Better state pictures facilitating state machine characteristic conjecture

Dang Duy Bui¹ · Kazuhiro Ogata¹

Received: 20 June 2020 / Revised: 10 February 2021 / Accepted: 30 April 2021 /

Published online: 26 May 2021

© The Author(s) 2021

Abstract

The mutual exclusion protocol invented by Mellor-Crummey and Scott (called MCS protocol) is used to exemplify that state picture designs based on which the state machine graphical animation (SMGA) tool produces graphical animations should be better visualized. Variants of MCS protocol have been used in Java virtual machines and therefore the 2006 Edsger W. Dijkstra Prize in Distributed Computing went to their paper on MCS protocol. The new state picture design of a state machine formalizing MCS protocol is assessed based on Gestalt principles, more specifically proximity principle and similarity principle. We report on a core part of a formal verification case study in which the new state picture design and the SMGA tool largely contributed to the successful completion of the formal proof that MCS protocol enjoys the mutual exclusion property. The lessons learned acquired through our experiments are summarized as two groups of tips. The first group is some new tips on how to make state picture designs. The second one is some tips on how to conjecture state machine characteristics by using the SMGA tool. We also report on one more case study in which the state picture design has been made for the mutual exclusion protocol invented by Anderson (called Anderson protocol) and some characteristics of the protocol have been discovered based on the tips.

Keywords Anderson protocol · Gestalt principles · Graphical animation · MCS protocol · SMGA · State machine · State picture design

This work was partially supported by JSPS KAKENHI Grant Number JP19H04082 and FY2020 grant-in-aid for new technology research activities at universities (SHIBUYA SCIENCE CULTURE AND SPORTS FOUNDATION).

✉ Kazuhiro Ogata
ogata@jaist.ac.jp

Dang Duy Bui
bddang@jaist.ac.jp

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

1 Introduction

Our group led by the second author of the present paper has been developing a state machine graphical animation (SMGA) tool [7, 22]. Given a state picture design for a state machine, SMGA basically takes a sequence of states in text and plays a graphical animation for the state machine by regarding the sequence as a movie film based on the state picture design. SMGA has been used to make graphical animations of state machines formalizing Alternating Bit Protocol (ABP) [22], Qlock shared-memory mutual exclusion protocol [4], MCS shared-memory mutual exclusion protocol [23] and Suzuki-Kasami distributed mutual exclusion protocol [7]. The main purpose of SMGA is to help experts in Formal Methods conjecture lemma candidates or state machine characteristics that can be used for formal verification by visually/graphically observing graphical animations. This is because humans are good at visual perception [6].

We summarize some tips on how to design state pictures in our paper [7] published in 2019. The tips have been accumulated through the case studies for ABP [22], Qlock protocol [4], MCS protocol [23] and Suzuki-Kasami protocol [7]. They are still useful but we realized that there is some room to improve the state picture design used for SMGA so far. Some part of each state of a state machine formalizing MCS protocol was not visualized sufficiently enough in that the part is almost the same as text representation. We have made a new state picture design of (the state machine formalizing) MCS protocol in which the part is visualized. We assess the new state picture design based on Gestalt principles (or principles of grouping) [26, 28, 29, 31]. The new state picture design can be expressed reasonably well based on two Gestalt principles: proximity principle and similarity principle. We then propose some more tips on how to make better state picture designs.

We have conducted a case study in which we conjecture several non-trivial characteristics or lemma candidates of (the state machine formalizing) MCS protocol. Some lemma candidates have been actually used in a real case study in which it is formally proved that MCS protocol enjoys the mutual exclusion property [27]. We also propose some tips on how to conjecture state machine characteristics with SMGA.

We also report on one more case study. The state picture design has been made for Anderson shared-memory mutual exclusion protocol [1] and some characteristics of the protocol have been discovered based on the tips in the case study.

The contributions of the paper are as follows. (1) We propose the new state picture design of MCS protocol and assess it based on Gestalt principles. (2) We report on a core part of a formal verification in which the new state picture design and the SMGA tool have been largely contributed to its successful completion. (3) We summarize two groups of tips on how to make better state picture designs and how to conjecture state machine characteristics with SMGA, respectively.

The rest of the paper is organized as follows. Section 2 reports on some related work, evaluation of information visualization and Gestalt principles as a summary of conducting a literature review. We finally summarize how we assess the new state picture design of MCS protocol and the SMGA tool in the section. Section 3 mentions some preliminaries such as state machines, Maude, SMGA, MCS protocol and two Gestalt principles (proximity principle and similarity principle). Section 4 mentions formal specification of MCS protocol in Maude. In Section 5, we review the tips summarized in the paper [7], argue that there is some room to improve the state picture design of MCS protocol used in the paper [23] and propose a new state picture design of MCS protocol. We assess the new state picture design based on Gestalt principles in the section. Section 6 reports on a case study in which we have conjectured several non-trivial characteristics of MCS protocol with SMGA that uses

the new state picture design. Section 7 summarizes some lessons learned. Section 8 reports on one more case study. Finally, we conclude the present paper in Section 9.

The present paper is an extended and revised version of the paper [8] published at DMS-VIVA 2020. SMGA, the state picture designs and input files for SMGA used in the present paper are available at the following website: <https://bddang.bitbucket.io/>.

2 Related work

This section first reports on our literature review on systems visualization, evaluation of visualization and usability, and Gestalt principles. We compare SMGA with some existing systems visualization tools. Based on the literature review, this section finally describes our way to evaluate the new state picture design of MCS protocol and the SMGA tool.

2.1 Literature review on systems visualization

SMGA is a systems visualization tool. Other systems visualization tools have been developed. As usual, we have conducted a literature review of some systems visualization tools related to SMGA. We mention the tools and compare SMGA with them.

ShiViz [5] is a tool to visualize logs generated by distributed systems. Logs in this context are basically sequences of events, hosts that carry out the events and timestamps when the hosts carried out the events. The most important events are message sending and receiving. Feeding a log into ShiViz, ShiViz generates a diagram that is similar to a sequence diagram. The diagram helps human users comprehend what events precede and/or succeed what events, some patterns of message passings, etc. ShiViz has a functionality to find three typical patterns of message passings: (a) Request Response, (b) Broadcast and (c) Gather. The authors conducted three experiments to assess ShiViz: (1) a controlled experiment with a mix of 39 undergraduate and graduate students in which one group of participants studied distributed system executions using ShiViz and another group without ShiViz; (2) two homework assignments in a distributed systems course conducted by 70 students who used ShiViz to help them debug and understand their implementations; (3) a case study conducted by two systems researchers who were developing complex distributed systems to evaluate the end-to-end usefulness of ShiViz to developers in their work. The evaluation results are positive in that ShiViz helps students understand distributed systems better and even expert engineers in distributed systems are able to discover subtle errors lurking in distributed systems unless otherwise it would be infeasible or take much longer time to do so. The visualization used by ShiViz (still visualization) and the one (graphical animation) used by SMGA can be complementary. One possible future direction is to find a good balanced combination of the two approaches to (distributed) system visualization.

Artho, et al. [2] propose an extended version of UML sequence diagrams so that multithreaded programs, especially interactions among multiple threads, can be visualized. Threads are two aspects in Java programs: data (or objects) and executable units. UML sequence diagrams do not have enough descriptive capabilities for threads as executable units. The authors propose hexagonal diagrams for threads as executable units. Their extended sequential diagrams make it possible to describe what threads as execution units start or resume parts of participants (threads as objects) lifelines and terminates. It is possible to describe some dependencies among events carried out by threads as execution units, which can be used to describe lock acquisition and release by threads as execution units. The authors suppose that their extended sequential diagrams could be helpful for human users

to comprehend counterexamples generated by model checkers or runtime verification tools. SMGA can graphically animate counterexamples generated by Maude LTL model checker [24]. Their extended sequence diagrams are also still visualization, while our visualization is graphical animations.

VA4JVM [3] is a tool that can visualize outputs generated by Java Pathfinder (JPF). JPF outputs can be lengthy and is not easy-to-read especially when JPF finds something wrong, such as race condition and deadlock. VA4JVM can zoom some specific part of JPF outputs, filter such outputs, leaving more interesting fragments only, and highlight some fragments of such outputs that look more interesting so that human users could comprehend JPF outputs better. As above-mentioned, SMGA can graphically animate counterexamples generated by Maude LTL model checker [24]. Although Maude LTL model checker is a classical model checker and JPF is a software model checker, it would be worth considering some VA4JVM functionalities, such as zooming, filtering and highlighting and adopting some of them for a future version of SMGA.

Magee et al. [18] have proposed a way to visualize the behavior of a Labeled Transition System (LTS) described in FSP and developed a tool to support their proposed technique. One novelty of their approach to graphical animation of the LTS behavior is to use Timed Automata as formal semantics of animations. Their proposed technique makes it possible to compose multiple animations by composing Timed Automata. Their tool has been implemented with SceneBeans¹, a library of JavaBeans. As written, their visualization is graphical animation like ours. SceneBeans could be used to implement a future version of SMGA.

2.2 Literature review on evaluation of visualization and usability

It is truly crucial to reasonably evaluate any new techniques proposed and tools that supports the techniques including information visualization techniques and tools. Several papers on evaluation of information visualization techniques and tools have been published because it is not straightforward but rather hard to reasonably evaluate them. We have conducted a literature review of some such papers. We summarize them, which partially made us decide how to evaluate the new state picture design of MCS protocol and the SMGA tool.

Carpendale [11] mentions challenges of information visualization evaluation and two kinds of methods to evaluate information visualization: quantitative evaluation and qualitative evaluation. She writes that reasons why current evaluations are not convincing enough to encourage widespread adoption of information visualization tools include that information visualizations are often evaluated using small datasets, with university student participants, and using simple tasks.

Isenberg, et al. [16] conducted a systematic review of 581 papers published at IEEE Visualization (now IEEE Scientific Visualization) conference for 10 years (2012–2006, 2003, 2000 and 1997) to assess the state and historic development of evaluation practices as reported in those papers. They found that there was a steady increase in evaluation methods that include participants, either by evaluating their performances and subjective feedback or by evaluating their work practices and their improved analysis and reasoning capabilities using visual tools for the six years (2012–2007). They also found that generally the studies reporting requirements analyses and domain-specific work practices are too informally reported that hinders cross-comparison and lowers external validity.

