DEDS 26(4):583-610
2016, Springer

On-line compositional controller synthesis for AGV*

Johan Girault · Jean-Jacques Loiseau · Olivier H. Roux

Received: date / Accepted: date

Abstract This paper deals with the on-line design of a supervisor to coordinate an automated guided vehicle (AGV) fleet. This supervisor ensures the system safety (no collision) and a good coordination between vehicles (no blocking situations). It is the so-called Wonham-Ramadge supervisor, it is the least restrictive, and ensures controllability and nonblocking. We propose a compositional procedure to resolve this problem allowing an efficient on-line synthesis. A calculation on the fly is made at every attribution of a new mission for an AGV, to actualize the supervisor and adapt it to the new situation. This compositional approach allows to increase the number of AGV taken on compared to the *monolithic* approach. We show on some tests the efficiency of this method for the on-line synthesis of supervisor to coordinate a fleet of mobile robots for real cases.

Keywords finite automata · control · Automated Guided Vehicle · supervisory control · on-line synthesis

1 Introduction

Automated Guided Vehicle (AGV) systems are widely spread in automatic material handling systems and production workshops. Compared to other solutions, such as the use of treadmills, the main advantage is to increase the system's flexibility and its operation requires a more complex control system.

* This paper is an extended version of Girault et al (2013) and was selected by the conference MSR'13 for this submission to the journal DEDS.

Johan Girault · Jean-Jacques Loiseau · Olivier H. Roux
Ecole Centrale de Nantes, UMR CNRS 6597, IRCCyN, France
E-mail: {johan.girault, jean-jacques.loiseau, olivier-h.roux}@irccyn.ec-nantes.fr

Johan Girault
BA Systèmes, Mordelles, France
E-mail: johan.girault@basystemes.fr

When designing a system for automated guided vehicles, it is required to make a good conflict-free routing of AGVs and to dispatch tasks (Reveliotis. 2000). The design of AGV fleet control systems has been the subject of many academic studies. Many solutions are based on the scheduling theory (Maza. 2003; Breton et al. 2006) where the authors merge predictive scheduling techniques with on-line reroutings to take into account breakdowns or delay which may occur in the system.

An alternative approach rests on the supervisory control theory, which is based on the use of state-transition models, such as finite automata. The supervisory control theory has been developed since the seminal work about 30 years ago (Wonham and Ramadge. 1984; Ramadge and Wonham. 1987). It has become a basic paradigm for the control of discrete event systems (DES). It essentially allows to design a supervisor to prevent conflicts. In an AGV system, it amounts to avoid collisions between vehicles and to avoid blocking situations. It naturally applies to the modelling of AGV systems with finite automata, but a similar approach was proposed in Krogh and Holloway (1991) with Petri net modelling.

The supervisory control theory has a main advantage, it is maximally permissive. Therefore, it gives a maximum degree of freedom compatible with the avoidance of conflicts. This permissiveness allows the system to have high flexibility. The supervisor implemented in this paper is designed for operation in real time. Distributed control design for DES in the supervisory control theory (SCT) framework has been studied in Cai and Wonham (2010). The central problem investigated is how to synthesize local controllers for individual independent agents in the context of AGV systems. Another approach has been proposed in Arnaud et al (2009), where the objective function is to minimize the energy expenditure. The main problem is the state-space explosion problem which depends on the number of AGVs in the system. Indeed, in this paper, it is impossible to synthesize a supervisor for four or more AGVs. This problem occurs when the product of the components (AGVs) results in an automaton with a huge number of states.

Compositional or modular approaches can help to overcome the problem of state space explosion. The modular supervisory control offers significant gains in computational complexity (Wonham and Ramadge. 1988; Queiroz and Cury. 2000). In Hill and Tilbury (2006), language projections are used to simplify finite-state machines during synthesis and to construct modular supervisors. To ensure that nonblocking and maximal permissiveness are preserved, the observer property and output-control consistency are imposed as additional requirements on the projection. Another compositional synthesis approach is used (Flordal et al. 2007; Malik and Flordal. 2008; Mohajerani et al. 2011) to remove unnecessary information and to reduce the size of systems. In Flordal et al (2007) and Malik and Flordal (2008), the authors propose automata equivalence for supervisor synthesis, where the synthesis is considered in a nondeterministic setting and leads to some problems when interpreting result and ensuring maximal permissiveness. The supervisor returned may be an over-approximation of the least restrictive solution that is

not automatically nonblocking. These problems are overcome in Mohajerani et al (2011), where synthesis abstraction is used to abstract automata. The method requires all automata and their abstraction results to be deterministic, which makes some desirable abstraction impossible. Actually, the approach to perform synthesis in several steps as proposed in this paper is described as abstraction and called "halfway synthesis" in the papers cited here. In our case, we do not use any abstraction. We calculate the supervisor using an iterative method and the synthesized previous supervisor. The supervisor is recalculated on-line when a change arises on the configuration of the AGV system.

This paper proposes a solution to the problem of supervisor synthesis using a compositional/incremental method. The specification is expressed in the form of pairs of forbidden states and the supervisor is synthesized on the fly. On our examples, our method allows to reduce the state-space explosion problem, which is well known for the supervisor synthesis.

1.1 Motivation

In the industrial world, a circuit is designed to define the traffic lanes of AGVs. These lanes allow to access different critical storage areas as shown in Fig. 1. Lettered nodes represent *intersections* and each lane is divided into *sections*. There are four workstations (F, H, J and L) that represent where AGVs pick up or drop cargo. Nodes $\{A, C, T, Q\}$ represent the entrances of the area and nodes $\{B, D, S, R\}$ the exits of the area.

In the industrial environment, we can extend this critical area (Fig. 1) with twenty or more workstations. It then becomes very difficult to manage the traffic in this area to ensure the absence of conflict (see section 2.4) between AGVs. As the number of AGVs wishing to access the area increases, blocking situations are more likely to happen. A blocking situation on an industrial site can stop the production of the company from ten to twenty minutes, which is not admissible for a manufacturer.

1.2 Our contribution

In this paper, using a compositional incremental approach, we propose to design the supervisor representing the largest behaviour (or the least restrictive) which is nonblocking and controllable for an AGV system. More precisely, we aim to generate the supervisor and compose intermediate supervisors step by step adding an AGV at every step to build the final supervisor. The synthesized supervisor is the same as the *monolithic* supervisor of Ramadge and Wonham but its computation is more effective in terms of time and memory. We call it $modified\ monolithic$ supervisor.

The structure of this paper is as follows. Firstly, in section 2, we present the basic concept of the SCT and the synthesis of a supervisor for an AGV system

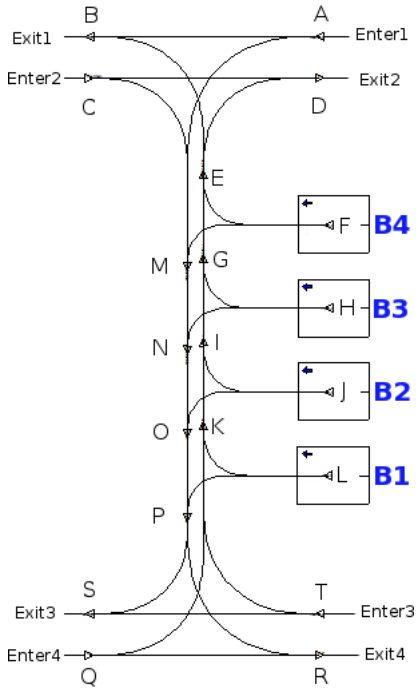


Fig. 1 An example of a storage area.

with forbidden states. In section 3, we propose a compositional approach to synthesize on-line a supervisor, we prove the compositionality and we discuss about the complexity. Finally, in section 4, this approach is applied to real cases from the industrial world.

2 Notations and preliminaries

This section presents notations and mathematical framework used in this paper. We will consider DES modelled by automata in the context of the SCT put forth by Ramadge and Wonham (Ramadge and Wonham. 1989; Cassandras and Lafortune. 2006).

2.1 Finite automata

A discrete event system is a dynamic system in which state changes occur at discrete points in time. They can be represented by an automaton denoted by G, which is defined as usually as a five-tuple.

Definition 1 $G = (Q, \Sigma, \delta, q_0, Q_m)$, where :

- Q is the finite set of states,
- Σ is the finite set of events,
- $\delta : Q \times \Sigma \to Q$ is the *transition function*,
- $q_0 \subseteq Q$ is the *initial* state set [1],
- $Q_m \subseteq Q$ is the set of *marked states* (final states).

In this paper, all automata are deterministic. Each $\sigma \in \Sigma$ is a label associated with an event. The occurrence of an event corresponds to the system changing by passing from one state to another. $\delta(x, \sigma) = y$ means that there is a transition labelled by event σ from state x to y. In general, δ is a partial function on its domain. We write $\delta(x, \sigma)!$ if $\exists y \in Q : \delta(x, \sigma) = y$. Let Σ^* denote the set of all finite strings s of elements of Σ, including the empty string, ϵ. For the sake of convenience, δ is extended from domain $Q \times \Sigma$ to domain $Q \times \Sigma^*$ in the following recursive manner:

$$\delta(x, \epsilon) = x$$
$$\delta(x, s\sigma) = \delta(\delta(x, s), \sigma) : s \in \Sigma^*, \sigma \in \Sigma \ .$$

A string $\sigma_1\sigma_2...\sigma_n \in \Sigma^*$ is often called a *word*. We call a *valid trace* a path from the initial state to a marked state ($\delta(q_0, s) = y$ where $s \in \Sigma^*$ and $y \in Q_m$). We denote $x \xrightarrow{s} y \Leftrightarrow \delta(x, s) = y$ where $x, y \in Q$, as a shorthand. Note that if the states set $Q = \emptyset$ is empty, then G is the *empty automaton*. The term "empty automaton" refers to an automaton whose state space is empty; an empty automaton necessarily generates and marks the empty set.

In the sequel, we denote $B \setminus A$, the relative complement of A in B where A and B are sets of states. Formally, $B \setminus A = \{x \in B : x \notin A\}$. In other words, the relative complement of A with respect to a set B is the set of elements in B but not in A. Then, the notation $\delta|_A$, means that we are restricting δ to the domain A. Formally, consider the previous automaton G, then we have $\delta|_A(x, \sigma) = y$ if $\delta(x, \sigma)!$ and $\delta(x, \sigma) = y$ where $x, y \in A$.

