Journal of Visual Language and Computing

journal homepage: www.ksiresearch.org/jvlc

Graphical Animations of an Autonomous Vehicle Merging Protocol*

Dang Duy Bui[a], Minxuan Liu[a], Duong Dinh Tran[a] and Kazuhiro Ogata[a,*]

[a]School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

ARTICLE INFO

Article History:
Submitted 3.21.2023
Revised 3.30.2023
Accepted 4.3.2023

Keywords:
state machines
graphical animations
Maude
autonomous vehicle merging protocol
state picture template
Gestalt principles
lockout freedom property
counterexample visualization

ABSTRACT

State machine graphical animation (SMGA) is a tool that takes a state picture template and a state sequence of a state machine as inputs and generates a graphical animation of the state machine by replacing each state in the sequence with the corresponding state picture made from the state picture template as an output. SMGA helps humans to discover characteristics of state machines. r-SMGA is an integration of SMGA and Maude, a formal specification language/tool. Maude is equipped with various functionalities, such as a reachability analyzer (the search command) and an LTL model checker. r-SMGA makes it possible to use such Maude functionalities inside r-SMGA. We suppose that we understand a system/protocol so that we can write a formal specification of the system/protocol. Thus, we know some characteristics of the system/protocol, such as the values that characterize each state of the state machine. Characteristics that can be observed through the formal specification of a system/protocol are called shallow characteristics. Even shallow characteristics positively affect the quality of state picture templates. In this paper, we use an autonomous vehicle merging protocol as an example to demonstrate the claim. We also rely on some Gestalt principles to design state picture templates. Based on our design and Gestalt principle, we describe how to discover deeper characteristics of the state machine that formalizes the protocol with r-SMGA and how to filter out false characteristics with the search command available in r-SMGA if characteristics are likely invariant properties. In addition to invariant properties, there are some other important classes of properties with respect to state machines, such as leads-to properties. We change the behavior of each vehicle (such change does not affect the essence of the protocol) and find that the protocol does not enjoy a leads-to property with the model checker available inside r-SMGA, finding a counterexample. The loop part of the counterexample is graphically animated, which makes us comprehend reasons why the protocol does not enjoy the property and come up with a revised version that enjoys the property.

© 2023 KSI Research

1. Introduction

State machine graphical animation (SMGA) [20] is a tool to visualize a system/protocol formalized as a state machine. The input of SMGA is a state picture template (designed by humans) and a state sequence (generated from a formal specification of a system/protocol by Maude, a formal specification language/tool [17]). The output is a graphical animation of the state machine by replacing each state in the sequence with the corresponding state picture made from the state picture template. SMGA can aid humans in discovering characteristics of systems/protocols [2, 7, 9, 18] through observing their graphical animations. r-SMGA [10] is an integration of SMGA and Maude. Maude is equipped with various functionalities, such as a reachability analyzer (the search command) and an LTL model checker, were LTL stands for linear temporal logic. r-SMGA makes it possible to use such Maude functionalities inside r-SMGA, and to automatically generate a state sequence from a Maude specification. The present paper reports on a case study in which r-SMGA is mainly used. Because designing state picture templates is the key task in r-SMGA [9], it is worth investigating this task. In this paper, we describe how to design state picture templates for r-SMGA by using a concrete non-trivial example, a revised version [16] of the autonomous vehicle merge protocol proposed by Aoki and Rajkumar [1]. The original protocol proposed by Aoki and Rajkumar is called

*The present paper is an extended and revised version of the paper [6] presented at DMSVIVA 2022

*Corresponding author

bddang@jaist.ac.jp (D.D. Bui); liuminxuan@jaist.ac.jp (M. Liu); duongtd@jaist.ac.jp (D.D. Tran); ogata@jaist.ac.jp (K. Ogata)

ORCID(s): 0000-0002-2700-1762 (D.D. Bui); 0000-0001-7092-2084 (D.D. Tran); 0000-0002-4441-3259 (K. Ogata)

DOI reference number: 10-18293/JVLC2023-N1-027

the AR protocol, while the revised version is called the r-AR protocol. The AR protocol depends on realtime information, while the revised version does not. The reason why Liu, et al. [16] made the revised version non-realtime was because they would like to focus on more basic mechanisms that have nothing to do with realtime and to this end, it would be reasonable to remove any realtime information on which the AR protocol relies. The goal of the AR protocol is to control autonomous vehicles to avoid crashing each other at a merge point where two lanes (a through lane and a non-through one) are merged, and so does the r-AR protocol.

To handle the goal of the r-AR protocol, the authors [16] use statuses for each vehicle. Let us briefly describe how each vehicle changes its status in the protocol. Each vehicle on each of the through and non-through lanes is initially in the *running* status, meaning that it is enough far from the merge point and it may go over the vehicles running in front of it. When it gets enough near the merge point, its status changes to the *approaching* one. When a vehicle is in the approaching status, we suppose that it never goes over the vehicles running in front of it. The vehicle on each lane just in front of the merge point will pass through the merge point or stop just before the merge point. In other words, if the former occurs, the vehicle's status changes to *crossing*; otherwise, it changes to *stopped*. Finally, when a vehicle passed the merge point, its status changes to *crossed* from the crossing status.

In the paper, we describe graphical animations of the r-AR protocol, where the main idea is to visualize each vehicle's status with its lane (i.e., through or non-through) based on Gestalt principles. We design state picture templates of the r-AR protocol based on these ideas and shallow characteristics of the protocol (obtained via its specification). By observing graphical animations, we conjecture some deep characteristics of the r-AR protocol to show that Gestalt principles is one factor affecting the design of state picture template.

In the formal verification of the r-AR protocol, we suppose that a small number of vehicles participate in the r-AR protocol, and each vehicle on each of the through and non-through lanes passes through the merge point once; and only consider invariant properties as desired properties. There are some more non-invariant desired properties of the r-AR protocol. One such possible property is that vehicles approaching the merge point or stopping just in front of the merge point on each lane will eventually pass through the merge point. The property belongs to a class of liveness properties, precisely leads-to properties, and is called the lockout freedom property in this paper. We use the Maude LTL model checker (integrated into r-SMGA) such that the r-AR protocol enjoys the lockout freedom property; and do not find any counterexamples. The result is not surprising because a small number of vehicles participate in the r-AR protocol and each vehicle passes through the merge point once. Thus, we modify the formal specification of the r-AR protocol such that each vehicle repeatedly tries to pass through the merge point. In other words, when each vehicle running on each lane has passed through the merge point, it goes back to the running status and repeatedly runs following the protocol's behavior. When we model check that such version enjoys the lockout freedom property, we find a counterexample whose form is a finite sequence of states plus a finite loop of states. We newly design the state picture template for the modified formal specification of the r-AR protocol. We also revise r-SMGA so that it can handle the finite loop of states in a counterexample. By observing the graphical animation of the finite loop of states, we notice reasons why the modified version does not enjoy the lockout freedom property. We then revise the r-AR protocol and model check that the revised version of the r-AR protocol enjoys the lockout freedom property, finding no counterexamples.

The present paper is an extended and improved version of the paper [6] accepted by DMSVIVA 2022. New contributions of the present paper that are not described in the DMSVIVA 2022 paper [6] are as follows:

- The formal specification of the r-AR protocol is modified such that each vehicle repeatedly tries to pass through the merge point;
- The state picture template is newly designed for the modified specification and new tips for designing state picture templates are discovered;
- A counterexample is found for the lockout freedom property for the r-AR protocol under the assumption that each vehicle tries to pass through the merge point repeatedly;
- r-SMGA is revised so that it can deal with the finite loop of states in a counterexample;
- By observing the graphical animation of the finite loop of states, we find reasons why the lockout freedom property is broken for the r-AR protocol under the assumption; and revise the r-AR protocol so that the lockout freedom property is satisfied under the assumption.

The rest of the paper is structured as follows. Section 2 mentions state machines, Maude, r-SMGA, and Gestalt principles as preliminaries. Section 3 introduces the r-AR protocol and its specification in Maude. In Section 4, we describe in detail how to design the state picture template of the r-AR protocol. By observing graphical animations generated based on the state picture template, we then guess some characteristics (likely invariant properties) of the r-AR protocol; and confirm them with the Maude search command integrated into r-SMGA in Section 5. Section 6 describes how we modify the formal specification of the r-AR protocol, model check the lockout freedom property of the modified protocol, show graphical animations of the counterexample, and explain reasons why the modified protocol does not enjoy such property. In Section 7, we revise the modified protocol so that it can enjoy the lock out freedom property. Section 8 mentions some related work. Finally, we conclude the present paper in Section 9.

