Doctoral Dissertation

State Machine Visualization Based on Gestalt Principles and Its Applications

Dang Duy Bui

Supervisor: Kazuhiro Ogata

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
[Information Science]

Dec, 2022

Abstract

This dissertation proposes an approach to visualizing/graphically animating state machines based on Gestalt principles for humans to find invariants of the state machines. This approach mainly addresses the question "Can Gestalt Principles help humans to understand state machines well?," where to understand state machines is defined as knowing invariant properties of the state machines. The more humans know invariant properties of the state machines, the better humans understand the state machines.

State machine visualization is one possible way to make humans gain insights into state machines because humans are good at visual perception. Graphically animating state machines is one approach to state machine visualization. To graphically animate state machines, it is necessary to design what are called state picture templates such that a series of state picture template instances (or state pictures) can be regarded as a movie film. Any state picture templates of a state machine do not work well for our purpose, and we need to carefully design a state picture template of a state machine so that graphical animations based on it help humans to find likely invariants of the state machine. We use Gestalt principles, a set of principles/laws that describes how humans group similar elements, recognize patterns, and simplify complex images when humans perceive visual objects, to design state picture templates of state machines. Because likely invariants of a state machine may not be true invariants of the state machine, we first use model checking to check if likely invariants have counterexamples and then use interactive theorem proving to judge if likely invariants left are true invariants of the state machine. We basically use a tool called State Machine Graphical Animation (SMGA) that takes a state machine template and a state sequence of a state machine, and generates a graphical animation of the state machine. However, SMGA is not mature enough.

This dissertation addresses how likely invariants of protocols/systems can be found by humans who use the graphical animation approach. In particular, this research shows the importance of the state picture template and gives practical tips for users to design complex protocols/systems. Those tips are inspired and evaluated by Gestalt principles. We also propose guidelines of how to use the tips for finding likely invariants. To make the guidelines more effective, SMGA is revised by integrating it with Maude, a specification/programming language and processor that is equipped with many useful facilities, such as a reachability analyzer (the search command), a parser for context-free grammars plus associative-commutative (AC) binary operators and a pattern matcher for the grammars plus AC binary operators. The revised version of SMGA is called r-SMGA in which such powerful features of Maude can be used. The search command can be used as an invariant model checker.

Case studies are conducted to demonstrate the usefulness of the proposed approach and r-SMGA. Based on experiments with new features of r-SMGA and the guidelines proposed, several likely invariants are found, and most of them survive with the search command feature of r-SMGA and are proven true invariants with interactive theorem proving. When conducting interactive theorem proving for a likely invariant that has survived with the search command, we can use some of other such likely invariants as lemmas, provided that we have found an enough number of likely invariants of a state machine.

Keywords: state machine graphical animations; r-SMGA; likely invariant discovery; Gestalt principles; interactive theorem proving.

Acknowledgments

Firstly and foremost I would like to gratitude to Professor Kazuhiro Ogata for his endless support and great encouragements not only in research but also in his experience of life. His immense knowledge and plentiful experience have encouraged me all the time in my academic research. Being a student in his laboratory is the proudest of my life. He cares so much about my work and gives me advice, comments, and great ideas to make my research keep on track. He always motivates me to improve myself day by day. I always remember his sentences when I am in a difficult time "We cannot change the past, although you may want to reflect on yourself to avoid the same mistake again. You should look forward.", his sentences help me to overcome the difficult time. No words can describe what he has done for me and I am indebted to him.

Secondly, I would like to give special thanks to Associate Professor Adrián Riesco who is my supervisor of the minor research topic, and the committee members Professor Kunihiko Hiraishi, Professor Toshiaki Aoki, Associate Professor Nakamura Masaki, and Associate Professor Fuyuki Ishikawa for their encouraging words, detailed comments, and rich suggestions that have been very important for me to improve my work as well as complete the dissertation. Especially, Associate Professor Adrián Riesco who is an expert in Maude, and his immense knowledge and plentiful experience have supported me in Maude integration.

Thirdly, I would like to express my special thanks to the Doctoral Research Fellow (DRF) at JAIST, the NEC C&C Foundation, and SHIBUYA Science Culture and Sports Foundation for financial support during my Ph.D. study at JAIST. I also would like to give deep thanks to the JAIST staff who helped directly and indirectly by giving me the best environment in my daily life.

Fourth, I would like to express my appreciation to my lab mates and Vietnamese friends. Thank you for sharing wonderful moments and interesting ideas, not only in research but daily life, being by my side, and helping me to overcome difficult periods. Without their support, sharing, and encouragement, it would be impossible for me to complete my Ph.D. study at JAIST. It was an unforgettable memory in my life. Thank you very much and love you all!

Last but not least, I would like to thank my family for the tremendous support and hope they have given me. I also would like to give the greatest thanks to the people who came and helped me when I was studying at JAIST. Thank you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The motivation of the dissertation is to propose an approach to graphically animating state machines based on Gestalt principles for humans to find likely invariants of the state machines. This dissertation focuses on the research question (**RQ**) "Can Gestalt principles help humans to understand state machines well?," where to understand state machines is defined as knowing properties of the state machines. This research concentrates on invariant properties only because invariant properties are the most fundamental ones and often used as lemmas to prove different classes of properties so as to exclude unreachable states. Therefore, the more humans know invariant properties of the state machines, the better humans understand the state machines.

In the dissertation, a likely invariant is considered as a characteristic that seems correct and is not proven in programs or specification of protocols/systems. Daikon [1] is a famous tool in Software Engineering and what Daikon does is to find likely invariants of programs. Several applications of Daikon [2, 3, 4, 5] show the usefulness of likely invariants and human understanding of program behaviors. Therefore, it is worth finding likely invariants. State machine visualization is one possible way to make humans gain insights into state machines because humans are good at visual perception [6]. Graphically animating state machines is one approach to state machine visualization. To graphically animate state machines, it is necessary to design what are called state picture templates such that a series of state picture template instances (or state pictures) can be regarded as a movie film. Any state picture templates of a state machine do not work well for our purpose, and we need to carefully design a state picture template of a state machine so that graphical animations based on it help humans to find likely invariants of the state machine. We use Gestalt principles [7, 8, 9], a set of principles/laws that describes how humans group similar elements, recognize patterns, and simplify complex images when humans perceive visual objects, to design state picture templates of state machines. Because likely invariants of a state machine may not be true invariants of the state machine, we first use model checking to check if likely invariants have counterexamples and then use interactive theorem proving to judge if likely invariants left are true invariants of the state machine. We basically use a tool called State Machine Graphical Animation (SMGA) [10] that takes a state machine template and a state sequence of a state machine, and generates a graphical animation of the state machine. However, SMGA is not mature enough.

1.2 Contributions

In summary, this study addresses **RQ** by providing an approach based on Gestalt principles to graphically animating state machines for humans to find likely invariants of the state machines and demonstrating the usefulness of our proposed approach by conducting case studies. In particular, based on **RQ**, two issues need to be concerned: (1) applying Gestalt principles for visualization of state machines and (2) using such visualization to find properties of state machines. Based on such issues and the graphical-animation-based visualization approach, two sub-questions are considered as follows:

RQ1: How to design state picture templates based on Gestalt principles?

RQ2: How to conjecture/find likely invariants based on graphical animations?

where **RQ1** focuses on tackling the issue (1) while **RQ2** concentrates on solving the issue (2). Note that likely invariants can become properties (or true invariants) when they are proven by theorem proving, which will be handled in the dissertation. Therefore, to answer **RQ**, it is equivalent to answer **RQ1** and **RQ2**. To be able to answer **RQ1** and **RQ2**, the dissertation focuses on three following tasks:

Designing State Picture Templates

This task shows the importance of state picture templates, and mainly provides two kinds of tips for designing state picture templates and conjecturing likely invariants of protocols/systems. The tips for designing state picture templates are our results from many case studies [11, 12, 13]. Such tips are inspired and evaluated by Gestalt principles, especially the common region, proximity, and similarity laws. Moreover, based on the tips for conjecturing likely invariants, this task also proposes guidelines as a generic way to orientate humans in finding likely invariants.

Integrating SMGA and Maude

To make the tips more effective, this task mainly provides several features that assist humans to conjecture likely invariants. The revised version of SMGA is called r-SMGA that consists of a special display feature (for displaying data structure, such as queue and array), interactive feature (for human users to focus on or hide visual objects that they are interested or less interested in), and some Maude features implemented by integrating SMGA and Maude [14], a specification/programming language and processor that is equipped with many useful facilities, such as a reachability analyzer (the search command), a parser for context-free grammars plus associative-commutative (AC) binary operators and a pattern matcher for the grammars plus AC binary operators, so that r-SMGA can use such powerful features of Maude.

Conducting Case Studies Using Our Approach

Conducting case studies is a metric to evaluate our approach. This task demonstrates the usefulness of our proposed approach by conducting case studies. For each case study, several likely invariants are found using our guidelines (with practical tips) and features in r-SMGA. Those likely invariants are confirmed by the search command and proved by interactive theorem proving. The results show that our approach can help humans to understand state machines based on true invariants.

1.3 Dissertation Structure

The study is structured into nine chapters that can be summarized as follows:

Chapter 1 introduces the motivation with **RQ** and our proposed approach to address **RQ** with two sub-questions **RQ1** and **RQ2**. This chapter also introduces our contributions and the structure of the dissertation.

Chapter 2 mentions some preliminaries for readers to comprehend the study such as state machines, Maude, SMGA, Proof scores in CafeOBJ, and Gestalt principles.

Chapter 3 investigates the literature review on automated theorem provers, systems visualization, evaluation of visualization and usability, and Gestalt principles. Then, based on the literature review, this chapter describes the way to evaluate the state picture template and SMGA.

Chapter 4 describes the first contribution. Firstly, the chapter shows how important state picture templates are. Then, this chapter focuses on how to design the state picture template by giving some tips based on Gestalt principles, such as the common region, proximity, and similarity laws. The chapter also gives some tips and practical guidelines to find likely invariants of protocols/systems in general. The TAS protocol is exemplified to explain the usefulness of some tips.

Chapter 5 describes the second contribution that makes the proposed tips more effective. This chapter introduces the revised version of SMGA (r-SMGA) where new and revised features have been implemented so that they can help human users to conjecture likely invariants of protocols/systems. Those new and revised features include special display features (e.g. users can display visually some data structures, such as array and queue), interactive features (e.g. users can interact with visual objects in the state pictures, such as focusing and hiding which visual objects that users are interested or less interested in), and some Maude features implemented by integrating SMGA and Maude so that r-SMGA can use some powerful features of Maude, such as a parser for the context-free grammar plus Associative-Commutative (AC) binary operators, a pattern matcher for the grammar plus AC binary operator, and a reachability analyzer (the search command)

Chapter 6 reports on a case study where the Mellor-Crummey-Scott (MCS) mutual exclusion protocol [15] is used as an example to demonstrate the usefulness of our proposed approach, especially the tips for designing state picture templates. Firstly, this chapter introduces the MCS protocol and its specification in Maude. Then, this chapter shows the usefulness of our tips for designing state picture templates by comparing our state picture template and state picture template of previous work. Based on the tips for conjecturing likely invariants, several likely invariants of the MCS protocol are conjectured and confirmed with the search command in r-SMGA. The chapter reports a case that a flawed version of the MCS protocol is used to show that our approach works well with the defective versions. Finally, most confirmed invariants are proven by proof scores in CafeOBJ. This chapter also reports reasons why the rest of the confirmed invariants are not proven and introduces one possible way to handle it.

Chapter 7 reports on a case study where the Suzuki-Kasami (SK) distributed mutual exclusion protocol [16] is used as an example to demonstrate the usefulness of our proposed approach, especially our proposed guidelines with new features in r-SMGA. Firstly, this chapter introduces the SK protocol and its specification in Maude. Then, based on guidelines with some features of r-SMGA, several likely invariants are found and explained in details. Those invariants are confirmed with the search command in r-SMGA. This chapter also reports a case that a flawed version of the SK protocol is used to show that humans still obtain some flawed states by observing graphical animations in spite of defective versions. Finally, all confirmed invariants are proven by proof scores in CafeOBJ. This chapter shows some ways to use our proposals to find likely invariants as lemmas when conducting interactive theorem proving.

Chapter 8 evaluates our proposals by analyzing the results from case studies. Two sub-questions **RQ1** and **RQ2** are answered and analyzed to show the helpfulness of our proposals. Then, **RQ** is addressed by answering **RQ1** and **RQ2** in the chapter. This chapter also mentions some limitations of the study.

Chapter 9 concludes the study and gives several future directions.

Chapter 2

Preliminaries

This chapter provides some backgrounds that can help readers to better comprehend this dissertation such as state machines, Maude, SMGA, proof scores in CafeOBJ, and Gestalt principles.

2.1 State Machines and Maude

A state machine $M \triangleq \langle S, I, T \rangle$ consists of a set S of states, a set $I \subseteq S$ of initial states, and a binary relation $T \subseteq S \times S$ over states. $(s, s') \in T$ is called a state transition. The set $R \subseteq S$ of reachable states with respect to (w.r.t.) M is inductively defined as follows: (1) for each $s \in I$, $s \in R$ and (2) for each $(s, s') \in T$, if $s \in R$, then $s' \in R$. A state predicate p is an invariant property w.r.t. M if and only if $p(s)$ holds for all $s \in R$. In this dissertation, characteristics of a state machine that are likely to be invariant properties of the state machine are called likely invariants. A finite sequence $s_0, \ldots, s_i, s_{i+1}, \ldots, s_n$ of states is called a state sequence of M if $s_0 \in I$ and $(s_i, s_{i+1}) \in T$ for each $i = 0, \ldots, n-1$.

In this dissertation, to express a state of S, we use a braced associative-commutative collection of name-value pairs although there are many possible ways to express states. Associative-commutative collections are called soups and name-value pairs are called observable components. That is, a state is expressed as a braced soup of observable components. The juxtaposition operator is used as the constructor of soups. Suppose $oc1, oc2, oc3$ are observable components and then $oc1$ $oc2$ $oc3$ is the soup of those three observable components. A state that can be characterized by the three observers can be expressed as $\{oc1\ oc2\ oc3\}$. There are many possible ways to specify state transitions. In the disertation, Maude is used to specify state transitions as rewrite rules. Maude [14] can specify complex systems flexibly and is also equipped with several formal analysis techniques, such as reachability analysis and linear temporal logic (LTL) model checking. A rewrite rule starts with the keyword `rl`, followed by a label enclosed by square brackets and a colon, two patterns (terms that may contain variables) connected with =>, and ends with a full stop. A conditional one starts with the keyword `crl` and has a condition following the keyword `if` before a full stop. The following is the form of a conditional rewrite rule:

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

where lb is a label, l and r are the patterns, and c_i is a part of the condition, which may be an equation $lc_i = rc_i$. The negation of $lc_i = rc_i$ could be written as $(lc_i$ =/= $rc_i) =$ `true`, where `= true` could be omitted. If the condition $\ldots$ /\ c_i /\ $\ldots$ holds under some substitution σ, $\sigma(l)$ can be replaced with $\sigma(r)$.

Maude provides the `search` command that allows users to find a reachable state from t such that the state matches the pattern p and satisfies the condition c:

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

where MOD is the name of the Maude module specifying the state machine, n and m are optional arguments stating a bound on the number of solutions and the maximum depth of the search, respectively. n typically is 1, m typically is unbounded (or equivalently omitted) and t typically represents an initial state of the state machine.

Maude provides LTL model checking so that we can check whether a system satisfies a property that is expressed as an LTL formula. Maude can check the system that starts from $init$ satisfies φ by the following command:

`reduce modelCheck(`$init$`,` φ`) .`

Maude returns true if the system satisfies φ provided that $init$ is only considered as its initial state. Otherwise, a counterexample is returned, which has a form of a state sequence and a loop[1]. When the system has multiple initial states, it suffices to conduct the model checking experiment for each initial state so that we can model check that the system enjoys the property.

2.2 State Machine Graphical Animation (SMGA)

State Machine Graphical Animation (SMGA) was originally developed by Nguyen and Ogata [10]. The main purpose of SMGA is to make formal-methods experts able to conjecture likely invariants of protocols/systems. There are two phases when using SMGA: preparation and control, as shown in Figure 2.1. In the preparation phase, SMGA requires a state picture template and a state sequence as the input. The state picture template is designed by users, while the state sequence is generated by Maude from a formal specification of a protocol/system. Designing the state picture template is an important part of SMGA [11] because if the state picture template is essentially composed of texts, then, it is boring and hard to observe likely invariants of protocols [13]. Based on the input, SMGA produces graphical animations as output. Observing such animations allows human users to conjecture likely invariants. In the control phase, users can control the animations, such as changing the speed of the animations (for running automatically) and running step by step, in which each step is a state transition. In addition,

[1]https://maude.lcc.uma.es/maude30-manual-html/maude-manualch12.html

given a state sequence input, SMGA allows us to search for some states in the state sequence such that such states satisfy some conditions. This feature uses regular expressions to conduct the search and is called Find Patterns in [11, 13].

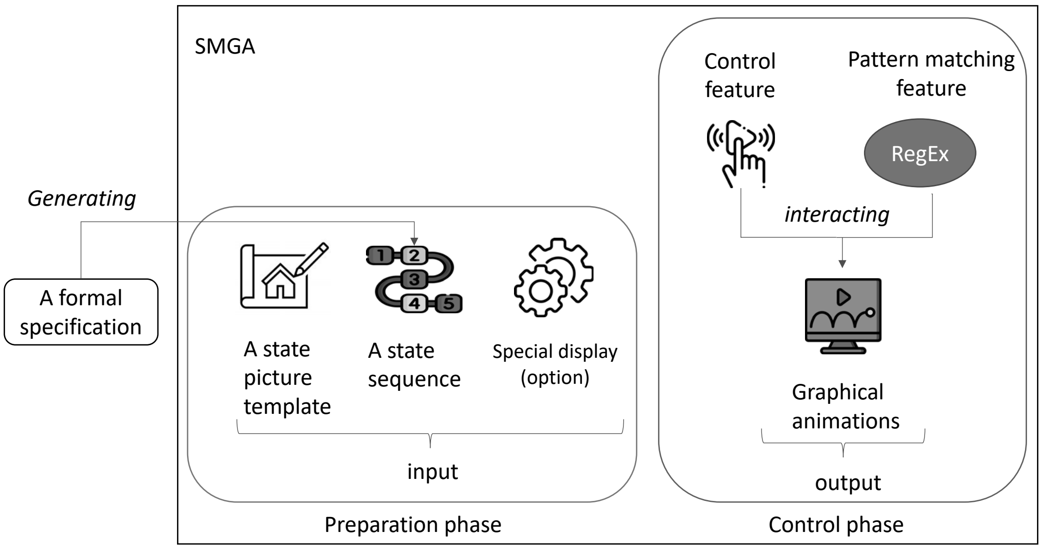
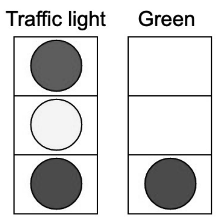


Figure 2.1: Overview of SMGA

SMGA basically provides two kinds of visualization for an observable component: (1) textual display and (2) visual display. (1) presents a value of an observable component as text, while (2) presents a value of an observable component visualized by what users expect, such as visual objects. For example, an observable component simulates a traffic light that exhibits one of the three values: Red, Yellow, and Green. The following figure displays the state picture template (on the left-hand side) and the state picture when the traffic light exhibits Green (on the right-hand side). In the figure, the text on the top ("Traffic light") is used as (1), while three circles with three different colors are used as (2).



2.3 CafeOBJ and Proof Scores

CafeOBJ [17] allows us to not only write formal specifications of protocols/systems but also verify that they satisfy some desired requirements/properties by writing and executing what are called proof scores [18]. Model checking only supports the verification of protocols/systems that have a finite number of reachable states, meaning that we need to, for example, limit the number of processes/nodes participating in a protocol/system under checking. Formal verification by writing proof scores [18] does not limit either the number of processes/nodes participating in a protocol/system or the number of sessions, for example, how many times each process/node enters a critical section. This subsection illustrates that advantage via an example with a mutual exclusion protocol. The protocol, called TAS (Test And Set), can be written in pseudo-code as follows:

```
loop { “Remainder Section”
  rs  :  repeat while test&set($locked$);
         “Critical Section”
  cs  :  $locked$ := false;  }
```

We suppose that each process is located at either rs (Remainder Section) or cs (Critical Section) and initially at rs. *locked* is a Boolean variable shared by all processes and initially false. test&set(*locked*) atomically does the following: if *locked* is false, then it sets *locked* to true and returns false; otherwise, it just returns true.

We want to prove that TAS enjoys the mutual exclusion property whose informal description is that there is always at most one process located at the Critical Section (cs), no matter how many processes participate in the protocol and no matter how many times each process tries to enter the cs. We first specify TAS in CafeOBJ and then prove that it enjoys the mutual exclusion property by writing proof scores. To specify the protocol, we use two observers with which we observe the location of each process and the value stored in *locked*. The two observers are expressed as the CafeOBJ operators declared as follows:

```
op pc : Sys Pid -> Label
op locked : Sys -> Bool
```

where `Sys` is the sort (or type) representing the state space, `Pid` is the sort of process IDs, and `Label` is the sort of locations such as rs and cs. For `s` of `Sys` and `p` of `Pid`, `pc(s,p)` is the location at which `p` is located at state `s` and `locked(s)` is the value stored in *locked* at state `s`.

An arbitrary initial state is expressed as a CafeOBJ operator (`init`). We use two transitions that are expressed as CafeOBJ operators (`enter` and `exit`). Those CafeOBJ operators are declared as follows:

```
op init : -> Sys {constr}
op enter : Sys Pid -> Sys {constr}
op exit : Sys Pid -> Sys {constr}
```

where `constr` stands for constructors. `init` and `enter` are defined by means of equations as follows:

```
eq pc(init,P) = rs .
eq locked(init) = false .

ceq pc(enter(S,P),Q) = (if P = Q then cs else pc(S,Q) fi)
 if c-enter(S,P) .
ceq locked(enter(S,P)) = true if c-enter(S,P) .
ceq enter(S,P) = S if not c-enter(S,P) .
eq c-enter(S,P) = (pc(S,P) = rs and not locked(S)) .
```

where `S` is a CafeOBJ variable of sort `Sys`, and `P` and `Q` are CafeOBJ variables of sort `Pid`. Note that in CafeOBJ, all variables are implicitly universally quantified. The equations for `exit` are defined likewise.

The mutual exclusion property is then defined by the following predicate:

```
op mutex : Sys PiD PiD -> Bool
eq mutex(S,P,Q) = (pc(S,P) = cs and pc(S,Q) = cs) implies P = Q .
```

We prove that the predicate `mutex` holds in all reachable state `S` and all processes `P` and `Q`, namely that `mutex` is an invariant property of the state machine formalizing TAS. It is proved by (simultaneous) structural induction on variable `S` by writing proof scores in CafeOBJ. There are one base case and two induction cases. The following is the proof of the base case written in CafeOBJ, which is called a proof score fragment (or just a fragment):

```
open TAS .
   ops p q : -> Pid .
   red mutex(init,p,q) .
close
```

where `TAS` is the CafeOBJ module in which the specification of TAS and the predicate `mutex` are available, `open` makes the given module available, `close` stops the use of the module, and `red` (an abbreviation of `reduce`) reduces the given term by applying equations. `p` and `q` are fresh constants of sort `Pid` representing arbitrary process IDs (which are possibly equal). Feeding this proof score fragment into CafeOBJ, CafeOBJ returns `true`, meaning that the case is discharged.

There are two induction cases needed to be tackled because we defined two operators with the `constr` attribute besides `init`. Let us consider the induction case in which `enter` is taken into account. The proof score fragment for this induction case is as follows:

```
open TAS .
   op s : -> Sys .
   ops p q r : -> Pid .
   red mutex(s,p,q) implies mutex(enter(s,r),p,q) .
close
```

where s represents an arbitrary state and mutex(s,p,q) is an instance of the induction hypothesis. However, feeding this proof score fragment into CafeOBJ, the returned result is neither true nor false, but instead a complicated term. Case splitting is used to overcome this situation. The proof of one sub-case is as follows:

```
open TAS .
   op s : -> Sys .
   ops p q r : -> Pid .

   eq pc(s,r) = rs .
   eq locked(s) = false .
   eq p = r .
   eq (q = r) = false .
   eq pc(s,q) = cs .
   red mutex(s,p,q) implies mutex(enter(s,r),p,q) .
close
```

The equations characterize the sub-case. For example, a Boolean term can be used to split a case into two sub-cases: (1) the term equals true and (2) it equals false. The induction case is first to split into two sub-cases with the Boolean term pc(s,r) = rs (distinguishing whether the process r is in the remainder section in the state s): (1) (pc(s,r) = rs) = true and (2) (pc(s,r) = rs) = false. (pc(s,r) = rs) = true can be replaced with pc(s,r) = rs that is used as the first equation of the fragment. CafeOBJ returns false for this fragment. We need to conjecture a lemma to discharge the sub-case. The lemma is as follows:

```
eq inv1(S,P) = (pc(S,P) = cs implies locked(S)) .
```

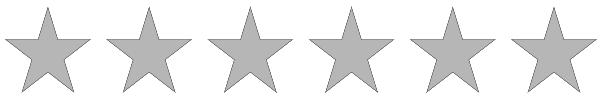
Then, in the fragment above, inv1 is used as a lemma to discharge the sub-case as follows:

```
red inv1(s,q) implies mutex(s,p,q) implies mutex(enter(s,r),p,q) .
```

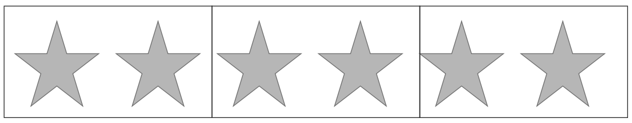
CafeOBJ now returns true for the fragment. The proof of inv1 needs to use mutex as a lemma. Although the proof of mutex uses inv1 and vice versa, our proof is not circular. The reason is that we use simultaneous (structural) induction to develop our proof. The correctness of this method has been mathematically proved in [18]. Note that it is not straightforward to find lemmas for complex and/or complicated protocols in reality.