¹<https://www.doc.ic.ac.uk/ltsa/scenebeans/>

Merino, et al. [20] conducted a systematic literature review of 181 full papers published at SOFTVIS/VISSOFT conferences on software visualization evaluation. They found that 68% of those papers lack of strong evaluation. They then propose guidelines to increase the evidence of the effectiveness of software visualization approaches, thus improving their adoption rate. Caine [9] conducted an analysis of all manuscripts published at CHI 2014 to determine local standards for sample size within the CHI community. She summarizes recommendations for authors as local standards in the CHI community on how to determine their sample size to evaluate their proposed techniques and/or tools. She also warns that relying on local standards should not be considered “best practice”.

Schmettow [25] points out that usability professionals and HCI researchers tend to use and/or want to have a magic number to determine the sample size to conduct usability studies, such as the 10 ± 2 rule of Hwang and Salvendy [15]. Resorting to such a magic number may make usability studies inaccurate for making predictions and underestimate required sample size as well. He recommend usability professionals and HCI researchers to conduct expensive, quantitatively managed studies when usability is critical. He, however, concludes the paper with the following sentence: “Most usability practitioners will likely continue to use strategies of iterative low-budget evaluation where quantitative statements are unreliable but also unnecessary”.

2.3 Literature review on Gestalt principles

Gestalt principles (or laws) (or principles of grouping) [26, 28, 29, 31] are a set of principles that govern humans perceiving an image as a whole, although the image is constituted of smaller visual objects and there do not seem any direct relations between the humans’ perception of the image and smaller visual objects. Note that “gestalt” is a German word meaning “form” or “group”. Gestalt principles have been used to design and assess visual interfaces, etc. in Computer Science.

Graphs are very common structures often used in many domains. Therefore, many software tools have been developed to draw graphs. Graphs may represent something dynamics, such as mobile networks. If so, whenever data represented as graphs change, the graph should change accordingly. Human users, however, may not follow such a change reasonably well, losing their mental maps. Nesbitt and Friedrich [21] have come up with how to visualize such a change by using Gestalt principles.

It is crucial to automatically identify some objects in a digital image for many purposes, such as security. To this end, it is necessary to make the boundaries between those objects and the others very distinct. Cao [10] invented a good algorithm for this aim by utilizing Gestalt principles and Helmholtz Principle, a quantitative version of the former [13].

Yalcinkaya and Singh [32] claim that information technologies have not been utilized reasonably well in AEC-FM industry, where AEC-FM stands for Architecture/Engineering/Construction (AEC) and Facilities Management (FM) Markets (see <https://www.ogc.org/>), in spite of many AEC-FM standards agreed in the industry, such as construction operations building information exchange (COBie) over its spreadsheet representation. They also claim that this is not just because of technical reasons caused by the standards but because of cognitive perception of COBie spreadsheet representation exchanged and processed by end-users. Then, they used Gestalt principles to analyze COBie spreadsheet representation, proposing more visual representation called VisualCOBie.

It is necessary to comprehend and track information or data that have been moved in cloud systems. It has been common to use logs and graphs that represent such logs. It has become the new trend for cloud users to comprehend “data provenance,” a historical record

of data and their origin. Then, Garae, et al. [14] proposed “User-centric Visualization of data provenance with Gestalt (UVisP),” a novel user-centric visualization technique for data provenance. UVisP aims at facilitating the missing link between data movements in cloud systems and end-users’ uncertain queries about their files’ security and life cycle in the cloud systems. It makes it possible for users to transform and visualize data provenance with implicit prior knowledge of Gestalt principles.

2.4 Our way to evaluate state picture design and SMGA

The purpose of SMGA partly overlaps that of ShiViz, which is to help human users comprehend distributed systems through visualization. The main purpose of SMGA is to help experts in Formal Methods conjecture lemma candidates that can be used for formal verification, however, while the main purpose of ShiViz is to help students and engineers understand implementations (or programs) of distributed systems. The developers of ShiViz conducted some experiments that involved human participants to assess it. We do not follow their approach to evaluating ShiViz so as to evaluate the new state picture design of MCS protocol and the SMGA tool partly because the main purposes of SMGA and ShiViz are different and partly because it is not straightforward to determine a good number of participants for our assessment purpose as we surveyed. Instead, we use Gestalt principles to assess the new state picture design of SMGA. Gestalt principles have been used to design and evaluate still images. Because SMGA produces graphical animations that are not still images but are dynamic images, there may be someone who wonders why we use Gestalt principles to evaluate the SMGA tool. We first need to make a state picture design to produce a graphical animation with SMGA. A state picture design is a template still image in which many visual objects are fixed. A small number of visual objects change in a series of concrete still images that are made from a state picture design and a sequence of states in text and that can be regarded as a movie film. We can use Gestalt principles to design and evaluate a state picture design because it is basically a still image. It is crucial to decide the positions of many visual objects that are fixed, for which we can use Gestalt principles. Even for visual objects that change their positions in a series of concrete still images, it is also important to decide the positions of the visual objects in each still image, for which we can use Gestalt principles as well. Note that Nesbitt and Friedrich [21] have used Gestalt principles to design animated visualization of network data that are represented as graphs, which is another case in which Gestalt principles have been used to design dynamic images.

It is a common practice to conduct a case study (or a few case studies) to evaluate newly proposed formal techniques and tools supporting them². We follow this practice to evaluate the SMGA tool because its main purpose is to help Formal Methods experts conjecture lemma candidates.

3 Preliminaries

This section describes some preliminaries needed to read what follows in the present paper: state machines, Maude, SMGA, MCS protocol and Gestalt principles. State machines are mathematical models used to formalize systems. Maude is a specification/programming language in which state machines can be described. Maude also refers to its processor equipped

²Please refer to some papers published at some Formal Methods conferences, such as FM and ICFEM.

with many functionalities. One of them is model checking. MCS protocol is a mutual exclusion protocol and used as one main example in the present paper. Gestalt principles can explain how humans perceive an image as a whole and are used to evaluate the new state picture design of MCS protocol and the SMGA tool.

3.1 State machines

A state machine $\mathcal{M} \triangleq \langle \mathcal{S}, \mathcal{I}, \mathcal{T} \rangle$ consists of a set $\mathcal{S}$ of states, a set $\mathcal{I} \subseteq \mathcal{S}$ of initial states and a binary relation $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{S}$ over states. The set $\mathcal{R}$ of reachable states with respect to $\mathcal{M}$ is inductively defined as follows: (1) for each $s \in \mathcal{I}$, $s \in \mathcal{R}$ and (2) for each $(s, s') \in \mathcal{T}$, if $s \in \mathcal{R}$, then $s' \in \mathcal{R}$. A state predicate p is an invariant property with respect to $\mathcal{M}$ if and only if $p(s)$ holds for all $s \in \mathcal{R}$.

A system can be formalized as a state machine. There are many possible ways to express states. We express states as braced associative-commutative collections of name-value pairs. Associative-commutative collections are called soups and name-value pairs are called observable components. So, states are expressed as braced soups of observable components. Let ob_1 , ob_2 and ob_3 be observable components and then the braced soup of the three observable components is expressed as $\{ob_1 \ ob_2 \ ob_3\}$, where the juxtaposition operator is used as the constructor of soups. Soups are associative-commutative collections in that the juxtaposition operator is associative and commutative. Therefore, $\{ob_3 \ ob_2 \ ob_1\}$, $\{ob_2 \ ob_1 \ ob_3\}$, etc. equal $\{ob_1 \ ob_2 \ ob_3\}$.

Let us consider a system such that we can observe two values val_1 and val_2 that are natural numbers and val_i for $i = 1, 2$ is updated as follows: if $val_i < val_{(i \% 2) + 1}$, then val_i is incremented and otherwise, it is decremented, where we suppose that if a value is 0 and decremented, the value does not change. Initially, val_1 and val_2 are set to 1 and 5, respectively. Let **Nat** be the set of natural numbers. Let us formalize the system, which we call TVS, as a state machine $\mathcal{M}_{TVS} \triangleq \langle \mathcal{S}_{TVS}, \mathcal{I}_{TVS}, \mathcal{T}_{TVS} \rangle$. $\mathcal{S}_{TVS}$ is $\{ \{ (val1 : x) \ (val2 : y) \} \mid x, y \in \mathbf{Nat} \}$, where $(val1 : x)$ and $(val2 : y)$ are observable components such that $val1$ and $val2$ are names, x and y are values that correspond to val_1 and val_2 and $:$ is used as a delimiter. $\mathcal{I}_{TVS}$ is $\{ \{ (val1 : 1) \ (val2 : 5) \} \}$. For all $x, y \in \mathbf{Nat}$, if $x < y$, then $\mathcal{T}_{TVS}$ has the following:

$$\{ \{ (val1 : x) \ (val2 : y) \}, \{ (val1 : x + 1) \ (val2 : y) \} \}$$

if $x \geq y$, then $\mathcal{T}_{TVS}$ has the following:

$$\{ \{ (val1 : x) \ (val2 : y) \}, \{ (val1 : \text{dec}(x)) \ (val2 : y) \} \}$$

if $y < x$, then $\mathcal{T}_{TVS}$ has the following:

$$\{ \{ (val1 : x) \ (val2 : y) \}, \{ (val1 : x) \ (val2 : y + 1) \} \}$$

if $y \geq x$, then $\mathcal{T}_{TVS}$ has the following:

$$\{ \{ (val1 : x) \ (val2 : y) \}, \{ (val1 : x) \ (val2 : \text{dec}(y)) \} \}$$

where $\text{dec}(0) = 0$ and $\text{dec}(n + 1) = n$ for all $n \in \mathbf{Nat}$.

3.2 Maude

Maude [12] is a specification and programming language based on rewriting logic. Its system (or environment) is also called Maude. Maude makes it possible to naturally describe soups, observable components and braced soups of observable components. $\mathcal{T}$ is described in terms of rewrite rules when specifying $\mathcal{M}$ in Maude. $\mathcal{T}_{TVS}$ is described as follows:


```

cr1 [inc-val1] : {(val1: X) (val2: Y)}
=> {(val1: (X + 1)) (val2: Y)} if X < Y .
cr1 [dec-val1] : {(val1: X) (val2: Y)}
=> {(val1: dec(X)) (val2: Y)}
if not (X < Y) .
cr1 [inc-val2] : {(val1: X) (val2: Y)}
=> {(val1: X) (val2: (Y + 1))} if Y < X .
cr1 [dec-val2] : {(val1: X) (val2: Y)}
=> {(val1: X) (val2: dec(Y))}
if not (Y < X) .