Definition 2 (Language generated and marked)
The language generated by an automaton G is denoted by $L(G)$.

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

The language marked by G is:

$$L_m(G) = \{s \in L(G) : \delta(q_0, s) \in Q_m\} \ .$$

The language $L(G)$ represents all the paths that can be followed along the automaton starting from the initial state, while the second language $L_m(G)$ is the subset of $L(G)$ (we indeed have $L_m(G) \subseteq L(G)$) corresponding to paths ending in a marked state and starting from the initial state. The marked language is also called the language *recognized* by the automaton. Hereinafter we denote that $L = L(G)$ and $L_m = L_m(G)$.

[1] In this paper, we consider automata with either a unique or an empty initial state, i.e., $q_0 \in Q$ or $q_0 = \emptyset$. Therefore, we denote $q_0 = x_0$ instead of $q_0 = \{x_0\}$ as a shorthand for the initial state.

Definition 3 (*Prefix-closure*)
Let $L \subseteq \Sigma^*$ a language, its prefix-closure denoted $\overline{L}$, is defined as

$$\overline{L} = \{s \in \Sigma^* : \exists t \in \Sigma^* (st \in L)\} .$$

In general, we have $L \subseteq \overline{L}$.

The notation $\overline{L}$ represents the set of all prefixes of strings in the language L. In the sequel, $|Q|$ denotes the number of states of the set Q and $|\delta|$ denotes the number of transitions of the considered automaton.

2.2 Operations on automata

To analyse DES modelled by automata, some operations on automata are used in this paper and presented in this section.

2.2.1 Accessible part of an automaton

The accessible part of an automaton is composed of all the states that are *accessible* or *reachable* from q_0 by some words in $L(G)$. When we "delete" a state, we also delete all the transitions that are attached to this state. We denote this operation by $Ac(G)$, where Ac stands for taking the "accessible" part (Cassandras and Lafortune. 2006). The $Ac(G)$ operation is defined as:

$$\begin{aligned} Ac(G) &= (Q_{ac}, \Sigma, \delta_{ac}, q_0, Q_{ac,m}) \\ Q_{ac} &= \{q \in Q : \exists s \in \Sigma^*, \delta(q_0, s) = q\} \\ Q_{ac,m} &= Q_m \cap Q_{ac} \\ \delta_{ac} &= \delta|_{Q_{ac}} \end{aligned}$$

Clearly, the Ac operation has no effect on $L(G)$ and $L_m(G)$. Thus, from now on, we will always assume, without loss of generality, that an automaton is *accessible*, that is, $G = Ac(G)$.

2.2.2 Coaccessible part of an automaton

A state q is said to be *coaccessible*, if there exists a path in the automaton G from state q to a marked state $q_m \in Q_m$. We denote the operation of deleting all the states of G that are not *coaccessible* by $CoAc(G)$, where $CoAc$ stands for taking the *coaccessible* part (Cassandras and Lafortune. 2006). This operation is defined as follows:

$$\begin{aligned} CoAc(G) &= (Q_{coac}, \Sigma, \delta_{coac}, q_{0,coac}, Q_m) \text{ where} \\ Q_{coac} \subseteq Q &= \{q \in Q : \exists s \in \Sigma^*, \delta(q, s) \in Q_m\} \\ q_{0,coac} &= \begin{cases} q_0 & \text{if } q_0 \in Q_{coac} \\ \emptyset & \text{otherwise} \end{cases} \\ \delta_{coac} &= \delta|_{Q_{coac}} \end{aligned}$$

If $G = CoAc(G)$, then G is said to be *coaccessible*.

2.2.3 Trim operation

An automaton which is both *accessible* and *coaccessible* is said to be *trim* (Cassandras and Lafortune. 2006). We define the $Trim$ operation by:

$$Trim(G) = CoAc[Ac(G)] = Ac[CoAc(G)]$$

where the commutativity of Ac and $CoAc$ is easily verified. An automaton is *nonblocking* if every state can reach a marked state. A *trim* automaton is *nonblocking*. Note that applying only the $CoAc$ operation is sufficient for an automaton to be *nonblocking*.

2.2.4 Composition of automata

A system may be composed of several subsystems. Each subsystem is represented by an automaton. We can design the whole process by the composition of all subsystems.

Definition 4 (Composition of automata)
Consider two automata $A = (Q_A, \Sigma_A, \delta_A, q_{0,A}, Q_{m,A})$ and $B = (Q_B, \Sigma_B, \delta_B, q_{0,B}, Q_{m,B})$ where Σ_A and Σ_B are disjoint. The composition of A and B is the automaton:

$$A \parallel B = (Q_A \times Q_B, \Sigma_A \cup \Sigma_B, \delta_{AB}, q_{0,AB}, Q_{m,A} \times Q_{m,B}) \ ,$$

where

- the initial state is $q_{0,AB} = q_{0,A} \times q_{0,B}$[2],
- for all $(x, y) \in Q_A \times Q_B$ and for all $(x', y') \in Q_A \times Q_B$, we have

$$\delta_{AB}((x, y), \sigma) = (x', y') \text{ iff } \begin{cases} y = y' \text{ and } \delta_A(x, \sigma) = x', \sigma \in \Sigma_A \\ x = x' \text{ and } \delta_B(y, \sigma) = y', \sigma \in \Sigma_B \end{cases}$$

This composition can be generalized to an arbitrary number of automata. It is well known that this operation is associative and commutative.

2.3 Supervisory control theory

Supervisory control theory (Wonham and Ramadge. 1984; Ramadge and Wonham. 1987; 1989) permits to design a supervisor from an automaton $M = (Q, \Sigma, \delta, q_0, Q_m)$ which represents the system. The set Σ is assumed to be separated into two distinct subsets: the set Σ_c of controllable events and the

[2] In this paper, we consider automata with either a unique or an empty initial state. Therefore, we denote $q_{0,AB} = (q_{0,A}, q_{0,B})$ as a shorthand for the initial state.

set Σ_u of uncontrollable events. This supervisor is an automaton representing the largest behaviour for the considered system and respecting the defined specifications. It generates the supremal controllable sublanguage, denoted in this paper $S(\cdot)$. Specifications can be defined by a list of forbidden states or a language. Let M be the system. Assume it is not satisfactory, then it must be modified by control. We have to construct a "supervisor" in order to alter the behaviour of M. We denote $S(M)$ the automaton that represents this supervisor. It generates a sublanguage of $L(M)$, that is $L(S(M))$. The sublanguage corresponding to paths which end in a marked state is denoted $L_m(S(M))$. Overall, we have the set inclusions:

$$\emptyset \subseteq L_m(S(M)) \subseteq \overline{L_m(S(M))} \subseteq L(S(M)) \subseteq L(M) \ .$$

For a better understanding of this article, we recall the concept of controllability defined in (Cassandras and Lafortune. 2006):

Definition 5 (Controllability)
Let K and $L = \overline{L}$ be languages over event set Σ. Let Σ_u be a designated subset of Σ. K is said to be *controllable* with respect to L and Σ_u if

$$\overline{K}\Sigma_u \cap L \subseteq \overline{K}$$

By definition, controllability is a property of the prefix-closure of a language. Thus K is controllable if and only if $\overline{K}$ is controllable. For this paper, we need a *nonblocking* supervisor, the definition is as follows:

Definition 6 (Nonblocking in controlled system)
The DES $S(M)$ is *nonblocking* if:

$$L(S(M)) = \overline{L_m(S(M))} \ .$$

We have to extend this definition if we have a set of uncontrollable events in M. To deal with uncontrollability, we recall the nonblocking controllability theorem (Cassandras and Lafortune. 2006) :

Theorem 1 (Nonblocking controllability theorem)
Consider the language $K \subseteq L_m(M)$ where $K \neq \emptyset$. There exists a nonblocking supervisor S for M such that:

$$L_m(S(M)) = K \text{ and } L(S(M)) = \overline{K}$$

if and only if the two following conditions hold:

- *Controllability:* $\overline{K}\Sigma_u \cap L(M) \subseteq \overline{K}$
- $L_m(M)$*-closure:* $K = \overline{K} \cap L_m(M)$

As it is well-known, the family of controllable and $L_m(M) - closed$ sublanguages of $L_m(M)$ is closed for the union of languages. As a consequence, if this family is non empty, it contains a supremal element that is a maximally permissive *nonblocking* supervisor. To build such a supervisor, simply apply the fixpoint procedure of Wonham and Ramadge (1984) (more details in section 2.4.4). Suppress the forbidden states (list of states) and keep the accessible and coaccessible part of the resulting automaton. If the system contains uncontrollable events, we remove also states that lead to a forbidden state through a series of uncontrollable events and apply the $Trim$ operation.

2.4 AGV fleet system

Nowadays, a common approach to ensure that the system is *nonblocking*, is to define the area like a zone of mutual exclusion which prevents any AGV from getting into the area if an AGV is already inside (only one AGV in the area). This method ensures to never stop the production process and in some cases it will even maintain an acceptable production rate. This is classical but restrictive, we propose to allow others AGVs to go inside the area. This permits to increase the production, but brings the risk of blocking situations. Using the supervisory control theory of Ramadge and Wonham, we guarantee no conflict and the most permissive behaviour. However, if there are too many AGVs, we can have a long computation time due to a combinatorial explosion (see section 3).

An AGV system is composed of a set of vehicles travelling on a circuit to accomplish various missions. Various tasks consist in delivering or picking-up manufactured part at different locations of the circuit. For the system to work correctly, the AGV fleet must be coordinated by a supervisor. The AGV fleet control uses several steps. Firstly, the system must assign different tasks to available AGVs that have no current job and are waiting for a mission. This function is not treated in this article, missions are assumed to be already dispatched. Then, the supervisor must define a path for each vehicle on the circuit, avoiding conflicts, to guarantee a secure behaviour of the production process. There are two types of conflicts :

- a collision, when two or more vehicles collide.
- a blocking situation, when there is no solution (no path, apart those leading to a collision) to achieve the mission. Fig. 2 illustrates a simple blocking situation between two AGVs.

