Deadlock Detection in the Scheduling of Last-Mile Transportation Using Model Checking

Koji Hasebe* hasebe@cs.tsukuba.ac.jp
Mitsuaki Tsuji† mits@osss.cs.tsukuba.ac.jp
Kazuhiko Kato* kato@cs.tsukuba.ac.jp

*Department of Computer Science, University of Tsukuba
†College of Information Science, University of Tsukuba
1-1-1 Tennodai, Tsukuba 305-8573, Japan

***Abstract*—We propose a formal verification method for deadlock detection in the scheduling of transportation systems in which vehicles run in fleets. Especially, as a prime example, we here consider the last-mile transportation system based on autonomous vehicles that we are currently developing. One of the major features of our transportation system is the ability for vehicles to run in a row without physical connecting points, which makes it possible to smoothly reorganize the vehicles. Meanwhile, owing to the rules for rearranging fleets and the traveling route of vehicles, the system may fall into a deadlock state where no vehicle can proceed to the next stop. To address this issue, we propose in this paper a method of detecting the possibility of a deadlock in a given operation schedule. For this objective, we use a model checking method to search all possible states of the system. We also show how to avoid deadlock by modifying specification based on the well-known idea of Coffman's deadlock prevention. Finally, we demonstrate the usefulness of the proposed method using examples.**

I. Introduction

A next-generation transportation system called a *last-mile mobility* system is beginning to be used as a supplement to conventional transportation systems. This system is a compact mobile transportation system operating at short to medium distances, such as those between the stations of conventional main transportation systems (e.g., train, airplane, and bus systems), and has departure points and destinations within relatively small areas. Typical examples are ParkShuttle [1] running near Rotterdam in the Netherlands and CityMobil2 [2] in La Rochelle, France.

Last-mile transportation systems offer a convenient transportation means especially for elderly people and people with disabilities because a stop can be finely moved to where there is passenger demand. Meanwhile, the transport network of such a system tends to be complicated, and there thus remains the problem that the speed of passenger transport is lost by the transfer operation occurring during the movement process.

To improve this situation, the authors are developing a last-mile public transportation system based on technologies of semi-autonomous driving (See also [8] for the overview of our research project.) Our vehicles consist of two types, those that lead a fleet and are driven manually and those that are driverless and follow another vehicle. One of the major features of our system is the ability for vehicles to run in a fleet without physical connecting points, which makes it possible to smoothly reorganize the vehicles. With this feature, even if passengers with different destinations get on the same fleet of vehicles, by appropriately detaching or reorganizing vehicles at branch points of the route, the passengers can reach their destinations without transferring between fleets.

Additionally, by installing a central server to consolidate the demand of passengers and periodically setting the operation schedule according to the demand, it is possible to realize efficient passenger transportation. However, when vehicles travel according to the determined schedule, some vehicles may concentrate at some stop points, resulting in a deadlock state (i.e., a state in which no vehicle can move to the next stop of the driving route). When arranging a schedule, therefore, it is necessary to verify in advance that there is no possibility of a deadlock. Meanwhile, in such a transportation system, vehicles run in parallel and rules such as the organization pattern of vehicles become complicated, while the number of combinations of states of the system becomes large. It is thus difficult to manually verify all behaviors in the system.

To address this issue, in this paper we present a method of detecting the possibility of deadlock for a given schedule by means of model checking. As a verification tool, in this study we use SPIN [9], one of the most popular model checkers for verifying the correctness of distributed systems. For this objective, we first give a model of the target system in the modeling language called Promela, and then explain the assumptions to be made and the properties to be verified. In particular, we discuss the assumption of fairness of each process (i.e., behavior of a vehicle) and explain how to describe fairness in the model. We next demonstrate how to avoid deadlock by modifying specifications when the possibility of a deadlock is found for a given model. This technique is based on the well-known idea of deadlock prevention [4]. Finally, we also demonstrate the usefulness of our method and discuss limitations due to state explosion using examples.

Even if we limit our attention to the formal verification of the transportation system, there are a number of methods for verifying safety-related properties of various systems in the literature. In particular, there have been many attempts at verifying the safety of railroads, which are close to our verification target. However, the issue of deadlock detection in a system where vehicles form fleets and reorganize frequently, as in our case, has not been thoroughly investigated. This issue is expected to become more important as automatic

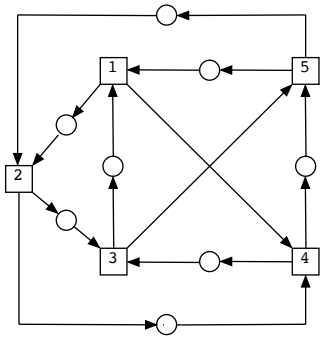

Fig. 1. Example of a traffic network (Example 1).

driving techniques develop in the future. Moreover, with the development of logistics, our method can be applied not only to passenger transport but also to various other transportation systems, such as logistics systems in a factory and product delivery systems.