2. Preliminaries

We briefly describe state machines, Maude, r-SMGA, and some Gestalt principles for readers to better follow the content of the paper.

2.1. State Machines and Maude

A state machine M is defined as $\langle S, I, T \rangle$, where S is a set of states, $I \subseteq S$ is the set of initial states, and $T \subseteq S \times S$ is a binary relation over states. We call $(s, s') \in T$ a state transition, where s, $s' \in S$. The set of reachable states with respect to M is inductively defined as follows: (1) $I \subseteq R$ and (2) if $(s, s') \in T$ and $s \in R$, then $s' \in R$. A state predicate p is an invariant property with respect to M if and only if $p(s)$ holds for all $s \in R$. A finite sequence of states $s_0, \dots, s_i, s_{i+1}, \dots, s_n$ is called a finite computation of M if $s_0 \in I$ and $(s_i, s_{i+1}) \in T$ for each $i = 0, \dots, n-1$.

There are many possible ways to express a state $s \in S$. In the paper, we use a braced associative-commutative collection of name-value pairs. Associative-commutative collections are called soups, and name-value pairs are called observable components. Then, a state is expressed as a braced soup of observable components and the juxtaposition operator is used as the constructor of soups. Suppose $oc1$, $oc2$, $oc3$ are observable components, and then $oc1\ oc2\ oc3$ is the soup of those three observable components. A state can be expressed as $\{oc1\ oc2\ oc3\}$. There are also many possible ways to specify state transitions. One possible way is to use Maude [17], a programming/specification language based on rewriting logic, to specify them as rewrite rules. A conditional rewrite rule (or just a rule) is in the form as follows:

```
crl [lb] : l => r if ... /\ c_i /\ ...
```

where lb is the label given to the rule and c_i is a part of the condition, which may be an equation $lc_i = rc_i$. The negation of $lc_i = rc_i$ could be written as lc_i =/= rc_i

Maude provides the `search` command for reachability analysis that allows finding a state reachable from one state *init* such that the state matches the pattern p and satisfies the condition c. The command can be expressed as follows:

```
search [n,m] in MOD : init =>* p such that c .
```

where n and m are a number of solutions and a bounding depth of a state space of a state machine under analysis, respectively; n is often 1 while m is often omitted meaning that the depth under traversal is not fixed; MOD is the name of the Maude module specifying the state machine; *init*, p, and c are the starting state, the pattern, and the condition, respectively. *init* typically is an initial state of a state machine under verification. The search command can be used as an invariant model checker. The patten p and the condition c are used to express the negation of an invariant property or a state predicate under verification.

Maude is also equipped with an LTL model checker, where LTL stands for linear temporal logic. We suppose that readers are familiar with LTL and Kripke structures [11]. We need to use the LTL model checker so as to formally verify that a system/protocol formalized as a Kripke structure, an extension of state machines, enjoys a liveness property that can be expressed as an LTL formula. Note that invariant properties can be expressed as LTL formulas. When we would like to formally verify that such a system/protocol, where *init* is the only initial state, enjoys a liveness property expressed as an LTL φ, the following command is used:

```
reduce modelCheck(init, φ) .
```

Maude returns true if the system/protocol satisfies φ. Otherwise, a counterexample is returned, which has the form of a finite sequence of states and a finite loop of states. When a system/protocol has multiple initial states, it suffices to conduct the model checking experiment for each initial state so that we can model check that the system/protocol enjoys the property.

2.2. Revised State Machine Graphical Animation (r-SMGA)

Bui, et al. [10] have integrated SMGA and Maude, extending/revising SMGA. The extended/revised version of SMGA is called r-SMGA. Some functionalities of Maude, such as the search command and the LTL model checker, can be used inside r-SMGA. For r-SMGA, inputs are a formal specification of a system/protocol in Maude and a state picture template. r-SMGA allows human users to flexibly generate finite computations and store them as a list. For example, human users can give the depth of each finite computation being generated and generate the graphical animation of one of the finite computations generated before. r-SMGA also makes it possible to extract states from multiple finite computations generated so far by using a given associative-commutative (AC) pattern. Furthermore, r-SMGA allows human users to use some interactive features when observing graphical animations, such as focusing on some observable components that users are interested in.

r-SMGA uses DrawSVG [14] as SMGA does. Thus, r-SMGA as well as SMGA requires human users to design state picture templates. We suppose that a formal specification of a protocol/system under consideration has been written by a human user. Thus, a braced soup of observable component, how to express/formalize each state, can be used to design a state picture template. Our approach to designing a state picture template or visualizing a state is as follows. As mentioned, a state is formalized as a braced soup of observable components, such as $\{oc1\ oc2\ oc3\}$. Thus, it suffices to visualize each observable component (such as $oc1$, $oc2$, $oc3$) to visualize a state. Then, we design a visualization template for each observable component to design a state picture template. Multiple instances of an observable component may be used, and even if that is the case, it suffices to design a visualization template for each observable component but not for each instance of it, where an observable component corresponds to a type and its instances correspond to values of the type. Fixing values used in an observable component, an instance of the observable component is made, and then a visualization template for an observable component becomes a concrete visualization object. Basically, there are two possible ways to visualize observable components: (1) textual

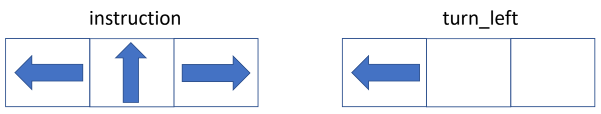


Figure 1: An example of an observable component *traffic-sign*, where the left-hand side is the state picture template and the right-hand side is one concrete state picture

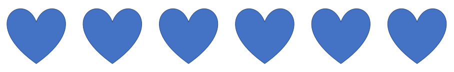


Figure 2: An example before applying gestalt principles

display and (2) visual display. Note that an instance of an observable component can be displayed in both (1) and (2). Option (1) displays values of an observable component instance as texts, while option (2) displays them as visual objects where such visual objects (such as circles and arrows designed by humans) correspond to the values.

For example, let us consider an observable component (`traffic-sign`: *instruction*), where `traffic-sign` is the name and *instruction* is the value that can be one of `turn_left`, `go_straight`, and `turn_right`. Therefore, there are the three instances: `(traffic-sign: turn_left)`, `(traffic-sign: go_straight)` and `(traffic-sign: turn_right)`. The observable component simulates a traffic sign that orders a vehicle to turn left, go straight or turn right. The left-hand side of Figure 1 shows a possible state picture template of the observable component, while the right-hand side shows the state picture of the instance `(traffic-sign: turn_left)`. For the state picture, the observable component is displayed in both (1) and (2) as seen.

2.3. Gestalt Principles

Gestalt principles [24, 25] are a collection of principles, such as the common region and the similarity principles, related to visual perception of humans about grouping. Let us use some concrete examples to describe the two principles in Gestalt principles. Taking a look at Figure 2, we can recognize that there are one group including six hearts. Based on the common region principle (grouping elements that are in the same closed region), Figure 3 can help us to recognize that there are three groups where each group contains two hearts. Similarly, based on the similarity principle (humans tend to build a relationship between similar elements via basic elements, such as color and size), Figure 4 and Figure 5 can also help us to recognize that there are three groups where each group contains two hearts, where Figure 4 uses color while Figure 5 uses size.

3. The r-AR Protocol

In this section, we first describe the r-AR protocol and its specification in Maude. We then introduce the observable components that are used to visualize the protocol.

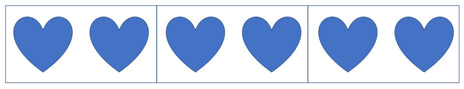


Figure 3: An example after applying the common region principle

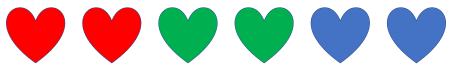


Figure 4: An example after applying the similarity principle (using color)

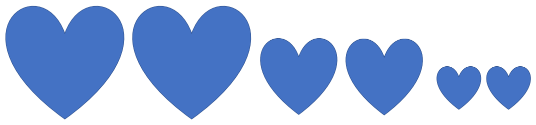


Figure 5: An example after applying the similarity principle (using size)

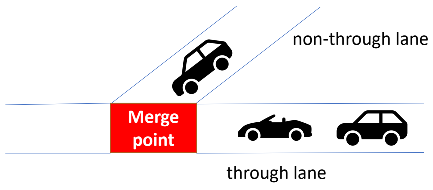


Figure 6: A merge point

3.1. Protocol Descriptions

S. Aoki and K. Rajkumar [1] have proposed an autonomous vehicle merging protocol (called the AR protocol) with two lanes called through and non-through lanes as depicted in Figure 6. The horizontal line refers to the through lane, the diagonal line refers to the non-through lane, and the intersection of those two lanes is called the merge point. Vehicles on both lanes are supposed to run toward the merge point and in one direction only. At the merge point, vehicles are controlled so that they never collide with each other. In other words, the protocol guarantees at most one vehicle located at the merge point.