2.4 Gestalt Principles

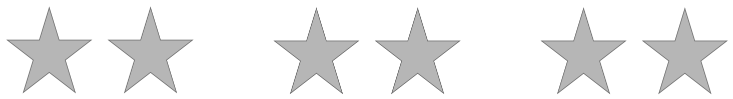
We use examples to introduce three Gestalt principles that are used to design and assess state picture template in the dissertation: common region principle, proximity principle and similarity principle. Let us take a look at the following image:



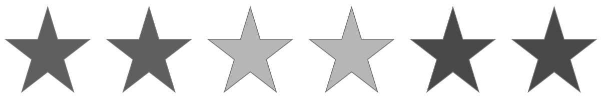
Six stars are aligned horizontally such that the distance of each adjacent pair of stars is the same. Then, we perceive that there are just six stars and did not recognize any sub-groups in it. Let us change the layout a little bit as follows:



We can recognize three rectangles as three sub-groups including two stars for each. This visual perception is call "common region" principle.
Let us change the first image of six stars in a different way. The changed image is 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".

Chapter 3

Related Work

This section first reports on our literature review on automated theorem provers, systems visualization, evaluation of visualization and usability, and Gestalt principles. Based on the literature review, this section finally describes our way to evaluate the new state picture template and the SMGA tool.

Literature Review on Automated Theorem Provers

We first give a literature review on automated theorem provers where their approaches mainly tackle proof search. Then, we investigate some approaches that close to our approach, and finally, report two recent studies on an integration of formal methods and visualization.

Automate theorem provers (ATPs) are generally based on propositional and first-order logic (FOL) and involve the development of computer programs that can automatically perform logical reasoning. Proof search is one bottleneck of ATPs because of search space (combinatorial) explosion [19, 20]. One approach called proof guidance is to use heuristics and strategies to guide the proof search [21, 22], while another approach is to use machine learning techniques to automatically determine heuristics for assisting with proof guidance even though the learned features have been manually designed [23]. Some studies [24, 25] use deep learning techniques to learn those features, which has the appeal of lessening the amount of expert knowledge needed compared with handcrafting new heuristics. Crouse et al. [26] have proposed TRAIL using deep reinforcement learning techniques to learn features from scratch to beat other reinforcement-learning based techniques. Wang et al. [27] have proposed an approach that uses represent mathematical formulas as graphs and embeds them into vector space. That approach achieves state-of-the-art results on the HolStep dataset [28]. Even the results of those techniques are promised, there still exist problems that state-of-the-art ATPs cannot tackle that makes Alemi et al. [29] stating that "we believe theorem proving is a challenging and important domain for deep learning methods,. . . ", which shows that ATPs are not mature enough in practice. The motivation of the work [27] (one of the models got state-of-the-art results) comes from an observation that a mathematical formula can be represented as a graph that encodes the syntactic and semantic structure of the formula, which is evidence to show that the visual perception is still a promising approach in theorem proving. Even though, those approaches and techniques are far of our approach, but, there still exists one state-of-the-art study that indirectly shows the usefulness of visual perception.

Let us introduce Daikon [1], a software can discover likely invariants from execution traces (sequences), which is closed to our approach (conjecturing likely invariants from observing graphical animations of sequences). Likely invariants in Daikon are detected from program executions by instrumenting the source program to trace the variables of interest, running the instrumented program over a set of test cases, and inferring likely invariants over both the instrumented variables and over derived variables that are not manifest in the original program. The essential idea is to test a set of possible likely invariants against the values captured from the instrumented variables; those invariants that are tested to a sufficient degree without falsification are reported to users. Win and Ernst [2] use likely invariants discovered from Daikon as lemmas in the proofs of more complex properties in a theorem prover even invariants can be so numerous and so simple that humans overlook them. They conduct case studies where some protocols and their desired properties are used as examples, to show the usefulness of their approach. They use (i) IOA language [30] to formally specify the protocols and generate execution traces, (ii) Daikon to discovery likely invariants via the execution traces, and (iii) Larch Prover [31] as a theorem prover where used lemmas come from (ii). The results of case studies show that likely invariants discovered by Daikon are useful (used in the proof) even most of them are redundant (not used in the proof). Because of the grammar of Daikon (that is used to generate likely invariants), there are three limitations of likely invariants discovered by Daikon: (1) number of variables in likely invariants are at most three, (2) some boolean connectives such as $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are not in likely invariants, and also with (3) existential quantifiers $\exists$. In our approach, we use Maude to specify a protocol/system, generate state sequences and visualize them in SMGA in a similar way of (i); observing the state sequences, we can conjecture likely invariants in a similar way of (ii); and use proof scores in CafeOBJ as a theorem prover where used lemmas are from (ii) in a similar way of (iii). One advantage of ours compares to Daikon approach is likely invariants in task (ii) that can contain limit properties while likely invariants discovered by Daikon cannot. Because of the dependence on humans, our approach cannot automate as so as our limitation.

Similar to Daikon with its application, Crème [32], an automatic invariant prover, can automatically generate lemmas used for proving invariant properties. The main technique of Crème to generate lemmas is based on fixed-point computation and case splitting. One advantage of Crème is to generate counterexamples if some properties does not hold. Because of fixed-point computation technique, it take much time to automatically generate lemma candidates. In addition, the implementation of Crème is not efficient so Crème may not work in complex protocols because of its limitations. Yang et al [33] have proposed an approach that can prove quantified theorems in programs with algebraic data types. The main technique is to use inductive reasoning where the prover decomposes given theorems into the base-case and inductive-case subgoals and uses a backtracking rewriter that sequentially simplifies each of the subgoals toward true. For cases that cannot succeed by the prover, the authors do two ways: (i) generalizing the rewritten formula by replacing certain concrete subterms in a formula with fresh variables and attempts to prove its validity from scratch. If this way is succeeded, the formula can be used as lemmas, otherwise (ii) performing a SyGuS [34]-based lemma enumeration driven by templates, i.e., formulas with unknowns potentially provided by the user. After tasks (i) and (ii) are done, lemma candidates are filtered to minimize number of lemmas to be proved. Task (ii) is similar to other automated lemma generating techniques that uses a grammar to generate lemma candidates, the different point is to use humans as a factor to make it well perform. It shows that humans are still needed in theorem proving. If possible, investigating the techniques of [32, 33] to be able to automatically generate lemmas used for proving invariant properties are one piece of our future.

Our approach with SMGA can be regarded as an integration of formal methods and visualization. We introduce two recent studies on an integration of formal methods and visualization. One [35] is a study on visualization of what is done inside by Vampire [36], an automated first-order logic theorem prover, and the other is a study on visualization of the structural operational semantics of a simple imperative programming language [37]. Although automated theorem provers are attractive because they may automatically prove theorems, they cannot truly fully automatically prove all possible theorems. Proof attempts may fail. If that is the case, human users need to comprehend why the proof attempts fail and need to change the format of input logical formulas and/or some internal proof strategies. It is very difficult for non-expert users and at least non-trivial for expert users to really comprehend why the proof attempts fail because it is necessary to understand what is done inside by an automated theorem prover, such as Vampire [36]. Gleiss, et al. [35] have then developed SATVIS, a tool to visualize what is done inside by Vampire. Students and even programmers should learn semantics of programming languages so as to understand programming languages better, which may make it possible for them to write better programs. However, it is hard for students to learn semantics of programming languages. Perhác and Zuzana Bilanová [37] have then developed an interactive tool for visualization of the structural operational semantics of a simple imperative programming language. SMGA partially shares the motivation of the first study [35]. This is because the main purpose of SMGA is to help human users to understand systems through graphical animations of state machines. Nothing special is directly shared by SMGA and the second study [37] except an integration of formal methods and visualization. However, several formal semantics of programing languages have been described in the $\mathbb{K}$ framework [38], where $\mathbb{K}$ has been implemented in Maude. SMGA basically graphical animates state machines specified in Maude. Therefore, it would be possible to integrate SMGA and the $\mathbb{K}$ framework so that formal semantics of programming languages can be visualized.

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 [39] 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.

Artho, et al. [40] propose an extended version of UML sequence diagrams so that multi-threaded 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 [41]. Their extended sequence diagrams are also still visualization, while our visualization is graphical animations.

VA4JVM [42] 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 [41].

Magee et al. [43] 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 SceneBeans[1], a library of JavaBeans. As written, their visualization is graphical animation like ours. SceneBeans could be used to implement a future version of SMGA.

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 template and SMGA.

Carpendale [44] 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. [45] 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.

Merino, et al. [46] conducted a systematic literature review of 181 full papers published at

[1]`https://www.doc.ic.ac.uk/ltsa/scenebeans/`

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 [47] 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 [48] 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 [49]. 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".

Literature review on Gestalt principles

Gestalt principles (or laws) (or principles of grouping) [50, 7, 8, 9] 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 [51] 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 [52] invented a good algorithm for this aim by utilizing Gestalt principles and Helmholtz Principle, a quantitative version of the former [53].

Yalcinkaya and Singh [54] 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. [55] 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.

Our way to evaluate state picture template 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 humans in Formal Methods conjecture likely invariants that make them better understand, 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 evaluate ShiViz so as to evaluate the state picture templates 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 state picture templates 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 template to produce a graphical animation with SMGA. A state picture template 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 template and a sequence of states in text and that can be regarded as a movie film. Based on Gestalt principles, we can design and evaluate a state picture template 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 [51] 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 them[2]. We follow this practice to evaluate the SMGA tool because its main purpose is to help humans to conjecture likely invariants that make them better understand.

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

Chapter 4

Designing State Picture Templates

This chapter mainly is inspired to answer **RQ1** and **RQ2**. As mentioned, a state picture template designed by human users is the input of SMGA. By conducting several case studies, this chapter shows how a state picture template is important and proposes two kinds of practical tips for designing a state picture template based on Gestalt principles, such as the common region, proximity, and similarity laws; and for conjecturing likely invariants of protocols/systems. The TAS protocol is used as a simple example to show how usefulness some of those tips are. Based on the tips for conjecturing likely invariants of protocols/systems, we propose guidelines to orientate humans to find likely invariants.

4.1 Designing State Picture Templates of the TAS Protocol

As introduced in Section 2.3, the TAS protocol has one global variable *locked* shared by processes participating in the protocols. Let us suppose that there are three processes participating in the protocol and we can specify the protocol in Maude with two observable components as follows:

- `(locked:` B`)` - a Boolean value B representing *locked*.
- `(pc[`i`]:` L`)` - a value L of process i indicating a location of process i, such as rs and cs.

An initial state can be expressed as follows:

```
(locked: false)
(pc[p1]: rs) (pc[p2]: rs) (pc[p2]: rs)
```

There is one simple way that can automatically generate a state picture template is to use textual display for all observable components as shown in Figure 4.1. It is boring to observe such kinds of text, especially in animations. One possible design is shown in Figure 4.2, where processes are designed as circles with ID inside, sections (rs and cs) are displayed in two designated places that contain values of processes in such sections, and *locked* is still used with textual display. This design is better and we can simply obtain locations of processes and easily count the number of process in each section when the number of process participating in the protocol is small enough. However, *locked* is still textual that is enough for this protocol though, it will be similar to first design if the number of observable components increases. To this end, Figure 4.3 is the best design for us, where processes are designed by circles with different colors and *locked* is design with a lock image, when *locked* is true, the image is displayed, otherwise nothing is displayed. This design follows some proposed tips that are inspired and evaluated by Gestalt principle, and will be introduced in the next section.

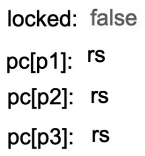

locked: false
pc[p1]: rs
pc[p2]: rs
pc[p3]: rs

Figure 4.1: A straightforward design of the TAS protocol.

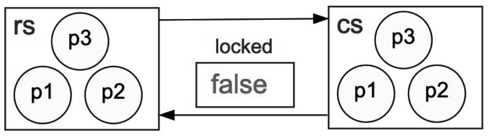


Figure 4.2: One possible design of the TAS protocol.

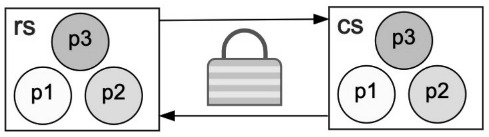


Figure 4.3: The most suitable design of the TAS protocol.

4.2 Practical Tips for Designing State Picture Templates

Based on conducting many case studies [11, 12, 13], there are two assumptions that users need to concern with when using our proposed approach:

- Understanding protocols/systems to be able to specify such protocols/systems as state machines in Maude and CafeOBJ.
- Numbers of values of observable components should be small enough at a moment [11]. For instance, number of processes participating systems/protocols should be two or three.

where the first assumption is to guarantee that humans know some basics of protocols while the second assumption is created based on our experience [11]. In addition, based on such case studies, we propose an essential lesson learned as a main orientation when designing state picture templates. The orientation can be expressed as follows:

- Values of observable components should be visual as much as possible and be arranged based on Gestalt principles, such as common region, proximity, and similarity laws. For example, 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

This orientation is crucial because animation-based visualization works well when humans easily catch and understand changes of still pictures [9] and texts are one of barriers for this issue. The rest of our lessons learned are classified into two categories: tips for designing areas and/or positions of values of observable components and tips for designing kinds of values of observable components. Those tips are summarized in Tables 4.1 and 4.2, where SPT-T stands for State Picture Template Tip. Let us explain some tips using observable components in TAS protocol as examples. Locations and values of processes are designed followed by SPT-T 1 and SPT-T 2, where locations can be trivial to observe based on the common region law of Gestalt principles, and processes can be easily to observe based on the proximity and similarity laws of Gestalt principles. Note that all proposed tips are our lessons learned when conducting many case studies [11, 12, 13, 56].

4.3 Practical Tips for Conjecturing Likely Invariants of Protocols/Systems

Given a state picture template and graphical animations produced based on the template, this section focuses on addressing a question of how to find likely invariants of protocols/systems. To do that, this section gives some practical tips to conjecture likely invariants of protocols/systems. Those tips are summarized in Table 4.3, where LIC-T stands for Likely Invariant Conjecture Tip. Let us use some tips applying for the TAS protocol. We focus on locations and values of processes based on **LTC-T 1**, and *locked* and values of processes based on **LTC-T 2**, we conjecture likely invariants as follows:

TAS-LI 1: There is at most one process in the cs.

TAS-LI 2.1: If *locked* is true, one node is in the cs.

TAS-LI 2.2: If one node is in the cs, *locked* is true.

Table 4.1: Tips for designing areas and/or positions of values of observable components

SPT-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 perceived and display some diagram on the designated pane.
SPT-T 2	To recognize what pieces of information, allocate the pane (or place) for each such piece of information such that we can visually perceive they are shared by all processes and nodes and display them on the designated panes adequately.
SPT-T 3	To recognize whether there are some that are more crucial than the others among the shared resources prepare the panes (or places) for them and display them there adequately.
SPT-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 perceive what pieces of information are owned by what processes or nodes and display them on the designated panes adequately.
SPT-T 5	If there exist observable components that have natural numbers as their values and the values are small enough, the values should be visually expressed nearby together so that we can see them simultaneously and compare them instantaneously.

TAS-LI 3.1: If *locked* is false, no node is in the cs.

TAS-LI 3.2: If no node is in the cs, *locked* is false.

Those tips can be confirmed Maude search command and they are confirmed when three processes participate in the protocol. Moreover, **TAS-LI 1**, **2.2**, and **3.1** are proven with CafeOBJ mentioned in Section 2.3. Note that the tips are proposed in [11] and applied to many case studies [11, 12, 56, 57, 58].

4.4 Guidelines for Discovering Likely Invairants of Protocols/Systems

From the proposed tips for conjecturing likely invariants and our experiences from case studies [11, 12, 56], the section propose guidelines as a generic way to find likely invariants of protocols/systems using graphical animations. Human users can utilize this guidelines to find likely invariants of protocols/systems. Note that this guidelines and good state picture templates are complements together because it depends on humans who are good at visual perception. Our proposed guidelines can be summarized as follows:

Table 4.2: Tips for designing kinds of values of observable components

SPT-T 6	When an observable component can have two different values, it should be visually/graphically represented as a light bulb.
SPT-T 7	If a value of an observable component does not change, it should be expressed at a fixed label.
SPT-T 8	When an observable component can have three or more (but moderate) different values, 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, and if the observable component has such special value, nothing should be displayed.
SPT-T 9	If there are some local variables, 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 and/or vertically.
SPT-T 10	When an observable component has a composite value which consists of more than one component value inside (i.e. $(name\text{: } value_1, value_2, ..., value_n,))$, we need to carefully select which component values (e.g. $value_n$) to visualize.

Step 1: Based on proposed tips (mainly with **LIC-T 1** and **LIC-T 2**), find as many likely invariants as possible by carefully observing graphical animations. Some conditions can be extracted from such likely invariants.

Step 2: Use the *Pattern matching* feature (which will introduced in the next chapter) in order to find states that satisfy each likely invariant found at **Step 1** (or any other steps) or a condition extracted from the likely invariant.

Step 3: By carefully observing the graphical animation of the states obtained at **Step 2** and/or the states as still pictures, where a likely invariant D_1 is used at **Step 2** to obtain the states, confirm the guessed likely invariant so far to some extent and/or find as many likely invariants as possible. Let D_2 be one of the likely invariant found. Then, $D_1 \Rightarrow D_2$ is one possible likely invariant candidate.

Step 4: Repeat **Step 2** with the likely invariants, such as D_1, found at **Step 3**, followed by **Step 3**, from which new likely invariants could be found.

The proposed guidelines aim to find likely invariants that have a form of the implication relation.

Table 4.3: Tips for conjecture likely invariants of protocols/systems using graphical animations

LIC-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, or some likely invariants related to one observable component.
LIC-T 2	By concentrating on two different-kind observable components, we may find a relation between them, from which we may conjecture some likely invariants.
LIC-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 likely invariants. It is necessary to use the *Pattern matching* feature of r-SMGA (that will discuss in section 5) 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.
LIC-T 4	By carefully investigating conditions of some guessed likely invariants and the states found by the *Pattern matching* feature of (part of) the conditions, we may conjecture some other likely invariants.

Then, humans can compose complex likely invariants (contain more than 2 variables) based on their understanding.

To make the guidelines more effective, in the next chapter, we will propose r-SMGA, a revised version of SMGA, which provides features inspired by some steps of the guidelines. Those features assist human users to find likely invariants leading by the proposed guidelines.

Chapter 5

Integration of SMGA and Maude

Based on two kinds of tips mentioned in the previous chapter, this chapter introduces the revised version of SMGA (r-SMGA) where new and revised features have been implemented that can help human users to find likely invariants of protocols/systems. Those new and revised features include special display features (e.g. users can display visually some data structures, such as array and queue), interactive features (e.g. users can interact visual objects in the state pictures, such as focusing and hiding which visual objects that users interest or less interest), and some Maude features implemented by integrating SMGA and Maude so that r-SMGA can use some powerful features of Maude, such as associative-commutative binary operators as well as context-free grammars, and reachability analysis.

5.1 Features of the Revised Version of SMGA

The main goal of the this section is to provide interactive features for helping users to conjecture likely invariants of a protocol/system for which graphical animations are prepared and controlled by r-SMGA. r-SMGA allows human users to interact with visual objects used in the state picture template, for example, focusing on or hiding them when the users are more or less interested in them. r-SMGA also helps human users to get more insights of a protocol/system by finding the pictures that satisfy some with powerful pattern matching empowered by Maude from its graphical animations. We have integrated r-SMGA and Maude so that r-SMGA can use powerful features of Maude, such as reachability analysis, parsing, and LTL model checking. Moreover, Maude has a powerful pattern matching feature based on rich grammars, such as context-free grammars and associative and/or commutative binary operators, so that users can search graphical animations for more various information as they want than regular expressions in the previous version. To this end, we use a server as a bridge to communicate between r-SMGA and Maude. The server uses Maude bindings [59] to communicate with Maude via APIs, while the server uses sockets to communicate with r-SMGA via message passing. r-SMGA's new features are described in the rest of the section.

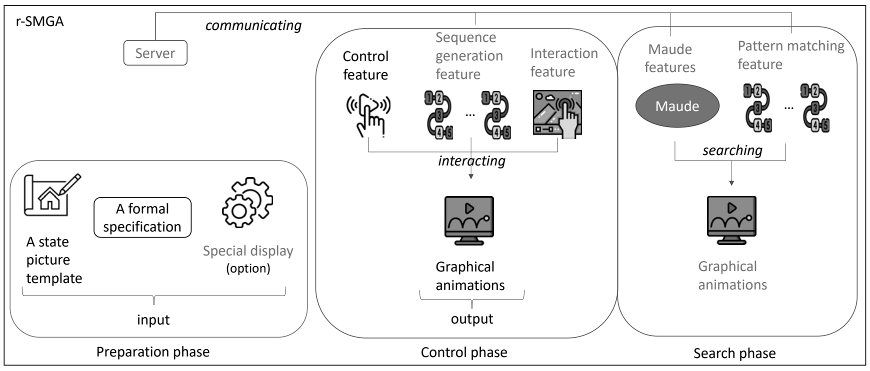
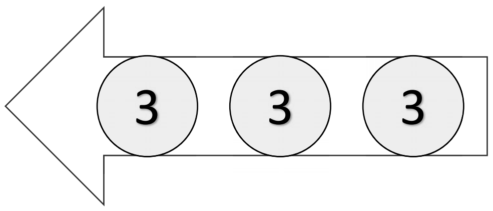
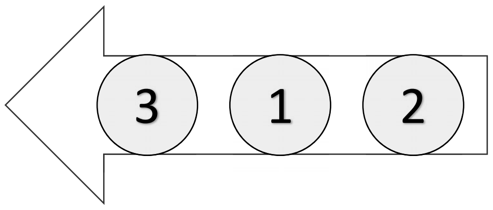


Figure 5.1: Overview of r-SMGA

Figure 5.1 displays an overview of r-SMGA in which light-blue texts refer to the new and revised features. There are three phases when using r-SMGA: preparation, control, and search. The main purpose of the preparation and control phases is to produce an input and an output, while the search phase focuses on analyzing data with the pattern matching feature. In the preparation phase, we use a state picture template and a formal specification of a protocol as the input that is fed into r-SMGA. Note that users do not need to prepare a state sequence as in the previous version. Moreover, we provide a feature called *Special display* to help users to visualize some specific data structures such as array and queue. Some other visualization features, such as displaying network containing huge messages in [13], can be reused in r-SMGA. Let us introduce some examples for the *Special display* feature. For queue, we provide two ways so that each element of a queue is displayed visually. The first way is for an observable component whose value is a queue only. For example, let look at the observable component `(q: 3 | 1 | 2 | empty)` where `q` is a name of the observable component, `3 | 1 | 2 | empty` is its value and `3` is the top of the queue. One possible way is to display the observable component as text in the same way of SMGA provided. In r-SMGA, users can display visually each element as they expect. Note that users must know the maximum possible elements in the queue to prepare such number of positions in the queue. Let look at the state picture template of the observable component (we assume that the queue has at most three elements so we prepare three positions in the queue) as follows:



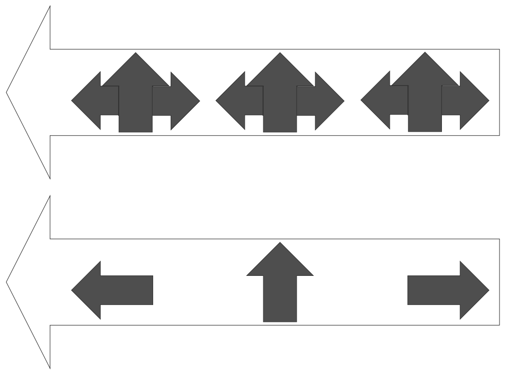
Where each circle in the arrow represents each element in the queue and the top element in the queue is displayed on the left-most. Each circle is designed for all possible elements in the queue so that the queue above can be visually displayed as users expect. The following figure is the state picture for the observable component mentioned above



The second way is for two observable components and queue is a value of one of them. Element is the queue represents a ID of the rest one and users want to display the queue based on the rest one. For example, let look at some observable components as follows:

```
(lane: 3 | 1 | 2 | empty)
(car[1]: straight) (car[2]: right) (car[3]: left)
```

where `lane` and `car[_]` are name of the observable component, `3 | 1 | 2 | empty` and directions are their values. Users expect to display the observable component `lane` with its value are values of the observable components `car`. In r-SMGA, the state picture template and the state picture of the queue for the case above can be prepared as follows:



where the figure on the top and bottom side are the state picture template and the state picture, respectively.

In the control phase, we implement two new features called *Sequence generation* and *Interaction*. The *Sequence generation* feature aims to automatically generate a state sequence based on the formal specification of a protocol on the fly. It consists of five functions:

- **Default generation**: The function automatically generates a random sequence (by selecting randomly one of the successor states for a next state) whose length is up to a fixed number (100 by default). This function guarantees that two consecutive states are different in the sequence. Once a new state sequence is generated, it will be added to a list where its index denotes the state sequence. Users can select any state sequence in the list by an index to reuse it without generating it again.

- **Update**: The function uses a selected state sequence from the list, then generates a new random state sequence and replaces the selected state sequence by the new one. Users can adjust the length of a state sequence before producing a new state sequence.

- **Add**: The function works similarly to **Default generation**, except that users can set a length of a new state sequence before generating.
- **Clear**: The function erases a selected state sequence from the list.
- **Reset list**: The function erases all state sequences from the list.