The paper is organized as follows. Sections II and III give an overview and the formal model of our target system, respectively. Section IV explains our verification method. Section V demonstrates deadlock detection and prevention using an example. Section VI presents related work. Section VII concludes the paper and discusses future research.

II. Overview of the Target System

This section outlines the last-mile transportation system targeted in this research.

A. Vehicles

Our transportation system consists of vehicles that are able to run in a row without having physical contact points. (See also [8] for a more detailed explanation of the system.)

The vehicles are classified according to their function into two types.

- *Lead vehicles*: vehicles driven manually at the head of a fleet of vehicles.
- *Trailing vehicles* (or *trailers*, for short): driverless vehicles that run autonomously behind another vehicle.

Vehicles can be arbitrarily connected through electronic control to a fleet as long as the fleet capacity (i.e., the maximum number of vehicles that can be connected to a lead vehicle) is not exceeded.

B. Traffic network

In this transportation system, a *traffic network*, consisting of routes on which the vehicles can move and some stops (i.e., boarding and alighting points), is determined. A specific example of the traffic network is depicted in Fig. 1. (Throughout this paper, this example is called *Example 1*.) In the figure, the routes the vehicles can travel are represented by arrows while stops are represented by circles and squares. Here, a stop represented by a circle is a regular stop whereas a stop represented by a square has branches to multiple routes or is where a plurality of routes join. The latter type of stop is called a *node*.

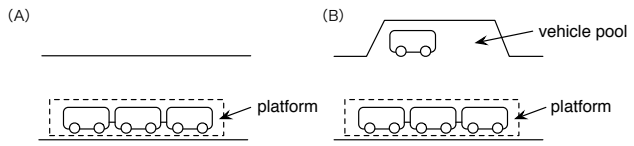

Fig. 2. Forms of a regular stop (A) and node (B).

TABLE I
Example of a schedule for the traffic network of Example 1.

HA	$1 \to 2 \to 3 \to 1$
HB	$3 \to 5 \to 1 \to 4 \to 3$
HC	$4 \to 5 \to 2 \to 4$
T1	$1 \to 2 \to 3 \to 1$
T2	$1 \to 2 \to 3 \to 5 \to 1$
T3	$3 \to 5 \to 2 \to 3$
T4	$3 \to 5 \to 1 \to 2 \to 3$
T5	$4 \to 5 \to 2 \to 4$
T6	$4 \to 3 \to 5 \to 1 \to 4$
T7	$2 \to 3 \to 5 \to 2$
T8	$5 \to 1 \to 2 \to 3 \to 5$

See Fig. 2 for the forms of a regular stop and node. The operations of detaching vehicles from a fleet and connecting new vehicles to a fleet are performed at nodes. For this purpose, a node provides not only a temporary stopping space for getting passengers on and off but also a parking space, called a *vehicle pool*, for keeping detached vehicles until the next time they are used. For each node, there is an upper limit on the number of vehicles that can stop at this stopping space and the vehicle pool and, as explained later, new vehicles exceeding this upper limit cannot enter the node.

C. Traveling route of a vehicle

For our transportation system, we assume there is a central server that aggregates the demand of passengers and periodically determines the traveling routes (i.e., operation schedule) of vehicles. An example of the traveling routes of vehicles in the traffic network of Example 1 is presented in Table I, where HA, HB, and HC represent lead vehicles while T1,...,T8 represent trailers. (Throughout the paper, this schedule is also used as the example to explain our method and it is often referred to as the *schedule of Example 1*.) In the table, the initial position of each vehicle is indicated as the starting point of the path of the route. To show the current traveling route to passengers, each lead vehicle and trailer has a destination board.

Here we note that the routes in this example are set so that passengers can be transported between any two stops without transfer. For example, if a passenger wishes to move from Node 2 to 4, she only has to board vehicle HC or T5. Furthermore, as seen in this example, routes are set according to the current demand of passengers for each node. For example, seven vehicles including Node 2 in their routes while

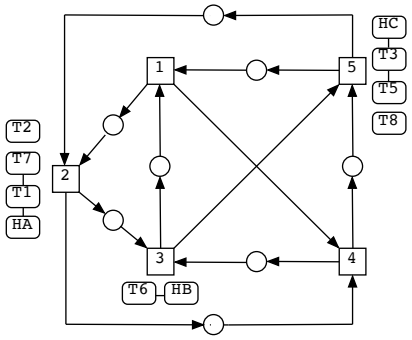

Fig. 3. A deadlock situation in Example 1.