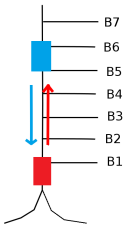


Fig. 2 Blocking situation between 2 AGVs

To optimize the system behaviour and allow a fluid circulation of AGVs, the supervisory control theory is a good solution since it will generate the most permissive supervisor.

2.4.1 Basic model

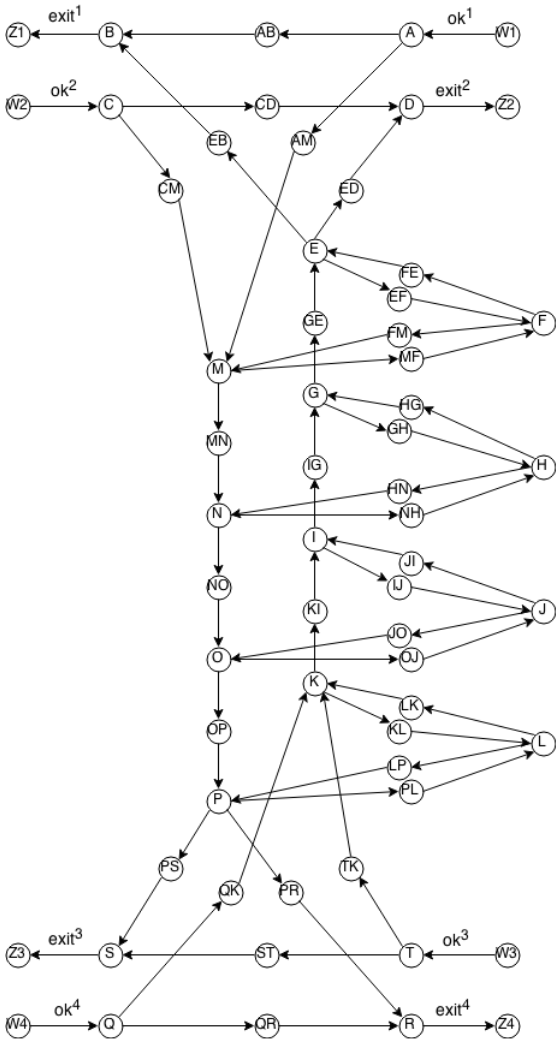


Fig. 3 One AGV's template

A model representing the movements of vehicles on the circuit is designed. Each AGV, which is an autonomous process in the overall process, is represented as a *template*. The circuit, according to its physical configuration, is partitioned into sections and intersections. Each time an AGV moves in a new intersection or in a new section, a new state in the template is occu-

pied, which reflects a change of the model's state. The template associated to the circuit shown in Fig. 1 is represented in Fig. 3. We denote this template $T = (Q_T, \Sigma_T, \delta_T)$, which can be seen as an automaton without initial and final states. It is the physical position of the AGV (resp. its mission) that will determine the initial state (resp. the final state).

For practical reasons, each state that represents a section in the template is named XY_i where X is the beginning of the section, Y the end and i the number of the AGV. For each section, we have two transitions, the first input transition C_{XY_i} is controllable and represents AGV_i moving on the section between X and Y from X to Y, and the second output transition U_{XY_i} is uncontrollable and represents the AGV arrived at the intersection Y from X. For example, if the location CM_1 is set, then the AGV_1 is moving on the section between C and M from C to M and we fired the transition C_{CM}. The transitions of the different AGVs are independent, they correspond to alphabets denoted Σ_i. Therefore, the different alphabets Σ_i are separated.

For each mission of an AGV, the template is instantiated as an automaton $(Q_T, \Sigma_T, \delta_T, q_0, Q_m)$ where we define the initial state and the final state set. All AGV instantiations are similar (same states and edges), except for the initial state q_0 that can be linked to any state according to AGVs positions on the circuit and the final state set Q_m which depends on their mission. An automaton which represents the AGV behaviour is an instance of the template. Each state in Fig. 3 represents the presence of an AGV on an intersection, a section or a workstation.

In the sequel, we consider a system M with n AGVs that run on the same circuit which is represented by a template as described previously. The automaton $G_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$ represents the automaton of AGV_i on the circuit where $q_{0,i}$ is the initial state (represents the starting position of the AGV on the circuit) and $Q_{m,i}$ the final state set (the goal of the AGV). Since the AGVs move on the same circuit, we have $(Q_i, \Sigma_i, \delta_i) = T$ for $\{1, \ldots, n\}$, and we can represent the global system as follows :

$$M = G_1 \parallel G_2 \parallel ... \parallel G_n \ .$$

A state of M is of the form:

$$(x_1, x_2, ..., x_n) \in Q_M \ .$$

where $x_i \in Q_T$, for $i = 1$ to n, is the position of AGV_i on the circuit and the positions x_i are arranged according to the arrival order of AGVs.

2.4.2 The waiting states and exit states

The *waiting states* and *exit states* of a template have a particularly meaning. We can see such a state as a parking with an adequate capacity for the

considered fleet. An AGV in such a state can not have any conflict with another AGV. In other words, all *waiting states* and *exit states* are safe, i.e., no collisions can appear with any AGVs (more details in the next section 2.4.3). These states are free-conflict states.

In the case of the circuit represented in Fig. 3, the entrances of the area are represented by the states $\{A, C, T, D\}$ whereas the exits are represented by the states $\{B, D, S, R\}$. The states $\{W1, W2, W3, W4\}$ represent the first physical point outside the area and before its associated entrance (e.g. the state $W1$ is associated to the entrance A). We name this set of states the *waiting states*, denoted $\mathbb{W}$, since an AGV which is in one of these states, is waiting for the authorization to go into the area. Therefore, we define a controllable event $ok_i^j \in \Sigma_c$. It means that the AGV_i has the authorization to go into the area and it is physically at the $Entrance_j$. In other words, it means that a supervisor was designed for all AGVs already inside the area and the new AGV. If no solution exists with the new AGV, then it will wait in this *waiting state* until a solution is found.

The states $\{Z1, Z2, Z3, Z4\}$ in Fig. 3, represent the first physical point outside the area and after its associated exit (e.g. the state $Z1$ is associated to the exit B). We name this set of states the *exit states*, denoted $\mathbb{E}$. Moreover, we define an uncontrollable event $exit_i^j \in \Sigma_u$ and that means the AGV_i has left the area by the $Exit_j$. If an AGV has left the area, the supervisor does not care anymore and has to remove it. For example, consider two AGVs where AGV_1 is in M and AGV_2 is in R (the system is in the state (M_1, R_2)). AGV_2 is leaving the area, when it will be in the exit state $Z4$, then the system will be in the state $(M_1, Z4_2)$. Therefore we can reduce this state to (M_1) since the AGV_2 is out of the area, i.e., AGV_2 was removed from the supervisor (more details in the section 3.4).

2.4.3 The specification

Our aim is to avoid blocking situations (see Fig. 2) and thus remove all paths that lead to a blocking situation and that do not permit to reach the final state.

The first step preliminary to the synthesis of a supervision system, consists in the determination of possible conflicts that lead to the specification. A supervisor, we have to express the specifications that must be satisfied. It consists in a list of forbidden states. Unlike most modular approaches, we have a single global specification and not a specification for local agents. This problem is dealt in Komenda et al (2008) for modular systems. The global system is $\bigcap_{i=1}^{n} P_i^{-1} L(G_i)$ (where P_i is the canonical projection $\bigcup_{i=1}^{n} \Sigma^* \to \Sigma_i^*$) that is a language over $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$, and we have only one specification $Spec \subseteq \Sigma^*$. Each AGV has only one alphabet. In our case, we want AGVs to be alone on a section or an intersection (beginning/end of a section). We need to delete

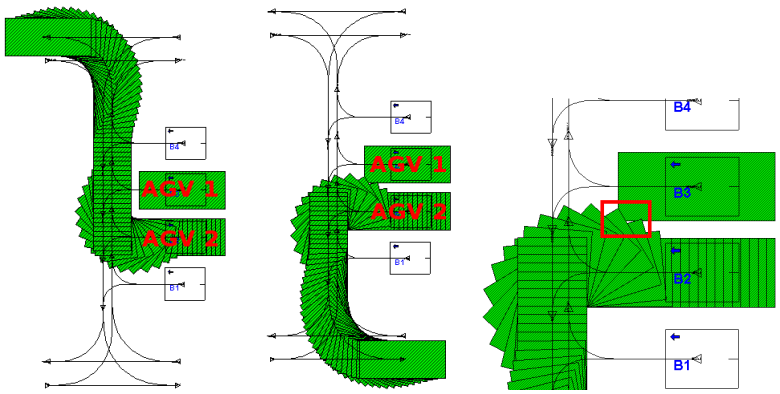
all states (and associated controllable transitions) $q = (x_1, x_2, ..., x_n)$ of the automaton M where:

$$\exists i, j \in \{1, \ldots, n\} : x_i = x_j \ .$$

We say that the states x_i and x_j are conflicting. Moreover, we say that two states are conflicting if there is a collision between both AGVs. Furthermore, the designer of the circuit has to apply a procedure to add the couple of states where a risk of collision exists. For example, consider the configuration shown in Fig. 4 where two AGVs are moving in a area (AGV_1 and AGV_2). In Fig. 4(a), there is no conflict, both AGVs are in their workstation and AGV_2 can get out without any problem. Its path is drawn by a series of rectangles representing the physical size of the AGV on the circuit, we note no collision between both AGVs. In contrast, in Fig. 4(b), AGV_2 must exit through the bottom of the area. By zooming the Fig. 4(b), we can see a collision between both AGVs in Fig. 4(c). In accordance with the template automaton T, the states LP and J are conflicting for example. The set of forbidden states $\mathcal{I}(T)$ is defined as follows:

$$\mathcal{I}(T) = \{(x_i, x_j) : x_i, x_j \text{ are conflicting}, (x_i, x_j) \in Q_T \times Q_T\}$$

$\mathcal{I}(T)$ is a list of pairs of states that are conflicting as described above. We generate this list of forbidden states from the template automaton T, the shape and the size of the AGVs.