Users can utilize the *Interaction* feature to interact with a state picture template or a state picture while observing the animations. This feature consists of two functions:

- **Focus**: This function allows users to focus on selected visual objects in a state picture template or a state picture by displaying only those visual objects and not displaying the remaining visual objects.
- **Hide**: Users can select visual objects in a state picture template or a state picture that they want to hide. Then, the selected visual objects are not displayed on the screen.

We also provide three more functions for each function above: undo, redo, and reset. Undo allows users to cancel the latest action (**Focus** or **Hide**), while redo allows users to carry out the latest action again. Reset allows users to go back to the original state picture template. Note that the function reset is different to **reset list** in the *sequence generation* feature. In addition, users can utilize the *Interaction* feature, while running some other features.

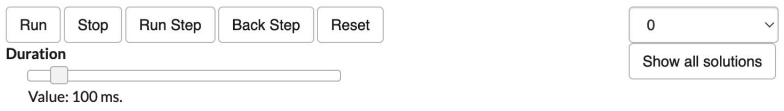
In the search phase, there are two features named *Maude* and *Pattern matching*. In the *Maude* feature, we provide two functions called search command (reachability analysis) and model checking as follows:

- **Search command**: The input of this function is a specification and a command including parameters mentioned in Section 2. Given the corresponding parameters, such as an initial state, a target state, and a number of solutions, the function calls to the Maude `search` command. If the number of solutions in the parameters is one, the function returns a state sequence leading to the target state from the initial state. When the number is greater than one, the function returns a list of state sequences where indices in the list denote the state sequences. In the list, users can utilize a state sequence by selecting the corresponding index of the list. If there is no solution, an alert message is returned.
- **Model checking**: Given a specification of a protocol, an LTL formula, and an initial state, we can conduct model checking with r-SMGA. If the protocol does not satisfy the formula with the initial state, a counterexample is returned that has the form of a state sequence with a loop mentioned in Section 2. Note that if the reachable state space of the protocol is huge, **Search command** and **Model checking** can stuck or take a long time to return results.

The purpose of *Pattern matching* feature is to find states satisfying some conditions from the state sequence being used. In r-SMGA, the *Pattern matching* feature is implemented based on some features empowered by Maude so that it can work on various cases, such as finding some specific messages in the network (formalized as a soup, an associative-commutative collection, of messages) that we cannot do with the previous version. This feature uses state sequences in the list maintained by r-SMGA and a pattern with a condition written in Maude as the input. The pattern and the condition are defined in the same way to the target state and the condition in the **Search command** function. The output is a list of states that match the pattern with the condition, or a message "no solution" if there is no such a state. The *Pattern matching* feature has the following three functions:

- **Pattern matching on a sequence**: Users select one state sequence from the list and fill a pattern with a condition. Then, the function returns a list of states that match the pattern and satisfy the condition from the selected state sequence.
- **Pattern matching on some sequences**: Users can select multiple state sequences from the list and fill a pattern with a condition. Then, the function returns a list of sequences of states that match the pattern and satisfy the condition from the selected state sequences. Users can select one sequence of the list returned to observe states that satisfy the pattern and the condition.
- **Pattern matching on all sequences**: This is the same as the second function provided that all state sequences in the list are selected.

In the search phase, we provide two ways to display the output: (i) a still picture that includes all states that match a pattern with a condition and (ii) graphical animations of such states. Displaying the output as animations can help human users to quickly recognize the difference between such states, which may be useful to conjecture likely invariants based on the proposed guideline in Section 4.4. We will show its usefulness in the rest of the dissertation. The following figure shows functions of the displaying the output as graphical animations.



where buttons on the left-hand side are similar to those used for the functions in the control feature; a list and a button on the right-hand side are a result (a list of solutions) returned from the *Maude* feature and the *Pattern matching* feature and a function to load all states (in the result-aboved) to the output display, respectively. We summarize the new and revised features with their purposes and functions in Table 5.1.

Table 5.1: Summary of new and revised features.

Feature	**Purpose**	**Functions**
Special display	Visualizing some specific data structures such as array and queue	
Sequence generation	Generating a state sequence on the fly and keeping state sequences in a list	Default generation, add, update, clear, and reset list
Interaction feature	Interacting with elements from a state picture template or a state picture	Focusing and hiding
Maude feature	Using Maude features that Maude supports	Search command and model checking
Pattern matching feature	Searching states by matching a pattern with a condition	Pattern matching on a sequence, some sequences, and all sequences in the list
Displaying the output of the search phase	Displaying the output in the search phase as animations	Control functions for animations and showing all output states

Chapter 6

Case study: the Mellor-Crummey-Scott Protocol

This chapter uses the Mellor-Crummey-Scott (MCS) protocol as an example to demonstrate the usefulness of both kinds of tips, especially tips for designing state picture templates. This chapter is mainly divided into two parts: graphical animations of the MCS protocol and formal verification of the confirmed invariants of the MCS protocol in CafeOBJ, where the former shows a flow of how to conjecture likely invariants using r-SMGA while the latter is to prove such likely invariants by proof scores in CafeOBJ. Firstly, this chapter introduces the MCS protocol and its specification in Maude. This chapter then graphically animates the MCS protocol where a state picture template is designed based on the proposed tips for designing state picture templates. This chapter also shows the importance of state picture templates by comparing our proposal with the previous work and evaluating them by Gestalt principles. Then, based on the proposed tips for conjecturing likely invariants, we can conjecture several likely invariants, and those likely invariants are confirmed with the **Search command** function in r-SMGA. One flawed version of the MCS protocol is used to show that our approach still works even the input is a defective version. Finally, most of the confirmed invariants (survived via the **Search command**) are theorem proved to demonstrate how well humans understand based on our proposed approach. We also report reasons that some of the confirmed invariants cannot be done with CafeOBJ even we intuitively know that they are correct.

6.1 Graphical Animations of the MCS Protocol

6.1.1 Description and Maude Specification

The MCS protocol [15] 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

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

Figure 6.1: The MCS protocol in Algol-like pseudo-code

W. Dijkstra Prize in Distributed Computing went to their paper[1]. It can be described in Algol-like pseudo-code as shown in Figure 6.1. 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 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 stored in 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. The MCS protocol uses a virtual queue basically composed of process IDs by using $next_p$. Figure 6.2 (a) 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 not atomic. Therefore, there may be some elements that have not yet been completely put into it. Figure 6.2 (b) 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 $pred_{p_1} = p_2$.

[1]`www.podc.org/dijkstra/2006-dijkstra-prize`

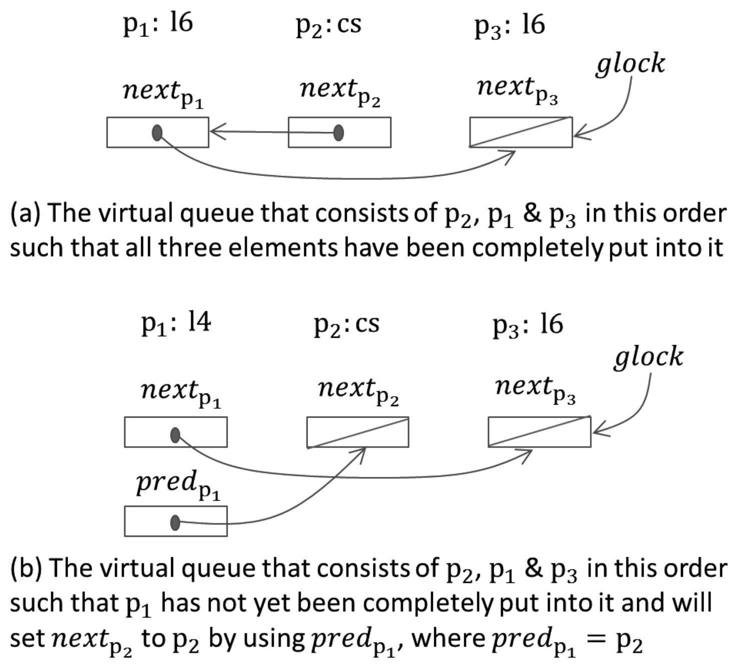

p_1: l6
p_2: cs
p_3: l6
glock
$next_{p_1}$
$next_{p_2}$
$next_{p_3}$
(a) The virtual queue that consists of p_2, p_1 & p_3 in this order such that all three elements have been completely put into it
p_1: l4
p_2: cs
p_3: l6
glock
$next_{p_1}$
$next_{p_2}$
$next_{p_3}$
$pred_{p_1}$
(b) The virtual queue that consists of p_2, p_1 & p_3 in this order such that p_1 has not yet been completely put into it and will set $next_{p_2}$ to p_2 by using $pred_{p_1}$, where $pred_{p_1} = p_2$

Figure 6.2: Virtual queue used in the MCS protocol

We suppose that there are three processes that participate in MCS protocol. Let M_{MCS} formalize the MCS protocol. Each state in S_{MCS} is expressed as follows:

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

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, l1 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. I_{MCS} consists of one state. Let `init` equal the initial state.

T_{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}
```

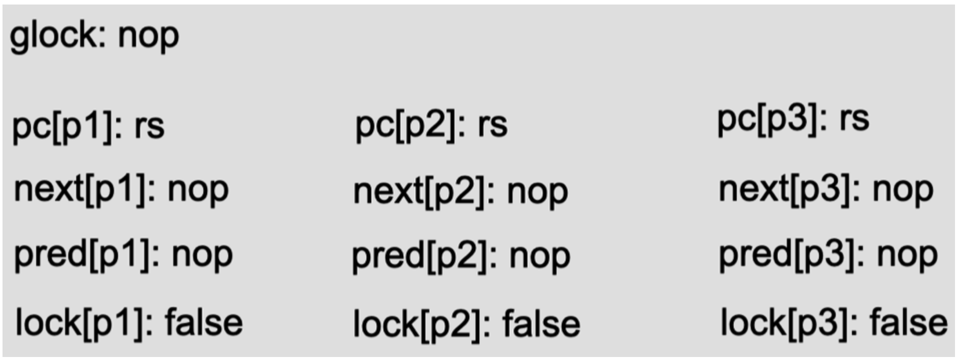


Figure 6.3: A straightforward state picture template for the MCS protocol

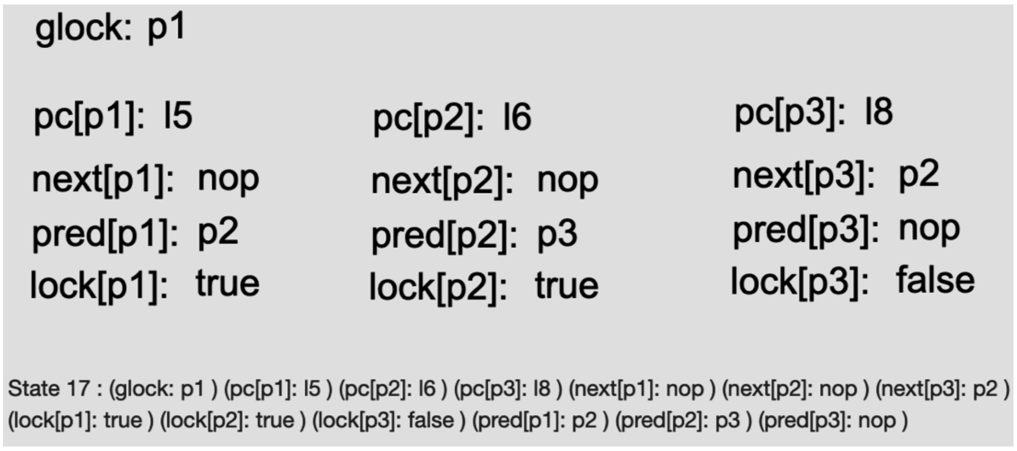
```
   => {(pc[P]: l6) (pred[P]: Q) (next[Q]: P) OCs} .
rl [chlck] : {(pc[P]: l6) (lock[P]: false) OCs}
   => {(pc[P]: cs) (lock[P]: false) OCs} .
rl [exit] : {(pc[P]: cs) OCs}
   => {(pc[P]: l7) OCs} .
rl [rpnxt] : {(pc[P]: l7) (next[P]: Q) OCs}
   => {(pc[P]: (if Q == nop then l8 else l11 fi)) (next[P]: 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} .
crl [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`.

6.1.2 Designing the State Picture Template

It is also possible to automatically generate a straightforward state picture template for the MCS protocol, such as the one shown in Figure 6.3. State pictures generated from the state picture template, such as the one shown in Figure 6.4 are almost the same as states in text:

```
{(glock: p1)
```



glock: p1

pc[p1]: l5 pc[p2]: l6 pc[p3]: l8

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]: l5) (pc[p2]: l6) (pc[p3]: l8) (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)

Figure 6.4: A straightforward state picture for the MCS protocol

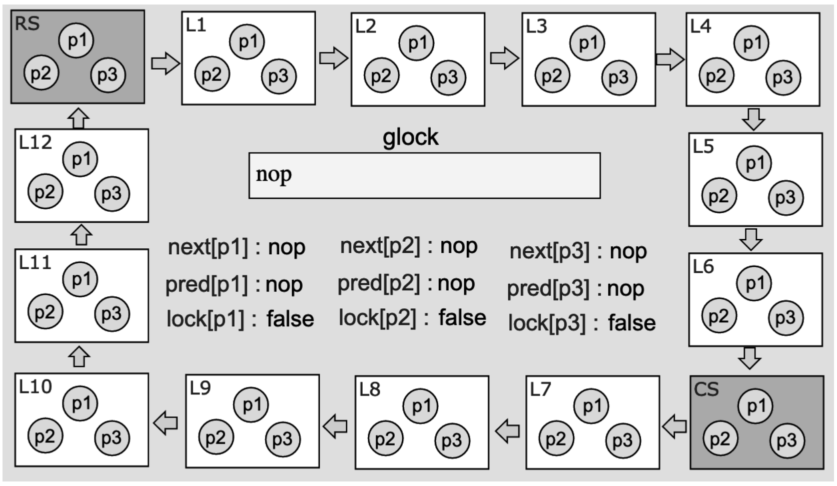


Figure 6.5: An old state picture template for the MCS protocol

```
(pc[p1]: l5) (next[p1]: nop) (pred[p1]: p2) (lock[p1]: true)
(pc[p2]: l6) (next[p2]: nop) (pred[p2]: p3) (lock[p2]: true)
(pc[p3]: l8) (next[p3]: p2) (pred[p3]: nop) (lock[p3]: false)}
```

Observing such texts, especially in animations, is too boring and so observable components should be visual as much as possible mentioned as an essential tip.

Nguyen and Ogata [60] made the state picture template shown in Figure 6.5 for the MCS protocol. A state picture generated from the state picture template is shown in Figure 6.6. The state picture allows us to immediately realize that processes p1, p2 and p3 are located at l5, l6 and l8, respectively. Nguyen and Ogata [60] conducted a case study in which several likely invariants of the MCS protocol can be discovered by observing graphical animations of the MCS protocol such that each state picture used in the graphical animation is generated from the state picture template. For example, one of the likely invariants found is as follows:

No state such that a process is at cs, l7, l8, l10, *or* l11 *and another process is at* cs, l7, l8,

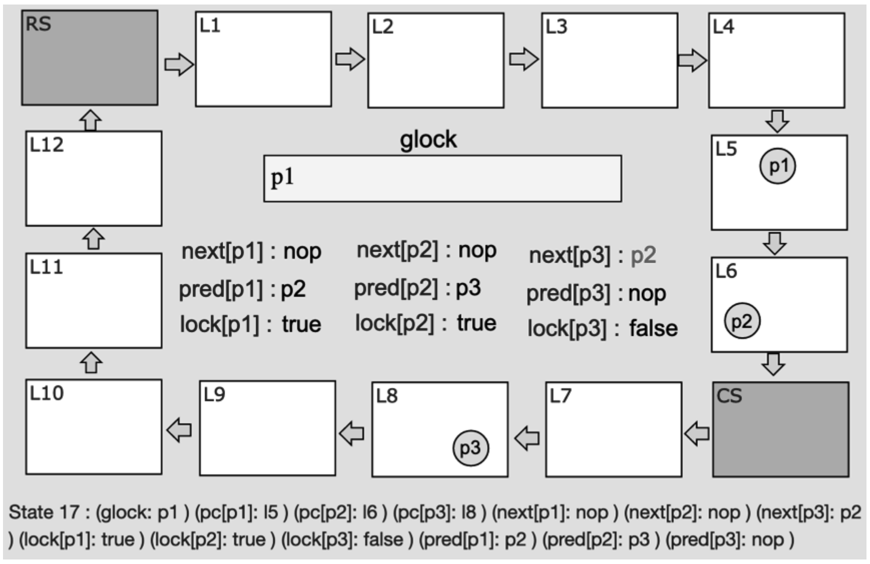

RS
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
CS
glock
p1
next[p1] : nop
pred[p1] : p2
lock[p1] : true
next[p2] : nop
pred[p2] : p3
lock[p2] : true
next[p3] : p2
pred[p3] : nop
lock[p3] : false
p1
p2
p3
State 17 : (glock: p1) (pc[p1]: l5) (pc[p2]: l6) (pc[p3]: l8) (next[p1]: nop) (next[p2]: nop) (next[p3]: p2) (lock[p1]: true) (lock[p2]: true) (lock[p3]: false) (pred[p1]: p2) (pred[p2]: p3) (pred[p3]: nop)

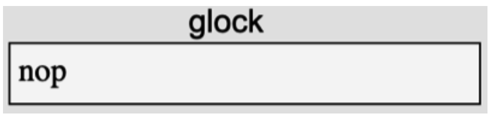
Figure 6.6: An state picture instance for the MCS protocol

l10*, or* l11.

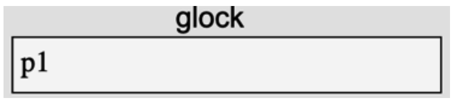
Let the likely invariant be called MCS-LI 0 in this section.

Although the state picture shown in Figure 6.6 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 templates, the representations must be able to be visually/graphically perceivable. We use proposed tips to come up with the state picture template shown in Figure 6.7. Figure 6.8 shows a state picture generated from the state picture template.

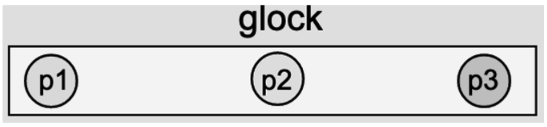
The design of the *glock* representation used in Figure 6.5 is as follows:



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 Figure 6.7 is as follows:



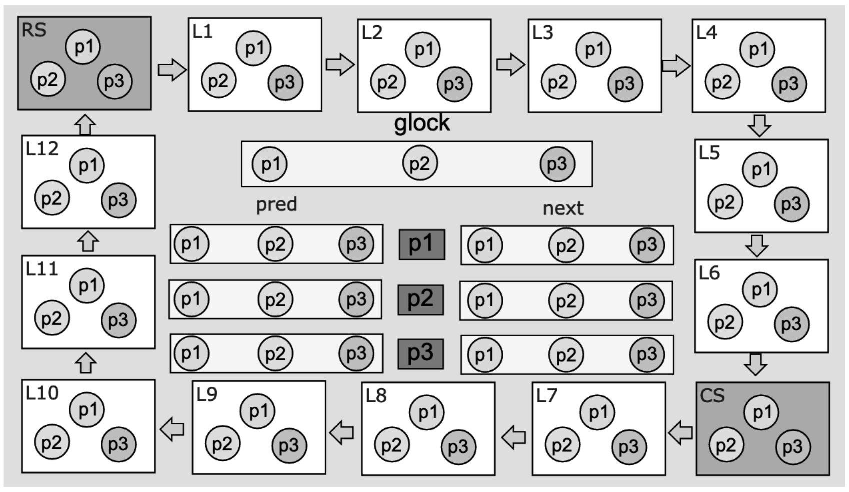
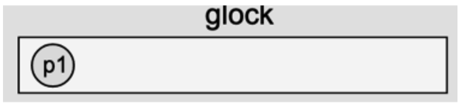


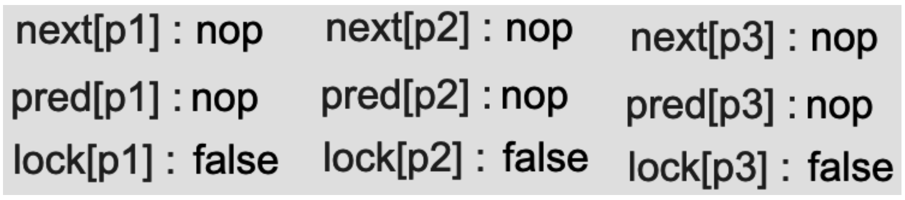
Figure 6.7: A new state picture design for the MCS protocol

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*. For example, when the value is p1, it is displayed as follows:



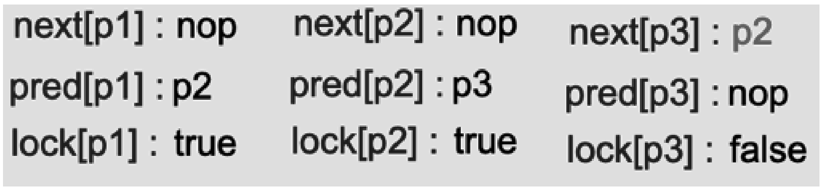
We can say that the *glock* representation used in Figure 6.7 and Figure 6.8 helps human users more visually/graphically perceive its value than the one used in Figure 6.5 and Figure 6.6.

The design of the $next_p$, $pred_p$ and $lock_p$ representations for each process p used in Figure 6.5 is as follows:

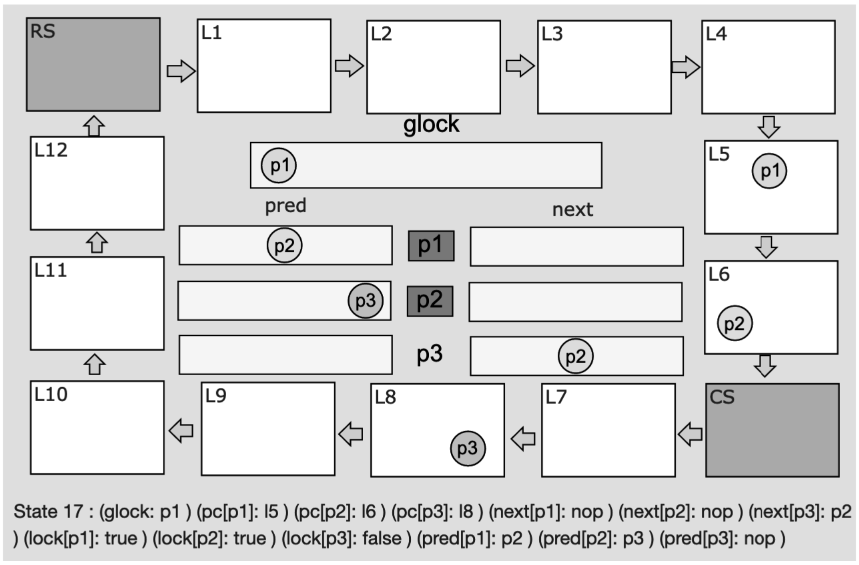


next[p1] : nop	next[p2] : nop	next[p3] : nop
pred[p1] : nop	pred[p2] : nop	pred[p3] : nop
lock[p1] : false	lock[p2] : false	lock[p3] : false

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



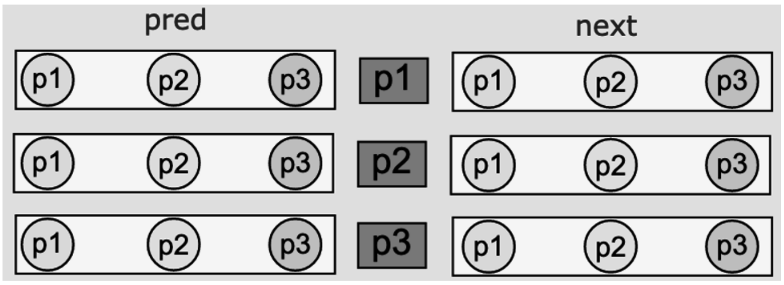
next[p1] : nop	next[p2] : nop	next[p3] : p2
pred[p1] : p2	pred[p2] : p3	pred[p3] : nop
lock[p1] : true	lock[p2] : true	lock[p3] : false