```

cr1 stands for conditional rewrite rule and is used to declare a conditional rewrite rule. An unconditional rewrite rule is declared with *rl* instead. *inc-val1* is the name or label given to the first rewrite rule. *X* and *Y* are Maude variables of **Nat**. Rule *inc-val1* says that if *X* is less than *Y*, a state denoted $\{(val1: X) (val2: Y)\}$ can change to another state denoted $\{(val1: (X + 1)) (val2: Y)\}$. The other three rewrite rules can be interpreted likewise.

Maude makes it possible to conduct reachability analysis for state machines and model check that state machines enjoy invariant properties. We may conjecture that val_1 is always less than or equal to 5 for $\mathcal{M}_{TVS}$. We can check it by the following search command:

```

search [1] in TVS : init
=>* {(val1: X) OCs} such that X > 5 .

```

where *init* is $\{(val1: 1) (val2: 5)\}$ and *OCs* is a Maude variable of observable component soups. The search command tries to find a reachable state such that *X* is greater than 5. It does not find any such a state and then we can conclude that $val_1 \leq 5$ is an invariant property with respect to $\mathcal{M}_{TVS}$. Let us model check that $val_1 < 5$ is an invariant property with respect to $\mathcal{M}_{TVS}$, which can be done by the following search command:

```

search [1] in TVS : init
=>* {(val1: X) OCs} such that X >= 5 .

```

It finds a reachable state $\{(val1: 5) (val2: 5)\}$ such that *X* is 5 and then $X \geq 5$ holds. Therefore, $val_1 < 5$ is not an invariant property with respect to $\mathcal{M}_{TVS}$.

3.3 State machine graphical animation (SMGA)

SMGA [22] is a tool that basically takes a sequence of states in text and plays a graphical animation by regarding the sequence as a movie film. To regard a sequence of states in text as a movie film, we are supposed to design state pictures. One possible state picture design for $\mathcal{M}_{TVS}$ is shown in Fig. 1. There are six rectangles on the state picture design such that each rectangle has a natural number (0, 1, 2, 3, 4 or 5) at the left upper corner. Let the rectangle that has *n* be called rectangle *n*. When val_i for $i = 1, 2$ is *n* that is 0,

Fig. 1 A state picture design for TVS

Fig. 2 First six state pictures of a graphical animation for TVS

1, 2, 3, 4 or 5, a circle on which val_i is written appears on rectangle n . Figure 2 shows the first six state pictures of a graphical animation for TVS. The state pictures allow us to visually/graphically perceive what value each val_i for $i = 1, 2$ has. We need to comprehend TVS to some extent in order to design such state pictures because otherwise we do not know how many rectangles should be prepared.

3.4 MCS protocol

MCS protocol [19] is a shared-memory mutual exclusion protocol invented by Mellor-Crummey and Scott. Partly because its variants had been used in Java VMs, the 2006 Edsger W. Dijkstra Prize in Distributed Computing went to their paper³. It can be described in Algol-like pseudo-code as shown in Fig. 3. It uses one global variable $glock$ and three local variables $next_p$, $pred_p$ and $lock_p$ for each process p . Process IDs are stored in $glock$, $next_p$ and $pred_p$, while Boolean values are stored in $lock_p$. In this paper, initially, $glock$, $next_p$ and $pred_p$ are set to nop, while $lock_p$ is set to false, where nop is a special ID that is

³www.podc.org/dijkstra/2006-dijkstra-prize

Fig. 3 MCS protocol in Algol-like pseudo-code

```

rs : "Remainder Section"
l1 :  $next_p := \text{nop}$ ;
l2 :  $pred_p := \text{fetch\&store}(glock, p)$ ;
l3 : if  $pred_p \neq \text{nop}$  {
l4 :    $lock_p := \text{true}$ ;
l5 :    $next_{pred_p} := p$ ;
l6 :   repeat while  $lock_p$ ; }
cs : "Critical Section"
l7 : if  $next_p = \text{nop}$  {
l8 :   if  $\text{comp\&swap}(glock, p, \text{nop})$ 
l9 :   goto rs;
l10 : repeat while  $next_p = \text{nop}$ ; }
l11 :  $lock_{next_p} := \text{false}$ ;
l12 : goto rs;

```

different from any real process IDs. $lock_{next_p}$ is $lock_q$ such that $next_p = q$ and $next_{pred_p}$ is $next_q$ such that $pred_p = q$. Note that because the protocol is used for shared-memory computers, any process p can read and write $lock_q$ and $pred_q$ even though q is different from p . The protocol uses two atomic operations (or instructions): fetch&store and comp&swap. For a variable v and a value a , fetch&store(v, a) atomically does the following: it sets v to a and returns the old value of v . For a variable v and two values a & b , comp&swap(v, a, b) atomically does the following: if the value stored in v equals a , then v is set to b and true is returned; otherwise false is just returned. MCS protocol uses a virtual queue basically composed of process IDs by using $next_p$. Figure 4a shows the virtual queue that consists of p_2, p_1 and p_3 in this order such that all of them have been completely put into it. $glock$ always refers to the bottom element whenever the virtual queue is not empty, while it is nop whenever the virtual queue is empty. Enqueuing and dequeuing for the virtual queue are

Fig. 4 Virtual queue used in MCS protocol

not atomic. Therefore, there may be some elements that have not yet been completely put into it. Figure 4b shows the virtual queue that consists of p_2 , p_1 and p_3 in this order such that p_1 has not yet been completely put into it and will set $next_{p_1}$ to p_2 by using $pred_{p_1}$, where $pred_{p_1} = p_2$. p_1 will conduct $next_{pred_p} := p$; at line 15 in the pseudo-code shown in Fig. 3, where because p is p_1 , $pred_{p_1}$ is p_2 and $next_{p_2}$ will be p_1 .

3.5 Gestalt principles

We use examples to introduce two Gestalt principles that are used to assess state picture designs in the present paper: proximity principle and similarity principle. Let us take a look at the following image:

Six stars are aligned horizontally such that the distance of each adjacent pair of stars is the same. Then, we perceive that there are just six stars and did not recognize any sub-groups in it. Let us change the layout a little bit as follows:

There are still six stars but we can recognize that there are three sub-groups each of which consists of two stars. This is because the distance between each adjacent pair of sub-groups is greater than the distance between the two stars in each sub-group. This visual perception is called “proximity principle”.

Let us change the first image of six stars in a different way. The changed image is as follows:

We do not change the layout but change the color of some stars. We can recognize that there are three different kinds of stars because there are three different colors. Thus, we may perceive that there are three sub-groups each of which consists of the same color stars. This visual perception is called “similarity principle”.

4 Specification of MCS protocol in Maude

We suppose that there are three processes that participate in MCS protocol. Let $\mathcal{M}_{\text{MCS}}$ formalize MCS protocol. Each state in $\mathcal{S}_{\text{MCS}}$ is expressed as follows:

$$\begin{aligned} &\{(glock : bp) \\ &\quad (pc[p1] : l_1)(pred[p1] : pp_1)(lock[p1] : b_1)(next[p1] : np_1) \\ &\quad (pc[p2] : l_2)(pred[p2] : pp_2)(lock[p2] : b_2)(next[p2] : np_2) \\ &\quad (pc[p3] : l_3)(pred[p3] : pp_3)(lock[p3] : b_3)(next[p3] : np_3)\} \end{aligned}$$

where bp , pp_i and np_i for $i = 1, 2, 3$ are process IDs, l_i for $i = 1, 2, 3$ is a label, such as rs, ll and cs, and b_i for $i = 1, 2, 3$ is a Boolean value. Initially, bp , pp_i and np_i are nop, l_i is rs and b_i is false. $\mathcal{I}_{\text{MCS}}$ consists of one state. Let `init` equal the initial state.

$\mathcal{T}_{\text{MCS}}$ is described in Maude as follows:

```

rl [want] : {(pc[P]: rs) OCs}
=> {(pc[P]: l1) OCs} .
rl [stnxt] : {(pc[P]: l1) (next[P]: Q) OCs}
=> {(pc[P]: l2) (next[P]: nop) OCs} .
rl [stprd] : {(glock: Q) (pc[P]: l2) (pred[P]: Q1) OCs}
=> {(glock: P) (pc[P]: l3) (pred[P]: Q) OCs} .
rl [chprd] : {(pc[P]: l3) (pred[P]: Q) OCs}
=> {(pc[P]: (if Q == nop then cs else l4 fi))
(pred[P]: Q) OCs} .
rl [stlck] : {(pc[P]: l4) (lock[P]: B) OCs}
=> {(pc[P]: l5) (lock[P]: true) OCs} .
rl [stnpr] : {(pc[P]: l5) (pred[P]: Q) (next[Q]: Q1) OCs}
=> {(pc[P]: l6) (pred[P]: Q) (next[Q]: P) OCs} .
rl [chlck] : {(pc[P]: l6) (lock[P]: false) OCs}
=> {(pc[P]: cs) (lock[P]: false) OCs} .
rl [exit] : {(pc[P]: cs) OCs}
=> {(pc[P]: l7) OCs} .
rl [rpnxt] : {(pc[P]: l7) (next[P]: Q) OCs}
=> {(pc[P]: (if Q == nop then l8 else l11 fi))
(next[P]: Q) OCs} .
rl [chglk] : {(glock: Q) (pc[P]: l8) OCs}
=> {(glock: (if Q == P then nop else Q fi))
(pc[P]: (if Q == P then l9 else l10 fi)) OCs} .
rl [go2rs] : {(pc[P]: l9) OCs}
=> {(pc[P]: rs) OCs} .
cr1 [rpnxt2] : {(pc[P]: l10) (next[P]: Q) OCs}
=> {(pc[P]: l11) (next[P]: Q) OCs}
if Q /= nop .
rl [stlnx] : {(pc[P]: l11) (next[P]: Q)
(lock[Q]: B) OCs}
=> {(pc[P]: l12) (next[P]: Q) (lock[Q]: false) OCs} .
rl [go2rs2] : {(pc[P]: l12) OCs}
=> {(pc[P]: rs) OCs} .