In the original protocol, there are two versions corresponding to two traffic environments: (1) only autonomous vehicles on the traffic (homogeneous traffic), and (2) autonomous vehicles and human-driven vehicles on the traffic (heterogeneous traffic). Liu, et al. [16] have revised the first version such that the revised version (called the r-AR protocol) does not rely on any real-time information, such as speed of vehicles running on both lanes. There are two modes in the r-AR protocol: prioritized and fair. In the prioritized mode, vehicles on the non-through lane (or non-through lane vehicles) cannot enter the merge point if some vehicles on the through lane (or through lane vehicles) are approaching the merge point. Basically, there are three situations in the prioritized

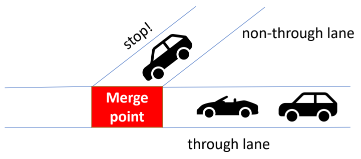


Figure 7: A case that a non-through lane vehicle stops before the merge point because of having some through lane vehicles approaching the merge point

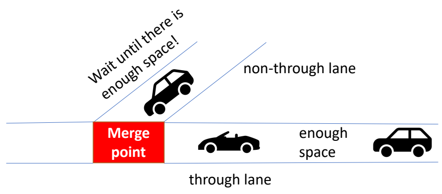


Figure 8: A case where there is enough space on the through lane for a non-through lane vehicle to enter the merge point

mode:

1. If some through lane vehicles are approaching the merge point and there is not enough space between any of two adjacent vehicles, then non-through lane vehicles must stop before the merge point until all through lane vehicles have passed through the merge point. Figure 7 is an example of this case.
2. If no through lane vehicle is approaching the merge point, non-through lane vehicles can enter the merge point.
3. If there is enough space between two adjacent through lane vehicles, then non-through lane vehicles can use the space to enter the merge point. Figure 8 is an example of this case.

When the traffic of the through lane becomes congested, the prioritized mode changes to the fair mode. In the fair mode, through and non-through lane vehicles can enter the merge point alternately. If the traffic of the through lane becomes less congested, the mode changes back to the prioritized mode.

In the r-AR protocol, each vehicle is assigned one of the following five values as its status:

- `running`: when a vehicle is far away from the merge point, its status is `running`.
- `approaching`: when a vehicle gets close to the merge point, its status changes from `running` to `approaching`.
- `stopped`: when a vehicle meets some conditions, it stops before the merge point and its status changes from `approaching` to `stopped`.
- `crossing`: if a vehicle has just entered the merge point, its status changes from `approaching` or `stopped` to `crossing`.
- `crossed`: when a vehicle has passed the merge point, its status changes from `crossing` to `crossed`.

Each vehicle whose status is `running` or `crossed` on each lane may go over those running on the same lane in front of it, while we suppose that each vehicle whose status is `approaching` never does so. Note that each vehicle whose status is `stopped` or `crossing` does not do so if the protocol works as intended. Therefore, we formalize the vehicles whose statuses are `approaching`, `stopped`, or `crossing` on each lane as a queue. In what follows, we will write that the through lane and the non-through lane are formalized as queues, which means that the vehicles on each lane whose statuses are `approaching`, `stopped` or `crossing` are maintained by putting them into a queue. The queue that consists of through lane vehicles is called the queue of the through lane or the through lane queue, while the queue that consists of non-through lane vehicles is called the queue of the non-through lane or the non-through lane queue. The through lane queue may consists of dummy vehicles. Congested traffic is defined as follows: the number of through lane vehicles whose statuses are `approaching` or `stopped` becomes greater than a specific number, making the mode fair. Otherwise, the mode is prioritized. In the fair mode, there is a turn for the through lane and the non-through lane. If the turn is through lane, then through lane vehicles have a higher priority to enter the merge point. If the turn is non-through lane, then non-through lane vehicles have a higher priority to enter the merge point. The turn is basically alternately changed between through lane and non-through lane in the fair mode. Therefore, vehicles on both lanes can enter the merge point alternately in the fair mode based on the turn.

The left-most flowchart in Figure 9 shows how statuses of through lane vehicles are changed. The changes between vehicle statuses, e.g., from `running` to `approaching`, are represented by arrows, possibly with some conditions that are not shown in the figure. The following describes in detail the conditions for changing the status of through lane vehicles. A vehicle status changes from `running` to `approaching` if the through lane vehicle gets close to the merge point. The status of a through lane vehicle approaching the merge point changes from `approaching` to `stopped`, which means that the vehicle must stop just before the merge point, if either:

- there is another vehicle at the merge point, or
- when the protocol is in the fair mode and currently the turn is non-through lane.

The status of a through lane vehicle approaching the merge point changes from `approaching` to `crossing`, which means that the vehicle enters the merge point, if either:

- when the protocol is in the prioritized mode and there is no vehicle at the merge point, or
- when the protocol is in the fair mode, the current turn is the through lane and there is no vehicle at the merge point.

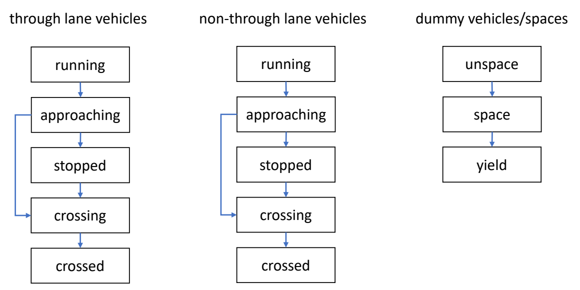


Figure 9: Transitions of status of through lane vehicles, non-through lane vehicles, and dummy vehicles (spaces).

The status of a through lane vehicle stopping the merge point changes from `stopped` to `crossing`, which means that the vehicle enters the merge point, if either:

- when the protocol is in the prioritized mode and there is no vehicle at the merge point, or
- when the protocol is in the fair mode, the current turn is through lane and there is no vehicle at the merge point, or
- when the protocol is in the fair mode, the current turn is non-through lane, there is no vehicle at the merge point, and there is no non-through lane vehicle in the non-through lane queue, which implies that there is no vehicle on the lane just before the merge point.

Vehicle status changes from `crossing` to `crossed` if the through lane vehicle whose status is `crossing` has passed the merge point.

Each space is assigned one of the three statuses: `unspace`, `space`, and `yield`, referring to the space that is not in the queue, is in the queue, and has just been out of the queue, respectively. Spaces on the through lane are treated as dummy vehicles running on the through lane, while it is unnecessary to explicitly deal with spaces on the non-through lane. The right-most flowchart in Figure 9 shows how the status of a space changes. As depicted in the figure, the status `unspace` cannot directly change to the status `yield`. For the change of the statuses of non-through lane vehicles (the middle flowchart in Figure 9) and dummy vehicles/spaces (the right-most flowchart in Figure 9), please refer to [16] in detail.

3.2. Formal Specification of the r-AR Protocol in Maude

Liu, et al. [16] have formally specified the r-AR protocol as a state machine in Maude. Two lanes that are close to the merge point are formalized as two queues of vehicles. The non-through lane queue consists of only real vehicles. Whereas, the non-through lane queue consists of real vehicles and spaces (also called dummy vehicles in this paper). The observable components used to formalize the r-AR protocol are as follows:

- (`lane[`l`]:` q) - l is one of `through` and `nonThrough`, corresponding to the through lane and the non-through lane, respectively. q is a queue of (possibly dummy) vehicle IDs. Initially, q is `empq` (denoting the empty queue) for both lanes.
- (`v[`id`]:` l, vs) - id is a vehicle ID (possibly a dummy vehicle ID). l and vs are the lane and the status information, respectively, of the vehicle ID. id is in the form of either `v(`i`)` when it is a real vehicle, or `dv(`i`)` when it is a dummy vehicle, where i is a natural number distinguishing the vehicle from each other. l is either `through` or `nonThrough`. For real vehicles, vs is `running`, `approaching`, `stopped`, `crossing`, or `crossed`; initially, vs is `running`. For dummy vehicles, vs is `unspace`, `space`, or `yield`; initially, vs is `unspace`.
- (`crossing?:` b) - it represents whether there exists a vehicle crossing the merge point. If so, b is `true`; otherwise, it is `false`. Initially, b is `false`.
- (`mode:` m) - it represents the mode in the r-AR protocol. m is one of `prioritized` and `fair` corresponding to the prioritized mode and the fair mode, respectively. Initially, m is `prioritized`.
- (`turn:` l) - it represents the turn when the system is in the fair mode. l is one of `through` and `nonThrough` corresponding to the turn of through lane and non-through lane, respectively. Initially, l is `through`.
- (`#uvcs:` n) - it represents the number of vehicles that have not yet passed the merge point. n is a natural number. Initially, n equals the number of vehicles participating in the protocol. It is reduced by one when a vehicle has just passed the merge point. When n is 0, all vehicles have crossed the merge point.
- (`gstat:` f) - it indicates that all vehicles have passed the merge point. When all vehicles have passed the merge point, f is `fin`; otherwise, it is `nFin`. Initially, f is `nFin`.