(a) No conflict (b) Collision (c) Zoom on collision (b)

Fig. 4 Example of potential conflict

In the previous section, we talked about the safety of *waiting states*, denoted $\mathbb{W}$, and *exit states*, denoted $\mathbb{E}$, of a template automaton. In the sequel, we assume that all *waiting states* and *exit states* are safe. We can now formalize this statement as follows.

$$\forall (x_i, x_j) \in \mathcal{I}(T),\ x_i, x_j \notin \mathbb{E} \cup \mathbb{W}$$

where $\mathbb{E} \cup \mathbb{W} \subseteq Q_T$. In other words, a *waiting state* or an *exit state* can not be conflicting with any states of the template.

2.4.4 Synthesis of supervisor with forbidden states

Once we have the specification, we have to synthesize the supervisor. It can be calculated using the standard fixpoint algorithm of the supremal controllable sublanguage (Cassandras and Lafortune. 2006), usually denoted as the $\uparrow C$ operation in the literature.

Consider the previous automaton M, where $M = (Q, \Sigma, \delta, q_0, Q_m)$. Let $\Gamma : Q \to 2^\Sigma$ be the *active event function* where $\Gamma(q) = \{\sigma \in \Sigma \ : \ \delta(q, \sigma)!, q \in Q\}$. In words, $\Gamma(q)$ is the set of all events σ for which $\delta(q, \sigma)$ is defined. We denote Γ_M the active event function restricted to the automaton M. We present the Standard Algorithm for $S(\cdot)$ applied to M where the specification is a list of forbidden states in Algorithm 1.

Algorithm 1 Standard Algorithm for $S(\cdot)$

Step 0:
Let $M = (Q, \Sigma, \delta, q_0, Q_m)$
Set $\tilde{\mathcal{I}}(T) = \{(x_1, \ldots, x_n) \in Q : \exists i, j, (x_i, x_j) \in \mathcal{I}(T), i \neq j\}$
Set $Q_0 = Q \setminus \tilde{\mathcal{I}}(T)$
Set $S^0 = (Q_0, \Sigma, \delta_0, q_{0,0}, Q_{m,0})$ where $\delta_0 = \delta|_{Q_0}, q_{0,0} = q_0, Q_{m,0} = Q_m)$
Set $i = 0$
Step 1:

$$\begin{aligned} Q'_i &= \{q \in Q_i : \Gamma_M(q) \cap \Sigma_u \subseteq \Gamma_{S_i}(q)\} \\ \delta'_i &= \delta_i|_{Q'_i} \\ q'_{0,i} &= q_{0,i} \cap Q'i \\ Q'_{m,i} &= Q_{m,i} \cap Q'_i \end{aligned}$$

Step 2:

$$S^{i+1} = Trim(Q'_i, \Sigma, \delta'_i, q'_{0,i}, Q'_{m,i}) \ .$$

If S^{i+1} is the empty automaton, i.e., $q_{0,i}$ is deleted in the above calculation, then $S(M) = \emptyset$ and STOP.
Otherwise, set

$$S^{i+1} = (Q_{i+1}, \Sigma, \delta_{i+1}, q_{0,i+1}, Q_{m,i+1})$$

Step 3:
If $S^{i+1} = S^i$, then $S(M) = S^{i+1}$ and STOP.
Otherwise, set $i \leftarrow i + 1$, and go to Step 1.

All states removed during the operation are named "bad states" since they can lead to a conflict in the system. The initial Step 0 removes the forbidden states of the set $\mathcal{I}(T)$. The Step 1 removes all uncontrollable states due to controllability issues, i.e., all states that lead to a bad state via a sequence of uncontrollable events. The Step 2 removes all states due to nonblocking issues, i.e., all states that are not coaccessible (nonblocking issues) and not accessible

(unnecessary states). Moreover, it checks if we obtain the empty automaton (no solution). Finally, the Step 3 checks the end of the operation. Note that an empty solution (i.e. $S(M) = \emptyset$), means that there exists no supervisor able to coordinate the AGV fleet to ensure the absence of conflict. The necessary and sufficient condition for the existence of a supervisor (and for the admissibility of the considered AGV set) is that $S(M) \neq \emptyset$.

We note that $L(S(M)) = \overline{L_m(S(M))}$. For example, we consider a fleet of three AGVs moving on the circuit as shown in Fig. 3. This automaton has 54 states and 68 transitions. To obtain the automaton M which represents the overall system (all behaviours), we do the composition of the automata G_i, $i \in \{1; 2; 3\}$. The automaton M has $157,464$ (54^3) states and $594,864$ ($3 \times 54^2 \times 68$) transitions. The supervisor $S(M)$ ensures there is no conflict between AGVs and $S(M)$ is maximally permissive and controllable. If no solution exists, then $S(M) = \emptyset$.

3 Compositional approach and synthesis of supervisor

3.1 Basic idea

The main difficulty to design such a supervisor is the great number of states and the computation time. Consider the previous system M with n AGVs. The automaton M represents all the behaviours on the circuit, and it has $|Q|^n$ states and $n \times |Q|^{n-1} \times |\delta|$ transitions. We easily see that the complexity is exponential and depends on the number of AGVs. In many cases, if $n \leq 3$ there is no problem, but for $n \geq 4$ we can meet the state-space explosion problem or a long computation time. To reduce this problem, we use a compositional approach. When a new task is assigned to an AGV, we synthesize the new controller from the previous controller and thus we reduce the computational time. This approach reduces the intermediate state-space, it is maximally permissive and may be faster than the standard approach (*monolithic* approach). We synthesize a new supervisor step by step, where each step adds a new AGV and composes it with the previous supervisor.

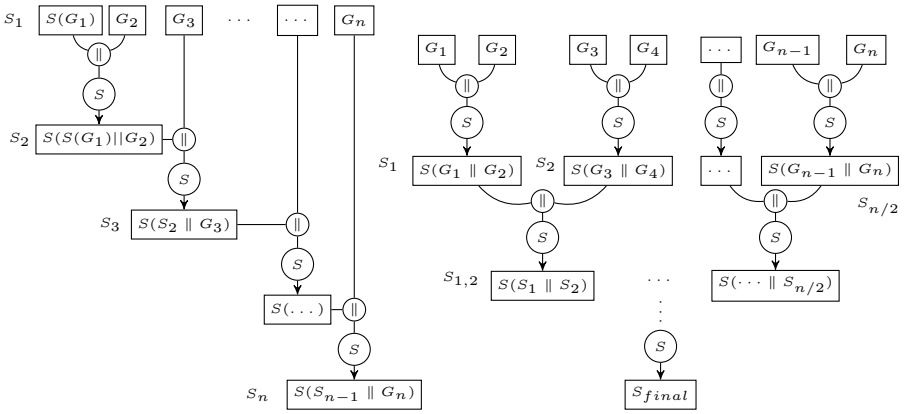
For example, consider a system with four AGVs modelled by G_1, G_2, G_3 and G_4. We have $M = G_1 \parallel G_2 \parallel G_3 \parallel G_4$. Let $S(M) \subseteq M$, the supervisor of M. We will demonstrate that:

$$\begin{aligned} S(M) &= S(S(S(S(G_1) \parallel G_2) \parallel G_3) \parallel G_4) \\ &= S(S(G_1 \parallel G_2) \parallel S(G_3 \parallel G_4)) \\ &= S(G_1 \parallel G_2 \parallel G_3 \parallel G_4)\ . \end{aligned}$$

3.2 Compositional synthesis

Consider the system M as described in section 2.4.1 with n AGVs and the automaton $G_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$ that represents the i'th *AGV* where $i \in$

$\{1, \ldots, n\}$. Let $S_G(M) = (Q_{S_G(M)}, \Sigma_{S_G(M)}, \delta_{S_G(M)}, q_{0,S_G(M)}, Q_{m,S_G(M)})$ be the supervisor obtained with the *monolithic* approach. Let $S_C(M) = (Q_{S_C(M)}, \Sigma_{S_C(M)}, \delta_{S_C(M)}, q_{0,S_C(M)}, Q_{m,S_C(M)})$ be the supervisor obtained with the compositional approach as described in Fig. 5(a). The synthesis algorithm is Algorithm 2.



(a) S_{k+1} from S_k (b) Synthesis in pairs

Fig. 5 Compositional synthesis

Algorithm 2 – Synthesis algorithm of S_C

$G \leftarrow \{G_1; G_2; \ldots; G_{n-1}; G_n\}$	*G is the set of AGVs' automata*
$S_C \leftarrow S(G_1)$	*Synthesis supervisor of* G_1
$G \leftarrow G \setminus \{G_1\}$	*Automaton* G_1 *of the set G is removed*
while $G \neq \emptyset$ **do**	
$x \leftarrow get(G)$	*get(G) return an element of G*
$S_C \leftarrow S(S_C \parallel x)$	*(the leftmost in the case of Fig. 5(a))*
$G \leftarrow G \setminus \{x\}$	
end while	

We obtain:

$$\begin{aligned} S_C(M) &= S(S(S(\ldots S(S(G_1) \parallel G_2) \parallel \ldots) \parallel G_{n-1}) \parallel G_n) \\ &= S(S(S(\ldots S(G_1 \parallel G_2) \parallel \ldots) \parallel G_{n-1}) \parallel G_n) \end{aligned}$$

The second equality is equivalent since $S(G_1)$ represents the supervisor for only one AGV, therefore, there is no conflict and $S(G_1) = G_1$.

Theorem 2 $S_G(M) = S_C(M)$.

Proof For this proof, we will denote $S_C(M) = S_C$ and $S_G(M) = S_G$ as shorthand. To prove this theorem, we have to demonstrate that S_C and S_G have

equal state sets, alphabets, transition relations, initial states, and marked state sets. By definition of the composition, S_C and S_G have the same initial state and the same alphabet. We will first prove that they have the same state set (potentially the empty set). By definition, the *monolithic* supervisor S_G is sound and complete. We will prove that the compositional supervisor S_C is sound (i.e. $Q_{S_C} \subseteq Q_{S_G}$) and complete too (i.e. $Q_{S_C} \supseteq Q_{S_G}$). We will denote $S_k = (Q_{S_k}, \Sigma_{S_k}, \delta_{S_k}, q_{0,S_k}, Q_{m,S_k})$ the compositional supervisor for the first k AGVs where $S_1 = S(G_1)$ and $S_{k+1} = S(S_k \parallel G_{k+1})$ (by definition of the compositional synthesis) and $k \in \{1, \ldots, n\}$. Note that the first supervisor S_1 is the supervisor for the first AGV in the area. Moreover, consider two automata A and B, we will denote $Q_{A\parallel B}$, the state set of the automaton $A \parallel B$. Let $k \in \{1, \ldots, n\}$ where n is the number of AGVs. We define the following projection $P_k : Q_M \longrightarrow Q_1 \times \cdots \times Q_k$:

$$P_k(x_1, \ldots, x_n) = (x_1, \ldots, x_k) \ .$$

where $P_k(q)$ represents the k first states of q, and consequently, the states of the k first AGVs.