```

where OCs is a Maude variable of observable component soups, P, Q and Q1 are Maude variables of process IDs, and B is a Maude variable of Boolean values. *if b then x else y fi* equals *x* if *b* equals true and *y* if *b* equals false.

5 State picture design

States are expressed as braced soups of observable components, where observable components are name-value pairs. Therefore, it would be possible to automatically produce a straightforward state picture design. Let us take TVS as an example. A straightforward state picture design that could be automatically generated is shown in Fig. 5. Figure 6 shows the

Fig. 5 A straightforward state picture design for TVS

first six straightforward state pictures of a graphical animation for TVS. The six straightforward state pictures of the graphical animation for TVS is almost the same as the text representation of the six states:

```
(val1: 1 val2: 5) ||
(val1: 2 val2: 5) ||
(val1: 2 val2: 4) ||
(val1: 2 val2: 3) ||
(val1: 3 val2: 3) ||
(val1: 3 val2: 2)
```

where `||` is the constructor of sequences. On the visual/graphical representation of each TVS state shown in Figs. 1 and 2, we can perceive that there is one group that consists of six rectangles due to both proximity and similarity principle. We also recognize that there are two circles on which `val1` and `val2` are shown. The two visual objects represent something similar because both of them are circle but they are not exactly the same because they are in different colors due to similarity principle. Each visual circle object is located on one of the six rectangles, which can be interpreted as follows: each `val` can have one of the six values (0, 1, 2, 3, 4, 5). Looking at the rectangles on which the two circles are located, we can perceive how different values `val1` and `val2` have. For two values x, y from the six values (0, 1, 2, 3, 4, 5), let the distance between x and y modulo 6 (or the distance (mod 6) between x and y) be the smallest k such that $x + k \equiv y \pmod{6}$ or $x \equiv y + k \pmod{6}$, where $k \in \{0, 1, 2, 3, 4, 5\}$. The bigger the distance between the locations of the two circles is, the bigger the difference (mod 6) between the values stored in `val1` and `val2` is. Let us confess that we need to comprehend TVS to some extent so that we can make the state picture design shown as in Figs. 1 and 2. On the other hand, just looking at each picture as shown in Fig. 6 or the text representation of each TVS

Fig. 6 First six straightforward state pictures of a graphical animation for TVS

glock: nop		
pc[p1]: rs	pc[p2]: rs	pc[p3]: rs
next[p1]: nop	next[p2]: nop	next[p3]: nop
pred[p1]: nop	pred[p2]: nop	pred[p3]: nop
lock[p1]: false	lock[p2]: false	lock[p3]: false

Fig. 7 A straightforward state picture design for MCS protocol

state does not immediately allow us to recognize that there are six possible values each val can have.

5.1 New State picture design of MCS protocol

It is also possible to automatically generate a straightforward state picture design for MCS protocol, such as the one shown in Fig. 7. State pictures generated from the state picture design, such as the one shown in Fig. 8 are almost the same as states in text:

```
{ (glock: p1)
  (pc [p1] : 15) (next [p1] : nop) (pred [p1] : p2) (lock [p1] : true)
  (pc [p2] : 16) (next [p2] : nop) (pred [p2] : p3) (lock [p2] : true)
  (pc [p3] : 18) (next [p3] : p2) (pred [p3] : nop) (lock [p3] : false) }
```

We summarize some tips on how to design state pictures for mutual exclusion protocols in our paper [7] published in 2019. The existing tips on state picture design are summarized in Table 1.

Nguyen and Ogata [23] made the state picture design shown in Fig. 9 for MCS protocol, which follows the tips. A state picture generated from the state picture design is shown in Fig. 10. The state picture allows us to immediately realize that processes p1, p2 and p3 are located at 15, 16 and 18, respectively. Nguyen and Ogata [23] conducted a case study in which several characteristics of MCS protocol can be discovered by observing graphical animations of MCS protocol such that each state picture used in the graphical animation is generated from the state picture design. For example, one of the characteristics found is as follows:

No state such that a process is at cs, 17, 18, 110, or 111 and another process is at cs, 17, 18, 110, or 111.

Let the characteristic be called Characteristic 0 in this paper.

glock: p1		
pc[p1]: 15	pc[p2]: 16	pc[p3]: 18
next[p1]: nop	next[p2]: nop	next[p3]: p2
pred[p1]: p2	pred[p2]: p3	pred[p3]: nop
lock[p1]: true	lock[p2]: true	lock[p3]: false

State 17 : (glock: p1) (pc[p1]: 15) (pc[p2]: 16) (pc[p3]: 18) (next[p1]: nop) (next[p2]: nop) (next[p3]: p2) (lock[p1]: true) (lock[p2]: true) (lock[p3]: false) (pred[p1]: p2) (pred[p2]: p3) (pred[p3]: nop)

Fig. 8 A straightforward state picture for MCS protocol

Table 1 Existing tips on state picture design [7]

SPD-T 1	To recognize what sections there are at which each process or node is located, allocate the pane (or place) for each section such that the relations among the sections are visually/graphically perceived and display some diagram, such as a circle on which a process or node ID is written, on the designated pane.
SPD-T 2	To recognize what pieces of information, such as the network for the Suzuki-Kasami protocol and variable <i>locked</i> for the test&set protocol, are shared by all processes or nodes, allocate the pane (or place) for each such piece of information such that we can visually/graphically perceive they are shared by all processes and nodes and display them on the designated panes adequately.
SPD-T 3	To recognize whether there are some that are more crucial than the others among the shared resources, such as the messages that have been just put into and deleted from the network, prepare the panes (or places) for them and display them there adequately
SPD-T 4	To recognize what pieces of information are owned by each process or node, allocate the panes (or places) for them to make it possible to visually/graphically perceive what pieces of information are owned by what processes or nodes and display them on the designated panes adequately.

SPD-T stands for State Picture Design Tip

Although the state picture shown in Fig. 10 also allows us to notice the values stored in the global variable *glock* and the three local variables *next_p*, *pred_p* and *lock_p* for each process *p*, their representations on the state picture are almost the same as the text representation. Since SMGA requires and/or permits human users to make state picture designs, the representations must be able to be visually/graphically perceivable. To this end, we came up with the state picture design shown in Fig. 11. Figure 12 shows a state picture generated from the state picture design.

The design of the *glock* representation used in Fig. 9 is as follows:

Fig. 9 An old state picture design for MCS protocol

Fig. 10 An state picture instance for MCS protocol

The value of *glock* is nop, p1, p2 or p3. Regardless of the value, the value is displayed on the same place. For example, when the value is p1, it is displayed as follows:

The design of the *glock* representation used in Fig. 11 is as follows:

If the value is nop, nothing is displayed on the rectangle or pane for *glock*. If the value is p1, p1 is displayed at the left-most place of the rectangle for *glock*. If the value is p2, p2 is displayed at the middle place of the rectangle for *glock*. If the value is p3, p3 is displayed at the right-most place of the rectangle for *glock*.

Fig. 11 A new state picture design for MCS protocol

Fig. 12 A new state picture instance for MCS protocol

at the right-most place of the rectangle for *glock*. Because p1, p2 and p3 are processes (or process IDs), we use exactly the same representation as those used for the three processes appearing on the 14 sections. For example, when the value is p1, it is displayed as follows:

The design of the *next_p*, *pred_p* and *lock_p* representations for each process *p* used in Fig. 9 is as follows:

next[p1] : nop	next[p2] : nop	next[p3] : nop
pred[p1] : nop	pred[p2] : nop	pred[p3] : nop
lock[p1] : false	lock[p2] : false	lock[p3] : false

Regardless of the values of *next_p*, *pred_p* and *lock_p*, their values are displayed on the same places. For example, when *next_{p1}* is nop, *pred_{p1}* is p2, *lock_{p1}* is true, *next_{p2}* is nop, *pred_{p2}* is p3, *lock_{p2}* is true, *next_{p3}* is p2, *pred_{p3}* is nop and *lock_{p3}* is false, those values are displayed as follows:

next[p1] : nop	next[p2] : nop	next[p3] : p2
pred[p1] : p2	pred[p2] : p3	pred[p3] : nop
lock[p1] : true	lock[p2] : true	lock[p3] : false

The design of the *next_p*, *pred_p* and *lock_p* representations for each process *p* used in Fig. 11 is as follows:

The $next_p$ representation appears at the right-most place, where there are three rectangles, the first, second and third ones of which from top are used for p_1 , p_2 and p_3 , respectively. For example, if the value of $next_{p_1}$ is nop , nothing is shown on the first rectangle; if the value is pi , the circle on which pi is written is shown at the designated place on the first rectangle. The $lock_p$ representation appears at the middle place, which also indicates that the first, second and third rows are used for p_1 , p_2 and p_3 , respectively. When $lock_{pi}$ is true, the background color of pi is red; otherwise the color is non-red (or light blue). The $pred_p$ representation appears at the left-most place, where there are three rectangles, the first, second and third ones of which from top are used for p_1 , p_2 and p_3 , respectively. For example, if the value of $pred_{p_1}$ is nop , nothing is shown on the first rectangle; if the value is pi , the circle on which pi is written is shown at the designated place on the first rectangle. For instance, when $next_{p_1}$ is nop , $pred_{p_1}$ is p_2 , $lock_{p_1}$ is true, $next_{p_2}$ is nop , $pred_{p_2}$ is p_3 , $lock_{p_2}$ is true, $next_{p_3}$ is p_2 , $pred_{p_3}$ is nop and $lock_{p_3}$ is false, those values are displayed as follows:

5.2 Evaluation of state picture design based on Gestalt principles

On both the old state picture design and the new state picture design of MCS protocol, we can perceive that there are 14 rectangles aligned along the edge of the whole image due to similarity principle. The 14 rectangles represent the 14 sections on which processes are located and each of the rectangles has its name on it, such as rs , $l1$ and cs . An arrow-shape visual object is shown between each pair of adjacent rectangles, meaning that processes basically move from one section to the other. Processes are represented as circles on which their IDs, such as $p1$, are shown. Taking a look at state pictures immediately allows us to recognize which sections processes are located. On the old state pictures, all circles representing processes are in one color, while on the new state pictures different colors are used to make it clear to distinguish different processes due to similarity principle. On both the old state picture design and the new state picture design of MCS protocol, the visual representation of $glock$ is arranged a bit far from the visual representations of the three local variables owned by processes, which makes it clear that $glock$ can be perceived as one independent group due to proximity principle.