Suppose that there are two vehicles participating in the non-through lane, and four vehicles and two spaces participating in the through lane. If so, the initial state can be expressed as follows:

```
(gstat: nFin) (#ucvs: 6) (crossing?: false)
(mode: prioritized) (turn: through)
(lane[through]: empq) (lane[nonThrough]: empq)
(v[v(0)]: through,running) (v[v(1)]: through,running)
(v[v(2)]: through,running) (v[v(3)]: through,running)
(v[v(4)]: nonThrough,running) (v[dv(0)]: through,unspace)
(v[v(5)]: nonThrough,running) (v[dv(1)]: through,unspace)
```

As mentioned in the previous sub-section, there are some specific conditions on which a status of a vehicle (including a dummy one) changes to another. In the formal specification, rewrite rules in Maude are used to describe state transitions, in which such conditions are embedded. We show here one of the rewrite rules with which the status of a through lane vehicle changes to `crossing` from `stopped`:

```
rl [enter-fairN-T] :
 {(v[v(I)]: through,stopped) (lane[through]: (v(I) ; TQ))
 (lane[nonThrough]: empq) (mode: fair)
 (turn: nonThrough) (crossing?: false) OCs}
 => {(v[v(I)]: through,crossing) (mode: fair)
 (lane[nonThrough]: empq) (turn: nonThrough)
 (lane[through]: (v(I) ; TQ)) (crossing?: true) OCs} .
```

where `I`, `TQ`, and `OCs` are Maude variables of natural numbers, queues, and soups of observable components, respectively. The rewrite rule says that if `mode` is `fair`, `turn` is `nonThrough`, `crossing?` is `false`, the non-through lane queue is empty, and the top vehicle on the through lane is `v(I)` whose status is `stopped`, then `v(I)` changes its status to `crossing`, and `crossing?` is updated to `true`. Meanwhile, in the fair mode, the non-through lane vehicle `v(I)` goes into the merge point when no through lane vehicle is the through lane. The other rewrite rules can be written likewise. The complete specification of the protocol is available at the URL[1].

4. Designing a State Picture Template of the r-AR Protocol

Designing state picture templates is the key task in r-SMGA and also a non-trivial task [9]. If a state picture template is too simple (it contains texts only [8]) or too complex (it contains many values for each observable component [9]), it takes so much effort to conjecture characteristics when observing graphical animations. This section first describes how to design the state picture template using the r-AR protocol as an example. We then show some factors affecting the design of the state picture template. The state picture template of the r-AR protocol is finally shown.

4.1. Idea

We suppose that we have formally specified a system/protocol as a state machine under visualization as described, namely that each state is formalized as a braced soup of observable components (precisely observable component instances) and state transitions are written in rewrite rules. The assumption implies that we comprehend the system/protocol such that we can make such a formal specification in Maude. We can design a state picture template based on observable components. Some previous work [7, 8, 9] have pointed out the usefulness of their state picture templates and also given some tips to make a good state picture template. In the present paper, we mainly use some of such tips for our design and summarize them as follows:

- Values of observable components should be visualized as much as possible.

[1]https://github.com/rSMGA/AVMP/blob/main/avmp.maude

- When an observable component has only two kinds of values, it should be visually/graphically represented as a light bulb.
- If a value of an observable component does not change, it should be expressed as a fixed label.

It is not necessary to visualize all observable components used. For example, we do not need to visualize the observable component `(crossing?: b)` (hereinafter referred to as `crossing?` and the same for other observable components). This is because we can easily know its value by checking if there is a vehicle at the merge point. We can also easily know the value of the observable component `#ucvs` by checking if there exist some vehicles on the two lanes. Hence, we do not need to visualize such observable component. Each of the observable components `turn`, `gstat`, and `mode` should be visualized as a light bulb as described. The lane information of each vehicle (either `through` or `nonThrough`) should be visualized as a fixed label. The remaining observable components needed to be visualized are the observable components `lane[through]`, the `lane[nonThrough]`, and `v[`*vid*`]`.In the rest of this section, we describe how to visualize them and show the final version of a state picture template of the r-AR protocol.

Furthermore, by observing graphical animations based on earlier drafts of the state picture template of the r-AR protocol, we comprehend that when the mode is prioritized, the `turn` observable component does not affect vehicles entering the merge point. Note that this shallow characteristic can be extracted from specification, however, there are many shallow characteristics that may be overlooked by human users. Therefore, we design two observable components `turn` and `mode` following this characteristic. The idea is that when the mode is prioritized, we do not display the observable component `turn`. Note, to design the state picture template in the present paper, we use some Gestalt principles [24, 25] (such as, the common region principle for elements in the queues of both lanes, and the similarity principle using colors for the status of vehicles, e.g., `stopped`). In the next sub-section, we describe designs of observable components and a state picture template of the r-AR protocol.

4.2. Designing a State Picture Template

Figure 10 shows fully our proposed state picture template. We suppose that there are four vehicles and two spaces participating in the through lane, and two vehicles participating in the non-through lane based on the `init` mentioned in Section 3. Figure 11 is our idea for visualizing the through and non-through lanes with vehicles. In the figure, the vertical rectangle and the horizontal rectangle represent the non-through and through lane, respectively; the red region represents the merge point; two arrows represent two meanings: (i) the directions and (ii) the turns of two lanes. The shapes of the arrows refer to (i) while the colors of the arrows refer to (ii), where the light-yellow color and the light-blue color indicate the turn of non-through and through lane vehicles, respectively; circles with numbers inside represent ve-

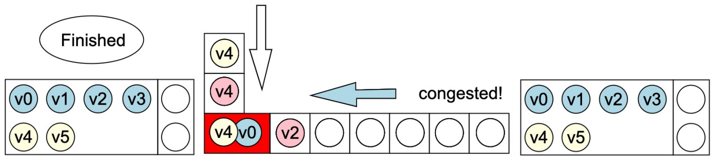


Figure 10: A state picture template for the r-AR protocol (1)

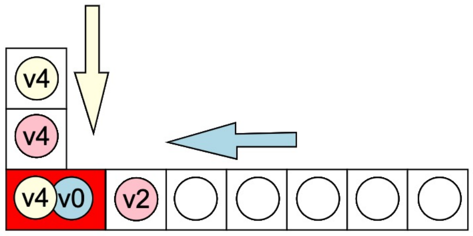


Figure 11: An idea of visualization of through and non-through lanes with vehicles

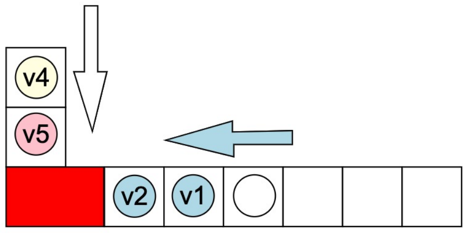


Figure 12: A concrete example of our proposed visualization for both lanes with vehicles

hicles with their IDs inside; white (or blank) circles represent spaces; blue and yellow circles represent through and non-through lane vehicles, respectively; pink circles represent vehicles whose status is `stopped`. Figure 12 shows two lanes when the value of the observable component `lane[through]` is `v(2);v(1);dv(1)` and the value of the observable component `lane[nonThrough]` is `v(5);v(4)`, where the semi-colon is used as the constructor of non-empty queues and the left-most is the top of each queue. In the figure, the status of `v(1)`, `v(2)`, `v(4)`, `v(5)`, and `dv(1)` are `approaching`, `approaching`, `approaching`, `stopped`, and `space`, respectively. Note that our design can help users to recognize a case such that two vehicles on both lanes collide at the merge point.

In Figure 10, as mentioned, both arrows indicate the turn of both lanes. In the prioritized mode, the light-blue arrow (nearing the through lane) and the white arrow (nearing the non-through lane) are displayed. In the fair mode, the text "congested!" is displayed and the arrows are displayed following the observable component `turn`, for example, if the value of `turn` is `nonThrough`, the light-yellow arrow (nearing the non-through lane) and the white arrow (nearing the through lane) are displayed. The rectangle in the left-most side of Figure 10 contains the vehicles and the spaces whose statues are `crossed` and `yield`, respectively. The rectangle in the right-most side of Figure 10 contains the vehicles and the spaces whose statues are `running` and `unspace`, respectively. The oval with the text "Finished" inside refers to the observable component `gStat`. When the value of `gStat` is `fin`, the oval is displayed; otherwise, it is not displayed.