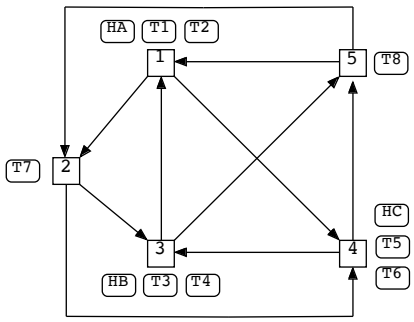

Fig. 4. Abstracted traffic network of Example 1.

only two vehicles include Node 4. This assumes a situation that the demand for passengers is high at Node 2 and low at Node 4.

D. Reorganization of vehicles

Vehicles autonomously form a new fleet at each node by the following procedure. When a fleet arrives at a node, among the trailers of this fleet and in the vehicle pool, the trailers whose next destination coincides with the destination of the lead vehicle are considered. Next, among these trailers, we select trailers that do not exceed the fleet capacity in descending order of the waiting time. However, owing to the space limitation, if the fleet cannot enter the next node, that fleet must wait at the current node.

E. Possibility of a deadlock

In the system described above, vehicles may be unevenly distributed in the traffic network depending on the traveling speed of vehicles. Furthermore, when this deviation becomes large, some vehicles stay at a stop and the system may eventually fall into a deadlock situation, where no fleet can move to the next stop because the parking space of the next stop is full.

An example of the deadlock situation is shown in Fig. 3. Here, the square drawn beside each node indicates the vehicles located there. In addition, squares connected by lines represent vehicles forming a fleet. For example, vehicles located at Node 5 are HC, T3, T5, and T8, among which HC, T3, and T5 form a fleet. Moreover, we assume that not more than five vehicles can enter a node at a time owing to the space limitation. In this setting, as a fleet is located at Nodes 2, 3, and 5 at the same time, no fleet can move to the next node.

III. Formal Model of the Transportation System

This section gives formal specifications of our transportation system. The Promela model shall be presented in the next section.

A. Vehicles

Let V be the set of vehicles and $V_L, V_T \subseteq V$ be the sets of lead vehicles and trailers, respectively, satisfying $V = V_L \cup V_T$ and $V_L \cap V_T = \emptyset$. We define a function $cap\_fleet : V_L \to \mathbb{N}$ that returns the maximum number of vehicles of the fleet containing a given lead vehicle.

B. Traffic network

As shown in Fig. 1, a traffic network can be formally represented as a graph $G = \langle S, E \rangle$, where the vertex and edge sets S and E are respectively the sets of stops and the routes for the vehicles to move. The sets $S_R, S_N \subseteq S$ are respectively the sets of regular stops and nodes. In addition, functions determining the maximum numbers of vehicles that can enter at each node at a time and park in the vehicle pool are defined as $max\_enter : S_N \to \mathbb{N}$ and $cap\_pool : S_N \to \mathbb{N}$, respectively.

C. Travel route

A travel route for a vehicle can be described as a sequence $e_1, \ldots, e_k \in E$. Here we define a function $pass : S_N \to 2^V$ that represents the set of vehicles passing through the nodes. Additionally, we define a function $route : V \to 2^{S_N}$ as $route(v) = \{s \in S_N \mid v = pass(s)\}$. As mentioned above, the travel route of each vehicle in our system should be set so that the passengers can move between any two stops without transfer. That is, for any $s, s' \in S_N$ with $s \neq s'$, there exists $v \in V$ such that $s, s' \in route(v)$. Clearly, the schedule of Example 1 satisfies this condition.

D. Model abstraction

In verification using model checking, model abstraction is often effective as a means of reducing the number of states in a target model. Also for our system, model abstraction is required because a state explosion occurs when modeling and verifying the system of Example 1. In the deadlock detection of our transportation system, because vehicles are not reorganized at regular stops, we omit them in the model. Therefore, the traffic network of Example 1 is abstracted as shown in Fig. 4, and we only consider this abstracted system in the remainder of the paper.