On the new state picture design of $next_p$, $pred_p$ and $lock_p$ for each process p , we can perceive that there are three groups due to similarity principle. The three groups correspond to $pred_p$, $lock_p$ and $next_p$ from left on the new state picture design. We also perceive that there are also three different groups due to proximity principle. The three different groups correspond to $p1$, $p2$ and $p3$ from top on the new state picture design.

Because the types of $glock$, $next_p$ and $pred_p$ are the same, process IDs or nop , we use the same visual representation for them so that we can perceive it due to similarity principle. When such a variable has a process ID, a circle on which the process ID is shown appears at the designated place on the rectangle that represents the variable. Since $p1$, $p2$ and $p3$ are processes, each of them is represented as a circle due to similarity principle. Since they are different processes, however, we use three different colors for the three different processes due to similarity principle. When such a variable has nop , nothing appears on the rectangle

that represents the variable. This is because nop is different from process IDs and should be distinguished from them. We can recognize that such a variable has nop thanks to similarity principle. Because of the visual representations of $glock$, $next_p$ and $pred_p$, we can perceive whether among those variables have a same value (namely a same process ID or nop) or different values.

The type of $lock_p$ is Boolean and then it has either true or false. We use two different colors to represent the two different values. Red is used to represent true, while non-red (or light blue) is used to represent false. This is because when $lock_p$ is true, process p is supposed to wait somewhere to proceed to the critical section and when $lock_p$ is false, p is allowed to enter the critical section. We can perceive which $lock_p$ is true or false thanks to similarity principle.

5.3 The number of processes

We supposed that there are three processes participating in MCS protocol. When we would like to change the number of processes, we need to revise the state picture design a little bit. However, the essence of the state picture design can be used even when the number of processes changes. For example, when there are seven processes participating in MCS protocol, the state picture design of MCS protocol is as shown in Fig. 13. Figure 14 shows a state picture generated from the state picture design. The state picture design as shown in Fig. 13 can be used for the case in which there are seven or less processes participating in MCS protocol. As you can imagine, although we could increase the number of processes, if there are many processes, such as 100, then state pictures become less understandable. Let us recall the main purpose of the SMGA tool: helping experts in Formal Methods conjecture lemma candidates or state machine characteristics that can be used for formal verification by visually/graphically observing graphical animations. When writing formal proofs, it suffices to suppose that there are 3 or 4 arbitrary processes. Therefore, it would suffice that SMGA can deal with 3 or 4 processes reasonably well for our purpose.

Fig. 13 A new state picture design for MCS protocol with 7 processes

Fig. 14 A new state picture instance for MCS protocol with 7 processes

6 Characteristic discovery based on new state pictures

The authors of the present paper and some others have conducted a case study in which it is formally proved that MCS protocol enjoys the mutual exclusion property, namely that there is always at most one process in the critical section. The case study is reported in the paper [27] accepted by 2020 Asia-Pacific Software Engineering Conference (APSEC 2020). The new state picture design and the SMGA tool contributed to the case study to some extent. This section describes how the new state picture design and the SMGA tool contributed to the case study.

Carefully observing graphical animations for MCS protocol in which new state pictures, such as Fig. 12, are used, we realize that there is always at most one process at cs, l7, l8, l10 or l11. This is exactly the same as Characteristic 0 discovered based on old state pictures, such as Fig. 10. We call these sections (cs, l7, l8, l10 and l11) CS region. We also notice that there exists at most one process p such that p is located at l3 and $pred_p$ is nop and there exists at most one process p such that p is located at l6 and $lock_p$ is false. Moreover, if p is located at l3 and $pred_p$ is nop, there is no process in CS region and there is no process q at l6 such that $lock_q$ is false, and if p is located at l6 and $lock_p$ is false, there is no process in CS region and there is no q at l3 such that $pred_q$ is nop. l3 and l6 are only the sections from which processes enter CS region. We call CS region plus l3 and l6 extended CS region. The first characteristic that can be conjectured by carefully observing graphical animations for MCS protocol in which new state pictures, such as Fig. 12, are used is as follows:

- Characteristic 1: There exists at most one process in extended CS region except for processes p such that (1) p is located at l3 and $pred_p$ is not nop and (2) p is located at l6 and $lock_p$ is not false.

Extended CS region is one key concept that captures one important aspect of MCS protocol and Characteristic 1 is very crucial in that several other characteristics can be discovered based on the characteristic. Figure 15 shows some state pictures that capture Characteristic 1. On the top left state picture, there are two processes p1 and p2 in extended CS region such that both are located at l6, $lock_{p1}$ is true and $lock_{p2}$ is false and therefore p2 is only the process in extended CS region in the sense of Characteristic 1 because p1 satisfies condition (2) in Characteristic 1. On the top right state picture, there are two processes p1 and p3 in extended CS region such that both are located at l3, $pred_{p1}$ is p2 and $pred_{p3}$ is nop and therefore p3 is only the process in extended CS region in the sense of Characteristic 1 because p1 satisfies condition (1) in Characteristic 1. Each of the other five state pictures shows that there exists one process at one section of CS region and for all processes q located at l3 and l6 if any, $pred_q$ is not nop and $lock_q$ is not false, respectively.

The first author of the present paper started the formal proof that MCS protocol enjoys the mutual exclusion property and got stuck somewhere for several months. One reason why the first author of the present paper got stuck for such a long time is as follows:

eq inv40(S, P)

= (($pc(S, P) = l3$ or $pc(S, P) = l4$ or $pc(S, P) = l5$ or $pc(S, P) = l6$ or $pc(S, P) = cs$ or $pc(S, P) = l7$ or $pc(S, P) = l8$ or $pc(S, P) = l10$ or $pc(S, P) = l11$) implies ($glock(S) = nop$) = false) .

The equation defines a state predicate inv40(S, P), where S is a variable for states and P is a variable for process IDs. $pc(S, P)$ denotes the section where a process P is located in a state S . $glock(S)$ denotes the value stored in $glock$ in a state S . or and implies denote the logical disjunction and the logical implication, respectively. false can be interpreted as usual. inv40(S, P) says that if P is located at l3, l4, l5, l6, cs, l7, l8, l10 or l11, then $glock$ is not nop. inv40(S, P) is likely to be an invariant with respect to MCS protocol and then the first author of the present paper started the proof. However, he did not converge the proof attempt because he encountered a situation in which he needed to use an unbounded number of lemmas that are similar but different a little bit. Please refer to the APSEC 2020 paper [27] in detail.

The key to tackle the situation is Extended CS region or Characteristic 1. The premise part of inv40(S, P) is similar to Characteristic 1 but not exactly the same. We can recognize that $glock$ is not nop on each state picture shown in Fig. 15. Therefore, we can guess the second characteristic of MCS protocol as follows:

- Characteristic 2: If there exists a process in extended CS region in the sense of Characteristic 1, then $glock$ is always not nop.

Characteristic 2 can be rephrased as follows:

eq inv4(S, P)

= (($(pc(S, P) = l3$ and $prede(S, P) = nop)$ or ($pc(S, P) = l6$ and $lock(S, P) = false$) or $pc(S, P) = cs$ or $pc(S, P) = l7$ or $pc(S, P) = l8$ or $pc(S, P) = l10$ or $pc(S, P) = l11$) implies ($glock(S) = nop$) = false) .

The difference between inv4(S, P) and inv40(S, P) is as follows: $pc(S, P) = l3$ is revised to $pc(S, P) = l3$ and $prede(S, P) = nop$, $pc(S, P) = l6$ is revised to $pc(S, P) = l6$ and $lock(S, P) = false$ and $pc(S, P) = l4$ & $pc(S, P) = l5$

Fig. 15 Some pictures for extended CS region

are deleted. The premise part of $\text{inv}_4(S, P)$ is now exactly the same as Characteristic 1. Although there is a small difference between $\text{inv}_4(S, P)$ and $\text{inv}_{40}(S, P)$, it is possible to successfully prove that $\text{inv}_4(S, P)$ is an invariant with respect to MCS protocol without any difficulties in contrast to $\text{inv}_{40}(S, P)$. Please refer to the APSEC 2020 paper [27] on how to formally prove that $\text{inv}_4(S, P)$ is an invariant with respect to MCS protocol.

We describe some more characteristics discovered based on the new state picture design. Focusing on a process in CS region or extended CS region, we can recognize the following characteristics:

- Characteristic 3.1: Whenever there is a process at l10, there is at least one process at l3, l4, l5 or l6;
- Characteristic 3.2: Whenever there is a process at l11, there is at least one process at l6.

The third-row right state picture of Fig. 15 is an example of Characteristic 3.1 and the bottom state picture of Fig. 15 is an example of Characteristic 3.2.

Carefully observing the $next_p$ and $pred_p$ representations for each process p , nothing may be displayed and the circle on which q that is different from p is written may be displayed but the circle on which p is written is never displayed. Thus, we can realize the following characteristics:

- Characteristic 4.1: The value of $next_p$ for each process p is never p ;
- Characteristic 4.2: The value of $pred_p$ for each process p is never p .

Some relations between two observable components can be discovered by carefully observing graphical animations, from which some characteristics can be conjectured. A relation between the $glock$ observable component and the $pc[p]$ observable component can be perceived by graphical animations and allows us to conjecture the following characteristics:

- Characteristic 5.1: If $glock$ is nop, then there is no process at l3, l4, l5 or l6 or in CS region;
- Characteristic 5.2: If $glock$ is a process p , then p is located at l3, l4, l5 or l6 or in CS region.

A relation between the $pred[p]$ observable component and the $pc[p]$ observable component allows us to conjecture the following characteristic:

- Characteristic 6: If $pred_p$ for each process p is nop, then p is never located at l4, l5, l6 or l12.

A relation between the $pc[p]$ observable component and the $next[p]$ observable component allows us to conjecture the following characteristics:

- Characteristic 7.1: If each process p is located at l2 or l9, then $next_p$ is nop;
- Characteristic 7.2: If $next_p$ for each process p is nop, then p is not located at l11 or l12.