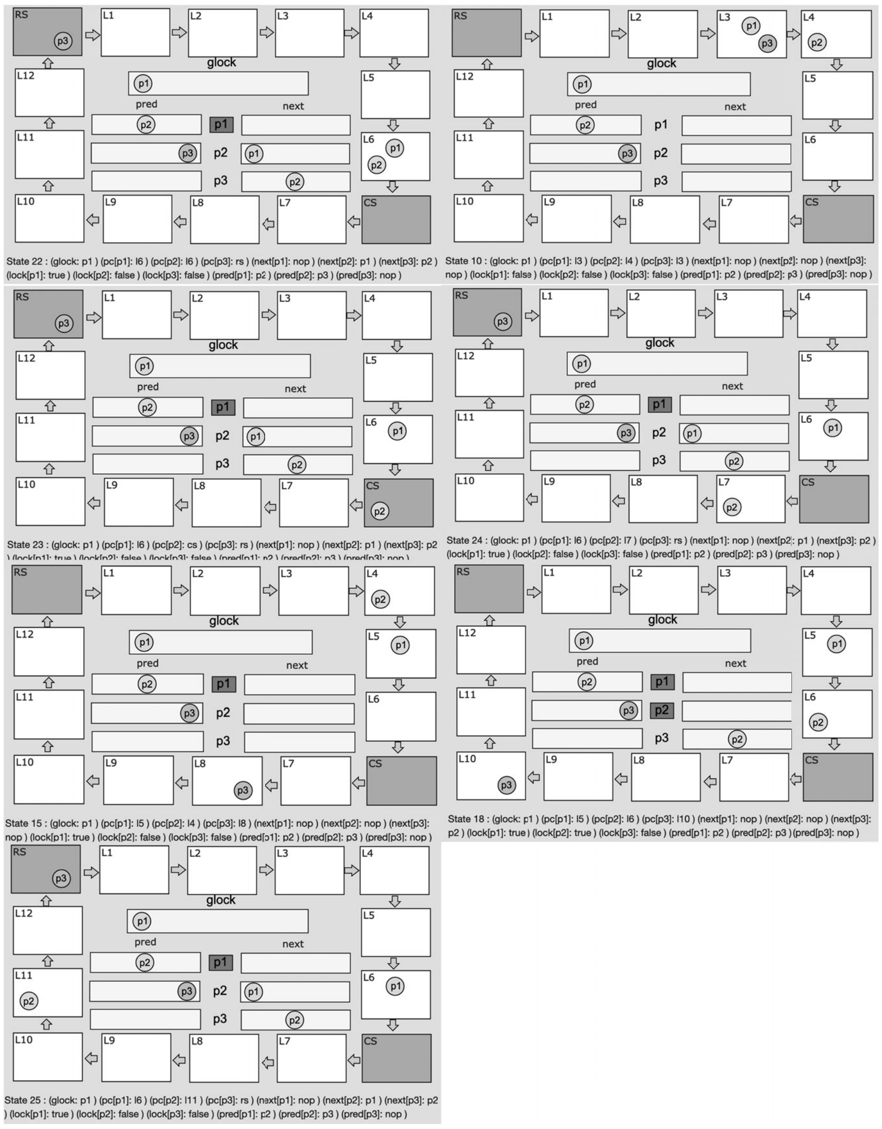
State 17 : (glock: p1) (pc[p1]: l5) (pc[p2]: l6) (pc[p3]: l8) (next[p1]: nop) (next[p2]: nop) (next[p3]: p2) (lock[p1]: true) (lock[p2]: true) (lock[p3]: false) (pred[p1]: p2) (pred[p2]: p3) (pred[p3]: nop)

Figure 6.8: A new state picture instance for the MCS protocol

The design of the $next_p$, $pred_p$ and $lock_p$ representations for each process p used in Figure 6.7 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 p1, p2 and p3, respectively. For example, if the value of $next_{\mathrm{p1}}$ 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 p1, p2 and p3, respectively. When $lock_{\mathrm{p}i}$ 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 p1, p2 and p3, respectively. For example, if the value of $pred_{\mathrm{p1}}$ 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 $next_{\mathrm{p1}}$ is nop, $pred_{\mathrm{p1}}$ is p2, $lock_{\mathrm{p1}}$ is true, $next_{\mathrm{p2}}$ is nop, $pred_{\mathrm{p2}}$ is p3, $lock_{\mathrm{p2}}$ is true, $next_{\mathrm{p3}}$ is p2, $pred_{\mathrm{p3}}$ is nop and $lock_{\mathrm{p3}}$ is false, those values are displayed as follows:



```
State 22 : (glock: p1 ) (pc[p1]: l6 ) (pc[p2]: l6 ) (pc[p3]: rs ) (next[p1]: nop ) (next[p2]: p1 ) (next[p3]: p2 )
(lock[p1]: true ) (lock[p2]: false ) (lock[p3]: false ) (pred[p1]: p2 ) (pred[p2]: p3 ) (pred[p3]: nop )
```

```
State 10 : (glock: p1 ) (pc[p1]: l3 ) (pc[p2]: l4 ) (pc[p3]: l3 ) (next[p1]: nop ) (next[p2]: nop ) (next[p3]:
nop ) (lock[p1]: false ) (lock[p2]: false ) (lock[p3]: false ) (pred[p1]: p2 ) (pred[p2]: p3 ) (pred[p3]: nop )
```

```
State 23 : (glock: p1 ) (pc[p1]: l6 ) (pc[p2]: cs ) (pc[p3]: rs ) (next[p1]: nop ) (next[p2]: p1 ) (next[p3]: p2
) (lock[p1]: true ) (lock[p2]: false ) (lock[p3]: false ) (pred[p1]: p2 ) (pred[p2]: p3 ) (pred[p3]: nop )
```

```
State 24 : (glock: p1 ) (pc[p1]: l6 ) (pc[p2]: l7 ) (pc[p3]: rs ) (next[p1]: nop ) (next[p2]: p1 ) (next[p3]: p2 )
(lock[p1]: true ) (lock[p2]: false ) (lock[p3]: false ) (pred[p1]: p2 ) (pred[p2]: p3 ) (pred[p3]: nop )
```

```
State 15 : (glock: p1 ) (pc[p1]: l5 ) (pc[p2]: l4 ) (pc[p3]: l8 ) (next[p1]: nop ) (next[p2]: nop ) (next[p3]:
nop ) (lock[p1]: true ) (lock[p2]: false ) (lock[p3]: false ) (pred[p1]: p2 ) (pred[p2]: p3 ) (pred[p3]: nop )
```

```
State 18 : (glock: p1 ) (pc[p1]: l5 ) (pc[p2]: l6 ) (pc[p3]: l10 ) (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 )
```

```
State 25 : (glock: p1 ) (pc[p1]: l6 ) (pc[p2]: l11 ) (pc[p3]: rs ) (next[p1]: nop ) (next[p2]: p1 ) (next[p3]: p2
) (lock[p1]: true ) (lock[p2]: false ) (lock[p3]: false ) (pred[p1]: p2 ) (pred[p2]: p3 ) (pred[p3]: nop )
```

Figure 6.9: Some pictures for extended CS region

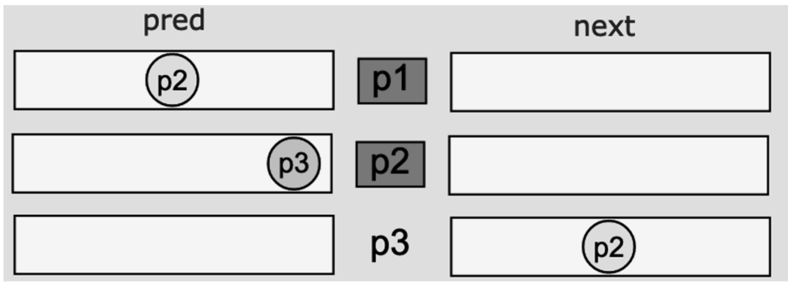


Table 6.1 summarizes observable components of the MCS protocol and the tips used to design their visualization.

Table 6.1: Observable components and their tips used to design

Observable components	**Tips**
pc[*i*]	SPT-T 1&2
lock[*i*]	SPT-T 4&6&9
pred[*i*]	SPT-T 4&6&8&9
next[*i*]	SPT-T 4&6&8&9
glock	SPT-T 3&8

6.1.3 Evaluation of State Picture Template Based on Gestalt Principles

On both the old state picture template and the new state picture template of the 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 the 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 template 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 template. 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 template.

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.

6.1.4 Likely Invariants Discovery Based on Our Proposed State Picture Template

Using **LIC-T 1** (focus on the location of processes, such as rem and cs) and carefully observing graphical animations for the MCS protocol in which new state pictures, such as Figure 6.6, were used, we realized that there is always at most one process at cs, l7, l8, l10 and l11. This is exactly the same as MCS-LI 0 discovered based on old state pictures, such as Figure 6.8. We call these sections (cs, l7, l8, l10 and l11) CS region. We also noticed 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 likely invariant that can be conjectured by carefully observing graphical animations for the MCS protocol in which new state pictures, such as Figure 6.6, are used is as follows:

- MCS-LI 1: There exists at most one process 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 in extended CS region.

Extended CS region is one key concept that captures one important aspect of the MCS protocol and MCS-LI 1 is very crucial in that several other likely invariants can be discovered based on this likely invariant. Figure 6.9 shows some state pictures that capture MCS-LI 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, $lock_{\mathrm{p1}}$ is true and $lock_{\mathrm{p2}}$ is false and therefore p2 is only the process in extended CS region in the sense of MCS-LI 1 because p1 satisfies condition (2) in MCS-LI 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, $pred_{\mathrm{p1}}$ is p2 and $pred_{\mathrm{p3}}$ is nop and therefore p3 is only the process in extended CS region in the sense of MCS-LI 1 because p1 satisfies condition (1) in MCS-LI 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.

Based on **LIC-T 1** and focusing on a process in CS region or extended CS region, we can recognize the following likely invariant:

- MCS-LI 2.1: Whenever there is a process at l10, there is at least one process at l3, l4, l5 or l6;
- MCS-LI 2.2: Whenever there is a process at l11, there is at least one process at l6.

The bottom left state picture of Figure 6.9 is an example of MCS-LI 2.1 and the bottom right state picture of Figure 6.9 is an example of MCS-LI 2.2.

Carefully observing the $next_p$ and $pred_p$ representations for each process p (based on **LIC-T 1**), 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 likely invariants:

- MCS-LI 3.1: The value of $next_p$ for each process p is never p.
- MCS-LI 3.2: The value of $pred_p$ for each process p is never p.

For example, taking a look at the $next_{\mathrm{p1}}$ representation or rectangle, it is visually/graphically observable that the circle on which p1 is written never comes into sight on the designated position. This is because $next_{\mathrm{p1}}$ is visually/graphically represented in the new state picture design.

Based on **LIC-T 2**, some relations between two observable components can be discovered by carefully observing graphical animations, from which some likely invariants 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 likely invariants:

- MCS-LI 4.1: If *glock* is nop, then there is no process at l3, l4, l5 or l6 or in CS region;
- MCS-LI 4.2: If *glock* is a process p, then p is located at l3, l4, l5 or l6 or in CS region;
- MCS-LI 4.3: If *glock* is not nop (or equivalently a process), then there exists a process in CS region.

Similarly, a relation between the pred[p] observable component and the pc[p] observable component (**LIC-T 2**) allows us to conjecture the following likely invariants:

- MCS-LI 5: If $pred_p$ for each process p is nop, then p is never located at l4, l5, l6 or l12.

Similarly, a relation between the pc[p] observable component and the next[p] observable component (**LIC-T 2**) allows us to conjecture the following likely invariants:

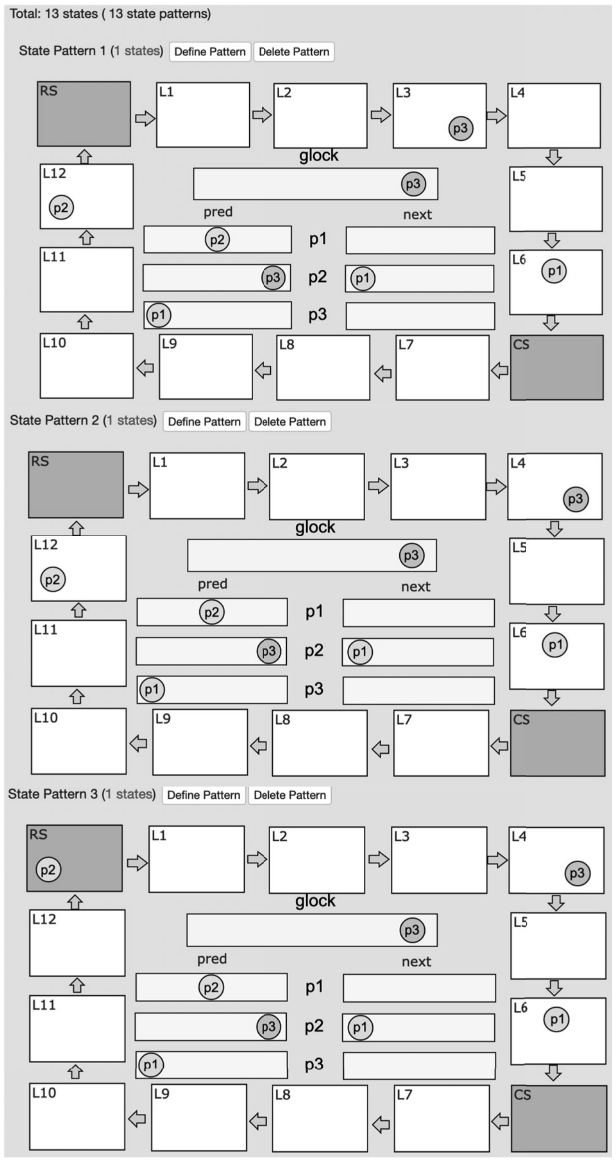

Total: 13 states (13 state patterns)
State Pattern 1 (1 states)
Define Pattern
Delete Pattern
RS
L1
L2
L3
L4
L5
L6
CS
L7
L8
L9
L10
L11
L12
glock
pred
next
p1
p2
p3
State Pattern 2 (1 states)
State Pattern 3 (1 states)

Figure 6.10: Three state pictures discovered by the *Pattern matching* feature.

- MCS-LI 6.1: If each process p is located at l2 or l9, then $next_p$ is nop;
- MCS-LI 6.2: If $next_p$ for each process p is nop, then p is not located at l11 or l12.

Similarly, a relation between the lock[p] observable component and the pc[p] observable component (**LIC-T 2**) allows us to conjecture the following likely invariants:

- MCS-LI 7.1: If $lock_p$ for each process p is true, then p is located at l5 or l6;
- MCS-LI 7.2: 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.

Based on **LIC-T 3**, 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 animations such that $pred_{\mathrm{p1}}$ is p2 and p2 is located at rs, l1, l2, l9 or l12. One possible remedy for it is to use the *Pattern matching* feature of r-SMGA. Given a pattern and a condition written by human users, the feature finds all states in an input sequence of states such that they match the pattern and the condition. When we would like to find all states in an input sequence of states such that $pred_{\mathrm{p1}}$ is p2 and p2 is located at rs, l1, l2, l9 or l12, it suffices to write the following the pattern and the condition in Maude:

```
pattern:
(pred[p1]: p2) (pc[2]: La:Label) OCs:Sys

condition:
La:Label = l1 or La:Label = l2 or La:Label = rs or La:Label = l9 or La:Label = l12
```

where `OCs:Sys` refers to the other observable components. Figure 6.10 shows three state pictures among the states discovered by the *Pattern matching* feature of r-SMGA.

It does not suffice, however, to use the feature abovementioned so as to conjecture non-trivial likely invariants. 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 MCS-LI 3.1. Let (p, q) be each of the six combinations to consider. Carefully observing all states discovered by the *Pattern matching* feature for the six combinations (**LIC-T 3**) to consider, we can conjecture some non-trivial likely invariants:

- MCS-LI 8.1: 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;
- MCS-LI 8.2: If $pred_p$ is q and *glock* is q, then p is not located at l3, l4 or l5;
- MCS-LI 8.3: 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 noticed 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 *Pattern matching* feature of the condition of MCS-LI 8.1, while there are only states in which $next_q$ is nop among the states found by the feature of the condition of MCS-LI 8.2. Then, finding the states with the feature of the second sub-condition only of MCS-LI 8.2 because the first sub-condition is shared by both likely invariants, we realized that $next_q$ is nop in all of them (**LIC-T 4**). Therefore, we came up with the following likely invariant:

- MCS-LI 9: If *glock* is a process p (or equivalently non-nop), $next_p$ is nop.

We have used the **Search command** of r-SMGA to model check that all likely invariants conjectured in this section are invariant properties with respect to the state machine formalizing the MCS protocol. Table 6.3 and Table 6.2 show all found likely invariants are not yet survived and survived with the **Search command** function of r-SMGA, respectively. The **Tips** column in Table 6.3 refers to specific tips to conjecture likely invariants.

Table 6.2: False invariants of the MCS protocol

No.	**Content**	**Tips**
1	there is at most one process in extended cs region, l9 and l12	LIC-T 1
2	if a process is in l5 or l6, its lock is true	LIC-T 2
3	there is no such case pred of process I is process J and pred of process J is process I	LIC-T 2
4	if process I is located at l3, process J is located at l6, next of process J is I	LIC-T 2

6.1.5 Graphical Animations of a Flawed Version of the MCS Protocol

This section aims to demonstrate that our approach can work well even the input is defective versions. In this section, we make the MCS protocol become a flawed version by making the function com&swap become non-atomic. The following is the correct rule in Maude:

```
rl [chglk] : (glock: Q) (pc[P]: l8) => (glock: (if Q == P
 then nop else Q fi))
 (pc[P]: (if Q == P then l9 else l10 fi)) .
```

We replace the correct rule by two following rules:

```
rl [chglk1] : (glock: Q) (pc[P]: l8) =>
 (pc[P]: (if Q == P then l8.5 else l10 fi)) (glock: Q) .
rl [chglk2] : (glock: Q) (pc[P]: l85) =>  (glock: nop) (pc[P]: l9) .
```

This version needs to add one more location `l85` and we need to change the state picture template by adding one more place for this location. Figure 6.11 shows the state picture template of this flawed version.

Observing graphical animations of this flawed version, some likely invariants are similar to the original version. For example, There is at most one process in cs, l7, l8, l85, l10, l11. However, we can obtain that whenever one process is located at l85, then another process is

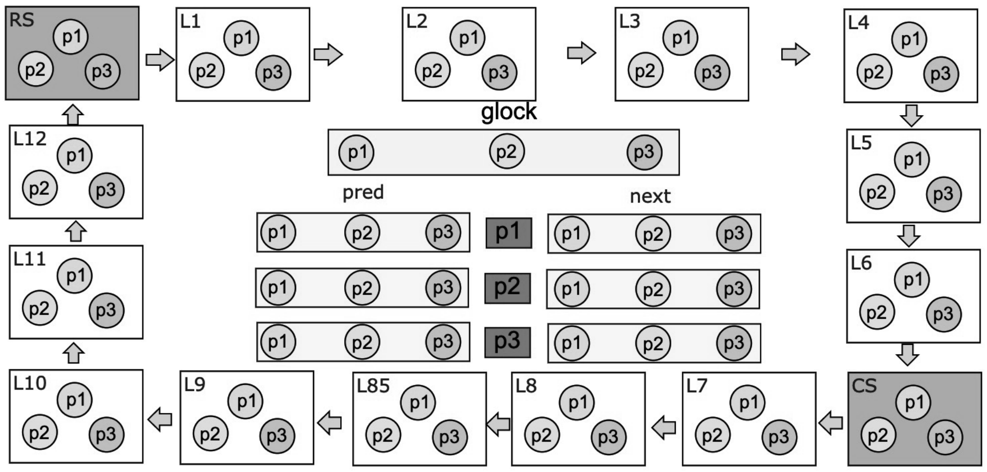


Figure 6.11: A state picture template of a flawed version of the MCS protocol

stuck at l6. It is because when a process I is located l85, another process J enters to the virtual queue so that process I cannot release process J because process I returns to rs instead of moving to l12. This characteristic is not invariant, but graphical animations help us to comprehend the reason. It implies that our approach is still help humans to understand the state machines in spite of defective versions.

6.2 Formal Verification of the Confirmed Likely Invariants of the MCS Protocol in CafeOBJ

6.2.1 Specification of the MCS Protocol in CafeOBJ

The specification of the MCS protocol in CafeOBJ can be made in the same way as what has been described in the Section 2.3. Let $S_{MCS} \triangleq \langle O_{MCS}, I_{MCS}, T_{MCS} \rangle$ be the state machine formalizing MCS.

O_{MCS} has the five observers that are declared as follows:

```
op glock : Sys -> Pid&Nop .
op next : Sys Pid -> Pid&Nop .
op prede : Sys Pid -> Pid&Nop .
op lock : Sys Pid -> Bool .
op pc : Sys Pid -> Label .
```

where `Sys` is the sort denoting $R_{S_{MCS}}$, `Pid` is the sort denoting process IDs, `Pid&Nop` is the sort denoting process IDs and nop and `Label` is the sort denoting the 14 labels.

I_{MCS} is specified as follows:

```
op init : -> Sys {constr} .
```

```
eq glock(init) = nop .
eq pc(init,P) = rs .
eq next(init,P) = nop .
eq prede(init,P) = nop .
eq lock(init,P) = false .
```

where `P` is a CafeOBJ variable of `Pid`.

T_{MCS} has the 14 transitions that are declared as follows:

```
op want : Sys Pid -> Sys {constr} .
op stnxt : Sys Pid -> Sys {constr} .
op stprd : Sys Pid -> Sys {constr} .
op chprd : Sys Pid -> Sys {constr} .
op stlck : Sys Pid -> Sys {constr} .
op stnpr : Sys Pid -> Sys {constr} .
op chlck : Sys Pid -> Sys {constr} .
op exit : Sys Pid -> Sys {constr} .
op chnxt : Sys Pid -> Sys {constr} .
op chglk : Sys Pid -> Sys {constr} .
op go2rs : Sys Pid -> Sys {constr} .
op chnxt2 : Sys Pid -> Sys {constr} .
op stlnx : Sys Pid -> Sys {constr} .
op go2rs2 : Sys Pid -> Sys {constr} .
```

The first (`want`) through sixth (`stnpr`) operators express state transitions such that a process (given as the second parameter) moves to l1, l2, l3, l4 or cs, l5 and l6 from rs, l1, l2, l3, l4 and l5, respectively, in a state (given as the first parameter). `chlck` expresses a state transition such that a process tries to move to cs from l6 in a state. The eighth (`exit`) through 11th (`go2rs`) operators express state transitions such that a process moves to l7, l8 or l11, l9 or l10 and rs from cs, l7, l8 and l9, respectively, in a state. `chnxt2` expresses a state transition such that a process tries to move to l11 from l10 in a state. `stlnx` and `go2rs2` express state transitions such that a process moves to l12 and rs from l11 and l12, respectively, in a state. Some more descriptions on the operators are given. For example, if a process p is at l2 in a state s, then `stprd(`s`,`p`)` denotes the state just after p has executed the statement at l2 and moved to l3 from l2; if p is at l6 in s, then `chlck(`s`,`p`)` denotes the state just after p has exited the loop at l6 and moved to cs if $lock_p$ is false and the state just after p has done one iteration of the loop at l6 if $lock_p$ is true.

The transitions are defined in terms of equations that specify how the values observed by the five observers change. For example, `stprd` is defined as follows:

```
ceq glock(stprd(S,P)) = P if pc(S,P) = l2 .
ceq pc(stprd(S,P),Q) = (if P = Q then l3 else pc(S,Q) fi) if pc(S,P) = l2 .
eq next(stprd(S,P),Q) = next(S,Q) .
eq lock(stprd(S,P),Q) = lock(S,Q) .
ceq prede(stprd(S,P),Q) = (if P = Q then glock(S)
                           else prede(S,Q) fi) if pc(S,P) = l2 .
ceq stprd(S,P) = S if (pc(S,P) = l2) = false .
```

where `S` is a CafeOBJ variable of `Sys` and `P` & `Q` are CafeOBJ variables of `Pid`. The remaining transitions can be defined likewise.

6.2.2 Formal Verification of the Confirmed Invariants of the MCS Protocol

In this section, we aim to use proof scores in CafeOBJ to prove all confirmed invariants in the previous section. We expected that all confirmed invariants will be proven, but some of them cannot be done with CafeOBJ. Let us introduce a situation that cannot be done and start with the true invariant as follows:

```
if a process is located at the extended cs region, glock is not nop.
```

It is straightforward to comprehend that a process in the extended cs region refers to the top of the virtual queue, then there exists an element in the virtual queue, and therefore *glock* is not nop. It is proven as inv4 shown in Table 6.5. However, the following likely invariant makes us stuck in a long time and cannot be done in CafeOBJ:

```
if a process is located at l3 and its pred is not nop, or l4, or l5,
or l6 and its lock is true, then glock is not nop.
```

Figure 6.12 shows state pictures that satisfy this likely invariant. We can obtain that, when a process p is located at l3 and $pred_p$ is not nop, or l4, or l5, or l6 and $lock_p$ is true, there exists another process in the extended cs region, therefore, *glock* is not nop. We can intuitively comprehend it, but it is not easy to formally prove it based on the current formal specification as it is. One such reason is that we need to identify a position of this process in the virtual queue. To comprehend the reason, let us rephrase the likely invariant as "if a process is located at l3 and its *pred* is not nop, or l4, or l5, or l6 and its *lock* is true, it refers to the virtual queue that contains at least two elements where this process is not the top of the virtual queue." The MCS protocol uses a virtual queue with *glock*, each process *next* and each process *pred*, where *glock* refers to the bottom element of the virtual queue if the queue is not empty. Therefore, when we concern the process in the likely invariant (make us stuck), we need to concern the number of processes in the virtual queue and the position of the process in the likely invariant (make us stuck) in such virtual queue. Currently, we cannot prove the likely invariant (make us stuck) in the original version of the MCS protocol by CafeOBJ. One possible solution is to use the other version that simulates the original version of the MCS protocol where a queue in simulated version simulates the virtual queue of the original version of the MCS protocol. We have done such simulation in [61] but not for proving the likely invariant (makes us stuck). We leave such work as a piece of our future directions.