Finally, our proposed design can be extended for more vehicles participating in the protocol. Users can design more squares when the number of squares can meet a case such that all vehicles are in the lane at the same time. For example, there are five vehicles and five spaces participating in the through lane, we need to prepare 10 positions (excluding the merge point) for a case that all of them are put into the queue. We prepare the state picture template shown in Figure 13 for a case in which there are five vehicles and five spaces in the through lane, and five vehicles in the non-through lane. Users can utilize it when the number of vehicles is up to five. Figure 14 shows a case that all through lane vehicles are in the queue. Note that we fix IDs of vehicles in the through and non-through lane from `0` to `4` and `5` to `9`, respectively. Therefore, users need to configure the initial state for such restriction.

5. Confirmation of Guessed Characteristics of the r-AR Protocol and Some Lessons Learned

In this section, we show the usefulness of our proposed design with factors (such as Gestalt principles) via conjecturing characteristics of the r-AR protocol. Finally, we confirm such characteristics by Maude search command integrated into r-SMGA.

5.1. Guessing Characteristics of the r-AR Protocol

Let us repeat that all examples are generated from the `init` as shown in Figure 10. We may recognize some characteristics of a protocol/system when formally specifying it. Graphical animations of the protocol/system make it possible to re-confirm such characteristics that are called shallow characteristics in the paper. It is worth doing so because formal specifications and state picture templates are supposed to be made by human beings who are subject to errors. Human errors may be detected by re-confirming shallow characteristics through observing graphical animations. For example, the following are two such shallow characteristics of the r-AR protocol that can be re-confirmed by ob-

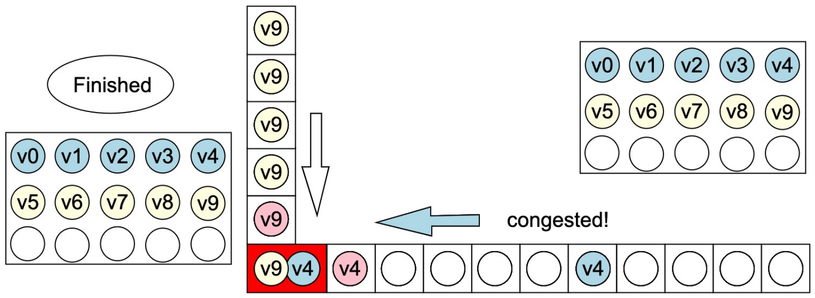


Figure 13: A state picture template for the r-AR protocol (2)

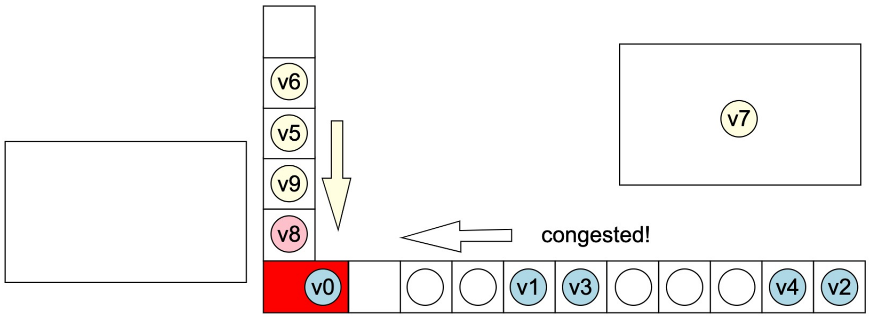


Figure 14: A state picture of when all through lane vehicles are in the queue

serving graphical animations, such as one of them shown in Figure 15:

- Characteristic 0.1: The turn is not concerned when the protocol is in the prioritized mode.
- Characteristic 0.2: There exists a case such that there is one vehicle whose status is `approaching` in the non-through lane, but this status changes to `stopped` even no vehicle is in the through lane.

Note that there are two sequences (two consecutive states for each) shown in Figure 15 obtained from two different sequences of states generated from the formal specification.

To conjecture some other characteristics, we use some tips [9]. They say that focusing on one or two observable components can help us to guess some relations of such observable components. We concentrate on two lanes and discover characteristics, some of which are as follows:

- Characteristic 1: There is at most one vehicle whose status is `stopped` in each lane.
- Characteristic 2: There are two vehicles whose statuses are `stopped` on both lanes, respectively, no vehicle is at the merge point.
- Characteristic 3: There are at most two vehicles whose statuses are `stopped` in the protocol.

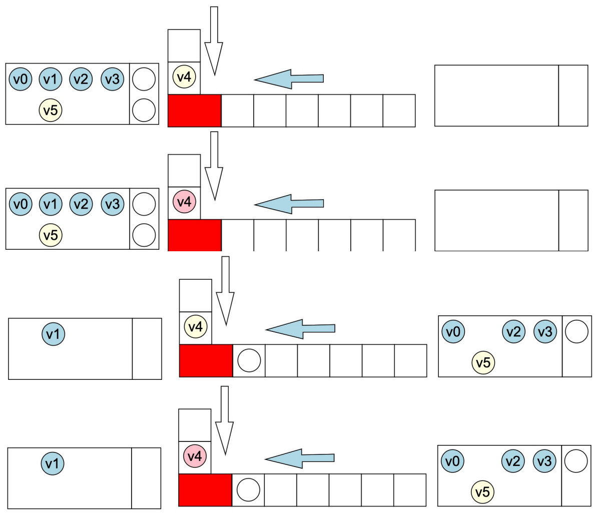


Figure 15: Some state pictures for Characteristic 0.2

To find the characteristics above, color is the main factor. Based on the colors designed based on the similarity principle, we can observe that there exists at most one light-pink color on each lane shown in Figure 16. Characteristic 3 can be conjectured based on two characteristics 1 and 2. The following characteristics are found by focusing on vehicles whose status is `stopped`.

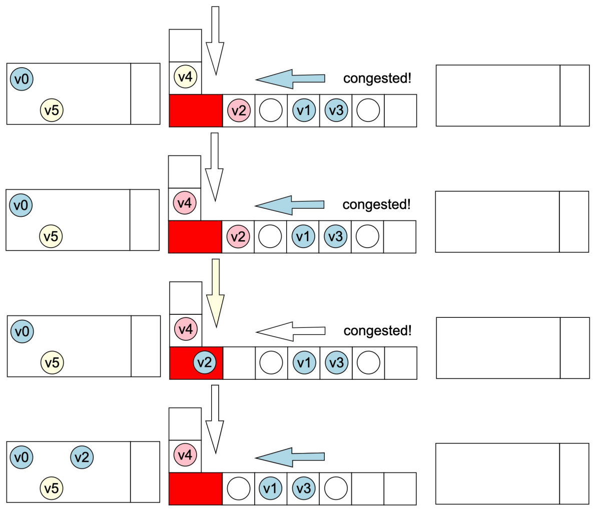


Figure 16: A piece of a finite computation

- Characteristic 4.1: If there is one vehicle whose status is `stopped` on the non-through lane and some vehicle is at the merge point, then this vehicle is on the through lane or the through lane queue is not empty.
- Characteristic 4.2: If there is one vehicle whose status is `stopped` on the through lane and some vehicle is at the merge point, then this vehicle is on the non-through lane or the non-through lane queue is not empty.

5.2. Confirmation of Guessed Characteristics

Characteristics guessed with r-SMGA may be wrong. The search command functionality available in r-SMGA makes it possible to check if there exists a counterexample for each characteristic. Note that we do not prove each characteristic and then a guessed characteristic for which no counterexample is found with the functionality may be wrong. Proof of characteristics for protocols/systems is out of the scope of the pater. For those who are interested in the topic, please refer to [5]. To sum up, when characteristics are confirmed (no counterexample is found) in the section, they are guaranteed for the case whose initial state is expressed as `init` as shown in Figure 10. The syntax of the functionality is the same as what is mentioned in Section 2. The following command is used to confirm Characteristic 1.

```
search [1] in AVMP : init =>*
{(v[X:Vid]: l1:Lane, stopped)
(v[Y:Vid]: l1:Lane, stopped) OCs:Soup{OComp}} .
```

where `AVMP` is the name of Maude module; `X:Vid` and `Y:Vid` are Maude variables of vehicle IDs; `l1:Lane` is a Maude variable of lanes; and `OCs:Soup{OComp}` is a Maude variable of observable component soups. The search command tries to find a reachable state from `init` in which there are two different vehicles whose statuses are `stopped` on one lane. If there is no such a reachable state from `init`, Characteristic 1 is an invariant property for the case whose initial state is `init`. It does not return any solution, meaning that no counterexample is found for the characteristic. Note that we use the Maude search command integrated into r-SMGA as shown in Figure 17 for Characteristic 1.