First, consider the case where a G_i automaton is empty, then the monolithic supervisor is empty. We understand easily that $S_G = \emptyset$ and the compositionnal supervisor is empty too, $S_C = \emptyset$, therefore $S_C = S_G$. For the next of the proof, we consider that a G_i automaton is not empty and it is *trim*.

1. The soundness
   By definition of the synthesis of supervisor (see section 2.4.4), the compositional supervisor S_C is sound since it is the result of some synthesis operations $S(\cdot)$. Therefore, it is nonblocking and controllable, and it contains no forbidden states because it removes all bad states. Therefore we have the following inclusion: $Q_{S_C} \subseteq Q_{S_G}$.
2. The completeness
   Now, we need to prove that S_C is complete, i.e., $Q_{S_C} \supseteq Q_{S_G}$. We will prove this result by induction on the number of AGVs $k \in \{1, \ldots, n\}$. We have to show that for every state $q = (x_1, \ldots, x_n) \in Q_{S_G}$, the part of q $P_k(q) = (x_1, \ldots, x_k) \in Q_{S_k}$ (for the rest of the proof, we denote $q^k = P_k(q)$ as shorthand). In other words, for every state q of the *monolithic* supervisor S_G, the part q^k of q is a state of the compositional supervisor S_k thus every state q of S_G is a state of $S_n = S_C$. Formally, we have to prove:

   $$\forall q \in Q_{S_G}, \forall k \in \{1, \ldots, n\}, \ q^k \in Q_{S_k} \tag{1}$$

   Clearly, nothing is to be shown for $k = 1$ as $S_1 = G_1$ contains all the possible states x_1 as discussed before the Theorem 2. So, $q^1 = (x_1) \in Q_{S_1}$.

   Now, assume the claim (1) has been shown for some $k \geq 1$ (it means that for every state $q = (x_1, \ldots, x_n)$ of S_G, it holds that $q^k = (x_1, \ldots, x_k)$ is a state of S_k). Then, we have to show that this holds true for $k = k + 1$, and so, $q^{k+1} \in Q_{S_{k+1}}$.

By definition, $x_{k+1} \in Q_{G_{k+1}}$. So, $q^{k+1} = (x_1, \ldots, x_{k+1}) \in Q_{S_k \| G_{k+1}}$. Therefore, it remains to be shown that this state survives the synthesis operation $S(S_k \parallel G_{k+1})$. It will be shown that for every state $q \in Q_{S_G}$, the part q^{k+1} is contained in every iteration of the synthesis fixpoint algorithm $S^l(S_k \parallel G_{k+1})$ (for $l \geq 0$), thus for every state q, q^{k+1} is contained in $S(S_k \parallel G_{k+1})$. We will prove the following formula by induction on l.

$$\forall q \in Q_{S_G}, \forall l \in \mathbb{N}, q^{k+1} \in Q_{S^l(S_k \| G_{k+1})} \tag{2}$$

First, we prove this for $l = 0$. It is clear that $q^{k+1} \in Q_{S^0(S_k \| G_{k+1})}$ since the state $q \in Q_{S_G}$ is a state of the *monolihic* supervisor and therefore, it is not a forbidden state. We recall that forbidden states are only removed during the Step 0 of the synthesis fixpoint algorithm (see section 2.4.4). As the state q is not a forbidden state, thus none of the pairs (x_i, x_j) are conflicting, i.e., $\forall x_i, x_j \in q, (x_i, x_j) \notin \mathcal{I}(T)$. So, the part q^{k+1} also is not a forbidden state.

Now assume the claim (2) has been shown for some $l \geq 0$ (i.e. every state $q = (x_1, \ldots, x_n) \in Q_{S_G}$, the part $q^{k+1} = (x_1, \ldots, x_{k+1})$ is a state of $S^l(S_k \parallel G_{k+1})$). Then, we have to show that this holds true for $l = l+1$, and so, $q^{k+1} \in Q_{S_{k+1}}$.
To show that q^{k+1} is contained in $S^{l+1}(S_k \parallel G_{k+1})$, we have to show that the state is never removed during this synthesis step. There is only two ways to remove a state during the synthesis: (i) due to the controllability (Step 1 of the synthesis algorithm, section 2.4.4), if a state may lead to a bad state via an uncontrollable event, (ii) due to the nonblocking issues (Step 2 of the synthesis algorithm, section 2.4.4), if a state is not coaccessible.

(i) For controllability, assume $\delta_{S_{k+1}}(q^{k+1}, u) = (y_1, \ldots, y_{k+1})$ for some $u \in \Sigma_u$ such that $\delta_{S_{k+1}}(q^{k+1}, u)!$. Then, we have $\delta_{S_G}(q, u) = (y_1, \ldots, y_{k+1}, x_{k+2}, \ldots, x_n)$ since the G_i do not share events. As $q = (x_1, \ldots, x_n) \in Q_{S_G}$, and S_G is controllable, it follows that $(y_1, \ldots, y_{k+1}, x_{k+2}, \ldots, x_n) \in Q_{S_G}$. By inductive assumption, the state $(y_1, \ldots, y_{k+1}) \in Q_{S^l(S_k \| G_{k+1})}$.

As this holds for all $u \in \Sigma_u$, by definition of synthesis (section 2.4.4), this means that $q^{k+1} \in Q_{S^{l+1}(S_k \| G_{k+1})}$ (denoted Q' in the step 1 of the Algorithm 1) as far as controllability is concerned.

(ii) For nonblocking, we recall that $q \in Q_{S_G}$, S_G is nonblocking. Therefore, there exists a path in S_G from $q = (x_1, \ldots, x_n)$ to a marked state:

$$(x_1^0, \ldots, x_n^0) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1) \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_p} (x_1^p, \ldots, x_n^p)$$

where $(x_1^0, \ldots, x_n^0) = q$, all the states $(x_1^r, \ldots, x_n^r) \in Q_{S_G}$ with $0 \leq r \leq p$, and the end state is a marked state, i.e., $(x_1^p, \ldots, x_n^p) \in Q_{m,S_G}$. By inductive assumption, we have the parts $(x_1^r, \ldots, x_{k+1}^r) \in Q_{S^l(S_k \| G_{k+1})}$. Since the G_i do not share events, the above path can be reduced by removing events that are not in $\Sigma_1 \cup \cdots \cup \Sigma_{k+1}$ to get a path from

$q^{k+1} = (x_1, \ldots, x_{k+1})$ to a marked state:

$$(x_1, \ldots, x_{k+1}) \xrightarrow{\tilde{\sigma}_{i_1}} (x_1^1, \ldots, x_{k+1}^1) \xrightarrow{\tilde{\sigma}_{i_2}} \ldots \xrightarrow{\tilde{\sigma}_{i_p}} (x_1^p, \ldots, x_{k+1}^p)$$

where $0 \leq i_1 \leq \cdots \leq i_p \leq p$, the end state is a marked state in $G_1 \parallel \cdots \parallel G_{k+1}$ and the notation "˜" means "restricted to the automaton $G_1 \parallel \cdots \parallel G_{k+1}$". As all the states on the above path are in $S^l(S_k \parallel G_{k+1})$, it follows by definition of synthesis that $q^{k+1} \in Q_{S^{l+1}(S_k \| G_{k+1})}$ as far as nonblocking is concerned.

Thus, the state q^{k+1} is not removed due to controllability nor due to nonblocking, so by definition of synthesis, the state $q^{k+1} = (x_1, \ldots, x_{k+1}) \in Q_{S^{l+1}(S_k \| G_{k+1})}$.

This proves the inclusion $Q_{S_C} \supseteq Q_{S_G}$, and therefore, since the reverse inclusion was already verified, S_C and S_G have exactly the same set of states. In other words, $Q_{S_G} = Q_{S_C}$ and S_C is complete.

We now prove that S_G and S_C have the same transitions. Let $q = (x_1, ..., x_n)$ and $q' = (x'_1, ..., x'_n)$ be two states of $Q_{S_G} = Q_{S_C}$. By definition of the composition (Definition 4), we have a transition between q and q' if and only if $\exists i \in \{1, \ldots, n\}$, $\exists \sigma \in \Sigma_i$ such that $\delta_{G_i}(x_i, \sigma) = x'_i$ and $\forall j \in \{1, \ldots, n\},\ j \neq i \Rightarrow x_j = x'_j$. Hence we have $\delta_{S_C}(q, \sigma) = \delta_{S_G}(q, \sigma) = q'$. Therefore $\delta_{S_C} = \delta_{S_G}$. We have finally checked the equalities: $Q_{S_C} = Q_{S_G}, \Sigma_{S_C} = \Sigma_{S_G}, \delta_{S_C} = \delta_{S_G}, q_{0,S_C} = q_{0,S_G}, Q_{m,S_C} = Q_{m,S_G}$ that ends the proof.

□

Let us denote $\otimes$ the operator of supervisor synthesis for two automata: $G_i \otimes G_j = S(G_i \parallel G_j)$. This operator is commutative, since the composition operation is commutative too:

$$G_1 \otimes G_2 \simeq G_2 \otimes G_1 \ .$$

To obtain the equality $G_1 \otimes G_2 = G_2 \otimes G_1$ we need to normalize the name of the states of the automata where we have to arrange the x_i elements in ascending order. For example, if $q_1 = (x_1, x_2)$ is a state of $G_1 \otimes G_2$ then $q_2 = (x_2, x_1)$ is its corresponding state in the product $G_2 \otimes G_1$. These two states represent exactly the same situation where AGV_1 is in x_1 and AGV_2 is in x_2. By sorting AGV numbers in ascending order, q_1 and q_2 can be written as (x_1, x_2). Therefore, the operator $\otimes$ is commutative :

$$G_1 \otimes G_2 = G_2 \otimes G_1 \ .$$

The operator $\otimes$ is associative. We obtain that:

$$\begin{aligned} S(G_1 \parallel G_2 \parallel G_3) &= S(S(G_1 \parallel G_2) \parallel G_3) \\ &= S(G_1 \parallel S(G_2 \parallel G_3)) \end{aligned}$$

which shows that:

$$(G_1 \otimes G_2) \otimes G_3 = G_1 \otimes (G_2 \otimes G_3)$$

As a consequence, the compositional synthesis method can be applied for every order chosen to compose subsystems. Every binary tree generated from the set of AGVs leads to the definition of a compositional synthesis resulting in a supervisor that actually coincides with the *monolithic* supervisor $S_G(M)$.