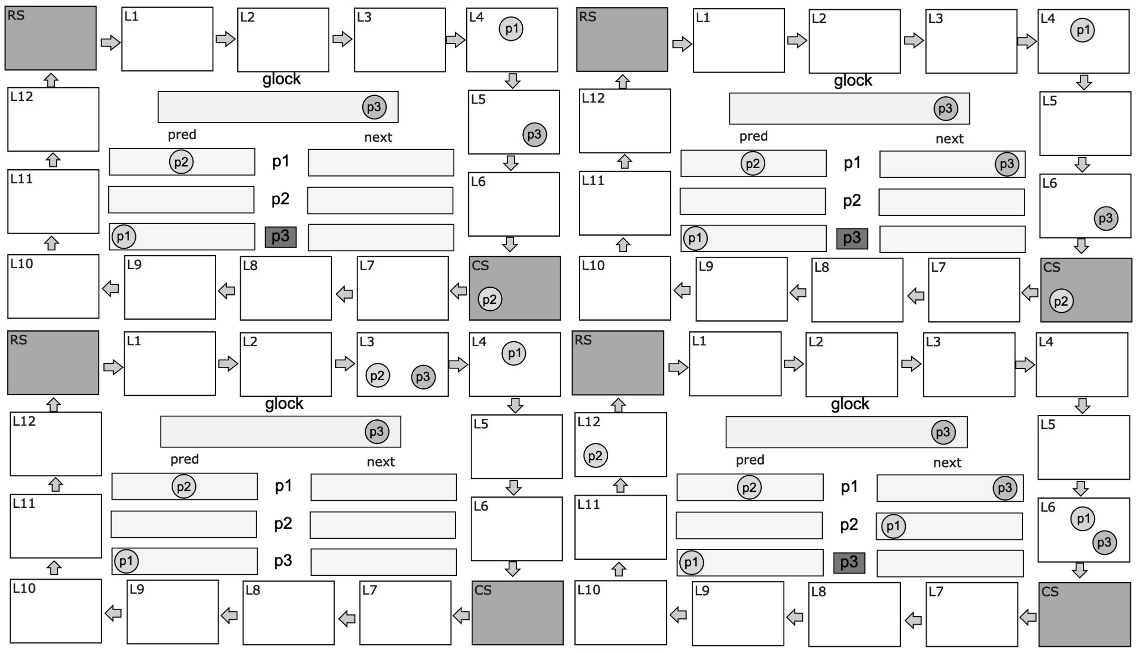


Figure 6.12: State pictures as examples of a likely invariant

Finally, Tables 6.5 and 6.6 show all properties (true invariants) of the MCS protocol. Note that the order of the properties is not same with the confirmed invariants. Table 6.4 shows information of a number of likely invariants and its relevance, such as numbers of such likely invariants that have counterexamples, are confirmed by model checking, and are proven by theorem proving.

6.3 Summary

This chapter shows the usefulness of both kinds of proposed tips, especially the proposed tips for designing state picture templates, for human users to find likely invariants of the MCS protocol. Most likely invariants found based on the tips survived with the search command in r-SMGA. We also report a case that a flawed version of the MCS protocol is conducted by our approach. This result shows that our approach still works for defective versions. Furthermore, most of the confirmed invariants are true invariants by interactive theorem proving. Some of the confirmed invariants cannot be proved even though we intuitively know that they are correct. It demonstrates that humans can understand the MCS protocol based on our approach.

Table 6.3: Confirmed invariants of the MCS protocol

No.	**Content**	**Tips**
1	there is at most one process in extended cs region	LIC-T 1
2	pred of process I is never process I	LIC-T 1
3	next of process I is never process I	LIC-T 1
4	if next of process I is process J, next of process J is not process I	LIC-T 1
5	if lock of process I is true, process I is located at l5 or l6	LIC-T 2
6	if process I is located at l5, its lock is true	LIC-T 2&4&3
7	if there is process is not in rs, l1, l2 and l9, then glock is not nop	LIC-T 2&3
8	if process I is in (l3 and its pred is not nop) or l4 or l5 or (l6 and its lock is true) and glock is process I, then there exists a process is in extended cs region	LIC-T 4&3&2
9	if process I is in rs, l1, l2, l9, l10, l11, l12, then glock is not process I	LIC-T 2&3
10	if a process is in l2 or l9, its next is nop	LIC-T 2&3
11	if a process is in l11 or l12, its next is not nop	LIC-T 2&3
12	if a process is in l4, l5, l6, its pred is not nop	LIC-T 2
13	if process is l3 and pred is nop, no other process is in extended cs region	LIC-T 4&3
14	if glock is a process I, its process I is nop	LIC-T 2&3
15	if pred of process I is process J and process J is located at rs, l1, l2, l9, l11 or l12, then process I is not located at l3, l4 or l5	LIC-T 4&3
16	if pred of process I is process J and glock is process J, then process I is not located at l3, l4 or l5	LIC-T 4&3
17	if pred of process I is process J, next of process J is not nop and process I is located at l3, l4, l5, l6, cs, l7, l8 or l10, then process J is not located at l3, l4 or l5	LIC-T 4&3

Table 6.4: A number of likely invariants and its relevance

No. of likely invariants	**A**	**B**	**C**	**D**
21	4	17	14	4

A: Numbers of likely invariants that have counterexamples

B: Numbers of likely invariants that are confirmed by invariant model checking

C: Numbers of likely invariants that are proven by theorem proving

D: Other invariants that are used as lemmas

Table 6.5: Proved properties of the MCS Protocol

Name	Lemma(s) used to prove	CafeOBJ syntax
inv1	inv2, inv3, inv4	eq inv1(S,P,Q) = ((pc(S,Q) = l3 and pred(S,Q) = nop and not(P = Q)) implies (not(pc(S,P) = cs or pc(S,P) = l7 or pc(S,P) = l8 or pc(S,P) = l10 or pc(S,P) = l11 or (pc(S,P) = l6 and lock(S,P) = false)))) .
inv2	inv4	eq inv2(S,P,Q) = ((pc(S,P) = l3 and pred(S,P) = nop and pc(S,Q) = l3 and (P = Q) = false) implies not(pred(S,Q) = nop))
inv3	inv5	eq inv3(S,P) = (pc(S,P) = l5 implies lock(S,P) = true) .
inv4	inv1, inv3, inv5	eq inv4(S,P) = (((pc(S,P) = l6 and lock(S,P) = false) or (pc(S,P) = l3 and pred(S,P) = nop) 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 not(glock(S) = nop)) .
inv5	inv3	eq inv5(S,P,Q) = ((not(pc(S,Q) = l12 or pc(S,Q) = l1 or pc(S,Q) = rs) and not(P = Q) and next(S,Q) = P) implies (pc(S,P) = l6 and lock(S,P) and pred(S,P) = Q)) .
inv6	inv1, inv3, inv7	eq inv6(S,P,Q) = ((pc(S,Q) = l6 and lock(S,Q) = false and (P = Q) = false) implies (not(pc(S,P) = cs or pc(S,P) = l7 or pc(S,P) = l8 or pc(S,P) = l10 or pc(S,P) = l11 or (pc(S,P) = l6 and lock(S,P) = false)))) .
inv7	inv1, inv3, inv6	eq inv7(S,P,Q) = (((pc(S,Q) = l11 or pc(S,Q) = l10 or pc(S,Q) = l8 or pc(S,Q) = l7 or pc(S,Q) = cs) and not(P = Q)) implies (not(pc(S,P) = cs or pc(S,P) = l7 or pc(S,P) = l8 or pc(S,P) = l10 or pc(S,P) = l11 or (pc(S,P) = l6 and lock(S,P) = false)))) .
inv8	inv3, inv9, inv11	eq inv8(S,P,Q) = ((next(S,Q) = P and (pc(S,Q) = l12 or pc(S,Q) = l1 or pc(S,Q) = rs) = false) implies (pc(S,P) = l6 and lock(S,P) = true and pred(S,P) = Q)) .
inv9	inv3, inv8, inv10, inv11	eq inv9(S,P,Q) = (not(pc(S,P) = l1 or pc(S,P) = l12 or pc(S,P) = rs) and (P = Q) = false and (next(S,Q) = nop) = false and not(pc(S,Q) = l1 or pc(S,Q) = l12 or pc(S,Q) = rs)) implies not(next(S,P) = next(S,Q)) .

Table 6.6: Proved properties of the MCS Protocol (cnt)

Name	**Lemma(s) used to prove**	**CafeOBJ syntax**
inv10	inv11	eq eq inv10(S,P) = ((next(S,P) = P) = false) .
inv11	inv12	eq inv11(S,P) = ((pred(S,P) = P) = false) .
inv12	inv13	eq inv12(S,P) = (pc(S,P) = l2 or pc(S,P) = l1 or pc(S,P) = rs or pc(S,P) = l9 or pc(S,P) = l12 or pc(S,P) = l11 or pc(S,P) = l10) implies (glock(S) = P) = false .
inv13	inv11, inv14, inv15	eq inv13(S,P) = ((glock(S) = P) implies (next(S,P) = nop)) .
inv14	inv13, inv15	eq inv14(S,P) = (pc(S,P) = l2 or pc(S,P) = l9) implies (next(S,P) = nop) .
inv15	inv11, inv12, inv13, inv15	eq inv15(S,P,Q) = (pred(S,Q) = P and not(P = Q) and (glock(S) = P or pc(S,P) = l2 or pc(S,P) = l1 or pc(S,P) = rs or pc(S,P) = l9 or pc(S,P) = l11 or pc(S,P) = l12 or ((pc(S,P) = l7 or pc(S,P) = l8 or pc(S,P) = cs or pc(S,P) = l6 or pc(S,P) = l10 or pc(S,P) = l3 or pc(S,P) = l5 or pc(S,P) = l4) and not(next(S,P) = nop)))) implies not(pc(S,Q) = l5 or pc(S,Q) = l4 or pc(S,Q) = l3) .
inv16	inv17	eq inv16(S,P,Q) = ((pc(S,Q) = l4 or pc(S,Q) = l5 or pc(S,Q) = l3) and not(P = Q) and (pc(S,P) = l5 or pc(S,P) = l4 or pc(S,P) = l3) and not(pred(S,P) = nop) and not(pred(S,Q) = nop)) implies not(pred(S,Q) = pred(S,P)) .
inv17	inv12, inv15	eq inv17(S,P) = ((pc(S,P) = l3 or pc(S,P) = l4 or pc(S,P) = l5) and not(pred(S,P) = nop)) implies not(glock(S) = pred(S,P)) .
inv18	none	eq inv18(S,P) = lock(S,P) implies (pc(S,P) = l5 or pc(S,P) = l6) .
inv19	none	eq inv19(S,P) = (pc(S,P) = l11 and pc(S,P) = l12) implies not(next(S,P) = nop) .
inv20	none	eq inv20(S,P) = (pc(S,P) = l4 or pc(S,P) = l5 or pc(S,P) = l6) implies not(pred(S,P) = nop) .

Chapter 7

Case Study: the Suzuki-Kasami Protocol

This chapter uses the Suzuki-Kasami (SK) protocol as an example to demonstrate the usefulness our approach, especially our proposed guidelines and new features in r-SMGA. This chapter is mainly divided the content into two parts: graphical animation of the SK protocol and formal verification of the confirmed invariants of the SK protocol in CafeOBJ. The former mainly shows the usefulness of new features with the proposed guidelines while the latter mainly shows one advantage of our proposed approach that can find lemmas when conducting theorem proving with proof scores in CafeOBJ. Firstly, this chapter introduces the SK protocol and its specification in Maude. Then, the chapter graphically animates the SK protocol where a state picture template is designed based on the proposed tips (for designing state picture template) with new features of r-SMGA. Some examples of how to find likely invariants using the guidelines and new features of r-SMGA are described in details. The found likely invariants are confirmed with the **Search command** in r-SMGA. This chapter also reports a case that a flawed SK protocol created by our intention is used for our approach. Finally, when conducting theorem proving for all confirmed invariants, we find that our proposed approach can be used to find likely invariants that can be used as lemmas.

7.1 Graphical Animations of the SK protocol

This section introduces a case study where the Suzuki-Kasami (SK) protocol is used as an example. The section first explain how the protocol works is specified in Maude. Then, the input of r-SMGA is prepared including the specification and the state picture template of the SK protocol. Based on some tips of [11], observing graphical animations, and using new and revised features of r-SMGA, some likely invariants are conjectured and confirmed to show the usefulness of those features.

```
try(i)      <-->  rem   procedure P1
setReq(i)   <-->  l1      requesting := true;
chkPrv(i)   <-->  l2      if ¬have_privilege then
incRN(i)    <-->  l3        rn[i] := rn[i] + 1;
                            for all j ∈ {1, …, N} − {i} do
sndReq(i)   <-->  l4          send request(i, rn[i]) to node j;
                            endfor
                            wait until privilege(queue, ln) is received;
wtPrv(i)    <-->  l5        have_privilege := true;
                          endif
exit(i)     <-->  cs      Critical Section;
cmpReq(i)   <-->  l6      ln[i] := rn[i];
                          for all j ∈ {1, …, N} − {i} do
                            if (j ∉ queue) ∧ (rn[j] = ln[j] + 1) then
updQ(i)     <-->  l7          queue := enq(queue, j);
                            endif
                          endfor
chkQ(i)     <-->  l8      if queue ≠ empty then
                            have_privilege := false;
trsPrv(i)   <-->  l9        send privilege(deq(queue), ln) to node top(queue);
                          endif
                          requesting := false;
rstReq(i)   <-->  l10   endproc

                        // request(j, n) is received; P2 is indivisible.
                        procedure P2
                          rn[j] := max(rn[j], n);
                          if have_privilege ∧ ¬requesting ∧ (rn[j] = ln[j] + 1)
recReq(i)   <-->          then have_privilege := false;
                                send privilege(queue, ln) to node j;
                          endif
                        endproc
```

Figure 7.1: Algol-like description of the Suzuki-Kasami protocol

7.1.1 Description

We first introduce the Suzuki-Kasami protocol and then describe how to specify it in Maude. The Suzuki-Kasami distributed mutual exclusion protocol (also known as the SK protocol) was proposed by Suzuki and Kasami [16]. In the protocol, if a node owns a privilege, then the node can enter its critical section. The privilege can be transferred to other nodes in the network. In the protocol, there are N node participants and $1, \ldots, N$ are used for their IDs. ***Node*** is defined as the set $\{1, \ldots, N\}$ of all node IDs. Each node can communicate with each other by exchanging messages in the network. In the protocol, there are two kinds of messages named request and privilege. A request message is in the form of request(j, n), where j is the ID of a node that sends the message and n is a natural number that identifies the request number. A privilege message is in the form of privilege(q, a), where q is a queue of node IDs and a is an array of natural numbers whose size is N.

In the protocol, there are two procedures P1 and P2 used for each node $i \in \boldsymbol{Node}$ and the two procedures are described in Figure 7.1. *requesting* and *have_privilege* are two Boolean variables. *requesting* is true if node i wants to enter the critical section; otherwise it is false. *have_privilege* is true if node i owns the privilege; otherwise it is false. *queue* is a queue of IDs of nodes that are requesting to enter the critical section. *ln* and *rn* are arrays of natural numbers whose size is N. $ln[j]$ for each node $j \in \boldsymbol{Node}$ is the number of node j's request granted most recently. *rn* records the largest request number received from each of the other nodes. For each node i, its *rn* is always meaningful, while its *queue* and *ln* are meaningful only when node i owns the privilege. For each node $i \in \boldsymbol{Node}$, initially, *requesting* is false, *have_privilege* is true if $i = 1$, otherwise it is false, *queue* is empty, and each element of *ln* and *rn* is 0.

Procedure P1 is used for node i if it wants to enter its critical section. First, the node sets *requesting* to true. If node i owns the privilege, it moves to the critical section. Otherwise, it increments $rn[i]$ and transfers the request message request$(i, rn[i])$ to all other nodes. Then, node i waits to receive the privilege and sets *have_privilege* to true if it receives the privilege. It moves to the critical section after that. Once node i leaves the critical section, it updates $ln[i]$ by $rn[i]$. Then, *queue* is updated by checking if each node j waits to enter its critical section ($rn[j] = ln[j] + 1$) and j is not in queue ($j \notin queue$) and putting such j into queue. After that, if *queue* is empty, node i sets requesting to false and leaves P1, keeping the privilege. Otherwise, *have_privilege* is set to false and node i transfers the privilege message privilege(deq$(queue), ln$) to the node that is the top of the queue.

Whenever the request message request(j, n) is transferred to node i, node i runs procedure P2. However, procedure P2 must be atomically executed. First, $rn[j]$ is updated if it is greater than n. Then, node i checks *have_privilege*, *requesting* and $rn[j]$, and if they satisfy the conditions described in P2, then, node i sets its *have_privilege* to false and sends the privilege message privilege$(queue, ln)$ to node j.

7.1.2 Specification of the Protocol in Maude

We formalize the Suzuki-Kasami protocol as a state machine in Maude. `Nat`, `Bool`, `Loc`, `Queue`, and `Array` are the sorts for natural numbers, Boolean values, locations (e.g., l1 and cs), queues of node IDs, and natural number arrays of size N, respectively. A message is in the form of msg$(i, body)$, where i is the receiver node and *body* is the message body that is either a request or a privilege. A request is in the form of req(j, k), where j is a node ID and k is a request number, while a privilege is in the form of priv(q, a), where q is a queue of node IDs and a is a natural number array of size N. The network is formalized as a soup of messages.

Let `Message` be a sort for soups of messages and `void` denote the empty soup of messages. The observable components used to formalize the SK protocol are classified into two groups:

(1) those storing values independent from each node, such as the `locked` observable component used to formalize TAS and (2) those storing values dependent on each node, such as the `pc` observable component. There are three observable components in group (1), while there are seven observable components in group (2) for each process. The observable components in group (1) are as follows:

- (`nw`: *ms*) says that the network is ms, a soup of messages. Initially, *ms* is empty.
- (`queue`: q) says that q is the meaningful queue. Initially, q is empty.
- (`ln`: a) says that a is the meaningful `ln`. Initially, a is the natural number array of size N such that each element is 0.

The observable components in group (2) for each node i are as follows:

- (`pc[`i`]`: l) says that node i is located at location l. Initially, l is rem.
- (`have_privilege[`i`]`: b) says that node i has the privilege when b is true and does not otherwise. Initially, b is true if $i = 1$ and false otherwise.
- (`requesting[`i`]`: b) says that node i wants to enter its critical section if b is true and does not otherwise. Initially, b is false.
- (`queue[`i`]` : q) says that the node i's *queue* is q. Initially, q is empty.
- (`rn[`i`]`: a) says that the node i's *rn* is a. Initially, a is the natural number array of size N such that each element is 0.
- (`ln[`i`]`: a) says that the node i's *ln* is a. Initially, a is the natural number array of size N such that each element is 0.
- (`idx[`i`]`: j) says that the node i's loop variable is j. Initially, j is 1.

To specify the SK protocol in Maude, we first divide the protocol into 13 regions as shown in Figure 7.1. The name of each region is put on the left side, such as try(i) and exit(i). We suppose that each node is located at one of 12 regions in P1. One region is expressed as one transition that is written as one rewrite rule in Maude. Thus, there are 13 rewrite rules in the formal specification of the protocol in Maude. For example, the rewrite rule (whose label is `updateQueue`) that corresponds to region updQ(i) is as follows:

```
rl [updateQueue] :
    (pc[I]: l7) (idx[I]: K) (rn[I]: RN) (ln[I]: LN)
    (queue[I]: Q) (queue: Q)
    => (pc[I]: if K == N then l8 else l7 fi)
       (idx[I]: if K == N then 1 else K + 1 fi)
```

```
(rn[I]: RN) (ln[I]: LN)
(queue[I]: if K =/= I and not(K \in Q)
                and (RN[K] == (LN[K]) + 1)
           then put(Q,K) else Q fi)
(queue: if K =/= I and not(K \in Q)
                 and (RN[K] == (LN[K]) + 1)
           then put(Q,K) else Q fi) .
```

If the node I's loop variable K equals N, then node I moves to l8 from l7, exiting the corresponding loop in the pseudo-code, and K is set to 1. Otherwise, node I stays at l7 and K is incremented to handle the next iteration of the loop. If K does not equal I, K is not in Q (the node I's queue) and the latest node K's request has not been yet granted (RN[K] == LN[K] + 1), then K is put into Q. Because the node I's queue is meaningful, the queue observable component stores the node I's queue and the one stored in the observable component is updated likewise. The other rules work similarly.

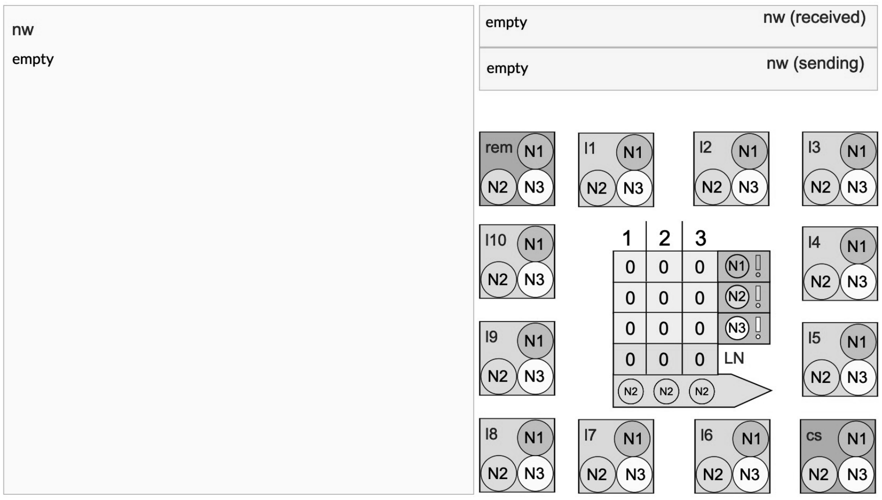


Figure 7.2: A state picture template of the SK protocol.

7.1.3 Conjecturing Likely Invariants of the SK Protocol

Designing the State Picture Template of the SK Protocol

Based on the observable components mentioned above and some proposed tips in the previous chapter, we design the state picture of the SK protocol shown in Figure 7.2.

We borrow some visualization techniques from [13] and redesign some observable components such as have_privilege, requesting, queue, rn, and ln. Based on tips for designing the state picture template in Table 4.1 and Table 4.2, such as observable components should be visual as much as possible, in the new state picture template, we redesign have_privilege

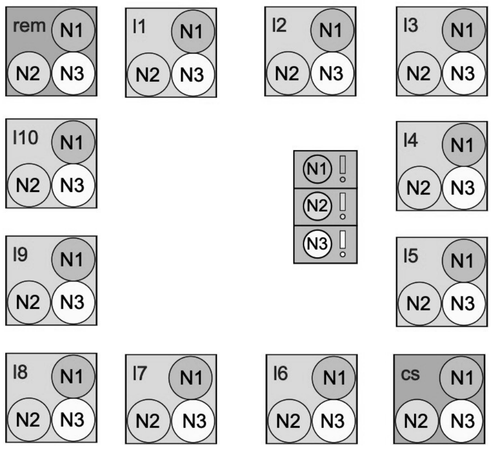
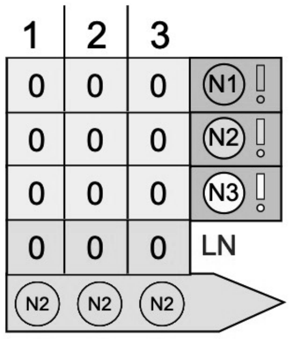
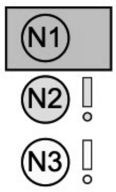


Figure 7.3: A state picture template of three observable components: pc[i], privilege[i], and requesting[i] for node i, where i = 1, 2, 3.

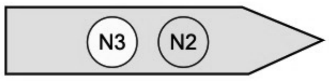
and `requesting` using the visual display. We use the *Special display* feature to visualize `queue`, `rn`, and `ln`. For other observable components, please refer to the work [13]. Based on SPT-T 9 in Table 4.2, the following figure shows the revised design for five observable components:



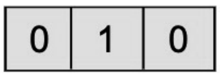
where three blue rectangles and exclamation marks on the right-hand side are `have_privilege[`i`]` and `requesting[`i`]` observable components of the three nodes, respectively where $i = 1, 2, 3$. We will describe later how exactly the blue rectangles express the `have_privilege[`i`]` observable components and the exclamation marks express the `requesting[`i`]` observable components. The three circles with different colors inside the blue rectangles represent the labels of the three nodes. Let us look at the nine light-pink rectangles. From top to down, three light-pink rectangles aligned horizontally with three numbers inside represent the `rn` observable component of each of nodes 1, 2 and 3. The numbers 1, 2, and 3 on the top of the figure represent indices of arrays starting from 1. The three light-orange rectangles aligned horizontally represent the `ln` observable component in which the meaningful array `ln` is stored, where the text "LN" appears right next to the three light-orange rectangles. The light-orange fat right-arrow represents the `queue` observable component in which the meaningful queue is stored. The following figure shows a case in which the `have_privilege[1]`, `have_privilege[2]`, and `have_privilege[3]` observable components are true, false and false, respectively; and the `requesting[1]`, `requesting[2]`, and `requesting[3]` observable components are false, true and true, respectively.



In the previous version, a queue was used with textual display only. r-SMGA makes it possible to show a queue with the visual display. Human users are supposed to design what visual objects as elements appear at each position in a visual object that represents a (bounded) queue. Because there are three nodes, it suffices that the queue stored in the `queue` observable component has at most three elements (three node IDs). Thus, there are three positions in the queue and then each of three node IDs can be located at each position in the queue. We use a circle on which a node ID is written as a visual object as a queue element. When the queue is empty, nothing appear at all positions of the queue, while the queue is not full, nothing appears at some positions of the queue. For example, the queue expressed as `2 | 3 | empty`, where there are two elements, 2 is the top and 3 is the second (and the last), is visualized as follows:



In the previous version, an array was shown with textual display only. In the formal specification of the SK protocol, an array is expressed as a soup of index-value pairs. For example, the natural number array `a` of size 3 such that `a[1] = 0`, `a[2] = 1`, and `a[3] = 0` is expressed as `(1 : 0), (2 : 1), (3 : 0)`. In the previous version, `a` is shown as the text `(1 : 0), (2 : 1), (3 : 0)` only. In the current version, `a` is visualized as follows:



The figure is the visual object for the `ln` observable component and the three `rn[`i`]` observable components, where $i = 1, 2, 3$, can be visualized likewise. Table 7.1 shows observable components of the SK protocol and which tips used to design the observable components.