IV. Deadlock Detection by Model Checking

This section explains our deadlock detection method using Example 1. We use the SPIN model checker as a verification tool in this paper.

```
proctype HeadA()
{
    do
    ::(turnHA == true) ->
        cnctTrailer(HA);
        tran(HA);
        dcnctTrailer(HA);
        turnHA = false;
    od
}
```

Fig. 5. Description of the behavior of a vehicle as a process.

```
proctype Select()
{
    do
    ::(turnHA == false)
        && (turnHB == false)
            && (turnHC == false) ->

        canTranHA = false;
        canTranHB = false;
        canTranHC = false;

        canTran(HA);
        canTran(HB);
        canTran(HC);

        if
        ::(canTranHA == true)
                -> turnHA = true
        ::(canTranHB == true)
                -> turnHB = true
        ::(canTranHC == true)
                -> turnHC = true
        fi;
    od
}
```

Fig. 6. Description of the process "`Select()`" that controls the execution of processes of vehicles.

A. Specifications in Promela

We explain how to describe the system defined in Section III with Promela.

In our model, the behavior of each lead vehicle is described as a process, and the vehicle moves to the next node when the process is executed. As an example, the process of the lead vehicle HA written in Promela is shown in Fig. 5.

In addition, the process of each lead vehicle is controlled by the process `Select()` that satisfies the fairness properties described in the next subsection. Process `Select()` is defined in Fig. 6. The process `Select()` examines whether each lead vehicle can be moved to the next node by the function `canTran()`. For example, if the lead vehicle HA can move, the variable `canTranHA` will be true, and the variable `turnHA` will be true if the move is permitted. The process `Select()` nondeterministically selects the lead vehicle that actually moves from the set of movable vehicles.

```
inline cnctTrailer(head)
{
    mtype t;

    do
    ::(head == HA) &&
        (currp_HA == h1) &&
        (len(h1_HA) > 0) &&
        (len(trailers_HA) < MAX_LINK) ->
            h1_HA?t; trailers_HA!t
    ::(head == HA) &&
        (currp_HA == h2) &&
        (len(h2_HA) > 0) &&
        (len(trailers_HA) < MAX_LINK) ->
            h2_HA?t; trailers_HA!t
    ::(head == HA) &&
        (currp_HA == h3) &&
        (len(h3_HA) > 0) &&
        (len(trailers_HA) < MAX_LINK) ->
            h3_HA?t; trailers_HA!t
    ::(head == HB) ... -> ...
    ::(head == HC) ... -> ...
    ::else -> break
    od
}
```

Fig. 7. Specifications of the reorganization of vehicles.

The lead vehicle selected by `Select()` moves to the next node. Then, by the function `cnctTrailer()`, the lead vehicle disconnects the current training vehicles and chooses vehicles in descending order of passengers' waiting time. Fig. 7 shows the Promela code of function `cnctTrailer()` that defines the operation of the reorganization of a new fleet. Here, the information on the position of each vehicle is treated using a channel. The channel manages the order of vehicles at a node in descending order of waiting time by message passing.

B. Fairness constraints

Modeling the behaviors of lead vehicles in a system as parallel processes may result in starvation owing to the concentration of executions in specific processes. Here, a starvation state is a state where even though a vehicle can move to the next node, some other vehicles continue to move indefinitely and block the vehicle from moving forever. In the verification without fairness constraints, a counterexample due to starvation may be output even for a model in which no counterexample due to a deadlock is detected. However, such a counterexample does not need to be considered for an actual system, except in the case of assuming a special situation such as the failure of a vehicle.

Therefore, in this paper, verification is performed on the premise that each process representing the behavior of a lead vehicle is executed evenly to some extent without considering a counterexample in the reachability of the vehicle due to such a starvation condition.

The notion of fairness is often classified into three categories.

(F1) *Unconditional fairness*: every process gets its turn infinitely often.

(F2) *Strong fairness*: every process that is enabled infinitely often gets its turn infinitely often.

(F3) *Weak fairness*: every process that is continuously enabled from a certain time often gets its turn infinitely often.

To exclude the starvation state, it is necessary to perform verification by assuming two fairnesses. The first is fairness in checking whether the lead vehicle can move to the next node. The second is fairness in the movement of the lead vehicle, which shall be discussed in Section IV-D.

As explained in Section III, each lead vehicle actually moves after confirming whether it can move to the next node. However, if the unconditional fairness of the confirmation is not satisfied, some of the lead vehicles will repeat the operation indefinitely, so that the other lead vehicles never get the opportunity to check whether they are ready to move to the next node. It is therefore necessary to perform verification on all the lead vehicles on the assumption of unconditional fairness that "at any time, each lead vehicle will eventually confirm whether it can move to the next node." In our Promela model, the process `Select()` presented in Fig. 6 is introduced to satisfy this assumption. The process `Select()` sequentially confirms whether each lead vehicle can move to the next node. Thus, by executing the process `Select()` alternately with the lead vehicle process, it is possible to satisfy the assumption of unconditional fairness of the confirmation.