To confirm Characteristics 2 and 3, we use two commands as shown in Figure 18 and Figure 19, respectively. Each search command tries to find a reachable state that satisfies the corresponding pattern. They do not return any solution, meaning that no example is found for the two characteristics. Similarly, to confirm Characteristics 4.1 and 4.2, we use two commands as shown in Figure 20 and Figure 21, respectively. The two commands also do not return any solution, meaning that no counterexample is found for the two characteristics.

Note that we need to confirm all guessed characteristics because human users may overlook flawed cases. One flawed characteristic we have conjectured is as follows: when there is a non-through lane vehicle whose status is `approaching`, the next status of the vehicle is always stopped whenever its status changes. The characteristic cannot be expressed as an invariant property and then it is impossible to check the characteristic with the search command. We should use the Maude LTL model checker that is available in r-SMGA. When we use the Maude model checker in r-SMGA to confirm it, r-SMGA returns a counterexample. The counterexample says that the status of a non-through lane vehicle can change to `crossing` from `approaching` when the `mode` is `fair` and the `turn` is `nonThrough`.

6. A Flawed Case of the r-AR Protocol

Beside invariant properties of a state machine, there are liveness properties, such as leads-to properties. In this section, we will check whether the r-AR protocol satisfies a property called the lock-out freedom that can be expressed as a leads-to property. An informal description of the property says that when a vehicle approaches or stops just before the merge point, it will eventually go into the merge point.

If there are a few vehicles running on the two lanes and each vehicle tries to go through the merge point once, the r-AR protocol enjoys the lock-out freedom property. Therefore, we modify the behavior of each vehicle as follows: when the status of each vehicle is `crossed`, it will change to `running`, meaning that each vehicle tries to go through the merge point repeatedly. Note that we do not essentially change the r-AR protocol. To do it, we add the following rewrite rule:

```
rl [return] :
  {(v[v(I)]: L1,crossed) OCs} =>
  {(v[v(I)]: L1,running) OCs} .
```

where `return` is the name of the rule, and `I` and `L1` are Maude variables of vehicles IDs and lanes, respectively. The rule says that if the status of a vehicle `I` on a lane (through or non-through) is `crossed`, it changes to `running`. To model check the lock-out freedom property, we define some propositions as follows:

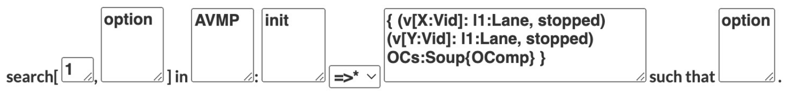

search[1 , option] in AVMP : init =>* { (v[X:Vid]: l1:Lane, stopped) (v[Y:Vid]: l1:Lane, stopped) OCs:Soup{OComp} } such that option .

Figure 17: A command in r-SMGA to confirm Characteristic 1

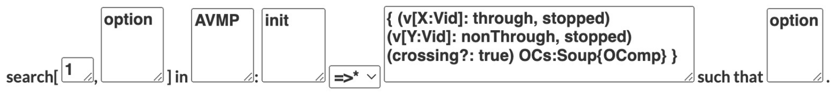

search[1 , option] in AVMP : init =>* { (v[X:Vid]: through, stopped) (v[Y:Vid]: nonThrough, stopped) (crossing?: true) OCs:Soup{OComp} } such that option .

Figure 18: A command in r-SMGA to confirm Characteristic 2

```
eq {(v[v(I)]: L,approaching) OCs} |= approaching(I,L)
                                                = true .
eq {(v[v(I)]: L,stopped) OCs} |= stopped(I,L) = true .
eq {(v[v(I)]: L,crossed) OCs} |= crossed(I,L) = true .
eq {OCs} |= PROP = false [owise] .
```

where `_|=_` is a Maude operator for assigning the state (the first parameter) to a proposition (the second parameter) and returns a Boolean value. `approaching(_,_)`, `stopped(_,_)`, and `crossed(_,_)` are Maude operators that denote the propositions meaning that the status of a vehicle on a lane is `approaching`, `stopped`, and `crossed`, respectively. `PROP` is a Maude variable of propositions. The first equation says that the proposition `approaching(I,L)` is `true` if a status of the vehicle `I` on lane `L` is `approaching`; Propositions `stopped(I,L)` and `crossed(I,L)` are defined likewise. `owise` stands for otherwise. We define the lock-out freedom property as follows:

```
eq lockout-free(I,L) =
  (approaching(I,L) \/ stopped(I,L)) |-> crossed(I,L) .
```

where `_|->_` is a Maude operator to denote the leads-to temporal connective. The formula says that whenever the status of the vehicle `I` on the lane `L` is `approaching` or `stopped`, the status will eventually become `crossed`.

We use the Maude model checking functionality in r-SMGA to check the property for each lane and each vehicle. For any vehicles on the through lane, no counterexample is returned. It implies that the r-AR protocol satisfies the lock-out freedom property for the through lane. However, a counterexample is returned when model checking for the non-through lane, which says that the protocol does not satisfy the lock-out freedom property for the non-through lane. Figure 22 shows the state pictures representing the loop part of the counterexample, where two vehicles on the non-through lane never enter the merge point. The following is the order of the loop. The first state of the loop is shown at the top-most on the left column, the second state is shown at the top-most on the right column, the third state is shown at the second row of the left column, etc. The next state of the final state of the loop shown at the bottom-most on the right column is the first state. Note that we have deleted the two observable components `gstat` and `#uvcs` in the protocol and in the state picture template because each vehicle tries to go through the merge point repeatedly and then we do not need to maintain them. Taking a look at those state pictures via graphical animations, we can observe situations where two vehicles on the non-through lane never go through the merge point. r-SMGA allows human users to observe the loop of the state shown in Figure 22 as graphical animations. Careful observation of the graphical animations helps us to know that there are two cases in the states in the loop:

1. The mode is fair and a through lane vehicle is passing the merge point. Therefore, even though the turn is non-through, the non-through lane vehicle just before

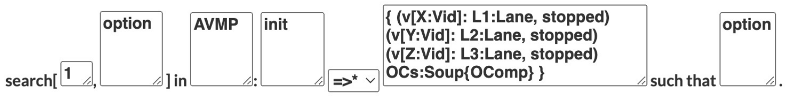

search[1 , option] in AVMP : init =>* { (v[X:Vid]: L1:Lane, stopped) (v[Y:Vid]: L2:Lane, stopped) (v[Z:Vid]: L3:Lane, stopped) OCs:Soup{OComp} } such that option .

Figure 19: A command in r-SMGA to confirm Characteristic 3

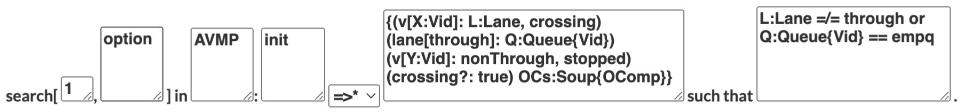
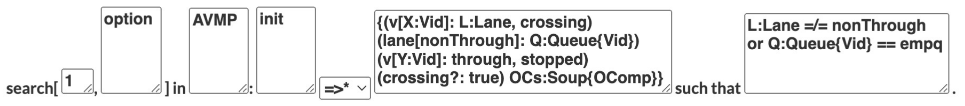

search[1 , option] in AVMP : init =>* {(v[X:Vid]: L:Lane, crossing) (lane[through]: Q:Queue{Vid}) (v[Y:Vid]: nonThrough, stopped) (crossing?: true) OCs:Soup{OComp}} such that L:Lane =/= through or Q:Queue{Vid} == empq .

Figure 20: A command in r-SMGA to confirm Characteristic 4.1



```
search[ 1 , option ] in AVMP : init =>* {(v[X:Vid]: L:Lane, crossing)
(lane[nonThrough]: Q:Queue{Vid})
(v[Y:Vid]: through, stopped)
(crossing?: true) OCs:Soup{OComp}} such that L:Lane =/= nonThrough
or Q:Queue{Vid} == empq .
```

Figure 21: A command in r-SMGA to confirm Characteristic 4.2

the merge point is not allowed to cross the merge point.

2. The mode is prioritized. If this is the case, a vehicle is passing the merge point or there is a through lane vehicle just before the merge point. Hence, the non-though lane just before the merge point has no chance to cross the merge point.