Conjecturing Likely Invariants of the SK Protocol

This section introduces some likely invariants that are conjectured using the proposed tips. Using **LIC-T 1** and **LIC-T 2**, the *Interaction* feature helps us to focus on two or more observable components without being distracted by the other observable components. For example, we focus on the `pc[`i`]`, `have_privilege[`i`]`, and `requesting[`i`]` observable components for

Table 7.1: Observable components and their tips used to design

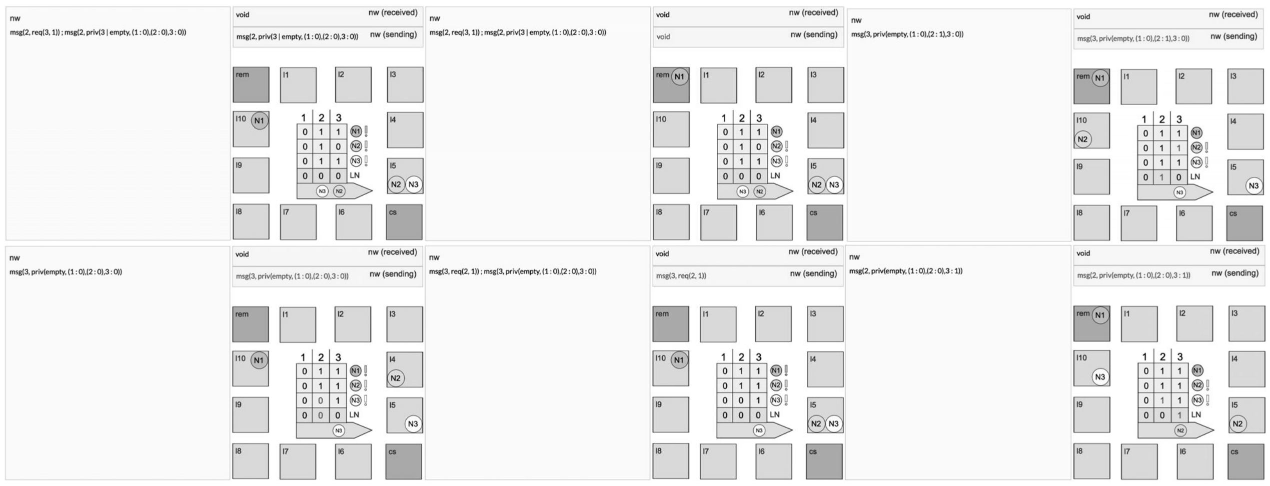
Observable components	**Tips**
pc[i]	SPT-T 1&2
have_privilege[i]	SPT-T 4&6&9
requesting[i]	SPT-T 4&6&9
rn[i]	SPT-T 4&6&9
ln	SPT-T 9
queue	SPT-T 9
nw	SPT-T 3

node $i = 1, 2, 3$ by using the **focus** function as shown in Figure 7.3. Based on **CC-T1**, we use the **hide** function to focus on one of the three kinds of observable components. By observing graphical animations, we conjecture some likely invariants as follows:

SK-LI 1: There is at most one node that is located at `cs`, `l6`, `l7`, `l8`, or `l9`.

SK-LI 2.1: There exists a case such that three nodes do not own the privilege.

SK-LI 2.2: If a node owns the privilege, there is no other nodes that owns the privilege.



```
nw
msg(2, req(3, 1)) ; msg(2, priv(3 | empty, (1 : 0),(2 : 0),3 : 0))
void                                              nw (received)
msg(2, priv(3 | empty, (1 : 0),(2 : 0),3 : 0))    nw (sending)

nw
msg(2, req(3, 1)) ; msg(2, priv(3 | empty, (1 : 0),(2 : 0),3 : 0))
void                                              nw (received)
void                                              nw (sending)

nw
msg(3, priv(empty, (1 : 0),(2 : 1),3 : 0))
void                                              nw (received)
msg(3, priv(empty, (1 : 0),(2 : 1),3 : 0))        nw (sending)

nw
msg(3, priv(empty, (1 : 0),(2 : 0),3 : 0))
void                                              nw (received)
msg(3, priv(empty, (1 : 0),(2 : 0),3 : 0))        nw (sending)

nw
msg(3, req(2, 1)) ; msg(3, priv(empty, (1 : 0),(2 : 0),3 : 0))
void                                              nw (received)
msg(3, req(2, 1))                                 nw (sending)

nw
msg(2, priv(empty, (1 : 0),(2 : 0),3 : 1))
void                                              nw (received)
msg(2, priv(empty, (1 : 0),(2 : 0),3 : 1))        nw (sending)
```

Figure 7.4: Some state pictures found with *Pattern matching* feature

Based on **LIC-T 2**, we focus on two of the three observable components shown in the Figure 7.3. By observing graphical animations, some likely invariants are conjectured as follows:

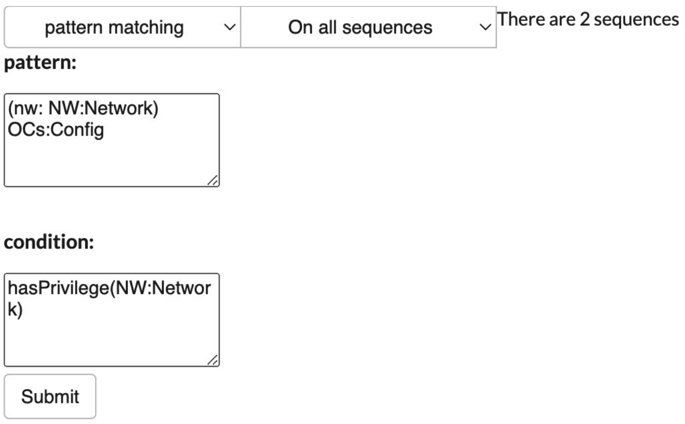
SK-LI 3.1: If a node is located at `cs`, `l6`, `l7`, `l8`, or `l9`, then the node owns the privilege.

SK-LI 3.2: If a node is located at `l3`, `l4`, or `l5`, the node does not own the privilege.

SK-LI 4.1: If `requesting` of a node is false, the node is located at `rem` or `l1`.

SK-LI 4.2: if a node is located at `rem` or `l1`, `requesting` of the node is false.

The *Pattern matching* feature is used for **LIC-T 3**. We use this feature to find states in which there exists a privilege message in the network. Note that, we cannot do it with the previous version because of limitation of regular expression. The following figure shows a command that is used for this case.



pattern matching | On all sequences | There are 2 sequences

pattern:

```
(nw: NW:Network)
OCs:Config
```

condition:

```
hasPrivilege(NW:Networ
k)
```

where the top of the figure shows information of the feature, such as the **Pattern matching on all sequences** function and a number of sequences on the current list. The pattern `(nw: NW:Network) OCs:Config` used is written in the rectangle just below "pattern:" and the condition `hasPrivilege(NW:Network)` used is written in the rectangle just below "condition:". The pattern is used to find states in which there exists an observable component that matches `(nw: NW:Network)`, where `NW:Network` is a Maude variable of sort `Network` declared on-the-fly. Because every state has such an observable component, all states are candidates to be found. The purpose of use of the pattern is to extract the contents, a soup of messages, stored in the observable component. The condition checks whether such a soup of messages has a privilege message. Because a soup of messages is an associative-commutative collection, we need to rely on the associative-commutative pattern matching of Maude, which is one unique feature empowered by Maude. Using **LIC-T 2** and observing the animations of such states shown in Figure 7.4, we conjecture some likely invariants as follows:

SK-LI 5: There is only one privilege message in the network.

SK-LI 6.1: If there is a privilege message in the network, no node owns the privilege.

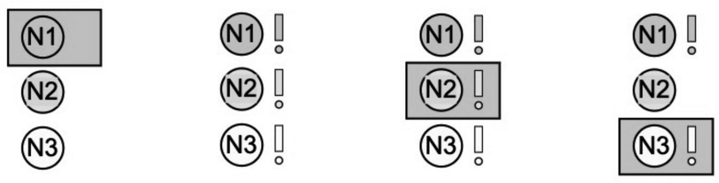
SK-LI 6.2: If a node owns the privilege, there is no privilege message in the network.

SK-LI 7: If there is a privilege message in the network, no node is located at `cs`, `l6`, `l7`, `l8`, and `l9`.

Note that we can conjecture SK-LIs 5-7 by observing states satisfying SK-LI 2.1 (we will explain the details of how to conjecture those in Section 7.1.4). SK-LI 7 can be guessed by SK-LIs 6.1 and 3.1 based on **LIC- 4**.

7.1.4 Finding Likely Invariants Using Guidelines

Let us start with a situation where we have conjectured some likely invariants of the SK protocol in Section 7.1.3 using guidelines. Given the state picture template and the specification of the SK protocol, r-SMGA then produces graphical animations. Based on **LIC-T 1**, observing the animations and focusing on the visual representations of the `privilege[`i`]` observable components for $i = 1, 2, 3$, we obtain four cases as follows:



Based on the second picture from left in the figure above, which indicates that no node owns the privilege, we conjecture SK-LI 2.1. Then, we use the *Pattern matching* feature to find states that match SK-LI 2.1 and some results are shown in Figure 7.4. Observing the animations made from the results and focusing on the network, we observe that there is a privilege message in the network in each state picture of the animations. Therefore, we can guess that if no node owns the privilege, there exists a privilege message in the network. We confirm it with the **Search command** function. Let us suppose that the confirmed invariant is denoted as $A \Rightarrow B$, where A and B are "no node owns the privilege" and "there exists a privilege message in the network," respectively; $\Rightarrow$ is the logical implication. Once again, we use the *Pattern matching* feature to find states that match B and the results obtained are the same as those shown in Figure 7.4. Then, we can conjecture SK-LIs 5–7. Repeating the process, we can find other likely invariants of the protocol. Note that we can find a new likely invariant by the transitive property of implication, namely that if $P \Rightarrow R$ and $R \Rightarrow Q$, then $P \Rightarrow Q$. For example, P, R, and Q are "there exists a node i located at cs, l6, l7, l8 or l9," "node i owns the privilege," and "no privilege message in the network," respectively; $P \Rightarrow R$ and $R \Rightarrow Q$ are SK-LIs 3.1 and 6.2, respectively. Then, we can get a new likely invariant $P \Rightarrow Q$ stating that "if there exists a node located at cs, l6, l7, l8 or l9, then no privilege message in the network." Note also that we can create a complex likely invariant by combining confirmed likely invariants together. For example, if $M \Rightarrow T$ and $N \Rightarrow T$, then $(M$ or $N) \Rightarrow T$. Those concrete likely invariants conjectured this way are in Section 7.1.3.

Let us use how to find `inv8` (that is extremely crucial and mentioned in Section 7.2.2) as an example to exemplify the guidelines. `inv8(`s`,`i`)` is as follows: if a node i is located at neither `rem` nor `l1` in a state s, then the value stored in `requesting[`i`]` is `true` in state s.

First let us focus on the `requesting[`i`]` and `pc[`i`]` observable components for $i = 1, 2, 3$. We carefully observe graphical animations, especially, the relation between the values stored in `requesting[`i`]` and `pc[`i`]`. We notice and guess that the following is likely to

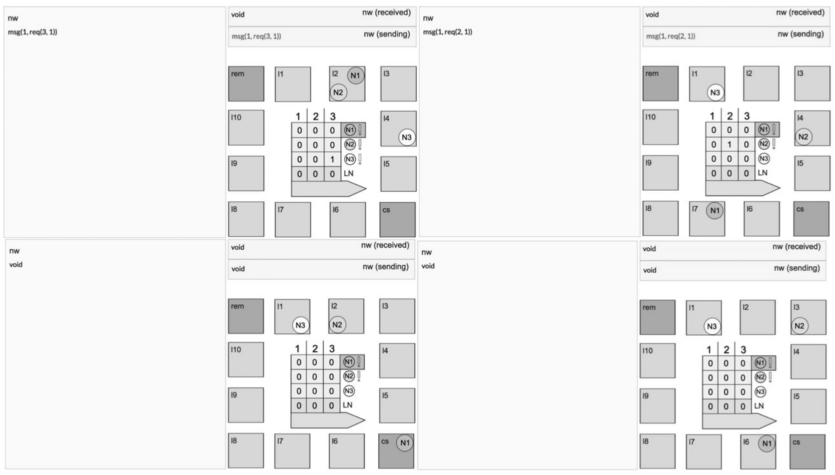


Figure 7.5: Some state pictures found with the condition “requesting[1] is true”.

be `true`: if the value stored in `requesting[`*i*`]` is `true`, then the value stored in `pc[`*i*`]` is neither `rem` nor `l1`. This is an example conducted at **Step 1**.

The condition that the value stored in `requesting[`*i*`]` is `true` can be extracted from the guessed likely invariant. We use the *Pattern matching* feature to generates states that satisfy the condition. See Figure 7.5 for some of the states generated. The figure shows some states that satisfy the condition “`requesting[1]` is `true`.” This is an example conducted at **Step 2**.

We then check whether the value stored in `pc[`*i*`]` is neither `rem` nor `l1` by observing the graphical animations made from the states found and/or the still pictures of the states, from which we can confirm the guessed likely invariant. This is an example conducted at **Step 3**.

We can also use “the value stored in `pc[`*i*`]` is neither `rem` nor `l1`” as a condition and generates states that satisfy the condition with the *Pattern matching* feature. This is another example conducted at **Step 2**. We carefully observe the graphical animations made from the states found and notice that the value stored in `requesting[`*i*`]` is `true` in each of the states, from which we guess that if the value stored in `pc[`*i*`]` is neither `rem` nor `l1`, then the value stored in `requesting[`*i*`]` is `true`. This is another example conducted at **Step 3**. The examples conducted at **Step 2** and **Step 3** is an example conducted at **Step 4** as well.

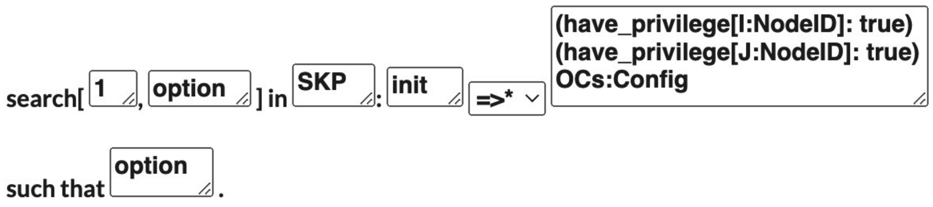
This is how we guess `inv8` based on the guidelines.

7.1.5 Confirmation of Guessed Likely Invairants Using Maude Features

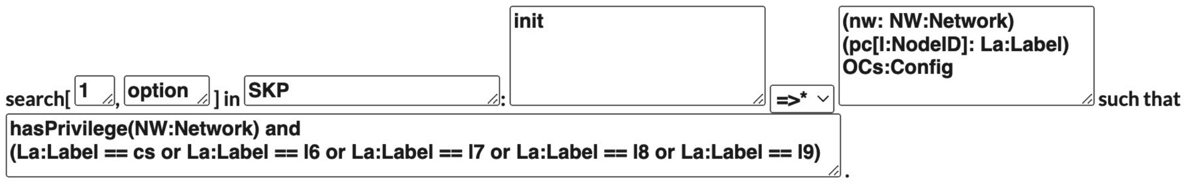
In the previous version, users can confirm guessed likely invariants by using the Maude search command independently. In r-SMGA, users can confirm the guessed likely invariants by using the **Search command** function in r-SMGA. For example, the command to confirm SK-LI 2.1 is as follows:

```
search [1] in SKP : init =>* (have_privilege[I:NodeID]: true)
(have_privilege[J:NodeID]: true) OCs:Config .
```

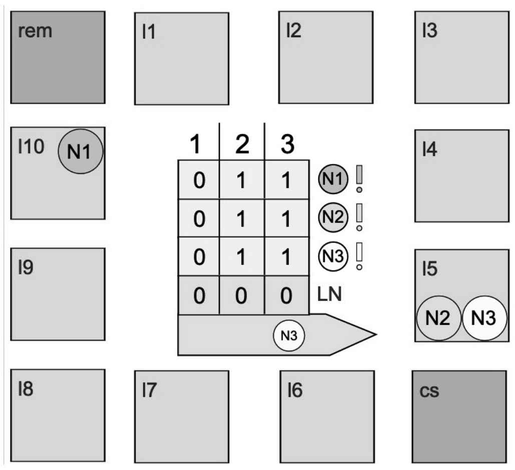
where `SKP` is a module, `init` is an initial state, `I` and `J` are Maude variables declared on-the-fly of sort `NodeID`, and `OCs` is a Maude variable declared on-the-fly of sort `Config`. The command can be used in **Search command** of r-SMGA as follows:


```
search[ 1 , option ] in SKP : init =>* (have_privilege[I:NodeID]: true)
                                      (have_privilege[J:NodeID]: true)
                                      OCs:Config
such that option .
```

The function tries to find a state such that both two different nodes `I` and `J` own the privilege. The function does not return any counterexample; hence, the guessed likely invariant is confirmed. The following figure shows the command for confirming SK-LI 7:


```
search[ 1 , option ] in SKP : init =>* (nw: NW:Network)
                                      (pc[I:NodeID]: La:Label)
                                      OCs:Config
such that hasPrivilege(NW:Network) and
(La:Label == cs or La:Label == l6 or La:Label == l7 or La:Label == l8 or La:Label == l9) .
```

where `hasPrivilege(_)` is an operation in specification to check whether the network contains a privilege message. The command does not return any counterexamples and then SK-LI 7 has been confirmed. Note that users need to check guessed likely invariants because they may not be correct. For example, observing the animations and focusing on locations of nodes. We obtain the following figure:



The figure makes us conjecture that there at most one process in cs, l6, l7, l8, l9, and l10. The counterexample that leads to l10 is pointed out by the **Search command**.

Table 7.2 and Table 7.3 show all likely invariants that are survived by the **Search command** and all false invariants that are found by the **Search command**, respectively. The **Step in guidelines** column refers to steps used to find likely conjecture.

7.1.6 Graphical Animations of a Flawed Version of the SK Protocol

The main purpose of our proposed approach is to help humans to understand state machines via likely invariants of the state machines. This section graphically animates a flawed version of the SK protocol to show that our approach still guarantees the goal in spite of defective versions. In this flawed version of the SK protocol, we do not update *have_privilege* of node i of the transition trsPrv(i) in Figure 7.1. The flawed version does not change much than the original version so that we still keep the current state picture template. Observing graphical animations, we immediately that there exists a case that two or three nodes can own the privilege at the same time. The following is the case that three nodes can own the privilege.

Table 7.2: Confirmed invariants of the SK protocol.

No.	Content	Step in guidelines
1	there is at most one privilege own by a node at the moment	Step 1 (LIC-T 1)
2	there is at most one privilege message in the network	Step 1 (LIC-T 1)
3	there is at most one node in cs, l6, l7, l8, l9	Step 1 (LIC-T 1)
4	if a node is in rem or l1, its requesting is false	Step 1 (LIC-T 2)
5	if requesting of a node is false, its location is rem or l1	Step 2&3 (LIC-T 2&3&4)
6	if requesting of a node is true, its location is not located at rem or l1	Step 2&3 (LIC-T 2&3&4)
7	if a node is not located at rem and l1, its requesting is true	Step 2&3 (LIC-T 2&3&4)
8	if there is the privilege message in the network, no node owns privilege	Step 2&3 (LIC-T 3&4)
9	if a node is located at cs, l6, l7, l8, l9, no privilege message in the network	Step 2&3 (LIC-T 3&4)
10	if a node owns a privilege, there is no privilege message in the network	Step 2&3 (LIC-T 3&4)
11	if a node is located at cs, l6, l7, l8, l9, no another node owns the privilege	Step 2&3 (LIC-T 3&4)
12	if no node owns a privilege, there is a privilege message in the network	Step 2&3 (LIC-T 3&4)

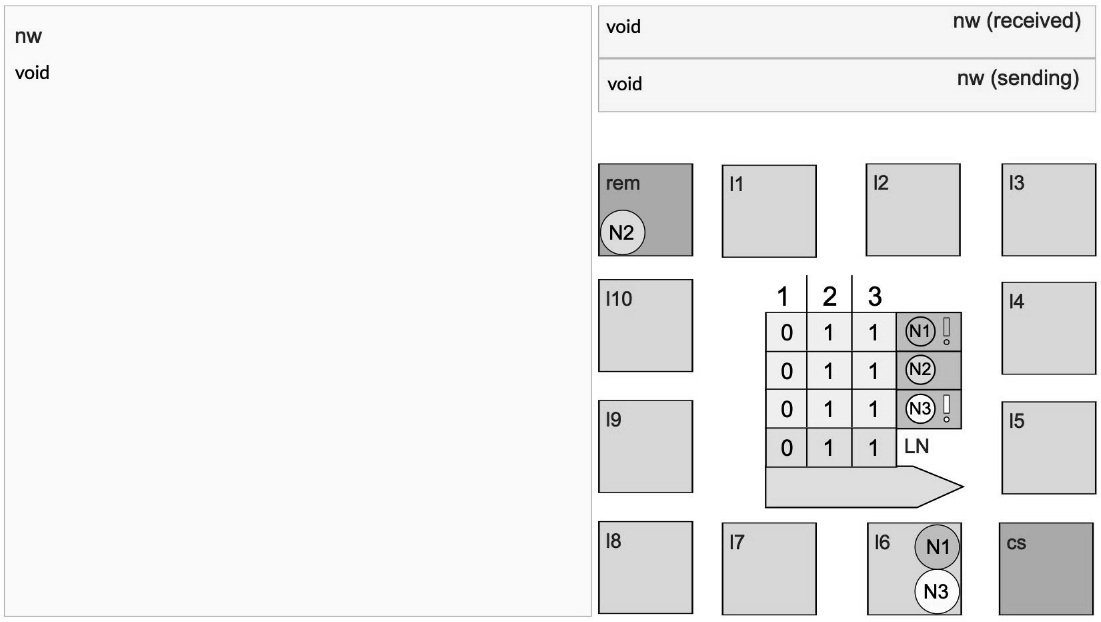


Table 7.3: False invariants of the SK protocol.

No.	**Content**	**Step in guidelines**
1	there is at most one process in cs, l6, l7, l8, l9, l10	Step 1 (LIC-T 1)
2	if a node does not own a privilege, its location is l3 or l4 or l5	Step 1 (LIC-T 2&4)
3	if a node is located at l3 or l4 or l5, there exists a node is located at cs, l6, l7, l8, l9, l10	Step 1 (LIC-T 2&4)
4	if a node is located at l5, it is in the queue	Step 2&3 (LIC-T 2&3&4)
5	if a node is in the queue, it is located at l5	Step 2&3 (LIC-T 2&3&4)

Moreover, some likely invariants are also found, such as there are two or more privilege messages in the network or there are two or more nodes in the cs. It implies that our approach still works even the input is defective versons.

7.2 Formal Verification of the Confirmed Invariants of the SK Protocol in CafeOBJ

7.2.1 Formal Specification of the Suzuki-Kasami Protocol in CafeOBJ

We first briefly present the formal specification of the protocol in CafeOBJ. Sorts `Sys`, `Network`, `Queue`, `Array`, and `Label` are introduced to represent the state space, the network, queues, arrays, and locations at which nodes are located, respectively. The meaning of `Sys` has been explained in Section 2.3 while the meaning of the remaining sorts is the same as the one described in Section 7.1. The observers used in the CafeOBJ formal specification are declared as follows:

```
op nw         : Sys       -> Network .
op pc         : Sys NzNat -> Label .
op havePriv   : Sys NzNat -> Bool .
op requesting : Sys NzNat -> Bool .
op queue      : Sys NzNat -> Queue .
op rn         : Sys NzNat -> Array .
op ln         : Sys NzNat -> Array .
op idx        : Sys NzNat -> NzNat .
```

where `NzNat` is the sort of non-zero natural numbers, representing node IDs.

The constant `init` is introduced to represent an arbitrary initial state. Let `I` is a CafeOBJ variable of sort `NzNat`, `init` is defined in terms of equations as follows:

```
op init : -> Sys {constr}

eq nw(init) = void .
eq pc(init,I) = rem .
eq havePriv(init,I) = (I = 1) .
eq requesting(init,I) = false .
eq queue(init,I) = empty .
eq rn(init,I) = ia .
eq ln(init,I) = ia .
eq idx(init,I) = 1 .
```

where `void`, `empty`, and `ia` are constants denoting the empty network, the empty queue, and the (initial) array such that each content is 0, respectively.

We specify 13 transitions, where each of them is defined in terms of equations that specify how the values observed by the eight observers change. For example, updQ(i) in Figure 7.1 is defined as follows:

```
op updateQueue : Sys NzNat -> Sys {constr} .
eq requesting(updateQueue(S,I),J) = requesting(S,J) .
eq havePriv(updateQueue(S,I),J) = havePriv(S,J) .
ceq queue(updateQueue(S,I),J) =
     if I = J then
      -- Check some conditions based on
      -- the algorithm, then update queue
      if not(idx(S,I) \in queue(S,I)) and not(idx(S,I) = I)
        and rn(S,I)[idx(S,I)] = s(ln(S,I)[idx(S,I)])
      then put(queue(S,I),idx(S,I))
      else queue(S,J) fi
     else queue(S,J) fi
     if c-updateQueue(S,I) .
eq ln(updateQueue(S,I),J) = ln(S,J) .
eq rn(updateQueue(S,I),J) = rn(S,J) .
ceq idx(updateQueue(S,I),J) =
     -- Update loop variable (index) by increment operator
     if I = J then s(idx(S,I)) else idx(S,J) fi
     if c-updateQueue(S,I) .
ceq pc(updateQueue(S,I),J) =
     -- Update location of node I if its index is over N
     if I = J then if idx(S,I) = N then l8 else l7 fi
     else pc(S,J) fi
     if c-updateQueue(S,I) .
eq nw(updateQueue(S,I)) = nw(S) .
```

```
ceq updateQueue(S,I) = S if not c-updateQueue(S,I) .
```

where `S` and `J` are CafeOBJ variables of sorts `Sys` and `NzNat`, respectively. `idx(S,I)` represents the value of the loop variable of node `I` in state `S`, `_\in_` is an operator defined to check whether the first parameter is in the second parameter, `s(_)` is the successor function of natural numbers, and `c-updateQueue(S,I)` is `pc(S,I) = l7`. The rest of the transitions can be defined likewise.