C. Safety properties

The property that our system must meet at the minimum is as follows.

(P1) All passengers can eventually reach their destination from any stop without transfer.

Therefore, for the property (P1) to be satisfied, it is sufficient if for any pair of nodes, there exists a vehicle that passes these nodes infinitely. That is, it is sufficient if the following property (P2) holds.

(P2) For any $s, s' \in S_N$ with $s \neq s'$, at least one vehicle $v \in V$ satisfying $v \in pass(s)$ and $v \in pass(s')$ moves forever.

That is, if the property (P2) is satisfied, there is at least one vehicle that transports any passenger without transfer, regardless of which two points the passenger travels through.

Property (P2), in other words, guarantees that the schedule does not require passengers to transfer and that the system does not fall into a deadlock state. However, this property is the minimum requirement and stronger properties need to be satisfied when considering the efficiency of transport in the system. Here, "efficient" means that all the vehicles continue moving on the designated route, so that the system can realize the transport function without waste, and does not mean that scheduled routes are optimized to minimize, for example, distance and time. In this sense, in addition to (P2), we consider the following property (P2′).

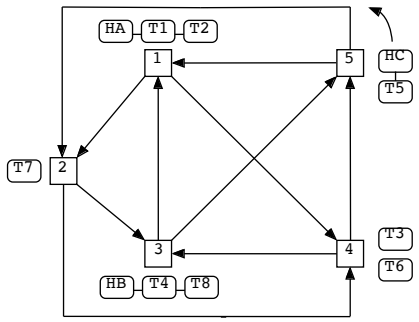

Fig. 8. Example of starvation.

(P2′) All vehicles move forever.

D. Safety properties with linear temporal logic formulae

To verify the properties (P2) and (P2′) presented in Section IV-C, it is necessary to describe them in linear temporal logic (LTL) so that SPIN can perform verification. However, the property "for any $s, s' \in S_N$ with $s \neq s'$, there exists a vehicle v such that $v \in pass(s)$ and $v \in pass(s')$", which is a part of (P2), can be easily verified with methods other than model checking, and we only describe the property (P2′) in LTL.

To verify whether a vehicle continues circling the route, it is necessary to select the node of initial position and another arbitrary node on the route and then verify the property that the vehicle continues visiting these two nodes alternately. Such a property can be described by LTL as a kind of response property (i.e., a property of the form "$\Box(\varphi \rightarrow \Diamond\psi)$" in LTL). . For example, the logical expression that T1 continues to travel in the order of Nodes 1, 2, and 3 is as follows.

$$\text{(L1)} \quad \Box(pos(\mathrm{T1}, 1) \rightarrow \Diamond pos(\mathrm{T1}, 2)) \wedge \Box(pos(\mathrm{T1}, 2) \rightarrow \Diamond pos(\mathrm{T1}, 1)),$$

When verifying (L1) using SPIN and the model obtained so far, a kind of starvation is detected as a counterexample such that when a vehicle (say, v) tries to move to the next node (say, n), other vehicles arrive at node n at that exact time, and v can never move. Such a counterexample is shown in Fig. 8, where the lead vehicle HC continues to visit Node 2 at the time that the lead vehicle HA is going to move.

This observation reveals that the unconditional fairness realized by the process `Select()` alone is insufficient. That is, this unconditional fairness just guarantees fairness with respect to the guarded command that "if the next node is not occupied, then the vehicle can move there." Meanwhile, in the case of a real system, it is hard to imagine circumstances in which specific vehicles continue to interfere with the progress of other vehicles indefinitely, and this counterexample is thus negligible. It is therefore also necessary to assume fairness regarding the actual movement of the vehicle. Specifically, it is necessary to introduce strong fairness that "if the lead vehicle is in a state where it can move, it will certainly move."

This strong fairness can be described in LTL as follows.

$$\text{(L2)} \qquad \bigwedge_{v \in V_L} (\Box\Diamond CanTran(v) \rightarrow \Box\Diamond Tran(v)),$$

where the predicate symbol $CanTran(v)$ means that "the lead vehicle v can move to the next node", while the symbol $Tran(v)$ means that "v actually moves to the next node." (Here we note that although the predicate symbols are used above, these statements can be expressed in propositional formulae.)