In Figure 22, there are four states that belong to Case 1, where you can see "Congested!" appearing. For each of the four states, its next state is in the prioritized mode. The r-AR protocol cannot make the best use of the fair mode, letting a non-through lane vehicle to cross the merge point. For each state that belongs to Case 2, a through lane vehicle is crossing the merge point or there is a through lane vehicle just in front of the merge point. Therefore, non-through vehicles are never allowed to cross the merge point. In the next section, we revise the r-AR protocol so that the it can enjoy the lockout freedom property.

7. A Revised Version of the r-AR Protocol

As written, the r-AR protocol does not make the best use of the fair mode, letting a non-through lane to cross the merge point. In the r-AR protocol, where the number of the through lane vehicles whose status is `approaching` is greater than a fixed number, the mode becomes fair, but if the number becomes less than the fixed number, the mode becomes back prioritized. Between two modes, nothing guarantees that a non-through lane vehicle can pass the merge point. Therefore, we modify how to change the prioritized mode to the fair mode and vice versa.

To handle such situation, we use how many through lane vehicles have entered the merge point instead. Let N be such number. When N becomes greater than a fixed number, the mode is changed to fair from prioritized, and non-through lane vehicles are given a higher priority than though lane vehicles in the fair mode. If the through lane queue is empty, the non-through lane just before the merge point is allowed to enter the merge point. So, let us suppose that the through lane queue is not empty and the mode is prioritized. If that is the case, though lane vehicles cross the merge point, increasing N. When N becomes larger than a fixed number, the mode changes to fair from prioritized. If the non-through lane queue is not empty, the top vehicles of the queue is given the highest priority to enter the merge point and then the non-through lane vehicle is allowed to enter the marge point. We add one observable component `(#tlvp N)` representing the number of through lane vehicles that have passed the merge point. The rewrite rule used when a through lane vehicle enters the merge point in the prioritized mode is modified as follows:

```
rl [enter-prio-T] :
  {(v[v(I)]: through,VS) (lane[through]: (v(I) ; TQ))
  (crossing?: false) (mode: prioritized) (#tlvp: N) OCs}
   => {(v[v(I)]: through,crossing)
   (lane[through]: (v(I) ; TQ)) (crossing?: true)
   (mode: (if N < 2 then prioritized else fair fi))
   (#tlvp: ( if N < 2 then s(N) else N fi)) OCs} .
```

where `TQ` is a Maude variable of queues, `N` is a Maude variable of natural numbers, and `s(_)` is the successor function of natural numbers. We use `2` as the fixed number and can use a different positive number.

When a non-through lane vehicle has entered the merge point in the fair mode, we change the mode to prioritized from fair and make `n` used in `(#tlvp n)` 0. Then, the rewrite rule used when a non-through lane vehicle enters the merge point in the fair mode is revised as follows:

```
rl [enter-fair-N] :
  {(v[v(I)]: nonThrough,stopped)
  (lane[nonThrough]: (v(I) ; NQ))
  (mode: fair) (#tlvp: N) (crossing?: false) OCs}
   => {(v[v(I)]: nonThrough,crossing)
  (lane[nonThrough]: (v(I) ; NQ)) (mode: prioritized)
  (#tlvp: 0) (crossing?: true) OCs} .
```

where `NQ` is a Maude variable of queues.

We check if the revised version of the r-AR protocol satisfies that lockout freedom property with the model checking functionality in r-SMGA, and then no counterexample is found. We confirm that this revised version satisfies the lockout freedom property.

Note that we do not use the observable component `turn` because we use the observable component `mode` instead. The formal specification of the revised version of the r-AR protocol is available at the URL[2].

8. Related Work

Bui, et al. [7] have used SMGA to graphically animate the intersection traffic control distributed mutual exclusion protocol or the LJPL protocol [15]. In the protocol, there are eight lanes in which each two of them can be conflicted or concurrent. Vehicles in the concurrent lanes are allowed to enter the intersection at the same time while vehicles in the conflicted lanes are prohibited. Moreover, vehicles in the

[2]https://github.com/rSMGA/AVMP/blob/main/avmp-rev.maude

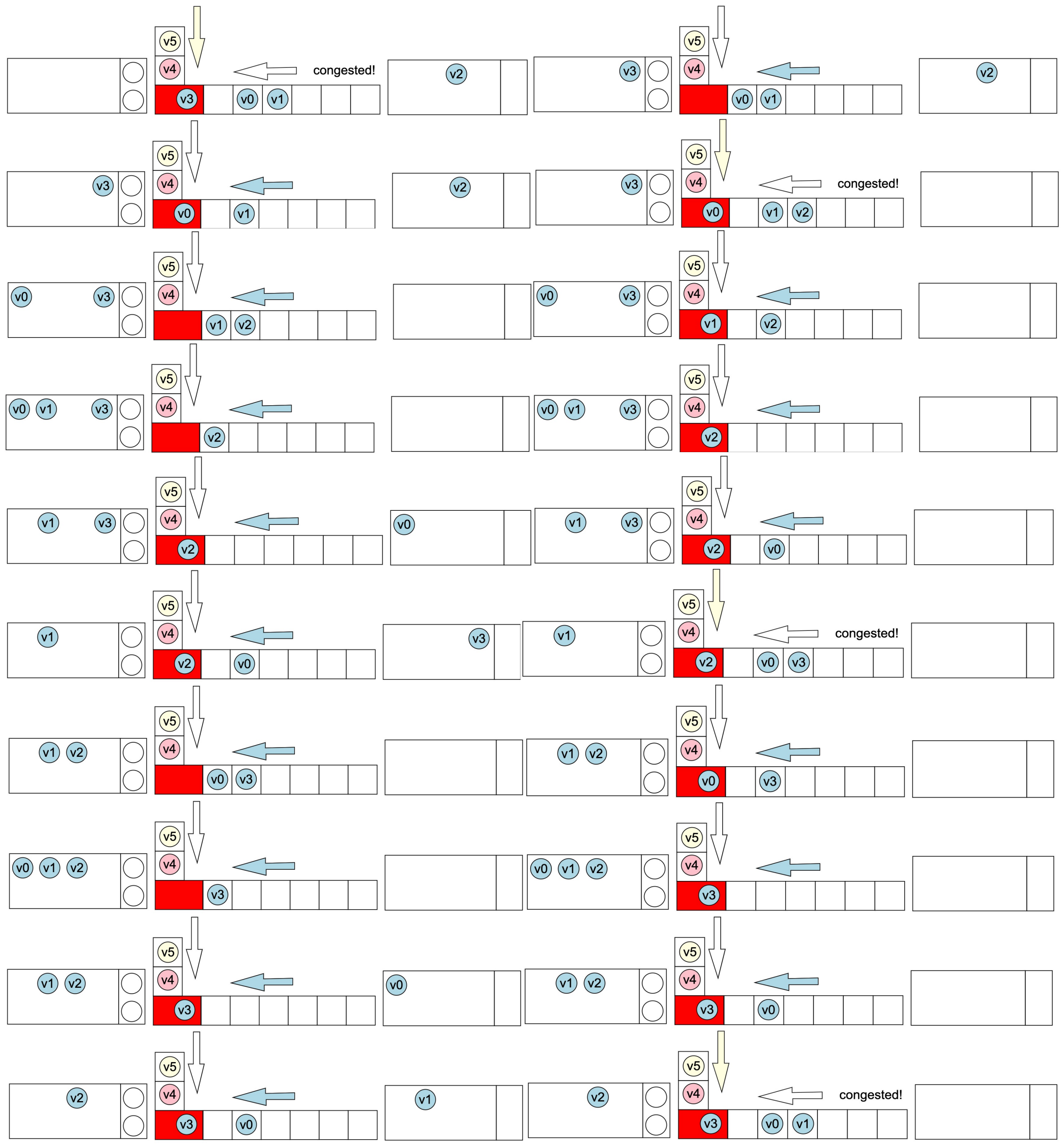


Figure 22: A loop generated by the model checking functionality in r-SMGA.

same lane can also enter the intersection at the same time. To guarantee the mutual exclusion property (e.g., no conflicted lane vehicles in the intersection), the protocol uses statuses for each vehicle (e.g., approaching and crossing) and controls them by their proposed algorithm. Based on some assumptions of the protocol, such as lanes of vehicles and vehicles' statuses, Bui, et al. [7] have revised SMGA and designed a state picture template so that the tool can visualize elements in the lanes (vehicles' IDs) followed by other elements (vehicles's statuses). By observing graphical animations based on their state picture template, the authors conjecture some non-trivial characteristics of the protocol and the characteristics have been confirmed with Maude. The r-AR protocol and the LJPL protocol share some ideas to design, such as using lanes of vehicles and statuses of vehicles, but, some assumptions of two protocols are different, and then we cannot apply all ideas to visualize the LJPL protocol for visualizing the r-AR protocol.