A relation between the $lock[p]$ observable component and the $pc[p]$ observable component allows us to conjecture the following characteristics:

- Characteristic 8.1: If $lock_p$ for each process p is true, then p is located at l5 or l6;
- Characteristic 8.2: If each process p is located at l5, then $lock_p$ is true.

Note that when a process p is located at l6, another process may set $lock_p$ to false.

There are cases such that it is necessary to fix the values of some observable components so as to discover some similarities of multiple states and/or some relations among observable components. It is not straightforward to do so by observing graphical animations because we need to remember states in which the former observable components have the fixed values. For example, it must not be reasonable to remember all states in a graphical

animation such that $pred_{p1}$ is $p2$ and $p2$ is located at $rs, l1, l2, l9$ or $l12$. One possible remedy for it is to use the Find Patterns functionality of SMGA [7, 22, 23]. Given a condition written by human users, the Find Patterns functionality finds all states in an input sequence of states such that they satisfy the condition. When we would like to find all states in an input sequence of states such that $pred_{p1}$ is $p2$ and $p2$ is located at $rs, l1, l2, l9$ or $l12$, it suffices to write the following condition:

```
state['pred[p1]'] == 'p2' &&
(state['pc[p2]'] == 'l1' ||
 state['pc[p2]'] == 'l2' ||
 state['pc[p2]'] == 'rs' ||
 state['pc[p2]'] == 'l9' ||
 state['pc[p2]'] == 'l12')
```

$state['pred[p1]'] == 'p2'$ says that $pred_{p1}$ equals $p2$. $\&\&$ and $||$ are logical conjunction and disjunction, respectively. Figure 16 shows three state pictures among the states discovered by the Find Patterns functionality for the condition.

It does not suffice, however, to use the condition above-mentioned so as to conjecture non-trivial characteristics. This is because the pair $(p1, p2)$ is one possible combination and there are five more combinations to consider: $(p1, p3)$, $(p2, p1)$, $(p2, p3)$, $(p3, p1)$ and $(p3, p2)$. Note that we do not need to take the three combinations $(p1, p1)$, $(p2, p2)$ and $(p3, p3)$ into account because of Characteristic 4.2. Let (p, q) be each of the six combinations to consider. Carefully observing all states discovered by the Find Patterns functionality for the six combinations to consider, we can conjecture some non-trivial characteristics:

- Characteristic 9.1: If $pred_p$ is q and q is located at $rs, l1, l2, l9, l11$ or $l12$, then p is not located at $l3, l4$ or $l5$;
- Characteristic 9.2: If $pred_p$ is q and $glock$ is q , then p is not located at $l3, l4$ or $l5$;
- Characteristic 9.3: If $pred_p$ is q , $next_q$ is not nop and p is located at $l3, l4, l5, l6, cs, l7, l8$ or $l10$, then q is not located at $l3, l4$ or $l5$.

We notice that there are both states in which $next_q$ is nop and those in which $next_q$ is not nop among the states found by the Find Patterns functionality of the condition of Characteristic 9.1, while there are only states in which $next_q$ is nop among the states found by the Find Patterns functionality of the condition of Characteristic 9.2. Then, finding the states with the Find Patterns functionality of the second sub-condition only of Characteristic 9.2 because the first sub-condition is shared by both characteristics, we realize that $next_q$ is nop in all of them. Therefore, we come up with the following characteristic:

- Characteristic 10: If $glock$ is a process p (or equivalently non-nop), $next_p$ is nop.

We have used the Maude reachability analyzer (or the search command) to model check that all characteristics conjectured in this section are invariant properties with respect to the state machine formalizing MCS protocol and have not found any counterexamples for each characteristic.

Moreover, we have formally proved that Characteristics 2, 4, 7.1, 8.2, 9 and 10 are invariants with respect to the state machine formalizing MCS protocol. Please refer to the APSEC 2020 paper [27] in detail.

7 Some lessons learned

Let us summarize some lessons learned through the case study with SMGA and MCS protocol.

7.1 Some new tips on state picture design

One important lesson learned can be directly derived from what is described in Section 5:

- State picture designs should be visualized as much as possible based on Gestalt principles, more specifically proximity principle and similarity principle; visual objects of observable components that are closely related should be closely arranged; visual objects of observable components whose types are the same should be the same, while different colors should be used when you would like to distinguish different instances; visual objects of observable components that are less related should be arranged apart.

We have described what effects we acquire from the new state picture design based on proximity principle and similarity principle in Section 5. We have also described how we discover characteristics of MCS protocol by observing graphical animations of MCS protocol generated based on the new state picture design and using the Find Patterns functionality of the SMGA tool in Section 6. The concept “extended CS region” is really crucial, which made it possible to conjecture Characteristic 2 or $\text{inv4}(S, P)$ that is a key lemma that can largely contribute to the completion of the formal proof that MCS protocol enjoys the mutual exclusion property.

To make the lesson more practical, we describe some more concrete lessons or guides.

- When an observable component can have two different values, such as lock_p for each process p , it should be visually/graphically represented as a light bulb.

For example, if an observable component has one value, such as on, we should use a fancy or light color; if it is the other value, such as off, we should use a plain or dark color.

- When an observable component can have three or more (but moderate) different values, such as glock and next_p & pred_p for each process p , we should prepare some designated area, such as a rectangle, and a specific position in the area for each value where some visual object, such as a circle on which the value is written, is displayed; if the observable component has a value, only the visual object for it should be displayed and the other visual objects for the other values should disappear; there may be some special value, such as nop, and if the observable component has such a value, nothing should be displayed.
- If there are some local variables, such as next_p , pred_p and lock_p for each process p , to processes or nodes, then we should design the layout of the visual representations for them so that we can visually/graphically identify what variables or observable components are local to what processes or nodes; for example, all local variables for each process should be aligned horizontally.

7.2 Some tips on characteristic conjecture

Some lessons we have learned from the case study are some tips on how to conjecture characteristics with SMGA:

- By concentrating on one observable component, we may find that it never has some specific value, it has some specific value much more often than the other values and whenever it has some specific value, any other same-kind observable components never have the value, from which we may conjecture some characteristics.

- By concentrating on two different-kind observable components, we may find a relation between them, from which we may conjecture some characteristics.
- By fixing some specific values of some observable components and taking a look at all state pictures in which the observable components have the specific values, we may find some relations among the observable components and some other observable components, from which we may conjecture some characteristics. It is necessary to use the Find Patterns functionality so as to take a look at all such state pictures because there may be many such states and it is almost impossible to remember all of them.
- By carefully investigating the conditions of some characteristics that have been already conjectured and the states found by the Find Patterns functionality of (part of) the conditions, we may conjecture some other characteristics.

All characteristics we can conjecture with SMGA may not be true properties of systems under examination. Therefore, we need to use a model checker to confirm that the characteristics are more likely to be true properties of the systems. For all of the characteristics described in Section 6, we have used the Maude search command to confirm that there is no counterexample. We found two plausible characteristics with SMGA:

- If a process p is located at l12, then there exists another process q ($\neq p$) in CS region.
- It is not the case that there are two different processes p_1 and p_2 such that $next_{p_1} = q$, $next_{p_2} = q$, $q \neq p_1$, $q \neq p_2$ and $q \neq \text{nop}$.

Therefore, it is mandatory to use a model checker to confirm each characteristic conjectured. It is not sufficient to use a model checker to do so because a standard model checker cannot prove characteristics or properties for an arbitrary number of each entities, such as processes. Hence, we need to use a theorem prover to prove each characteristic conjectured with SMGA and confirmed with a model checker. In the Qlock case study [4], it has been formally verified that Qlock protocol enjoys the properties conjectured with SMGA and confirmed with a model checker by writing what is called proof scores in CafeOBJ, an interactive theorem proving technique. Among the characteristics found for MCS protocol, it has been formally proved that Characteristics 2, 4, 7.1, 8.2, 9 and 10 are invariants with respect to the state machine formalizing MCS protocol as mentioned.

7.3 Summary of tips

The new tips on how to make good state picture designs are summarized in Table 2. We have one new generic tip and three new concrete tips with which we can make the generic tip more effective. In the table, the generic tip is called SPD-T 5, while the three concrete tips are called SPD-T 5.1, SPD-T 5.2 and SPD-T 5.3, respectively. The tips on how to conjecture state machine characteristics with SMGA are summarized in Table 3.

8 One more case study

Anderson protocol [1] is a shared-memory mutual exclusion protocol. The protocol uses a finite Boolean array whose size is the same as the number of processes participating in the protocol. It also uses the modulo operation of natural numbers and an atomic operation `fetch&incmod`. `fetch&incmod` takes a natural number variable x and a non-zero natural number constant N and atomically does the following: setting x to $(x + 1) \% N$, where $\%$ is the modulo operation, and returning the old value of x .

Table 2 New tips on state picture design

SPD-T 5	State picture designs should be visualized as much as possible based on Gestalt principles, more specifically proximity principle and similarity principle; visual objects of observable components that are closely related should be closely arranged; visual objects of observable components whose types are the same should be the same, while different colors should be used when you would like to distinguish different instances; visual objects of observable components that are less related should be arranged apart.
SPD-T 5.1	When an observable component can have two different values, such as $lock_p$ for each process p , it should be visually/graphically represented as a light bulb.
SPD-T 5.2	When an observable component can have three or more (but moderate) different values, such as $glock$ and $next_p$ & $pred_p$ for each process p , we should prepare some designated area, such as a rectangle, and a specific position in the area for each value where some visual object, such as a circle on which the value is written, is displayed; if the observable component has a value, only the visual object for it should be displayed and the other visual objects for the other values should disappear; there may be some special value, such as <i>nop</i> , and if the observable component has such a value, nothing should be displayed.
SPD-T 5.3	If there are some local variables, such as $next_p$, $pred_p$ and $lock_p$ for each process p , to processes or nodes, then we should design the layout of the visual representations for them so that we can visually/graphically identify what variables or observable components are local to what processes or nodes; for example, all local variables for each process should be aligned horizontally.