From the above discussion, the formula for verifying the movement of vehicles is of the form "$(L2) \rightarrow (L1)$", but it is difficult to directly use such a long formula. In general, SPIN verifies by converting an LTL formula to a Büchi automaton, but the size of the automaton increases exponentially with the length of the formula [3]. Meanwhile, because the formula used to verify the movement of all vehicles at the same time becomes prolonged in proportion to the number of vehicles, there is a possibility that the process of converting the LTL formula to a Büchi automaton does not end in a feasible time. Indeed, verified as it is with the formula above, conversion to the Büchi automaton did not end. However, verifying simultaneously that all vehicles keep moving is equivalent to verifying that each vehicle keeps moving individually. Therefore, when the formula is long as in our example, it is enough to describe the formula individually for each vehicle and perform the verification one by one.

E. Verification using labels

The model described using Promela presented above defines the behavior of each lead vehicle as a process, which is to be repeatedly executed as much as possible. Thus, if no process can be executed, it can be considered that a deadlock state in the system occurs. Section IV-D describes the verification method using the LTL formulae, but SPIN also provides a verification method with the End label. More precisely, by specifying the End label in the model described in Promela, SPIN can judge whether or not the system is in a normal termination state with the presence or absence of the End label when the process stops. Employing this technique, we can also detect a deadlock by verifying the model described without specifying the End label.

V. Verification Results

This section presents verification results and discusses the problem of state explosion in verifying our target system.

A. Results obtained for initial settings

We first considered the abstracted traffic network and the schedule of Example 1. In addition, we assume that $cap\_fleet(v) = 3$ for all $v \in V_H$ and $max\_enter(s) = 4$ for all $s \in S_N$. For this model, we have verified property (P2′). As explained above, SPIN found the deadlock situation presented in Fig. 3 to be a counterexample.

B. Deadlock prevention

Regarding the occurrence of deadlocks, it is well known that a deadlock situation can arise if and only if all the following conditions hold simultaneously [4].

(D1) Mutual exclusion: only one process may use a resource at a time.
(D2) Hold and wait: a process already holding a resource may request a new resource.
(D3) No preemption: only the process holding a resource can release it.
(D4) Circular waiting: multiple processes form a circular chain where each process waits for a resource that is held by the next process in the chain.

Generally, we can prevent deadlocks by arranging the system specifications so that one of these conditions does not hold. In our case, however, it is hard to eliminate the first three conditions by simple modification of a system. Therefore, in the remainder of this section, we consider a way of preventing condition (D4) from being established.

C. Arrangement of model

For (D4) not to be satisfied, we consider the following arrangements of the model.

(A1) Add a restriction that at each of Nodes 2, 3, and 5, there must be no situation where at least one lead vehicle is stopped at the same time.
(A2) For each $s \in S_N$ and for any $v \in V_H$, $cap\_pool(s) - cap\_fleet(v) > 0$.
(A3) For each node, increase the value of cap_pool.

The arrangement of (A1) eliminates the situation in which, at each of Nodes 2, 3, and 5, a fleet cyclically waits for the next node to be released. In the case of our model, we have to consider only the situation that the lead vehicles HA, HB, and HC are respectively present at Nodes 2, 3, and 5 simultaneously. Meanwhile, arrangements (A2) and (A3) are related to the number of vehicles that can stop at each node. This prevents the situation that a fleet cannot move due to occupation of the next node.

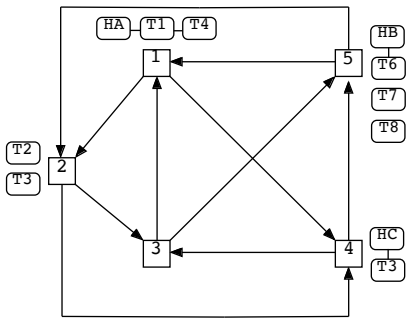

Fig. 9. Detected counterexample after modification (A1).

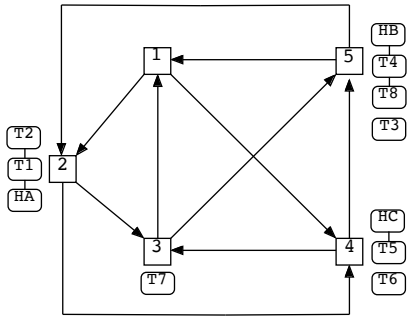

Fig. 10. Detected counterexample after modifications (A1) and (A2).

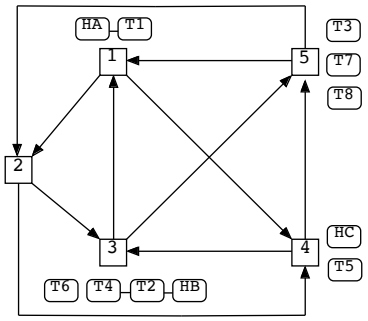

Fig. 11. Detected starvation situation as a counterexample.