Frank, et al. [13] have proposed a method to visualize state transition systems of protocols/systems. They aim to let users observe global properties of protocols/systems by visualizing whole state spaces. They use cone tree [21] to form state transition structures in three dimensions so that users can observe the visualization in three dimensions. The main algorithm of the method focuses on the symmetry property and aims to let users identify the symmetrical and similar sub-structures in the tree. To do that, they first rank all nodes to make the systems become hierarchical system structures [23]. Then, they cluster such nodes following some local properties to reduce the visual complexity of the tree. Based on the clusters, they aim to visualize the whole state spaces as a backbone tree, where clusters are visualized as circles whose sizes are decided by their volumes. Then, users can observe and interact with the tree by focusing and zooming into some clusters to be able to analyze paths inside. The method is extended to deal with a large state space [12]. The results of both versions allow users to observe state spaces of protocols visualized as a backbone tree with the cone tree concept for each node. Then users can find some global properties of the protocols, such as obtaining some clusters that do not return to initial nodes after starting some executions. This method and r-SMGA share the idea to help users find properties or characteristics of protocols. r-SMGA graphically animates state sequences (paths) while this method visualizes whole state spaces. The idea of the method motivates us to extend r-SMGA so that r-SMGA can give users an overview of state spaces. Then users can select some paths and graphically animate them by our approach. One piece of our future work is to extend r-SMGA based on the mentioned ideas.

r-SMGA can be considered a way to comprehend counterexamples via graphical animations where the counterexamples are returned by Maude. Nguyen, et al. [19] have shown that observing graphical animations of a shorter counterexample can make humans easier to comprehend. The idea is to use meta level in Maude to generate a shorter counterexample based on the search command in Maude. First, they use a model checker equipped with Maude to generate a counterexample when a system does not satisfy some property. Then, they use meta-search in Maude that uses a breadth-first search way (this search can make the counterexample shorter) to find a shorter state sequence leading the loop part of the counterexample. For the loop, it can work likewise. Finally, the shorter counterexample is graphically animated with SMGA.. In the present paper, the size of the counterexample is large (over 450 states) and then the approach to making a counterexample shorter would be useful for r-SMGA as well. It is one piece of our future work to utilize the approach in r-SMGA.

9. Conclusion

We have graphically animated the r-AR protocol with r-SMGA. Designing a state picture template is a non-trivial task in r-SMGA. To design the state picture template of the r-AR protocol, we have used the tips [9] and Gestalt principles [24, 25]. We also have shown some factors that affect the state picture template, such as colors in the similarity principles of Gestalt principles. Observing graphical animations helps us to reconfirm some shallow characteristics of the r-AR protocol that have been noticed when writing the formal specification and guess some deep characteristics of the protocol. Those characteristics have been confirmed with the Maude search command in r-SMGA. We have checked that the r-AR protocol does not enjoy the lockout freedom property using the Maude model checking function available in r-SMGA. By observing graphical animations of the counterexample, we can comprehend reasons why the protocol does not satisfy the property and revise the r-AR protocol so that the revised version enjoys that property. There are two main pieces of our future directions: one is to implement some factors that help humans to better comprehend graphical animations of a counterexample, such as size and causality [3]; another direction is to conduct more advanced case studies, such as quantum teleportation protocol [4] and Shor algorithm [22], to demonstrate the usefulness of our approach [5].

References

[1] Aoki, S., Rajkumar, R.R., 2017. A merging protocol for self-driving vehicles, in: ICCPS, p. 219–228. doi:10.1145/3055004.3055028.

[2] Aung, M.T., Nguyen, T.T.T., Ogata, K., 2018. Guessing, model checking and theorem proving of state machine properties – a case study on Qlock. IJSECS 4, 1–18. doi:10.15282/ijsecs.4.2.2018.1.0045.

[3] Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R., 2012. Explaining Counterexamples Using Causality. Formal Methods in System Design 40, 20–40. doi:10.1007/s10703-011-0132-2.

[4] Bennett, C.H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., Wootters, W.K., 1993. Teleporting an Unknown Quantum State via Dual Classical and Einstein-Podolsky-Rosen Channels. Phys. Rev. Lett. 70, 1895–1899. doi:10.1103/PhysRevLett.70.1895.

[5] Bui, D.D., 2022. State Machine Visualization Based on Gestalt Principles and Its Applications. Ph.D. thesis. Japan Advanced Institute of Science and Technology. URL: http://hdl.handle.net/10119/18189.

[6] Bui, D.D., Liu, M., Ogata, K., 2022a. Graphical Animations of an Autonomous Vehicle Merging Protocol, in: DMSVIVA 2022, KSI Research Inc.. pp. 16–22. doi:10.18293/DMSVIVA22-009.

[7] Bui, D.D., Myint, W.H.H., Tran, D.D., Ogata, K., 2022b. Graphical animations of the Lim-Jeong-Park-Lee autonomous vehicle intersection control protocol. JVLC 2022, 1–15. doi:10.18293/JVLC2022-N1-004.

[8] Bui, D.D., Ogata, K., 2019. Graphical animations of the Suzuki-Kasami distributed mutual exclusion protocol. JVLC 2019, 105–115. doi:10.18293/JVLC2019-N2-012.

[9] Bui, D.D., Ogata, K., 2022. Better state pictures facilitating state machine characteristic conjecture. Multimedia Tools and Applications 81, 237–272. doi:10.1007/s11042-021-10992-z.

[10] Bui, D.D., Tran, D.D., Ogata, K., Riesco, A., 2022c. Integration of SMGA and Maude to Facilitate Characteristic Conjecture, in: DMSVIVA 2022, KSI Research Inc.. pp. 45–54. doi:10.18293/DMSVIVA22-006.

[11] Clarke, E.M., Grumberg, O., Kroening, D., Peled, D.A., Veith, H., 2018. Model checking, 2nd Edition. MIT Press. URL: https://mitpress.mit.edu/books/model-checking-second-edition.

[12] Groote, J., Ham, F., 2006. Interactive visualization of large state spaces. STTT 8, 77–91. doi:10.1007/s10009-005-0198-5.

[13] van Ham, F., van de Wetering, H., van Wijk, J.J., 2002. Interactive Visualization of State Transition Systems. IEEE Transaction on Visual and Computer Graphics 8, 319–329. doi:10.1109/TVCG.2002.1044518.

[14] Liard, J., . Draw-svg the free online drawing tools. https://www.drawsvg.org/. Accessed: 2022-04-25.

[15] Lim, J., Jeong, Y., Park, D., Lee, H., 2018. An efficient distributed mutual exclusion algorithm for intersection traffic control. J. Supercomput. 74, 1090–1107. doi:10.1007/s11227-016-1799-3.

[16] Liu, M., Bui, D.D., Tran, D.D., Ogata, K., 2021. Formal specification and model checking of an autonomous vehicle merging protocol, in: QRS-C, pp. 333–342. doi:10.1109/QRS-C55045.2021.00057.

[17] M. Clavel, et al. (Ed.), 2007. All About Maude. volume 4350 of *LNCS*. Springer. doi:10.1007/978-3-540-71999-1.

[18] Mon, T.W., Bui, D.D., Tran, D.D., Ogata, K., 2021. Graphical animations of the ns(l)pk authentication protocols. JVLC 2021, 39–51. doi:10.18293/JVLC2021-N2-005.

[19] Nguyen, T.T.T., Ogata, K., 2017a. A Way to Comprehend Counterexamples Generated by the Maude LTL Model Checker, in: 2017

International Conference on Software Analysis, Testing and Evolution (SATE), pp. 53–62. doi:10.1109/SATE.2017.15.

[20] Nguyen, T.T.T., Ogata, K., 2017b. Graphical animations of state machines, in: 15th DASC, pp. 604–611. doi:10.1109/DASC-PICom-DataCom-CyberSciTec.2017.107.

[21] Robertson, G.G., Mackinlay, J.D., Card, S.K., 1991. Cone trees: animated 3d visualizations of hierarchical information, in: SIGCHI, pp. 189–194. doi:10.1145/108844.108883.

[22] Shor, P., 1994. Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. doi:10.1109/SFCS.1994.365700.

[23] Sugiyama, K., Tagawa, S., Toda, M., 1981. Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics 11, 109–125. doi:10.1109/TSMC.1981.4308636.

[24] Todorovic, D., 2008. Gestalt principles. Scholarpedia 3, 5345. doi:10.4249/scholarpedia.5345.

[25] Ware, C., 2012. Information Visualization: Perception for Design. 3 ed., Morgan Kaufmann.