7.2.2 Formal Verification of the Confirmed Invariants of the SK Protocol

Similarly to the TAS case study presented in Section 2.3, we specify the confirmed invariants from Section 7.1.3 and prove them by writing proof scores in CafeOBJ.

Similarly to what has been presented in Section 2.3, we also use simultaneous structural induction on variable `S` of sort `Sys` to conduct the formal verification. There are one base case and 13 induction cases (from try(i) to recReq(i) in Figure 7.1). As also described in Section 2.3, we use case splitting to make each of CafeOBJ fragments return either `true` or `false`. For each case in which CafeOBJ returns `false`, we need to use lemmas to discharge the case. There are four factors used to construct lemmas: equations (assumptions) in the fragment of a case that CafeOBJ returns false, output from CafeOBJ, confirmed invariants, and our approach with features of r-SMGA. When conducting theorem proving all confirmed invariants, most lemmas are constructed from the confirmed invariants in Section 7.1.3. Let us consider a fragment (or a sub-case) of the induction case waitPriv(i) in which `false` is returned as follows:

```
open INV .
   op s : -> Sys .
   ops i j p : -> NzNat .
   op pri : -> Privilege .

   eq pc(s,p) = l5 .
   eq (i = p) = false .
   eq j = p .
   eq (msg(p,pri) \in nw(s)) = true .
   eq pc(s,i) = l9 .

   red inv7(s,i,j) implies inv7(waitPriv(s,p,pri),i,j) .
close
```

where `pc(s,p) = l5`, . . . , `pc(s,i) = l9` are our assumptions used to characterize the sub-case (or the fragment). To discharge this fragment, we use `inv2` defined as follows:

```
op inv2 : Sys NzNat NzNat Privilege -> Bool
```

```
eq inv2(S,I,J,Pri) = msg(I,Pri) \in nw(S)
                        implies (not(pc(S,J) = cs or pc(S,J) = l9 or pc(S,J) = l8
                        or pc(S,J) = l7 or pc(S,J) = l6)) .
```

inv2 is constructed by SK-LI 7. We also use some parts of combination of the confirmed invariants. Let us consider the fragment that returns false as follows:

```
open INV .
   op s : -> Sys .
   ops i j p : -> NzNat .

   eq pc(s,p) = l1 .
   eq i = p .
   eq (j = p) = false .
   eq havePriv(s,p) = true .
   eq pc(s,j) = l6 .

   red inv7(s,i,j) implies inv5(s,j,p) implies inv7(setReq(s,p),i,j) .
close
```

where `pc(s,p) = l1`, ..., `pc(s,j) = l6` are our assumptions used to characterize this fragment. To discharge this fragment, we use `inv5` defined as follows:

```
op inv5 : Sys NzNat NzNat -> Bool
eq inv5(S,I,J) = ((pc(S,I) = l9 or pc(S,I) = l8 or pc(S,I) = l7 or pc(S,I) = l6)
                            and havePriv(S,J)) implies (I = J) .
```

`inv5` is constructed based on SK-LIs 2.2 and 3.1.

There are two lemmas `inv6` and `inv8` that are constructed in different ways. `inv6` is defined as follows:

```
op inv6 : Sys NzNat NzNat Privilege Privilege -> Bool
eq inv6(S,I,J,Pri,Pri1) = msg(I,Pri) \in nw(S)
          implies not (msg(J,Pri1) \in del(nw(S),msg(I,Pri))) .
```

where `del(_,_)` is the operator defined to delete a specific message (the second parameter) in the network (the first parameter). `inv6` says that if there exists a privilege message M from the network, there is no more privilege messages in the network just after deleting the privilege message M. From `inv6`, we can derive that there always exists at most one privilege message in the network. The main reason to be able to construct the lemma is to discharge a sub-case as follows:

```
open INV .
   op s : -> Sys .
   ops i j p : -> NzNat .
   ops pri pri1 : -> Privilege .

   eq pc(s,p) = l5 .
   eq (msg(p,pri1) \in nw(s)) = true .
   eq (j = p) = false .
   eq (msg(i,pri) \in nw(s)) = false .
   eq (pc(s,j) = cs) = false .
   eq pc(s,j) = l9 .
red inv2(s,i,j,pri) implies inv2(waitPriv(s,p,pri1),i,j,pri) .
close
```

In the sub-case, CafeOBJ returns a term as follows:

```
true xor (msg(i,pri) \in del(nw(s),msg(p,pri1)))
```

We can use `inv6` so that CafeOBJ can return `true` for the sub-case. Note that this lemma is inspired from SK-LI 5.

Lastly, `inv8` is defined as follows:

```
op inv8 : Sys NzNat -> Bool
eq inv8(S,I) = not(pc(S,I) = rem or pc(S,I) = l1)
            implies (requesting(S,I) = true) .
```

The meaning of the lemma is similar to some confirmed invariants, such as SK-LIs 4.1 and 4.2 and the meaning of this lemma is as follows: when a node is not located at either `rem` or `l1`, its `requesting` is true. However, it took time for us to find that the lemma is crucial. When we use the lemma, we can discharge the most annoying sub-cases we have encountered. We construct this lemma by using some features of r-SMGA. Let us consider a sub-case when proving the induction case recReq(i) as follows:

```
open INV .
   op s : -> Sys .
   ops i j p : -> NzNat .
   op pri : -> Privilege .
   op re : -> Request .

   eq (msg(p,re) \in nw(s)) = true .
   eq (node(re) = p) = false .
   eq (i = p) = false . eq j = p .
```

Table 7.4: A number of likely invariants and its relevance

No. of likely invariants	**A**	**B**	**C**	**D**
17	5	12	12	3

A: Numbers of likely invariants that have counterexamples
B: Numbers of likely invariants that are confirmed by invariant model checking
C: Numbers of likely invariants that are proven by theorem proving
D: Other invariants that are used as lemmas

```
   eq msg(i,pri) \in nw(s) = false
   eq pc(s,p) = cs .
   eq requesting(s,p) = false .
   ...
red inv2(s,i,j,pri) implies inv2(receiveReq(s,p,re),i,j,pri) .
close
```

where `node(_)` is defined to get the node from the request message denoted `re`. "`...`" are assumptions that we do not list at all. First, in this sub-case, we make an attempt by using a simplification of the confirmed invariants based on all assumptions (similarly to what we do with most lemmas). The assumptions make us construct some likely invariants that are hard or complicated to verify. After we have analyzed the situation, we found that `requesting(s,p)` is the core source of the situation. If `requesting(s,p)` is true, we can discharge the case; otherwise, it makes the case become complicated. We have used *Pattern matching* feature of r-SMGA to search for states that satisfy the condition "`requesting` is true" and we have finally constructed `inv8` by observing graphical animations of such states (the details of how to find this lemma are described in Section 7.1.4). To this end, we summarize all properties of the SK protocol proved and their lemmas used in Table 7.5 and 7.6. Note that we do not keep the same order with the confirmed invariants. Table 7.4 shows information of a number of likely invariants and its relevance, such as numbers of such likely invariants that have counterexamples, are confirmed by model checking, and are proven by theorem proving.

All proof scores of the SK protocol are available at: `https://gitlab.com/duydang12/skp-cafeobj-cafeinmaude`.

7.3 Summary

In this chapter, we have conducted a case study where the SK protocol is used as an example. In this case study, we mainly show the usefulness of our proposed approach, especially using proposed guidelines with new features in r-SMGA, where some parts cannot be done by the

Table 7.5: Proved properties of the SK Protocol

Name	**Lemma(s) used to prove**	**CafeOBJ syntax**
inv1	inv2, inv3, inv4	eq inv1(S,I,J) = ((pc(S,I) = cs or (pc(S,I) = l2 and havePriv(S,I))) and havePriv(S,J) implies (I = J)) .
inv2	inv1, inv4, inv6, inv7, inv5, inv8	eq inv2(S,I,J,Pri) = msg(I,Pri) \in nw(S) implies (not(pc(S,J) = cs or pc(S,J) = l9 or pc(S,J) = l8 or pc(S,J) = l7 or pc(S,J) = l6)) .
inv3	inv4	eq inv3(S,I,J) = havePriv(S,I) and havePriv(S,J) implies (I = J) .
inv4	inv3, inv6	eq inv4(S,I,J,Pri) = msg(I,Pri) \in nw(S) implies (not(havePriv(S,J))) .
inv5	inv1, inv2	eq inv5(S,I,J) = ((pc(S,I) = l9 or pc(S,I) = l8 or pc(S,I) = l7 or pc(S,I) = l6) and havePriv(S,J)) implies (I = J) .
inv6	inv2, inv4	eq inv6(S,I,J,Pri,Pri1) = msg(I,Pri) \in nw(S) implies not (msg(J,Pri1) \in del(nw(S),msg(I,Pri))) .
inv7	inv1, inv2, inv3, inv4, inv5	eq inv7(S,I,J) = ((pc(S,I) = cs or pc(S,I) = l6 or pc(S,I) = l7 or pc(S,I) = l8 or pc(S,I) = l9 or (pc(S,I) = l2 and havePriv(S,I))) and not (I = J)) implies not(pc(S,J) = cs or pc(S,J) = l6 or pc(S,J) = l7 or pc(S,J) = l8 or pc(S,J) = l9) .
inv8	no lemma	eq inv8(S,I) = not(pc(S,I) = rem or pc(S,I) = l1) implies (requesting(S,I) = true) .
inv9	inv2, inv7	eq inv9(S,I,J) = (pc(S,I) = cs or pc(S,I) = l6 or pc(S,I) = l7 or pc(S,I) = l8 or pc(S,I) = l9) and (pc(S,J) = cs or pc(S,J) = l6 or pc(S,J) = l7 or pc(S,J) = l8 or pc(S,J) = l9) implies I = J .
inv10	no lemma	eq inv10(S,I) = pc(S,I) = rem or pc(S,I) = l1 implies requesting(S,I) = false .
inv11	no lemma	eq inv11(S,I) = (requesting(S,I) = false) implies (pc(S,I) = rem or pc(S,I) = l1) .
inv12	no lemma	eq inv12(S,I) = pc(S,I) = l3 or pc(S,I) = l4 or pc(S,I) = l5 implies havePriv(S,I) = false .
inv13	no lemma	eq inv13(S,I) = (requesting(S,I) = true) implies not(pc(S,I) = rem or pc(S,I) = l1) .

Table 7.6: Proved properties of the SK Protocol (cnt)

Name	Lemma(s) used to prove	CafeOBJ syntax
inv14	inv3, inv4	eq inv14(S,I,J) = havePriv(S,I) and (J \in queue(S,I)) and not(I = J) implies havePriv(S,J) = false .
inv15	inv3, inv6	eq inv15(S,I,J,Pri) = msg(I,Pri) \in nw(S) and J \in q(Pri) implies havePriv(S,J) = false .
inv16	inv7, inv8	eq inv16(S,I) = pc(S,I) = cs or pc(S,I) = l6 or pc(S,I) = l7 or pc(S,I) = l8 or pc(S,I) = l9 implies havePriv(S,I) .
inv17	inv2, inv3, inv4	eq inv17(S,I,J) = (pc(S,I) = cs or pc(S,I) = l6 or pc(S,I) = l7 or pc(S,I) = l8 or pc(S,I) = l9) and not(I = J) implies not(havePriv(S,J)) .
inv18	inv3, inv4	eq inv14(S,I,J) = havePriv(S,I) and (J \in queue(S,I)) and not(I = J) implies havePriv(S,J) = false .
inv19	inv1, inv5, inv6, inv8, inv9, inv18	eq inv19(S,I,J,Pri) = (pc(S,I) = cs or pc(S,I) = l6 or pc(S,I) = l7 or pc(S,I) = l8 or pc(S,I) = l9) implies not(msg(J,Pri) \in nw(S)) .
inv20	inv2, inv4, inv6	eq inv20(S,I,J,Pri,Pri1) = msg(I,Pri) \ in nw(S) and msg(J,Pri1) \ in nw(S) implies (I = J and Pri = Pri1) .

original version (e.g. searching a specific message in the network). Several likely invariants are found based on our proposed approach and features of r-SMGA. Some invariants that cannot survive with the search command, show one limitation of our approach, but it can be avoided by confirming with the search command. We also have conducted a flawed version of the SK protocol to show that our approach could be applied to defective versions. The results of this chapter show that our approach can help humans to find true invariants of the SK protocol where all confirmed invariants (survived with the search command) are proven as true invariants. One promising result is that our approach can find new likely invariants used as lemmas when conducting interactive theorem proving.

Chapter 8

Evaluation

8.1 Analysis of the Case Studies

Based on the results from case studies, this section analyzes the results with each of proposed tips (for designing state picture templates and conjecturing likely invariants), accordingly.

For the proposed tips for designing state picture templates, Table 8.1 shows all tips and their number of uses when conducting case studies in the dissertation. The first column refers to the names of the tips while the second and the third column refer to the number of the uses of the tips in the MCS protocol and the SK protocol, respectively. In the table, most proposed tips are shared for both protocols because both protocols are mutual exclusion protocols. There are some of the tips that are never used for both protocols but used for other case studies shown in the Table 8.2. Table 8.2 shows the tips and the number of case studies for which the tips are used. In the table, **SPT-T 5**, **7**, and **10** are mainly proposed for autonomous vehicles protocols and then not used for the two protocols in the dissertation. **SPT-T 6** is the tip that is used most frequently because it can be applied for any protocols where they have at least one variable that can have either of two values, such as Boolean. For us, this tip is also crucial, especially in finding likely invariants (used with the tips for conjecturing likely invariants), because (1) this tip is a form of Gestalt principles that helps us to recognize two groups (two kinds of values, such as Boolean) via animations and the design from this tip (note that two kinds of values, such as Boolean are the most fundamental value and used most frequently in systems/protocols), (2) based on the design from the content, this tip also reduces the complexity of state picture templates (numbers of visual objects) when the number of observable components (satisfies this tip) increases, and (3) it is inspired to propose **SPT-T 8** where observable components can have either of three or more than three values. Note that each tip is proposed based on the orientation mentioned in Chapter 4, where Gestalt principles are a main part in this orientation. Table 8.2 also shows that all proposed tips are used to design state picture templates.

For the proposed tips for conjecturing likely invariants, Table 8.3 shows all tips and their number of uses when conducting case studies in the dissertation. The first column refers to the names of the tips while the second and the third column refer to the number of the uses of the tips in the MCS protocol and the SK protocol, respectively. The table says that all tips are used because the proposed tips are built to help humans to find likely invariants including simple likely invariants (consist of one or two variables/observable components) using **LIC-T 1** & **LIC-T 2** (by observing animations from state sequences only) and complex likely invariants (consist of two or more variables/observable components) using **LIC-T 3** & **LIC-T 4** (by utilizing pattern matching feature in r-SMGA). All tips are also used in the other case studies as shown in Table 8.4. For all case studies, **LIC-T 3** is used most frequently because most likely invariants that consist of two or more observable components are found by this tip. In particular, when using **LIC-T 2** (focus on two observable components), it is not enough to conjecture likely invariants that have relations of two observable components, such as $A \Rightarrow B$ or $B \Rightarrow A$, where A & B and $\Rightarrow$ are two observable components and the implication relation, respectively, therefore, it is necessary to use **LIC-T 3** beside **LIC-T 2**. For us, **LIC-T 3** is crucial because (1) it is a main part to find complex likely invariants as described in Section 4.4, (2) its content helps humans to find likely invariants that have a form of implication, and (3) this tip also helps us to recognize counterexamples in (2). When we use **LIC-T 1**, most likely invariants are related to the mutual exclusion property. For the SK protocol, we most frequently use features of r-SMGA with **LIC-T 3** because the features work well for some observable components of the SK protocol, such as the network (an AC collection of messages). Note that the features in r-SMGA make the tips more effective as shown in Chapter 7.

After theorem proving the confirmed invariants (likely invariants survived with invariant model checking), most confirmed invariants become properties (true invariants). For the SK protocol, all confirmed invariants are properties. When conducting theorem proving the confirmed invariants of the SK protocol, lemmas are constructed by the confirmed invariants and the tips with the features of r-SMGA as described in Chapter 7.2. For the MCS protocol, even though we cannot prove all confirmed invariants by CafeOBJ, we are likely to be able to prove the remaining confirmed invariants that have not yet been proved by adding auxiliary variables to the MCS protocol as described in [61]. It shows that when likely invariants found by our approach are survived with invariant model checking, they are likely to be properties of protocols/systems under consideration.

8.2 Answer to the Research Question

As mentioned in Chapter 1, we can answer **RQ** by answering **RQ1** and **RQ2**. Contents from Chapter 4 and Chapter 5 answer two sub-questions while the results from Chapter 6 and Chapter 7 show the usefulness of our proposals applied to case studies. Let us summarize the answers and our contributions.

RQ1: How to design state picture templates based on Gestalt principles?
(A1): This study proposes the tips based on Gestalt principles to help humans to design state picture templates. All tips based on the orientation where Gestalt principles are a main part are as follows:

> Values of observable components should be visual as much as possible and be arranged based on Gestalt principles, such as common region, proximity, and similarity laws.

When humans use the tips, they can easily recognize which observable components have just changed based on Gestalt principles, and then can conjecture/find likely invariants based on such changes of observable components. The results in Section 8.1 show the usefulness of the tips via non-trivial case studies where Tables 8.1 and 8.2 are some results from case studies and have been analyzed in Section 8.1. Note that those tips are proposed based on the Gestalt principles that have been established in the psychology field.

RQ2: How to conjecture/find likely invariants based on graphical animations?
(A2): Based on the state picture templates designed by our proposed tips, this study also proposes tips for conjecturing/finding likely invariants of state machines. The tips help humans to focus on values of observable components in the Gestalt principles-based state picture template and conjecture/find likely invariants based on such values of observable components, especially likely invariants with a form of implication. Tables 8.3 and 8.4 are some results from case studies that have been analyzed to show the usefulness of those tips when applying them to case studies. The results of the case studies also show that when likely invariants can survive by invariant model checking, they are most likely to be properties of either of the two mutual exclusion protocols. Therefore, Gestalt principles can help humans to understand state machines via knowing properties of state machines based on our proposals.

8.3 Limitations

Besides, there are some limitations of our proposed approach and some threats that may affect to our results. In this section, let us discuss some factors involving to our proposals.

Humans

Humans are a main factor that cannot be ignored in our study because we mainly rely on the visual perception of humans, especially Gestalt principles about grouping. In our proposed approach, three tasks require humans:

- Designing a state picture template: as mentioned, it is a non-trivial task [11] and there is no perfect solution for all protocols in general, therefore, our proposals currently mainly depend on Gestalt principles established in psychology, which is relied on the visual perception of humans about grouping.

Table 8.1: Tips for designing state picture templates and number of their uses in the MCS protocol and the SK protocol

Name	**the MCS protocol**	**the SK protocol**
SPT-T 1	1	1
SPT-T 2	1	1
SPT-T 3	1	1
SPT-T 4	3	3
SPT-T 5	0	0
SPT-T 6	3	3
SPT-T 7	0	0
SPT-T 8	3	0
SPT-T 9	3	5
SPT-T 10	0	0

- Conjecturing/finding likely invariants via the state picture template: this task mainly helps humans to conjecture/find likely invariants based on state sequences. Finding likely invariants based on state sequences is similar to most state-of-the-art approaches mentioned in Section 3. Currently, about proposed tips, we do not limit kinds of characteristics of likely invariants that humans should focus on so that it totally depends on human observation. Such kinds of characteristics will be analyzed and such task will be a piece of our future work.

- Verifying guessed likely invariants: this task takes much efforts and time because (1) we have manually prepared proof scores and (2) finding lemmas for fragments that return false is a still a non trival task. Invariant Proof Score Generator (IPSG) [64] is a tool that can automatically generate proof scores and point out sub-cases or fragments that return false. (1) can be done by using IPGS in the future while (2) is still not

Assessment Methods

Currently, conducting case studies and analyzing results is a current way to to evaluate our proposals. Conducting usability evaluation involving humans is one possible way however, this approach that has been pointed out its limitations and difficulties in Chapter 3. For our knowledge, there is no standard way to evaluate our proposals. Finding assessment methods is one piece of our future work.

Table 8.2: Tips for designing state picture templates and case studies use them

Name	**Number of case studies**
SPT-T 1	5 (SKP [13, 58], MCS [11], Anderson [11], LJPL [12], AR [56])
SPT-T 2	5 (SKP [13, 58], MCS [11], Anderson [11], LJPL [12], AR [56])
SPT-T 3	3 (SKP [13, 58], MCS [11], Anderson [11])
SPT-T 4	5 (SKP [13, 58], MCS [11], Anderson [11])
SPT-T 5	1 (LJPL [12])
SPT-T 6	6 (SKP [13, 58], MCS [11], Anderson [11], LJPL [12], AR [56], NSLPK [57, 63])
SPT-T 7	2 (LJPL [12], AR [56])
SPT-T 8	1 (MCS [11])
SPT-T 9	6 (SKP [13, 58], MCS [11], Anderson [11], LJPL [12], AR [56], NSLPK [57, 63])
SPT-T 10	2 (LJPL [12], AR [56])

Table 8.3: Tips for conjecturing likely invariants and number of their uses in the MCS protocol and the SK protocol

Name	**the MCS protocol**	**the SK protocol**
LIC-T 1	5	5
LIC-T 2	8	12
LIC-T 3	10	11
LIC-T 4	12	6

Table 8.4: Tips conjecturing likely invariants and case studies use them

Name	**Number of case studies**
LIC-T 1	6 (SKP [58], MCS [11], Anderson [11], LJPL [12], AR [56], NSLPK [57])
LIC-T 2	6 (SKP [58], MCS [11], Anderson [11], LJPL [12], AR [56], NSLPK [57])
LIC-T 3	6 (SKP [58], MCS [11], Anderson [11], LJPL [12], AR [56], NSLPK [57])
LIC-T 4	6 (SKP [58], MCS [11], Anderson [11], LJPL [12], AR [56], NSLPK [57])

Chapter 9

Conclusion and Future Work

9.1 Conclusion

To answer **RQ**, this dissertation has proposed an approach based on visual information to help humans to find likely invariants of state machines. There are three contributions to assist the proposed approach: (1) giving some practical tips for designing state picture templates and conjecturing likely invariants, (2) providing new features to SMGA, by developing r-SMGA by integrating Maude with SMGA, to make those tips more effective, and (3) conducting multiple case studies using our proposals. The previous chapter has answered **RQ** by answering **RQ1** and **RQ2**, where **RQ1** and **RQ2** have been answered based on the the results of the case studies.

Design state picture templates

This contribution has shown that the state picture template is extremely crucial in our approach. After conducting many case studies, we have summarized our lessons learned as practical tips and provided them for users to design the state picture template. The tips are mainly built and evaluated based on Gestalt principles, especially the common region, proximity and similarity laws. Moreover, we also have provided some more tips for users to conjecture likely invariants. From those tips, we have built guidelines as a generic way for human users to find likely invariants more systematically.

Integration of SMGA and Maude

In this contribution, to make the proposed tips more effective, some features are provided to assist humans to conjecture/find likely invariants. We have integrated SMGA and Maude so that the revised version (r-SMGA) can use some powerful features of Maude to assist to find likely invariants based on the proposed guidelines. By using Maude, the pattern matching feature of SMGA has been revised so that context-free grammars, instead of regular expressions, can be used as patterns, and associative-commutative binary operators can also be used in the patterns. Some more interactive features have been provided to help users to concentrate on some visual objects in which users are interested. Special display features are also provided to display some data structures, such as array and queue.

Conducting the case studies

Two case studies where the MCS protocol and the SK protocol are used as two non-trivial examples have been conducted to demonstrate the usefulness of our approach. Firstly, for each protocol, we produce graphical animations based on state picture template designed by our proposed tips. Using guidelines and features in r-SMGA, several likely invariants are found and confirmed with invariant model checking (the search command). The confirmed invariants are also proven by interactive theorem proving. When conducting theorem proving, some likely invariants are constructed as lemmas by our proposed approach. We have shown that our approach also works with flawed or defective protocols.

9.2 Future Work

As usual, there are many things left we need to do in the future. There are two directions we aim to do: (i) keeping revising the current version by supporting some more other visualization for human users to have various options to visualize observable components, and (ii) finding some factors affecting animations and/or state picture templates to help human users in designing state picture templates systematically. Last but not least, we need to conduct more classes of protocols, such as security protocols to demonstrate the usefulness of our proposed approach.

Support More Visualization Techniques for Observable Components

Tree graphs are a common kind of diagram to visualize pieces of information that link together. Hernando et al. [65] have proposed a novel method using a tree graph to visualize huge information from related documents, such as news. Keywords or sentences are nodes where each node can be seen as raw texts or related images. There is one main node that is displayed as a picture containing related texts and images. The other nodes are displayed as raw texts or displayed in the same way as the main node such that the main node is displayed larger than the others. Users can observe nodes and navigate the graph to understand the relations between such nodes. When users navigate the graph, the main node is updated to let users mainly focus on such node. ABETS [66] is a prototype for checking the correctness of Maude programs. The main purpose of the work is to improve the diagnosis of erroneous Maude programs. It uses tree graphs to visualize state sequences when nodes and edges correspond to states and rules, respectively. If an error occurs, the tool generates a tree graph that contains states which lead to the error. To understand the error, users can observe the paths and click on states to expand the information of such states displayed as raw texts. One direction of our future work is to combine both approaches above to r-SMGA where state sequences can be used in the way as proposed in [66], be displayed in the way as proposed in [65], and be graphically animated by our approach when users select one concrete state sequence.