D. Verification results after modifications to the model

1) Result of (A1): We verified the model with modification by (A1) in the first model, and as a result, a deadlock as shown in Fig. 9 was detected as a counterexample. In this counterexample, there is no circular waiting of lead vehicles as shown in the previous counterexample in Fig. 3. However, owing to the existence of disconnected trailers at Node 2, another instance of circular waiting occurs, resulting in a deadlock.

2) Result of both (A1) and (A2): We also verified the model with modifications by both (A1) and (A2), and as a result, a deadlock as shown in Fig. 10 was detected again as a counterexample.

3) Detected starvation: A deadlock did not occur when upper limits of the numbers of vehicles in vehicle pools at Nodes 1 to 5 were set as four, six, six, three, and four, respectively. However, the starvation situation presented in Fig. 11, where only HA continues to move, was detected. This situation is not due to the fairness condition of the model, but due to constraints on the number of vehicles that can enter the node at the same time.

As mentioned earlier, such a situation cannot happen for real systems, and it should thus be ignored as a false counterexample. However, a method of efficiently eliminating such false counterexamples has not been established and its development remains a future task.

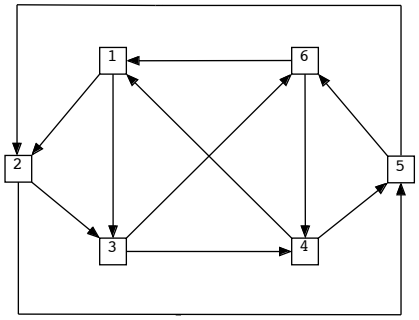

Fig. 12. More complicated model.

E. Other results

The verification results presented above reveal that deadlock occurs regardless of other settings when the upper limit of the number of vehicles remaining at each node is relatively small. We therefore further conducted various arrangements including (A3) and verified the cases that these upper limits were changed at each node.

The results are summarized in Table II. For the verification, we used a Mac Pro having 3.7GHz Quad-Core Intel Xeon E5 and 64GB memory. In the table, for each case, if a deadlock situation and a starvation situation are detected, there is a check mark in the columns of "D" and "S" respectively. These results indicate that adding restrictions to the upper limits can reduce the number of occurrences of deadlock in many cases. However, it is seen that there are deadlocks even if the constraint is too strong.

F. Problem of the state explosion

Generally, in model checking, the verification time increases exponentially with respect to the size of the target model and the LTL formula used in the verification. Therefore, when the model becomes large, it is necessary to reduce the verification time using a technique such as abstraction and division of the model. In response to this problem, in this paper we conduct a simple abstraction of ignoring regular stops.

However, a state explosion may occur if verification is performed on a model more complicated than that in the above example. For example, when verifying the behavior of T1 for the model shown in Fig. 12, verification did not terminate even after 1 hour. The depth of the search at that time was about 5.6 million, the number of states was about 600 million, and the amount of memory used was 120 GB. The solution we are currently considering is to describe only the circulation route of the target vehicle in the model when verifying the movement of each vehicle. For other vehicles, we describe only the movements for travel on this route, and abstract the actions for traveling elsewhere. The details of this method are the subject of future work.

TABLE II
VERFICATION RESULTS OF VARIOUS CASES.

The values of max_enter					Modifications	Num. of states	Execution time (sec)	Num. of counterexamples	D	S
Node 1	Node 2	Node 3	Node 4	Node 5						
4	4	4	4	4	–	14620031	45.7	348	✓	
					(A1)	12661	0.04	4	✓	
					(A2)	13777	0.05	2	✓	
					(A1),(A2)	12661	0.04	4	✓	
5	5	5	5	5	–	3.9e+08	1.3e+03	8239	✓	
					(A1)	12661	0.04	4	✓	
					(A2)	3165420	4.02	489	✓	
					(A1),(A2)	2756053	3.46	717	✓	
6	6	6	6	6	–	1.0e+10	4.2e+03	0		
					(A1)	6.8e+08	2.7e+03	36586	✓	
					(A2)	4.8e+08	2.4e+03	23912	✓	
					(A1),(A2)	4.2e+08	1.7e+03	29410	✓	
4	6	6	3	5	–	6.5e+08	3.4e+03	0		
4	6	6	2	6	–	7.7e+08	3.1e+03	1	✓	
4	6	6	3	4	–	4.5e+08	1.8e+03	1		✓

VI. RELATED WORK

A number of studies on deadlock detection have been conducted by taking formal approaches. Deadlock detection for railroads was studied in [13] and [12]. In these papers, the authors modeled the operation of trains on actual railway lines and described the basic patterns of and solutions to deadlock. To detect and prevent deadlocks in a flexible manufacturing system (FMS), [6] proposed a method that uses Petri nets. Similar to [6], in [14] and [11], the FMS was modeled using Petri nets, but the transitions falling into a deadlock were found using a reachability graph.