For instance, let us consider the alternative compositional synthesis, illustrated in Figure 5(b). The figure must be viewed as a tree that should be as balanced as possible. We note $S_{AC}(M)$ the supervisor synthesized with alternative compositional approach. We have $S_{AC}(M) = S(S(\dots S(S(G_1) \parallel G_2) \parallel \dots S(S(G_{n-3} \parallel G_{n-2}) \parallel S(G_{n-1} \parallel G_n))\dots)$. Thus, we have the following property.

Corollary 1 $S_G(M) = S_{AC}(M)$.

As said in the introduction, our method is a modified *monolithic* supervisor because it generates the same language as the standard *monolithic* supervisor. Every compositional supervisor is actually equal to $S_G(M)$, so it is maximally permissive. This is particularly the case of $S_C(M)$ and $S_{AC}(M)$. In the case of an AGV system, one needs to have a centralized supervisor but the computation is "modular". It is necessary to have only one supervisor because AGVs move on the same physical part of a circuit, therefore a centralized supervisor allows to avoid conflicts between AGVs. The on-line application of this method is its main advantage (because a new AGV may arrive at any time in the critical area) without having to rebuild the supervisor from scratch (as described in section 3.2).

3.3 On-line supervisor synthesis

The proposed compositional approach offers three advantages. The first one is a reduction of the intermediate state-space (necessary during the various steps of synthesis) and thus (in our experiments) the computation time. The second one is the possibility to design intermediate supervisors in any order. It always produces the maximally permissive and controllable supervisor. The construction order may be designed to minimize the total computation time, which goes beyond the aim of the present paper.

The main advantage of this approach is to be especially suitable for real-time systems. Indeed, there is no need to recompute the supervisor if a new AGV arrives in the area. Changing the initial state of the current supervisor is enough in order to maintain its new accessible and coaccessible parts and to combine it with the automaton of the new AGV. The new initial state is the state in which the system is located at the time of introduction of the new AGV. Thus the computation gain may be significant according to the progress in which the system is. If the first AGVs have performed a large part of their mission, then the computation time will be significantly reduced, and vice versa. Then the computation time depends on the progress of various

missions. This property is essential for a real-time system. This approach allows to design a supervisor for a large number of AGVs whereas with a standard approach, we cannot model such a supervisor for five or more AGVs. We have to define a new function to change the initial state of an automaton:

Definition 7 (NIS - New Initial State)
Let $A = (Q, \Sigma, \delta, q_0, Q_m)$ be an automaton. Let q_0' be the desired new initial state of A.

$$NIS(A, q_0') = (Q, \Sigma, \delta, q_0', Q_m)$$

In other words, $NIS(A, q_0')$ just changes the initial state of A from q_0 to q_0'.

For example, consider a critical area with n AGVs, it is assumed that n AVGs are moving in the area and they execute instructions sent by the supervisor S_n (corresponding supervisor for n AGVs) and a new AGV_{n+1} arrives (goes into the area). Let G_{n+1} be the automaton for AGV_{n+1}. Now, we need to change the initial state of S_n. Let q_c be the current state of S_n when $AGVn + 1$ arrives, i.e., the current state of the system. The on-line synthesis of the new supervisor is:

$$S_{n+1} = S(Trim(NIS(S_n, q_c)) \parallel G_{n+1})$$

This supervisor is the compositional supervisor S_c where q_c is the new initial state. We can easily see the usefulness of the $Trim$ operation to keep only accessible and coaccessible states. As shown in the Theorem 2, this supervisor is nonblocking, controllable and maximally permissive.

3.4 Entrance and exit of an area

In this section, we discuss about entrances and exits of an area. First, we will discuss about the meaning of a *waiting state*. Next, we will discuss about the meaning of an *exit state*.

When an AGV is in a *waiting state*, the AGV is outside the area. It is waiting to go into the area to accomplish its mission. If an AGV has no way to leave this state, it means that the system has not found a solution without conflict which authorizes it to go into the area, i.e., the supervisor synthesized with this new AGV is empty and the system uses its old supervisor. Therefore, an AGV which is in a *waiting state* has to wait in this place until a nonblocking and conflict free supervisor is found. Once such a supervisor is synthesized, the *AGV* has the authorization to go into the area and the system can fire the *ok* transition as discussed in the section 2.4.1.

When an AGV has exited the area, the supervisor has fired an *exit* transition and the AGV is in an *exit state*. As we said previously, an AGV which is in an *exit state* is out of the area and we need to remove all elements of this AGV from the supervisor states. The first step preliminary to the deletion is

to apply the NIS and $Trim$ operation to keep only the new accessible part of the system. Consider the system M described in the section 2.4.1 which is supervised by the supervisor $S(M)$. Let r be the AGV number which has exited the area, where $1 \leq r \leq n$. Then, AGV_r is in an *exit state* $x_r \in \mathbb{E}$ and the system is in a state $\mathbf{x}$ which is of the form:

$$(x_1, \ldots, x_r, \ldots, x_n) \ .$$

We apply the NIS operation where $\mathbf{x}$ is the new initial state. Then, we have the new supervisor $S'(M) = Trim(NIS(S(M), \mathbf{x}))$. For the second step we have to remove all elements of AGV_r from all states of $S'(M)$. Let $q \in Q_{S'(M)}$ be a state of the new supervisor. After the NIS and $Trim$ operation, no transition can be fired for AGV_r since an *exit state* has no output transition. Therefore, we have to apply Algorithm 3. The AGV number r has been removed from the supervisor, and then the system contains $n-1$ AGVs in the area.

Algorithm 3 Deletion of AGV_r in a supervisor $S'(M)$

for all $q = (x_1, \ldots, x_r, \ldots, x_n) \in Q_{S'(M)}$ **do**
  $q = (x_1, \ldots, x_{r-1}, x_{r+1}, \ldots, x_n)$
end for

3.5 A simple example

We present, in this section, a simple example to illustrate the results of the previous section. The "template" automaton T is shown in Fig. 6.

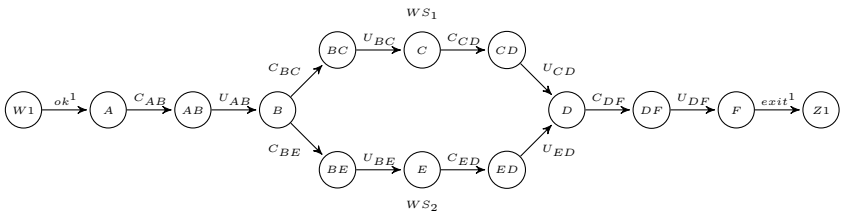


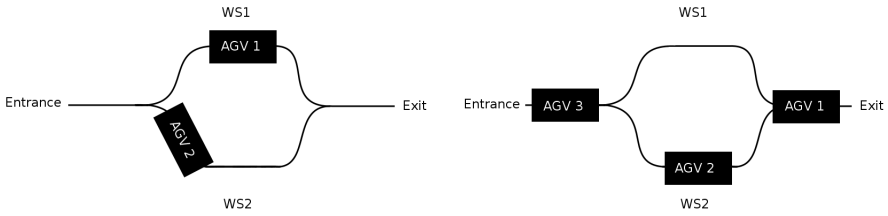
Fig. 6 "Template" automaton T

Two AGVs are moving on the area, AGV_1 has to go to workstation WS_1 and AGV_2 has to go to workstation WS_2 (Fig. 7(a)). Let G_1 be the automaton of AGV_1 and G_2 be the automaton of AGV_2. Let $S_2 = S(G_1 \parallel G_2)$. We recall that when an AGV is in an *exit state* ($Z1$ in this case) then the AGV has left the area. Therefore it is removed from the supervisor and the system has $n-1$ AGVs in the area. The AGV_3 arrives when AGV_1 is in the state D and AGV_2 is in WS_2. The AGV_3 has to go to WS_2. Let G_3 be the automaton of

AGV_3. If we apply the compositional approach described previously, then we have to compose the automaton G_3 of AGV_3 to the supervisor S_2 of AGV_1 and AGV_2 with the new initial state of S_2. Let us assume that the new initial state denoted q, where $q = (D, E) : D \in Q_1, E \in Q_2$ and $q \in Q_1 \times Q_2$. Therefore the new supervisor for the three AGVs is as follows:

$$S_3 = S(Trim(NIS(S_2, q)) \parallel G_3)$$

where $Trim(NIS(S_2, q))$ is shown in Fig. 8. The Table 1 shows the states of the automaton of the Fig. 8. The list of forbidden states is generated as described on section 2.4.4. We note that the following pairs of states are forbidden states: $\{(BC, BE), (CD, ED)\} \subset \mathcal{I}(T)$.



