldsymbol{G})$ is defined as follows:

(1) $\epsilon \in L(V/\boldsymbol{G})$;

(2) if $s \in L(V/\boldsymbol{G})$, $\sigma \in V(s)$, and $s\sigma \in L(\boldsymbol{G})$, then $s\sigma \in L(V/\boldsymbol{G})$;

(3) no other strings belong to $L(V/\boldsymbol{G})$.

Now let $M \subseteq L_{\mathrm{m}}(\boldsymbol{G})$; we say that V is a marking supervisory control for $(M, \boldsymbol{G})$ if $L_{\mathrm{m}}(V/\boldsymbol{G}) = L(V/\boldsymbol{G}) \cap M$. In addition, V is nonblocking if $\overline{L_{\mathrm{m}}(V/\boldsymbol{G})} = L(V/\boldsymbol{G})$.

Let $K \subseteq L_{\mathrm{m}}(\boldsymbol{G})$ be a specification language which represents constraints on the behavior of plant $\boldsymbol{G}$. The goal of supervisory control is to synthesize V such that $L_{\mathrm{m}}(V/\boldsymbol{G}) = K$. For this, controllability turns out to be key. We say that K is controllable with respect to $\boldsymbol{G}$ if

$$\overline{K}\Sigma_{\mathrm{u}} \cap L(\boldsymbol{G}) \subseteq \overline{K}.$$

The following is the main result of the supervisory control theory:

Theorem A1 Let $K \subseteq L_{\mathrm{m}}(\boldsymbol{G})$ and suppose $K \neq \varnothing$. Then there exists a marking nonblocking supervisory control V for $(K, \boldsymbol{G})$ such that $L_{\mathrm{m}}(V/\boldsymbol{G}) = K$ if and only if K is controllable.

Whether or not a specification language K is controllable, let $\mathcal{C}(K)$ denote the family of all controllable sublanguages of K; namely,

$$\mathcal{C}(K) := \{K' \subseteq K \mid K' \text{ is controllable}\}.$$

Since the empty language $\varnothing$ is trivially controllable, $\mathcal{C}(K)$ is nonempty. Moreover, controllability is closed under arbitrary set union: if K_i is controllable for all i's in some index set I, then the union $\cup\{K_i \mid i \in I\}$ is also controllable. Hence, $\mathcal{C}(K)$ contains a (unique) supremal element

$$\sup \mathcal{C}(K) := \cup\{K' | K' \in \mathcal{C}(K)\}.$$

Theorem A2 Let $K \subseteq L_{\mathrm{m}}(\boldsymbol{G})$ and $K_{\sup} := \sup \mathcal{C}(K)$. If $K_{\sup} \neq \varnothing$, then there exists a marking nonblocking supervisory control V for $(K_{\sup}, \boldsymbol{G})$ such that $L_{\mathrm{m}}(V/\boldsymbol{G}) = K_{\sup}$.

Standard (polynomial) algorithms and software, based on automata, are available for computing the supremal sublanguage $K_{\sup}$: a generator **SUP** is computed with $L_{\mathrm{m}}(\mathbf{SUP}) = K_{\sup}$ and $L(\mathbf{SUP}) = \overline{K_{\sup}}$. The generator **SUP** is called the optimal (i.e., maximally permissive) and nonblocking monolithic supervisor for the pair $(\boldsymbol{G}, K)$.