Meanwhile, our research proposes a deadlock detection method for last-mile transportation systems. Unlike a railroad or FMS, such systems have complicated behaviors where vehicles can form fleets and the formation is frequently reorganized. In this sense, the problem of deadlock addressed in this research is not that of railroads and the FMS that have been dealt with so far.

VII. CONCLUSIONS AND FUTURE WORK

We proposed a deadlock detection method for last-mile transportation systems. In particular, using model checking, it was possible to comprehensively verify the behavior of the system that is complicated by the reorganization of fleets and the timing of movements of the vehicles. In addition, we discussed specific issues to be solved in actual verification, such as fairness assumptions required for verification and how to describe the safety properties using LTL. Although this paper treated our last-mile transportation system, the method proposed in the paper is expected to be available not only to the transportation system but also to various objects. In particular, it is thought that a system responsible for logistics involving unmanned traveling is essentially the same as the transportation system dealt with in this research.

In future work, we plan to address issues such as a method of dividing and verifying the model to make it possible to verify more complex systems.

REFERENCES

[1] http://www.2getthere.eu/projects/rivium-grt/

[2] http://www.citymobil2.eu/

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

[4] E. G. Coffman, JR., M. J. Elphick, and A. Shoshani. System Deadlocks, *ACM Computing Surveys*, vol.3, issue 2, pp.67–78, 1971.

[5] S. Colin, A. Lanoix, O. Kouchnarenko, and J. Souquières. Using CSP||B Components: Application to a Platoon of Vehicles, *13th International Workshop on Formal Methods for Industrial Critical Systems*, pp.103–118, 2009.

[6] J. Ezpeleta, J. M. Colom, and J. Martínez. A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems, *IEEE Transactions on Robotics and Automation*, vol.11, issue 2, pp.173–184, 1995.

[7] S. Gnesi, G. Lenzini, D. Latella, C. Abbaneo, A. Amendola, and P. Marmo. An Automatic SPIN Validation of a Safety Critical Railway Control System. *The International Conference on Dependable Systems and Networks (DSN 2000)*, pp.119–124, 2000.

[8] K. Hasebe, K. Kato, H. Abe, R. Akiya, and M. Kawamoto. Traffic Management for Last-Mile Public Transportation Systems Using Autonomous Vehicles. *3rd IEEE International Smart Cities Conference (ISC2 2017)*, 8 pages, 2017.

[9] G. J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, vol.23, issue 5, pp.279–295, 1997.

[10] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres. Formal Verification of Autonomous Vehicle Platooning, *ArXiv e-prints*, 2016.

[11] H. Lei, K. Xing, L. Han, F. Xiong, and Z. Ge. Deadlock-free scheduling for flexible manufacturing systems using Petri nets and heuristic search. *Computers & Industrial Engineering*, vol.72, pp.297–305, 2014.

[12] F. Mazzanti, G. O. Spagnolo, S. D. Longa, and A. Ferrari. Deadlock Avoidance in Train Scheduling: A Model Checking Approach. *19th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2014)*, pp.109–123, 2014.

[13] F. Moller, H. N. Nguyen, M. Roggenbach, S. Schneider, and H. Treharne. Defining and Model Checking Abstractions of Complex Railway Models Using CSP||B, *Haifa Verification Conference (HVC2012)*, pp.193–208, 2013.

[14] M. Uzam. An Optimal Deadlock Prevention Policy for Flexible Manufacturing Systems Using Petri Net Models with Resources and the Theory of Regions. *International Journal of Advanced Manufacturing Technology*, vol.19, pp.192–208, 2002.

[15] K. Winter and N. J. Robinson. Modeling Large Railway Interlockings and Model Checking Small Ones. *26th Australasian Computer Science Conference (ACSC2003)*, pp.309–316, 2003.

[16] M. E. Zaher, J. M. Contet, P. Gruer, F. Gechter, and A. Koukam. Compositional Verification for Reactive Multi-Agent Systems Applied to Platoon non Collision Verification. *Studia Informatica Universalis*, vol.10, no.3, pp.119–141, 2012.