(a) 2 AGVs moving (b) AGV_3 arrives

Fig. 7 Straightforward example

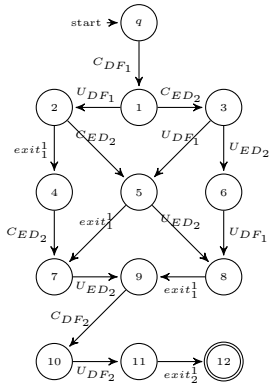


Fig. 8 $Trim(NIS(S_2, q))$

q	(D_1, E_2)
1	(DF_1, E_2)
2	(F_1, E_2)
3	(DF_1, ED_2)
4	$(Z1_1, E_2)$
5	(F_1, ED_2)
6	(DF_1, D_2)
7	$(Z1_1, ED_2)$
8	(F_1, D_2)
9	$(Z1_1, D_2)$
10	$(Z1_1, DF_2)$
11	$(Z1_1, F_2)$
12	$(Z1_1, Z1_2)$

Table 1 States of Fig. 8

The automaton $Trim(NIS(S_2, q))$ has 13 states and 16 transitions. At this step, we do not need to recalculate the composition with $Trim(NIS(G_1, q_1))$ and $Trim(NIS(G_2, q_2))$, as in the *monolithic* approach, where $q_1 \in Q_1$ (resp. $q_2 \in Q_2$) is the new initial state of G_1 (resp. G_2). Formally, for the *monolithic* approach, we have $Trim(NIS(G_1, q_1)) \parallel Trim(NIS(G_2, q_2))$ and the result of this operation is an automaton with 24 states and 38 transitions. At this step, we can clearly see the gain for the intermediate space used. The compositional method uses the previous supervisor and previous forbidden states that have already been removed whereas the *monolithic* approach rebuilds the previous forbidden states that were removed. The supervisor shown in Fig. 8, is the supervisor when the third AGV arrives at the entrance of the area and it does not have the authorization to go into the area yet (S_3 not synthesized). Then, we have to compose it with G_3 (the automaton of the AGV_3) and synthesize to obtain the final supervisor S_3.

3.6 Complexity of monolithic and compositional controller synthesis for AGV

This section estimates the complexity of our on-line compositional synthesis and compares it with the monolithic one. We describe the complexity in terms of generator M and a sublanguage K of $L(M)$. M has m states and K has n states. Consider the problem of synthesizing a supervisor so that the closed loop behaviour is a prescribed language K. In our case K is not controllable and $K \subseteq L(M)$, therefore the computation of $K\uparrow$ (the largest controllable sublanguage) is of time complexity $O(m^2n^2)$ and can be computed in polynomial time (Ramadge and Wonham. 1989). In the case of this paper, the forbidden states are not represented by a separate automaton K but only by a list, therefore the complexity does not get multiplied by n^2. If each G_i has k states, then M has k^p states since $M = \bigparallel_{i=1}^{p} G_i$. Therefore the complexity to synthesize a system of p AGVs is $O(k^{2p})$. It depends on the number of AGVs and on the size of the system. For the compositional approach, we perform $p-1$ times the operation to synthesize the supervisor S ($K^{\uparrow}$). Then we can decompose the complexity for p AGVs as follows: $O(k^{2\times2}) + O(k^{3\times2}) + \cdots + O(k^{2p}) = O(k^{2p})$. Then the compositional approach does not increase the theoretical complexity to synthesize a supervisor.

In practice we may have a large gain in terms of time and intermediate space. It depends on the topology of the circuit and therefore on the number of forbidden and bad states. If there are many forbidden states, the gain will be important. Indeed, in this case, the size of intermediate supervisors will be significantly reduced. If a new AGV arrives, numbered $p + 1$, as we said previously, we set the initial state of S_p, apply the $Trim$ operation to S and synthesize the new supervisor S_{p+1} from S_p. Assume that $Trim(NIS(S_p, q))$ (where q is the new initial state) has m' states and G_{p+1} has k states. The complexity for the compositional approach for the step $p+1$ is $O(km'^2)$ where

$m' \leq m$ because the $Trim$ operation cannot increase the size of an automaton. Moreover, m' may be much lower than m, it depends on the progress of the system. Let $M = \prod_{i=1}^{p} NIS(G_i, q_i)$ be the composition of the first p AGVs in the *monolithic* approach where q_i is the new initial state of G_i. Assume that M has m'' states, therefore we have $m'' \geq m'$. The complexity at the step $p+1$ for the *monolithic* approach is $O(km''^2)$ where m'' may be much larger than m'. For the *monolithic* approach, m'' is the result of p composition operations whereas, for the proposed approach, we have only one composition operation. This operation may be very costly because it directly depends on the number of AGVs and it is exponential. We can clearly see the gain for an on-line synthesis when a new AGV has to go into the area.

For the compositional approach, some forbidden and bad states are never built and therefore some paths generated with the *monolithic* approach will never be generated in our approach. As we said previously, the synthesis is associative, then the order of generation of intermediate supervisors is not important but the execution time and intermediate space may depend on this order. For example, if an AGV_1 and an AGV_2 have many forbidden states then it is preferable to generate this supervisor first. Of course, this order should respect the arrival order of the vehicles and in this case, we can not choose the order.

3.7 Optimization for successive missions.

To optimize the paths of AGVs, it is essential to know where the AGV will go after completing its mission. We consider in this section that we know the exit of the AGV which often happens. For example, when the mission of an AGV is to move an object from one point to another outside the area. If we treat this problem by considering successively two separate missions for the AGV (one is ending in the area and the other one finishes outside the area), depending on the physical configuration of the circuit, it is possible to encounter situations where two AGVs have completed their first mission, one of the AGV is forced to retreat to let out another. Therefore, the path of the AGV is not optimal.

To solve this problem, it is no longer a question to find a supervisor to ensure reachability for a final state but it is a more complex problem. Let G_T be the template automaton. Let q_0 be the initial state. Let q_1 be the state to be reached for its first mission, and q_2, the exit state of the second mission. The automaton G, corresponding to the possible movements of the AGV, is an instance of the template G_T with the initial state q_0 and the final state q_2. The supervisor must keep only all paths without blocking from the initial state q_0 to the state q_1 and from q_1 to the final state q_2. To formalize this problem we express this specification in the temporal logic called *computation tree logic*

(CTL) (Clarke et al. 1986). The system must guarantee the CTL formula:

$$(AF\ q_1)\ and\ (AG\ (q_1 \Rightarrow AF\ q_2))\ . \tag{3}$$

In other words, $AF\ q_1$ means for all paths, we reach the state q_1 and $AG\ (q_1 \Rightarrow AF\ q_2)$ is the classic response property that means starting from state q_1 system necessarily reaches the state q_2. The implementation of the supervisor must guarantee the previous formula. However, a theoretical maximally permissive model of a supervisor allows loops in the model and then, our supervisor only guarantees the following property:

$$AG\ (EF\ q_1\ or\ q_1 \Rightarrow AG\ EF\ q_2)$$

Since with the supervisory control theory we know effectively how to deal with a reachability problem, we decompose this complex problem into two simple reachability problems. The method used is to consider two separate missions that we sequentialize before computing the supervisor. The construction will ensure the reachability of the objective state of the first mission (q_1) provided that the objective of the second mission (q_2) is reached. The synthesis is as follows: we create two instances of the template $G_T = (Q_T, \Sigma_T, \delta_T)$ that we name G^1 and G^2. The initial state of G^1 is q_0 and its final state is q_1. The initial state of G^2 is q_1 and its final state is q_2. Thus, we have $G^1 = (Q_T, \Sigma_T, \delta_T, q_0, \{q_1\})$ and $G^2 = (Q_T, \Sigma_T, \delta_T, q_1, \{q_2\})$. We perform chaining by applying composition restricted on the set of states $Q_\oplus$ as follows:

$$G = (Q_\oplus, \Sigma_T, \delta_\oplus, (q_0, q_1), \{(q_1, q_2)\}) = G^1 \oplus G^2\ ,$$

where

$$Q_\oplus \subseteq Q_T \times Q_T = \{(q, q') \in Q_T \times Q_T \text{ such as } q = q_1 \text{ or } q' = q_1\}\ ,$$

and

$$\delta_\oplus((x, y), \sigma) = \begin{cases} (x', y) & \text{, if } \delta_T(x, \sigma) = x' \text{ and } (x', y) \in Q_\oplus\ , \\ (x, y') & \text{, if } \delta_T(y, \sigma) = y' \text{ and } (x, y') \in Q_\oplus\ . \end{cases}$$

Theorem 3 *The supervisor $S(G^1 \oplus G^2)$ ensures the properties $(AG\ (EF\ (q_1, q_1)\ or\ (q_1, q_1) \Rightarrow AG\ EF\ (q_1, q_2))$ and it is maximally permissive.*

Proof First, the state (q_2, q_1), if it exists in $Q_\oplus$, is not final because only (q_1, q_2) is the final state. The construction $G^1 \oplus G^2$ ensures that if a path leads to the state (q_1, q_2) then this path goes through the state (q_1, q_1). The supervisor $S(G^1 \oplus G^2)$ by definition ensures the property $AG\ EF\ (q_1, q_2)$ and therefore, by construction of $G^1 \oplus G^2$, for $i \neq 1$, $AG\ (q_i, q_1) \Rightarrow AG\ EF\ (q_1, q_1)$ and $AG\ (q_1, q_1) \Rightarrow AG\ EF\ (q_1, q_2)$ and so the properties $(AG\ (EF\ (q_1, q_1)\ or\ (q_1, q_1)$ $\Rightarrow AG\ EF\ (q_1, q_2))$. Furthermore, any path in G^1 between q_0 and q_1 exists in $G^1 \oplus G^2$ between (q_0, q_1) and (q_1, q_1), and any path in G^2 between q_1 and q_2 exists in $G^1 \oplus G^2$ between (q_1, q_1) and (q_1, q_2). The computation of S retains all safe paths and therefore the supervisor $S(G^1 \oplus G^2)$ is maximally permissive. □

From this maximally permissive supervisor, a strategy to obtain an implementation satisfying the formula (3), classically consists in avoiding loop in the automaton. Another way that we will study in further works to solve this problem consists in adding some criteria such as the energy consumption hence avoiding infinite loop.

4 Experimental results

The compositional approach presented in this paper has been implemented in C++ and has been used on a standard PC with a single core (3.2GHz) and 8Gb of memory (RAM). We studied four different real circuits from the industrial world and more precisely in the company BA Systèmes (France). However, the circuits 1 and 4 are the most significant because the configuration is often found in an industrial environment. The four configurations are different in terms of potential conflicts.

For the first circuit, we tested our approach on the area presented in Fig. 1. The circuit structure is not restrictive because interleaving is large whatever the system's state of progress. Indeed, looking at Fig. 3, we can access most states from any state of the automaton, which may cause a combinatorial explosion. For the second, the circuit respects the LIFO model. The first AGV to arrive will be the last to leave the area, therefore interleaving is limited. In this model, we have a lot of forbidden states and our method should be effective. For the third case, it is similar to the simple example presented in the section 3.5. In this configuration, there are few forbidden states and the compositional supervisor should be less effective than the first two configurations. Finally, in the fourth configuration, the template authorizes to go into reverse everywhere whereas as we can see in Fig. 3 it is not possible. For example, if an AGV comes from CM it cannot directly go to EB. In the industrial world, the interdiction to go into reverse is often a security requirement because a human who needs to go to the site must be able to predict the direction of an AGV which is moving. Therefore this configuration is less realistic than the circuit 1 but the combinatory explosion is the biggest. However, this model contains 10 workstations so it is more realistic in terms of number of workstations.