SPD-T stands for State Picture Design Tip

We suppose that there are N processes participating in Anderson protocol. Figure 17 shows the protocol written in Algol-like pseudo-code. We suppose that each process is located at *rs*, *ws* or *cs* and initially located at *rs*. *place* is an array whose size is N , each of whose indexes is a process IDs and each of whose elements stores one from $\{0, 1, \dots, N-1\}$. Initially, each element of *place* can be any from $\{0, 1, \dots, N-1\}$ but is 0 in this paper. Although *place* is an array, each process i only uses *place*[i] and then we can regard *place*[i] as a local variable to each process i . *array* is a Boolean array whose size is N and each of whose indexes is as usual. Initially, *array*[0] is true and *array*[j] is false for any $j \in \{1, \dots, N-1\}$. *next* is a natural number variable and initially set to 0. $x, y := e_1, e_2$

Table 3 Tips on characteristic conjecture

CC-T 1	By concentrating on one observable component, we may find that it never has some specific value, it has some specific value much more often than the other values and whenever it has some specific value, any other same-kind observable components never have the value, from which we may conjecture some characteristics.
CC-T 2	By concentrating on two different-kind observable components, we may find a relation between them, from which we may conjecture some characteristics.
CC-T 3	By fixing some specific values of some observable components and taking a look at all state pictures in which the observable components have the specific values, we may find some relations among the observable components and some other observable components, from which we may conjecture some characteristics. It is necessary to use the Find Patterns functionality so as to take a look at all such state pictures because there may be many such states and it is almost impossible to remember all of them.
CC-T 4	By carefully investigating the conditions of some characteristics that have been already conjectured and the states found by the Find Patterns functionality of (part of) the conditions, we may conjecture some other characteristics.

CC-T stands for Characteristic Conjecture Tip

```

Loop “Remainder Section”
  rs : place[i] := fetch&incmod(next, N);
  ws : repeat until array[place[i]];
    “Critical Section”
  cs : array[place[i]], array[(place[i] + 1) % N] := false, true;

```

Fig. 17 Anderson protocol in Algol-like pseudo-code

is a concurrent assignment that is processed as follows: calculating e_1 and e_2 independently and setting x and y to their values, respectively.

We suppose that there are three processes that participate in Anderson protocol. Let $\mathcal{M}_{\text{Anderson}}$ formalize Anderson protocol. Each state in $\mathcal{S}_{\text{Anderson}}$ is expressed as follows:

$$\{(pc[p1] : l_1)(pc[p2] : l_2)(pc[p3] : l_3) \\ (place[p1] : pl_1)(place[p2] : pl_2)(place[p3] : pl_3) \\ (array[0] : b_1)(array[1] : b_2)(array[2] : b_3) \\ (next: npl)\}$$

where l_i for $i = 1, 2, 3$ is a label such as rs, ws and cs, pl_i for $i = 1, 2, 3$ is a natural number (precisely 0, 1 or 2), npl is a natural number (precisely 0, 1 or 2) and b_i for $i = 1, 2, 3$ is a Boolean value. Initially, each l_i is rs, each pl_i is 0, npl is 0, b_1 is true and each b_j for $j = 2, 3$ is false. $\mathcal{I}_{\text{Anderson}}$ consists of one state. Let *init* equal the initial state.

$\mathcal{T}_{\text{Anderson}}$ is described in Maude as follows:

```

r1 [stPl&Nx+] : {(pc[I]: rs) (place[I]: X) (next: Y) S}
=> {(pc[I]: ws) (place[I]: Y) (next: ((Y + 1) rem N)) S} .
r1 [wait] : {(pc[I]: ws) (place[I]: X) (array[X]: true) S}
=> {(pc[I]: cs) (place[I]: X) (array[X]: true) S} .
cr1 [chArray] : {(pc[I]: cs) (place[I]: X) (array[X]: B1)
(array[X1]: B2) S}
=> {(pc[I]: rs) (place[I]: X) (array[X]: false)
(array[X1]: true) S}
if X1 = (X + 1) rem N .

```

where I is a Maude variable of process IDs, X , $X1$ & Y are Maude variables of natural numbers, $B1$ & $B2$ are Maude variables of Boolean values, S is a Maude variable of observable component soups, N is the number of processes participating in the protocol (it is 3 in this case study) and *rem* is the modulo operation of natural numbers.

Once we have decided how to formalize or represent each state of Anderson protocol, we can make a simple state picture design for the protocol. Figure 18 shows a state picture

next: 1		
pc[p1]: ws	pc[p2]: ws	pc[p3]: rs
array[0]: false	array[1]: false	array[2]: true
place[p1]: 2	place[p2]: 0	place[p3]: 0
State 53 : (next: 1) (array[0]: false) (array[1]: false) (array[2]: true) (place[p1]: 2) (place[p2]: 0) (place[p3]: 0) (pc[p1]: ws) (pc[p2]: ws) (pc[p3]: rs)		

Fig. 18 A simple state picture for Anderson protocol

based on the simple design. The simple state picture design does not make best use of the human image perception capability. We then use the tips on how to make state picture designs summarized in Tables 1 and 2. Figure 19 shows a state picture based on one state picture design made according to the tips.

We visualize the *array* observable components as follows:

Three colors (light pink, violet and light blue) are used to represent the three indexes (0, 1 and 2, respectively) instead of natural numbers because colors are visually/graphically more perceivable than natural numbers. Each element of *array* is either true or false. If it is true, a red box appears at the designated place. If it is false, nothing appears at the designated place. On the above visualization of *array*, because $array[0] = \text{false}$, $array[1] = \text{false}$, $array[2] = \text{true}$, a red box appears at the index 2 (or the light blue color place) and nothing appears at the other indexes (or the light pink and violet places).

We visualize the *next* observable component as follows:

The *next* observable component stores one of the three indexes 0, 1 and 2. When it is 0, a light pink box appears at the left-most place and is aligned vertically with the light pink rectangle representing the index 0 of the *array* visualization. When it is 1, a violet box appears at the middle place and is aligned vertically with the violet rectangle representing the index 1 of the *array* visualization. When it is 2, a light blue box appears at the right-most place and is aligned vertically with the light blue rectangle representing the index 2 of the *array* visualization. On the above visualization of *next*, because $next = 1$, a violet box appears at the middle place (see also Fig. 19).

Fig. 19 A revised state picture for Anderson protocol

We visualize the *place* observable components as follows:

Each *place*[*p*] observable component stores one of the three indexes of *array*. The three indexes are represented by the three colors (light pink, violet and light blue). Therefore, the value of each *place*[*p*] observable component is represented by a box whose color is one of light pink, violet and light blue. When *place*[*p*] = 0, a light pink box appears at the left-most position. When *place*[*p*] = 1, a violet box appears at the middle position. When *place*[*p*] = 2, a light blue box appears at the right-most position. On the above visualization of *place*, because *place*[*p1*] = 2, a light blue box appears at the right-most position on the pane of *place*[*p1*] and is aligned vertically with the light blue rectangle representing the index 2 of the *array* visualization (see also Fig. 19); because *place*[*p2*] = 0, a light pink box appears at the left-most position on the pane of *place*[*p2*] and is aligned vertically with the light pink rectangle representing the index 0 of the *array* visualization (see also Fig. 19); because *place*[*p3*] = 0, a light pink box appears at the left-most position on the pane of *place*[*p3*] and is aligned vertically with the light pink rectangle representing the index 0 of the *array* visualization (see also Fig. 19).

Because of the visual representations of the *place*, *next* and *array* observable components described so far, it is possible to visually/graphically perceive the value of each *place*[*p*] for *p* = *p1*, *p2*, *p3*, the value of *next*, the value of each *array*[*x*] for *x* = 0, 1, 2 and some relations among those values. For example, we can visually/graphically perceive that two boxes of the *place*[*p*] observable component and the *next* observable component such that their colors are the same and a red box of the *array* observable components are aligned vertically; we can visually/graphically perceive that the three values of the *place* observable components are different from each other.

Observing graphical animations of Anderson protocol, we often find that at least one of the three boxes representing the values of the three *places* and the two boxes representing the values of *next* and some *array* element, namely the three boxes, are aligned vertically. Figure 20 shows four such state pictures. For example, on the top right state picture of Fig. 20, the left-most three boxes (two pink boxes and one red box) are aligned vertically. On the four state pictures of Fig. 20, the process concerned, for example *p1* on the top right state picture of Fig. 20, is located at either *ws* or *cs*. Thus, we first guess that whenever *place*[*p*] = *next* and *array*[*place*[*p*]] = true, *p* is located at either *ws* or *cs* for any process *p*. This guess of ours is, however, refuted by the initial state whose picture is shown in Fig. 21, where all five boxes are aligned vertically and all three processes are located at *rs*.

Because our intuition says that there must be some interesting relation among *place*[*p*] = *next*, *array*[*place*[*p*]] = true and that *p* is located at either *ws* or *cs*, however, we ask SMGA with its Find Patterns functionality to find all states that satisfy the following condition:

```
state['place[p]'] == state['next']
state['array[x]'] == 'true'
state['pc[p]'] == '1'
```

Fig. 20 Some state pictures for Anderson protocol

for $p = p1, p2, p3$, $x = 0, 1, 2$ and $l = ws, cs$. In all states found by the Find Patterns functionality, we notice that there is no process located at rs . From this observation, we conjecture the following characteristics:

Fig. 21 Initial state picture of Anderson protocol

- Characteristic 11.1: If a process p is located at ws, $place[p] = x$, $next = x$ and $array[x] = \text{true}$, then there is no process located at rs;
- Characteristic 11.2: If a process p is located at cs, $place[p] = x$, $next = x$ and $array[x] = \text{true}$, then there is no process located at rs.

In all those states found by the Find Patterns functionality, we also find that the three values of the `place` observable components are different from each other, which can be visually/graphically perceived because of the visual representations of the `place` observable components (see Fig. 20). This observation guides us to the following characteristics:

- Characteristic 12.1: If two different processes p, q are located at ws, $place[p] \neq place[q]$;
- Characteristic 12.2: If two different processes p, q are located at ws and cs, respectively, $place[p] \neq place[q]$;
- Characteristic 12.3: If two different processes p, q are located at cs, $place[p] \neq place[q]$.

On the top right state picture of Fig. 20, $p1$ is located at ws, $place[p1] = 0$, $array[0] = \text{true}$ and there is no process located at cs. On the bottom right state picture of Fig. 20, $p1$ is located at ws, $place[p1] = 2$, $array[2] = \text{true}$ and there is no process located at cs. We find several other state pictures like them, from which we can conjecture the following characteristic:

- Characteristic 13: If a process p is located at ws, $place[p] = x$ and $array[x] = \text{true}$, then there is no process located at cs.