Frank et al. [67] have proposed a method to visualize state transition systems. They aim to let users observe the global properties of protocols by visualizing state spaces. They use cone, the tree concept [68] to form state transition structures in three dimensions. The main algorithm of the method focuses on the symmetry property and aims to let users identify the symmetrical and similar sub-structures in the tree. First, they rank all nodes to make the systems become hierarchical system structures [69]. Then, they cluster such nodes following some local properties to reduce the visual complexity of the tree. Based on the clusters, they aim to visualize the state spaces as a backbone tree in which clusters are visualized as circles whose sizes are decided by their volumes. Then, users can observe and interact with the tree by focusing and zooming into some clusters to be able to analyze paths inside. The method is extended to deal with a large state space [70]. The results of both versions allow users to observe state spaces of protocols visualized as a backbone tree with the cone tree concept for each node. Then users can find some global properties of the protocols, such as obtaining some clusters that do not return to initial nodes after starting some executions. This method and r-SMGA share the idea to help users find properties or invariants of protocols. r-SMGA graphically animates state sequences (paths) while this method visualizes whole state spaces. The idea of the method motivates us to extend r-SMGA so that r-SMGA can give users an overview of state spaces. Then users can select some paths and graphically animate them by our approach. One piece of our future work is to extend r-SMGA based on the mentioned ideas.

Factors affecting animations and/or state picture templates

There are many attributes of animations that can help humans to recognize some relations of visual objects. We investigate two attributes: subitizing and the law of common fate of the Gestalt principle. In the psychology field, subitizing is a term to introduce the ability of humans to enumerate a small number of items rapidly and accurately. The terminology was first used in [71] to distinguish it from counting that is time-consuming and error-prone, where the subitizing range is fewer than four items and the counting range is more than four items [72]. Many studies have shown that some factors can expand the subitizing range, such as grouping [73], bilateral and two-item advantage [74]; and the original subitizing range (1-3) is still the fastest and the most accurate in the experiments. The law of common fate (LCF) [75] is one of the Gestalt principles and it is the only one that can deal with dynamic (i.e. animations) properties instead of static properties [76]. This law states that visual elements that move with the same velocity (i.e. same speed and same direction) are perceived as the same group. The work [77] has conducted two empirical experiments to demonstrate that LCF is not restricted to mere motion and some dynamic visual properties (such as luminance and size) have been affected also. In the second experiment, participants are required to observe some animated scatterplots representing the change of data and answer some questions related to group, such as “what patterns do you see?” and “which variable(s) create the most visible patterns?” The result of the second experiment makes the authors claim that the power of dynamic visual variables might shift when applied to more realistic visualization scenarios, such as pattern identification and conjunction search.

When a protocol/system, such as one of all case studies we have conducted, is graphically animated, the number of active entities, such as processes and nodes, is a few, such as three. Our way to visualize each state of the protocol allows us to immediately recognize how many nodes there are at each location, such as cs (Critical Section), without counting the nodes because of subitizing. r-SMGA makes it possible for human users to rely on subitizing so as to find likely invariants, although we heavily depend on state picture templates. This illustrates how important state picture templates are [11]. Some non-invariant characteristics (e.g. liveness characteristics), such as eventual characteristics, can be perceived in a similar way to LCF, where a group or a pattern can be regarded as a characteristic. LCF can be one possible factor to guide human users in conjecturing liveness characteristics.

Bibliography

[1] M.D. Ernst, J. Cockrell, W.G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, 2001, doi:10.1109/32.908957.

[2] Toh Ne Win and Michael D. Ernst. Verifying Distributed Algorithms via Dynamic Analysis and Theorem Proving. Technical Report 841, MIT Laboratory for Computer Science, Cambridge, MA, May 25, 2002.

[3] Marat Boshernitsan, Roongko Doong, and Alberto Savoia. From Daikon to Agitator: Lessons and Challenges in Building a Commercial Tool for Developer Testing. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, page 169–180, New York, NY, USA, 2006. Association for Computing Machinery, doi:10.1145/1146238.1146258.

[4] Y. Kataoka, M.D. Ernst, W.G. Griswold, and D. Notkin. Automated support for program refactoring using invariants. In *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001*, pages 736–743, 2001, doi:10.1109/ICSM.2001.972794.

[5] M. Harder, J. Mellen, and M.D. Ernst. Improving test suites via operational abstraction. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 60–71, 2003, doi:10.1109/ICSE.2003.1201188.

[6] K. W. Brodlie, L. Carpenter, R. A. Earnshaw, J. R. Gallop, R. J. Hubbold, A. M. Mumford, C. D. Osland, and P. Quarendon. *Scientific Visualization: Techniques and Applications.* Springer-Verlag, Berlin, Heidelberg, 1992, doi:10.1007/978-3-642-76942-9.

[7] Johan Wagemans, James H. Elder, Michael Kubovy, Stephen E. Palmer, Mary A. Peterson, Manish Singh, and Rüdiger von der Heydt. A Century of Gestalt Psychology in Visual Perception: I. Perceptual Grouping and Figure–ground Organization. *Psychological Bulletin*, 138(6):1172–1217, 2012, doi:10.1037/a0029333.

[8] Johan Wagemans, Jacob Feldman, Sergei Gepshtein, Ruth Kimchi, James R. Pomerantz, Peter A. van der Helm, and Cees van Leeuwen. A century of Gestalt Psychology in

Visual Perception: II. Conceptual and Theoretical Foundations. *Psychological Bulletin*, 138(6):1218–1252, 2012, doi:10.1037/a0029334.

[9] Colin Ware. *Information Visualization: Perception for Design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[10] Tam Thi Thanh Nguyen and Kazuhiro Ogata. Graphical Animations of State Machines. In *15th DASC*, pages 604–611, 2017, doi:10.1109/DASC-PICom-DataCom-CyberSciTec.2017.107.

[11] Dang Duy Bui and Kazuhiro Ogata. Better State Pictures Facilitating State Machine Characteristic Conjecture. *Multimedia Tools and Applications*, 81(1):237–272, 2022, doi:10.1007/s11042-021-10992-z.

[12] Dang Duy Bui, Win Hlaing Hlaing Myint, Duong Dinh Tran, and Kazuhiro Ogata. Graphical Animations of the Lim-Jeong-Park-Lee Autonomous Vehicle Intersection Control Protocol. *JVLC*, 2022(1):1–15, 2022, doi:10.18293/JVLC2022-N1-004.

[13] Dang Duy Bui and Kazuhiro Ogata. Graphical Animations of the Suzuki-Kasami Distributed Mutual Exclusion Protocol. *JVLC*, 2019(2):105–115, 2019, doi:10.18293/JVLC2019-N2-012.

[14] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott, editors. *All about Maude - a High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer-Verlag, Berlin, Heidelberg, 2007, doi:10.1007/978-3-540-71999-1.

[15] John M. Mellor-Crummey and Michael L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. *ACM Trans. Comput. Syst.*, 9(1):21–65, 1991, doi:10.1145/103727.103729.

[16] Ichiro Suzuki and Tadao Kasami. A Distributed Mutual Exclusion Algorithm. *ACM Trans. Comput. Syst.*, 3(4):344–349, 1985, doi:10.1145/6110.214406.

[17] Razvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report.* World Scientific, Singapore, 1998, doi:10.1142/3831.

[18] Kazuhiro Ogata and Kokichi Futatsugi. Compositionally Writing Proof Scores of Invariants in the OTS/CafeOBJ Method. *J. Univers. Comput. Sci.*, 19(6):771–804, 2013, doi:10.3217/jucs-019-06-0771.

[19] Adam Naumowicz and Artur Korniłowicz. A Brief Overview of Mizar. In *Theorem Proving in Higher Order Logics*, pages 67–72, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg, doi:10.1007/978-3-642-03359-9_5.

[20] John Harrison. HOL Light: An Overview. In *Theorem Proving in Higher Order Logics*, pages 60–66, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg, doi:10.1007/978-3-642-03359-9_4.

[21] Kryštof Hoder and Andrei Voronkov. Sine Qua Non for Large Theory Reasoning. In *Automated Deduction – CADE-23*, pages 299–314, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg, doi:10.1007/978-3-642-22438-6_23.

[22] Alex Roederer, Yury Puzis, and Geoff Sutcliffe. Divvy: An ATP Meta-system Based on Axiom Relevance Ordering. In *Automated Deduction – CADE-22*, pages 157–162, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg, doi:10.1007/978-3-642-02959-2_13.

[23] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient Semantic Features for Automated Reasoning over Large Theories. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 3084–3090. AAAI Press, 2015, doi:10.5555/2832581.2832679.

[24] Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep Network Guided Proof Search. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 85–105. EasyChair, 2017, doi:10.29007/8mwc.

[25] Karel Chvalovský, Jan Jakubův, Martin Suda, and Josef Urban. ENIGMA-NG: Efficient Neural and Gradient-Boosted Inference Guidance for E. In *Automated Deduction – CADE 27*, pages 197–215, Cham, 2019. Springer International Publishing, doi:10.1007/978-3-030-29436-6_12.

[26] Maxwell Crouse, Ibrahim Abdelaziz, Bassem Makni, Spencer Whitehead, Cristina Cornelio, Pavan Kapanipathi, Kavitha Srinivas, Veronika Thost, Michael Witbrock, and Achille Fokoue. A Deep Reinforcement Learning Approach to First-Order Logic Theorem Proving. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7):6279–6287, May 2021.

[27] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise Selection for Theorem Proving by Deep Graph Embedding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 2783–2793, Red Hook, NY, USA, 2017. Curran Associates Inc., doi:10.5555/3294996.3295038.

[28] Cezary Kaliszyk, François Chollet, and Christian Szegedy. HolStep: A Machine Learning Dataset for Higher-order Logic Theorem Proving. In *International Conference on Learning Representations*, 2017.

[29] Alexander A. Alemi, François Chollet, Niklas Een, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - Deep Sequence Models for Premise Selection. In *Pro-*

ceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, page 2243–2251, Red Hook, NY, USA, 2016. Curran Associates Inc., doi:10.5555/3157096.3157347.

[30] Stephen J. Garland, Nancy A. Lynch, and Mandana Vaziri. IOA: A Language for Specifying, Programming, and Validating Distributed Systems , 1997.

[31] Stephen J. Garland and John V. Guttag. A Guide to LP, The Larch Prover, 1991.

[32] Masahiro Nakano, Kazuhiro Ogata, Masaki Nakamura, and Kokichi Futatsugi. Crème: an Automatic Invariant Prover of Behavioral Specifications. *International Journal of Software Engineering and Knowledge Engineering*, 17(06):783–804, 2007, doi:10.1142/S0218194007003458.

[33] Weikun Yang, Grigory Fedyukovich, and Aarti Gupta. Lemma Synthesis for Automating Induction over Algebraic Data Types. In *Principles and Practice of Constraint Programming*, pages 600–617, Cham, 2019. Springer International Publishing, doi:10.1007/978-3-030-30048-7_35.

[34] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided Synthesis. In *2013 Formal Methods in Computer-Aided Design*, pages 1–8, 2013, doi:10.1109/FMCAD.2013.6679385.

[35] Bernhard Gleiss, Laura Kovács, and Lena Schnedlitz. Interactive Visualization of Saturation Attempts in Vampire. *CoRR*, abs/2001.04100, 2020, doi:10.1007/978-3-030-34968-4_28.

[36] Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. In *Computer Aided Verification*, pages 1–35, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg, doi:10.1007/978-3-642-39799-8_1.

[37] Ján Perháč and Zuzana Bilanová. Another Tool for Structural Operational Semantics Visualization of Simple Imperative Language. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 513–518, 2020, doi:10.1109/ICETA51985.2020.9379205.

[38] Xiaohong Chen and Grigore Roşu. $\mathbb{K}$: A Semantic Framework for Programming Languages and Formal Analysis. In *Engineering Trustworthy Software Systems: 5th International School, SETSS 2019, Chongqing, China, April 21–27, 2019, Tutorial Lectures*, page 122–158, Berlin, Heidelberg, 2019. Springer-Verlag, doi:10.1007/978-3-030-55089-9_4.

[39] Ivan Beschastnikh, Perry Liu, Albert Xing, Patty Wang, Yuriy Brun, and Michael D. Ernst. Visualizing Distributed System Executions. *ACM Trans. Softw. Eng. Methodol.*, 29(2), 2020, doi:10.1145/3375633.

[40] Cyrille Artho, Klaus Havelund, and Shinichi Honiden. Visualization of Concurrent Program Executions. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 2, pages 541–546, 2007, doi:10.1109/COMPSAC.2007.236.

[41] Tam Thi Thanh Nguyen and Kazuhiro Ogata. A Way to Comprehend Counterexamples Generated by the Maude LTL Model Checker. In *2017 International Conference on Software Analysis, Testing and Evolution (SATE)*, pages 53–62, 2017, doi:10.1109/SATE.2017.15.

[42] Cyrille Artho, Monali Pande, and Qiyi Tang. Visual Analytics for Concurrent Java Executions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1102–1105, 2019, doi:10.1109/ASE.2019.00112.

[43] Jeff Magee, Nat Pryce, Dimitra Giannakopoulou, and Jeff Kramer. Graphical Animation of Behavior Models. In *Proceedings of the 22nd International Conference on Software Engineering*, page 499–508, New York, NY, USA, 2000. Association for Computing Machinery, doi:10.1145/337180.337368.

[44] Sheelagh Carpendale. *Evaluating Information Visualizations*, pages 19–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[45] Tobias Isenberg, Petra Isenberg, Jian Chen, Michael Sedlmair, and Torsten Möller. A Systematic Review on the Practice of Evaluating Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2818–2827, 2013, doi:10.1109/TVCG.2013.126.

[46] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz. A Systematic Literature Review of Software Visualization Evaluation. *Journal of Systems and Software*, 144:165–180, 2018, doi:10.1016/j.jss.2018.06.027.

[47] Kelly Caine. Local Standards for Sample Size at CHI. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, page 981–992, New York, NY, USA, 2016. Association for Computing Machinery, doi:10.1145/2858036.2858498.

[48] Martin Schmettow. Sample Size in Usability Studies. *Commun. ACM*, 55(4):64–70, 2012, doi:10.1145/2133806.2133824.

[49] Wonil Hwang and Gavriel Salvendy. Number of People Required for Usability Evaluation: The 10+/-2 Rule. *Commun. ACM*, 53(5):130–133, 2010, doi:10.1145/1735223.1735255.

[50] D. Todorovic. Gestalt Principles. *Scholarpedia*, 3(12):5345, 2008, doi:10.4249/scholarpedia.5345. revision #91314.

[51] K.V. Nesbitt and C. Friedrich. Applying Gestalt Principles to Animated Visualizations of Network Data. In *Proceedings Sixth International Conference on Information Visualisation*, pages 737–743, 2002, doi:10.1109/IV.2002.1028859.

[52] Frédéric Cao. Application of the Gestalt Principles to the Detection of Good Continuations and Corners in Image Level Lines. *Computing and Visualization in Science*, 7(1):3–13, 2004, doi:10.1007/s00791-004-0123-6.

[53] A. Desolneux, L. Moisan, and J.-M. More. A Grouping Principle and Four Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):508–513, 2003, doi:10.1109/TPAMI.2003.1190576.

[54] Mehmet Yalcinkaya and Vishal Singh. Exploring the Use of Gestalt's Principles in Improving the Visualization, User Experience and Comprehension of COBie Data Extension. *Engineering, Construction and Architectural Management*, 26(6):1024–1046, 2019, doi:10.1108/ECAM-10-2017-0226.

[55] Jeffery Garae, Ryan K.L. Ko, and Sivadon Chaisiri. UVisP: User-centric Visualization of Data Provenance with Gestalt Principles. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1923–1930, 2016, doi:10.1109/TrustCom.2016.0294.

[56] Dang Duy Bui, Minxuan Liu, and Kazuhiro Ogata. Graphical Animations of an Autonomous Vehicle Merging Protocol. In *The 28th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2022, KSIR Virtual Conference Center, USA, June 29-30, 2022*, pages 16–22. KSI Research Inc., 2022, doi:10.18293/DMSVIVA21-004.

[57] Thet Wai Mon, Dang Duy Bui, Duong Dinh Tran, and Kazuhiro Ogata. Graphical Animations of the NSLPK Authentication Protocol. In Shi-Kuo Chang, editor, *The 27th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2021, KSIR Virtual Conference Center, USA, June 29-30, 2021*, pages 29–35. KSI Research Inc., 2021, doi:10.18293/DMSVIVA21-005.

[58] Dang Duy Bui, Duong Dinh Tran, Kazuhiro Ogata, and Adrian Riesco. Integration of SMGA and Maude to Facilitate Characteristic Conjecture. In *The 28th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2022, KSIR Virtual Conference Center, USA, June 29-30, 2022*, pages 45–54. KSI Research Inc., 2022, doi:10.18293/DMSVIVA22-006.

[59] Rubén Rubio. Maude as a Library: an Efficient All-purpose Programming Interface. In *14th WRLA*, pages 274–294, Cham, 2022. Springer International Publishing, doi:10.1007/978-3-031-12441-9_14.

[60] Tam Thi Thanh Nguyen and Kazuhiro Ogata. Graphically Perceiving Characteristics of the MCS Lock and Model Checking Them. In *7th SOFL+MSVL*, volume 10795 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2017, doi:10.1007/978-3-319-90104-6_1.

[61] Duong Dinh Tran, *Dang Duy Bui*, and Kazuhiro Ogata. Simulation-Based Invariant Verification Technique for the OTS/CafeOBJ Method. *IEEE Access*, 9:93847–93870, 2021, doi:10.1109/ACCESS.2021.3093211.

[62] Duong Dinh Tran, Dang Duy Bui, Parth Gupta, and Kazuhiro Ogata. Lemma Weakening for State Machine Invariant Proofs. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 21–30, 2020, doi:10.1109/APSEC51365.2020.00010.

[63] Thet Wai Mon, Dang Duy Bui, Duong Dinh Tran, Canh Minh Do, and Kazuhiro Ogata. Graphical Animations of the NSLPK Authentication Protocols. *J. Vis. Lang. Comput.*, 2021(2):39–51, 2021, doi:10.18293/jvlc2021-n2-005.

[64] Duong Dinh Tran and Kazuhiro Ogata. Ipsg: Invariant proof score generator. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1050–1055, 2022, doi:10.1109/COMPSAC54236.2022.00164.

[65] A. Hernando, J. Bobadilla, F. Ortega, and A. Gutiérrez. Method to Interactively Visualize and Navigate Related Information. *Expert Systems with Applications*, 111:61–75, 2018, doi:10.1016/j.eswa.2018.01.034.

[66] María Alpuente, Demis Ballis, Francisco Frechina, and Julia Sapiña. Debugging Maude Programs via Runtime Assertion Checking and Trace Slicing. *J. Log. Algebraic Methods Program.*, 85(5):707–736, 2016, doi:10.1016/j.jlamp.2016.03.001.

[67] Frank van Ham, Huub van de Wetering, and Jarke J. van Wijk. Interactive Visualization of State Transition Systems. *IEEE Transaction on Visual and Computer Graphics*, 8(4):319–329, 2002, doi:10.1109/TVCG.2002.1044518.

[68] George G Robertson, Jock D Mackinlay, and Stuart K Card. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194, 1991.

[69] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981, doi:10.1109/TSMC.1981.4308636.

[70] Jan Groote and Frank Ham. Interactive Visualization of Large State Spaces. *STTT*, 8:77–91, 02 2006, doi:10.1007/s10009-005-0198-5.

[71] E. L. Kaufman, M. W. Lord, T. W. Reese, and J. Volkmann. The Discrimination of Visual Number. *The American Journal of Psychology*, 62(4):498–525, 1949, doi:10.2307/1418556.

[72] Lana M Trick and Zenon W Pylyshyn. Why Are Small and Large Numbers Enumerated Differently? A Limited-capacity Preattentive Stage in Vision. *Psychological review*, 101(1):80, 1994, doi:10.1037/0033-295x.101.1.80.

[73] Paula Maldonado Moscoso, Elisa Castaldi, David Burr, Roberto Arrighi, and Giovanni Anobile. Grouping Strategies in Number Estimation Extend the Subitizing Range. *Scientific reports*, 10:14979, 09 2020, doi:10.1038/s41598-020-71871-5.

[74] Henry Railo. Bilateral and Two-item Advantage in Subitizing. *Vision Research*, 103:41–48, 2014, doi:10.1016/j.visres.2014.07.019.

[75] Kurt Koffka. Perception: An Introduction to the Gestalt Theory. *Psychological Bulletin*, 19:531–585, 1922, doi:10.1037/h0072422.

[76] Johan Wagemans, James H. Elder, Michael Kubovy, Stephen E. Palmer, Mary A. Peterson, Manish Singh, and Rüdiger von der Heydt. A Century of Gestalt Psychology in Visual Perception: I. Perceptual Grouping and Figure-ground Organization. *Psychological bulletin*, 138 6:1172–217, 2012, doi:10.1037/a0029333.

[77] Amira Chalbi, Jacob Ritchie, Deokgun Park, Jungu Choi, Nicolas Roussel, Niklas Elmqvist, and Fanny Chevalier. Common Fate for Animated Transitions in Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):386–396, 2020, doi:10.1109/TVCG.2019.2934288.

First-author Publications

[1] Dang Duy Bui and Kazuhiro Ogata. Graphical Animations of the Suzuki-Kasami Distributed Mutual Exclusion Protocol. *J. Vis. Lang. Comput.*, 2019(2):105–116, 2019, doi:10-18293/JVLC2019-N2-012.

[2] Dang Duy Bui and Kazuhiro Ogata. Better State Pictures Facilitating State Machine Characteristic Conjecture. *Multim. Tools Appl.*, 81(1):237–272, 2022, doi:10.1007/s11042-021-10992-z.

[3] Dang Duy Bui, Win Hlaing Hlaing Myint, Duong Dinh Tran, and Kazuhiro Ogata. Graphical Animations of the Lim-Jeong-Park-Lee Autonomous Vehicle Intersection Control Protocol. *J. Vis. Lang. Comput.*, 2022(1):1–15, 2022, doi:10.18293/JVLC2022-N1-004.

[4] Dang Duy Bui and Kazuhiro Ogata. Graphical Animations of the Suzuki-Kasami Distributed Mutual Exclusion Protocol. In Joseph J. Pfeiffer Jr., editor, *The 25th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2019, Hotel Tivoli, Lisbon, Portugal, July 8-9, 2019*, pages 125–137. KSI Research Inc., 2019, doi:10.18293/DMSVIVA2019-012.

[5] Dang Duy Bui and Kazuhiro Ogata. Better State Pictures Facilitating State Machine Characteristic Conjecture. In Shi-Kuo Chang, editor, *The 26th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2020, KSIR Virtual Conference Center, USA, July 7-8, 2020*, pages 7–12. KSI Research Inc., 2020, doi:10.18293/DMSVIVA20-007.

[6] Dang Duy Bui, Minxuan Liu, and Kazuhiro Ogata. Graphical Animations of an Autonomous Vehicle Merging Protocol. In *The 28th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2022, KSIR Virtual Conference Center, USA, June 29-30, 2022*, pages 16–22. KSI Research Inc., 2022, doi:10.18293/DMSVIVA22-009.

[7] Dang Duy Bui, Duong Dinh Tran, Kazuhiro Ogata, and Adrian Riesco. Integration of SMGA and Maude to Facilitate Characteristic Conjecture. In *The 28th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2022, KSIR Virtual Conference Center, USA, June 29-30, 2022*, pages 45–54. KSI Research Inc., 2022, doi:10.18293/DMSVIVA22-006.

Other Co-author Publications

[1] Thet Wai Mon, Dang Duy Bui, Duong Dinh Tran, Canh Minh Do, and Kazuhiro Ogata. Graphical Animations of the NSLPK Authentication Protocols. *J. Vis. Lang. Comput.*, 2021(2):39–51, 2021, doi:10.18293/jvlc2021-n2-005.

[2] Duong Dinh Tran, Dang Duy Bui, and Kazuhiro Ogata. Simulation-Based Invariant Verification Technique for the OTS/CafeOBJ Method. *IEEE Access*, 9:93847–93870, 2021, doi:10.1109/ACCESS.2021.3093211.

[3] Duong Dinh Tran, Dang Duy Bui, Parth Gupta, and Kazuhiro Ogata. Lemma Weakening for State Machine Invariant Proofs. In *27th Asia-Pacific Software Engineering Conference, APSEC 2020, Singapore, December 1-4, 2020*, pages 21–30. IEEE, 2020, doi:10.1109/APSEC51365.2020.00010.

[4] Minxuan Liu, Dang Duy Bui, Duong Dinh Tran, and Kazuhiro Ogata. Formal Specification and Model Checking of an Autonomous Vehicle Merging Protocol. In *21st IEEE International Conference on Software Quality, Reliability and Security, QRS 2021 - Companion, Hainan, China, December 6-10, 2021*, pages 333–342. IEEE, 2021, doi:10.1109/QRS-C55045.2021.00057.

[5] Thet Wai Mon, Dang Duy Bui, Duong Dinh Tran, and Kazuhiro Ogata. Graphical Animations of the NSLPK Authentication Protocol . In Shi-Kuo Chang, editor, *The 27th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2021, KSIR Virtual Conference Center, USA, June 29-30, 2021*, pages 29–35. KSI Research Inc., 2021, doi:10.18293/DMSVIVA21-005.

[6] Win Hlaing Hlaing Myint, Dang Duy Bui, Duong Dinh Tran, and Kazuhiro Ogata. Graphical Animations of the Lim-Jeong-Park-Lee Autonomous Vehicle Intersection Control Protocol . In Shi-Kuo Chang, editor, *The 27th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2021, KSIR Virtual Conference Center, USA, June 29-30, 2021*, pages 22–28. KSI Research Inc., 2021, doi:10.18293/DMSVIVA21-004.