We present in Table 2, the maximum memory requested and the time used to synthesize the supervisors for the previous cited cases. The memory requested represents the maximum number of states and transitions generated during the synthesis. It is important to recall that our compositional approach provides exactly the same supervisor as the *monolithic* supervisor which is the most permissive.

1st circuit: In this example, we consider four AGVs and the fourth arrives when the first three are performing their missions, and a second case with a fifth AGV. All AGVs have two tasks, one going to the area and the second out-

Configuration		Monolithic			Compositional		
Circuit	AGVs	States	Trans.	Time	States	Trans.	Time
1	4	332,856	1,443,504	44 s	127,068	372,645	3.3 s
1	5	explosion	-	-	1,686,462	2,529,639	42 s
2	4	782,130	4,000,492	49 s	19,538	43,171	0.3 s
2	5	explosion	-	-	57,366	71,192	0.9 s
2	6	explosion	-	-	122,538	324,620	3.5 s
3	5	27,720	120,031	1 s	24,920	107,341	0.4 s
3	6	194,040	1,006,537	10.1 s	174,440	900,907	3.4 s
4	3	15,372	47,854	1.1 s	12,646	36,182	0.2 s
4	4	937,692	4,148,854	180 s	560,023	2,113,323	19 s

Table 2 Experiments results. AGVs is the number of AGVs. The States, Trans. columns represent the maximum number of states and transitions generated during the synthesis and the time column is the time to synthesize the supervisor.

side the area. AGV_1 (resp. AGV_2, AGV_3, AGV_4) starts on $Entrance_1$, (resp. $Entrance_2$, $Entrance_3$, $Entrance_4$), has to go to the workstation B_1 (resp. B_4, B_2, B_3), and leaves out by $Exit_1$ (resp. $Exit_2$, $Exit_3$, $Exit_4$). For this example, we first consider three AGVs are performing their missions and a fourth arrives. AGV_1 completed its first mission and is leaving the area (EB), AGV_2 starts its second mission (K) and AGV_3 is performing its first task (KI). The results are shown in Table 2. Next, we add a fifth AGV which arrives on $Entrance_2$ and has to go to the workstation $B1$. For this AGV we don't know its second mission.
Note that the compositional approach is approximately thirteen times faster in this case, which provides an interesting time saver. This is explained by the reasons mentioned above, i.e., the reduction of the intermediate state space. These results confirm the explanation provided in section 3.3. Note that the compositional supervisor requires roughly four times less memory than the *monolithic* approach whereas for five AGVs the *monolithic* synthesis explodes, i.e, the synthesis needs more than 8Gb of memory.

2nd circuit: The second case is also a realistic case, but it should be more efficient. Indeed, the automaton *template* has 30 states and 42 transitions but its main feature is to respect the LIFO model, there is only one exit from the area. This circuit is shown in Fig. 9. This type of configuration is often present in industrial sites with limited space. If two or more AGVs go into the area, the first will be the last to leave because it will be blocked by the others (an AGV on the workstation blocks the central lane of the area). There is no solution except to wait for the others AGVs having evacuated the area. For example, in Fig. 9, AGV_2 has to go to the workstation W_6 but it is impossible since AGV_1 blocks the central lane. AGV_2 has to wait at the entrance of the area until AGV_1 leaves the area. We consider in this case four AGVs and the fourth

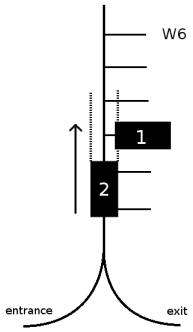


Fig. 9 An area which respects the LIFO model

arrives when the first three are performing their missions. AGV_1 achieved its first mission (0 % of its second mission), AGV_2 performed 80 % of its first mission and AGV_3 70 % of its first mission. The results are shown in Table 2. We also test for five and six AGVs with the following configuration: an AGV has completed its first mission, two are at 90 % and two at 50 % of their first mission, and the last AGV arrives at the entrance of the area.
With this circuit structure, the compositional approach is very efficient (almost 163 times faster). This is explained by the number of states that are forbidden caused by the LIFO model. Intermediate supervisors help to maintain a system with few states while the standard approach will calculate many forbidden states and will remove them very late. We note that for five or six AGVs, the *monolithic* approach explodes whereas our approach allows to generate the supervisor in $4.2s$. We also note very significant gains in terms of memory space as we can see in the Table 2.

3rd circuit: For the third example, the circuit used is similar to the simple example. It contains six workstations and we can have conflicts only on two places in the circuit (for example, the states B and D of the Fig. 6). Therefore the list of forbidden states is very small. The configuration is as follows: AGV_1 achieved its first mission (50%), AGV_2 (resp. AGV_3, AGV_4, AGV_5) is achieving its first mission (45%, resp. 20%, 25%, 25%), and the AGV_6 arrives at the entrance of the area.
The template used has 46 states and 50 transitions but G_i automata are very small because each workstation has a unique path to reach it. Therefore an automaton G_i has a maximum of 11 states and 10 transitions, that is why the *monolithic* approach can generate the supervisor for six AGVs in this case. As we can see in the Table 2, the space required for the *monolithic* approach is near to the space required for the compositional approach because the list of forbidden states is small. However, the compositional approach is still faster

than the *monolithic* one.

4th circuit: In the 4th case, three AGVs are moving on the area and a fourth arrives. The template automaton contains 58 states and 76 transitions. The configuration is as follows: AGV_1 is leaving the area (95% of its mission has performed), AGV_2 achieved its first mission (50%, it is in a workstation), AGV_3 is performing its first mission (15%) and the AGV_4 arrives at the entrance of the area.
As explained previously, this circuit causes a large combinatorial explosion and that is mainly why the *monolithic* approach is very slow. As we can see in the Table 2, the standard approach has twice more transitions and states than the compositional approach. Therefore, the compositional approach is faster than the standard approach.

5 Conclusion

In this paper, we propose a compositional approach (modified *monolithic* supervisor) to synthesize an on-line supervisor to manage a fleet of mobile robots. We prove compositionality of supervisor so that the supervisor is the same as the *monolithic* approach of Ramadge and Wonham. This method has several advantages. First, it is more efficient than the standard approach, and according to the case it allows to synthesize a supervisor for four or more AGVs in a critical area, which is not possible with the *monolithic* approach. But the on-line application in a real time context is the biggest advantage for this method as explained in previous sections. Indeed, in an industrial environment, a mission is often generated in real time, we cannot know it in advance. Therefore, it is essential to be able to generate a supervisor when a new task appears in its critical areas.

This paper focuses only on critical areas, if we extend to the non-critical areas (lack of interaction), we could use local controllers. This will be the subject of a future study.

In our further work, we would like to add a notion of weight on transitions to choose the best path in real time according to some criteria, such as energy or time and we will implement this approach in company BA Systèmes (France).

References

Arnaud Y, Cury JER, Loiseau JJ, Martinez C (2009) Pilotage sûr et optimal d'une flotte de véhicules autoguidés. In: JD-JN-MACS 2009, Angers : France

Breton L, Maza S, Castagna P (2006) A multi-agent based conflict-free routing approach of bi-directional automated guided vehicles. In: American Control Conference, 2006

Cai K, Wonham WM (2010) Supervisor localization: A top-down approach to distributed control of discrete-event systems. Automatic Control, IEEE Transactions on 55(3):605–618, DOI 10.1109/TAC.2009.2039237

Cassandras CG, Lafortune S (2006) Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA

Clarke EM, Emerson EA, Sistla AP (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst 8(2):244–263, DOI 10.1145/5397.5399

Flordal H, Malik R, Fabian M, Åkesson K (2007) Compositional synthesis of maximally permissive supervisors using supervision equivalence. Discrete Event Dynamic Systems 17(4):475–504, DOI 10.1007/s10626-007-0018-z

Girault J, Loiseau JJ, Roux OH (2013) Synthèse en ligne de superviseur compositionnel pour flotte de robots mobiles. In: European Journal of Automation, MSR'13, vol 47/1-3, pp 195–210

Hill RC, Tilbury DM (2006) Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In: Discrete Event Systems, 2006 8th International Workshop on, pp 399–406, DOI 10.1109/WODES.2006.382507

Komenda J, Van Schuppen J, Gaudin B, Marchand H (2008) Supervisory control of modular systems with global specification languages. Automatica 44:1127–1134, DOI 10.1016/j.automatica.2007.09.004

Krogh BH, Holloway LE (1991) Synthesis of feedback control logic for discrete manufacturing systems. Automatica 27(4):641–651

Malik R, Flordal H (2008) Yet another approach to compositional synthesis of discrete event systems. In: Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on, pp 16–21, DOI 10.1109/WODES.2008.4605916

Maza S (2003) Analyse du comportement d'un système de transport par chariot bidirectionnels en vue de sa commande. PhD thesis, Université de Nantes

Mohajerani S, Malik R, Ware S, Fabian M (2011) Compositional synthesis of discrete event systems using synthesis abstraction. In: Control and Decision Conference (CCDC), 2011 Chinese, pp 1549–1554, DOI 10.1109/CCDC.2011.5968439

Queiroz MHD, Cury JER (2000) Modular supervisory control of large scale discrete event systems. In: International Workshop on Discrete Event Systems: Analysis and Control. Proc. WODES'00, Kluwer Academic, pp 103–110

Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230

Ramadge PJ, Wonham WM (1989) The control of discrete event systems. Proceedings of the IEEE 77(1):81–98

Reveliotis SA (2000) Conflict resolution in agv systems. IIE Transactions 32:200–0

Wonham WM, Ramadge PJ (1984) On the supremal controllable sublanguage of a given language. In: Decision and Control, 1984. The 23rd IEEE Conference on, vol 23, pp 1073–1080

Wonham WM, Ramadge PJ (1988) Modular supervisory control of discrete-event systems. Mathematics of Control, Signals and Systems 1(1):13–30