We also conjecture the following characteristics:

- Characteristic 14: There is always at most one process located at cs.
- Characteristic 15: If there is a process p located at cs and $place[p] = x$, $array[x] = \text{true}$.

From Characteristic 14, we know that the conditional part of Characteristic 12.3 is always false and then Characteristic 12.3 is vacantly true and becomes less interesting. From Characteristic 15, Characteristic 11.2 can be simplified as follows:

- Characteristic 11.2': If a process p is located at cs, $place[p] = x$ and $next = x$, then there is no process located at rs.

We have used the Maude reachability analyzer (or the search command) to model check that all characteristics conjectured in this section are invariant properties with respect to the state machine formalizing Anderson protocol and have not found any counterexamples for each characteristic.

9 Conclusion

We have used MCS protocol, not only a literature protocol but also used in Java virtual machines, to exemplify that state picture designs based on which the SMGA tool produces graphical animations should be better visualized. The new state picture design of a state machine formalizing MCS protocol has been assessed based on Gestalt principles, more specifically proximity principle and similarity principle. We have reported on a core part of a formal verification case study in which the new state picture design and the SMGA

tool largely contributed to the successful completion of the formal proof that MCS protocol enjoys the mutual exclusion property. We have summarized the lessons learned as two groups of tips. The first group is some new tips on how to make state picture designs. The second one is some tips on how to conjecture state machine characteristics by using the SMGA tool. We have also reported on one more case study in which the state picture design has been made for Anderson protocol and some characteristics of the protocol have been discovered based on the tips.

As usual, there are many left to do as future work. The current implementation of the SMGA tool is not interactive in that state picture designs cannot be revised, the layouts of visual objects cannot be changed, any visual objects cannot be zoomed-in/-out, any visual objects cannot be hide, etc. while playing graphical animations and even when pausing graphical animations. We will plan to make SMGA interactive so that we can do what has been just mentioned. The Find Patterns functionality of SMGA is useful as described but only allows us to use a limited class of regular expressions. Maude allows us to use context-free grammars plus some more, such as binary operators that are associative and/or commutative, and has a rich pattern matching functionality between terms constructed based on rich grammars. Maude also provides several formal methods functionalities, such as a reachability analyzer and an linear temporal logic (LTL) model checker. We will plan to integrate SMGA with Maude so that rich Maude functionalities can be used by SMGA. We should conduct some more case studies in which different types of protocols, such as PAXOS [17], as well as some more mutual exclusion protocols, such as a mutual exclusion protocol for ad hoc mobile networks [30], are tackled. Case studies should include formal proofs in which the SMGA tool is used to conjecture lemmas.

Acknowledgements The authors would like to thank the anonymous reviewers who carefully read an earlier version of the paper and gave them valuable comments without which they were not able to complete the present paper.

Competing interests The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Anderson TE (1990) The performance of spin lock alternatives for shared-memory multiprocessors. *IEEE Trans Parallel Distrib Syst* 1(1):6–16. <https://doi.org/10.1109/71.80120>
2. Artho C, Havelund K, Honiden S (2007) Visualization of concurrent program executions. In: 31st international computer software and applications conference (COMPSAC 2007). IEEE, pp 541–546. <https://doi.org/10.1109/COMPSAC.2007.236>
3. Artho C, Pande M, Tang Q (2019) Visual analytics for concurrent Java executions. In: 34th IEEE/ACM international conference on automated software engineering (ASE 2019). IEEE, pp 1102–1105. <https://doi.org/10.1109/ASE.2019.00112>

4. Aung MT, Nguyen TTT, Ogata K (2018) Guessing, model checking and theorem proving of state machine properties – a case study on Qlock. *Int J Softw Eng Comput Syst* 4(2):1–18. <https://doi.org/10.15282/ijsecs.4.2.2018.1.0045>
5. Beschastnikh I, Liu P, Xing A, Wang P, Brun Y, Ernst MD (2020) Visualizing distributed system executions. *ACM Trans Softw Eng Methodol* 29(2):38. <https://doi.org/10.1145/3375633>
6. Brodlie K. W., Carpenter L. A., Earnshaw R. A., Gallop J. R., Hubbard R. J., Mumford A. M., Osland C. D., Quarendon P. (eds) (1992) *Scientific Visualization: Techniques and Applications*. Springer, Berlin. <https://doi.org/10.1007/978-3-642-76942-9>
7. Bui DD, Ogata K (2019) Graphical animations of the Suzuki-Kasami distributed mutual exclusion protocol. *J Vis Lang Comput* 2019(2):105–115. https://doi.org/10.1007/978-3-319-90104-6_1
8. Bui DD, Ogata K (2020) Better state pictures facilitating state machine characteristic conjecture. In: The 26th international DMS conference on visualization and visual languages (DMSVIVA 2020). KSI research inc, pp 7–12. <https://doi.org/10.18293/DMSVIVA20-007>
9. Caine K (2016) Local standards for sample size at CHI. In: 2016 CHI Conference on human factors in computing systems. ACM, pp 981–992. <https://doi.org/10.1145/2858036.2858498>
10. Cao F (2004) Application of the Gestalt principles to the detection of good continuations and corners in image level lines. *Comput Vis Sci* 7:3–13. <https://doi.org/10.1007/s00791-004-0123-6>
11. Carpendale S (2008) Evaluating information visualizations. In: *Information visualization, LNCS*, vol 4950. Springer, pp 19–45. https://doi.org/10.1007/978-3-540-70956-5_2
12. Clavel M, Durán F, Eker S, Lincoln P, Martí-Oliet N, Meseguer J, Talcott C (2007) All About Maude – A High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic, LNCS, vol 4350. Springer <https://doi.org/10.1007/978-3-540-71999-1>
13. Desolneux A, Moisan L, Morel J (2003) A grouping principle and four applications. *IEEE Trans. Pattern Anal. Mach Intell* 25(4):508–513. <https://doi.org/10.1109/TPAMI.2003.1190576>
14. Garae J, Ko RKL, Chairis S (2016) UVisp: User-centric visualization of data provenance with Gestalt principles. In: 2016 IEEE TrustCom/BigDataSE/ISPA, pp 1923–1930. <https://doi.org/10.1109/TrustCom.2016.0294>
15. Hwang W, Salvendy G (2010) Number of people required for usability evaluation: the 10+/-2 rule. *Commun ACM* 53(5):130–133. <https://doi.org/10.1145/1735223.1735255>
16. Isenberg T, Isenberg P, Chen J, Sedlmair M, Moller T (2013) A systematic review on the practice of evaluating visualization. *IEEE Trans Vis Comput Graph* 19(12):2818–2827. <https://doi.org/10.1109/TVCG.2013.126>
17. Lampert L (1998) The part-time parliament. *ACM Trans Comput Syst* 16(2):133–169. <https://doi.org/10.1145/279227.279229>
18. Magee J, Pryce N, Giannakopoulou D, Kramer J (2000) Graphical animation of behavior models. In: 22Nd international conference on software engineering (ICSE 2000). IEEE, pp 499–508. <https://doi.org/10.1145/337180.337368>
19. Mellor-Crummey JM, Scott ML (1991) Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans Comput Syst* 9(1):21–65. <https://doi.org/10.1145/103727.103729>
20. Merino L, Ghafari M, Anslow C, Nierstrasz O (2018) A systematic literature review of software visualization evaluation. *J Syst Softw* 144:165–180. <https://doi.org/10.1016/j.jss.2018.06.027>
21. Nesbitt KV, Friedrich C (2002) Applying Gestalt principles to animated visualizations of network data. In: *International conference on information visualisation (IV 2002)*. IEEE, pp 737–743. <https://doi.org/10.1109/IV.2002.1028859>
22. Nguyen TTT, Ogata K (2017) Graphical animations of state machines. In: 15Th IEEE intl conf.depend., auton. & secure compu. (DASC 2017). IEEE, pp. 604–611. <https://doi.org/10.1109/DASC-PICOM-DataCom-CyberSciTec.2017.107>
23. Nguyen TTT, Ogata K (2017) Graphically perceiving characteristics of the MCS lock and model checking them. In: 7Th intl wksh SOFL+MSVL, LNCS, vol 10795. Springer, pp 3–23. https://doi.org/10.1007/978-3-319-90104-6_1
24. Nguyen TTT, Ogata K (2017) A way to comprehend counterexamples generated by the Maude LTL model checker. In: 2017 International conference on software analysis, testing and evolution (SATE 2017). IEEE, pp 53–62. <https://doi.org/10.1109/SATE.2017.15>
25. Schmettow M (2012) Sample size in usability studies. *Commun ACM* 55(4):64–70. <https://doi.org/10.1145/2133806.2133824>
26. Todorovic D (2008) Gestalt principles. *Scholarpedia* 3(12):5345. <https://doi.org/10.4249/scholarpedia.5345>
27. Tran DD, Bui DD, Gupta P, Ogata K (2020) Lemma weakening for state machine invariant proofs. In: 27Th asia-pacific software engineering conference (APSEC 2020). IEEE

28. Wagemans J, Elder JH, Kubovy M, Palmer SE, Peterson MA, Singh M, von der Heydt R (2012) A century of Gestalt psychology in visual perception: I. perceptual grouping and figure-ground organization. *Psychol Bull* 138(6):1172–1217. <https://doi.org/10.1037/a0029333>
29. Wagemans J, Feldman J, Gepshtein S, Kimchi R, Pomerantz JR, van der Helm PA, van Leeuwen C (2012) A century of Gestalt psychology in visual perception: II. conceptual and theoretical foundations. *Psychol Bull* 138(6):1218–1252. <https://doi.org/10.1037/a0029334>
30. Walter JE, Welch JL, Vaidya NH (2001) A mutual exclusion algorithm for ad hoc mobile networks. *Wirel Netw* 7:585–600. <https://doi.org/10.1023/A:1012363200403>
31. Ware C (2004) *Information Visualization: Perception for Design*, 2nd edn. Morgan Kaufmann
32. Yalcinkaya M, Singh V (2019) Exploring the use of Gestalt's principles in improving the visualization, user experience and comprehension of COBie data extension, Engineering, Construction and Architectural Management. <https://doi.org/10.1108/ECAM-10-2017-0226>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.