Abstract

In this research project, we created formal specifications of some leader election algorithms and then verified that they enjoy desired properties using model checking. By doing this, we will show how to create a formal specification of a leader election algorithm and how to model check that the algorithm enjoys desired properties based on the formal specification in a realistic execution time.

In distributed systems, centralized management is not a good idea in terms of system load and failure, and it is considered better to design a system with multiple processes that have the same role. However, it is easier to make sure to keep data integrity if there is one special process that is in charge of it.

The process of selecting one elected process, or "leader," from among multiple processes that comprise a distributed system is called leader election. Various algorithms have been proposed for leader election.

Since leader election has been used in recent years for distributed databases and for building fault tolerance into systems, the accuracy of the leader election algorithm has become very important from the perspective of system operation. For this reason, leader election algorithms are modeled and model checking tools are used to verify that such algorithms satisfy desired properties. However, the difficulty of modeling and the unrealistically long verification time due to state explosion have become problems.

In the experiments described in this report, we created formal specifications using Maude for three leader election algorithms: the Bully algorithm, the Chang-Roberts algorithm, and the Franklin algorithm. In addition, based on the created formal specifications, we used model checking to verify whether each algorithm satisfies the guaranteed properties.

In the Bully algorithm, when a process detects that a leader has stopped, it starts the next election process. The process that started the election sends a message to all processes whose IDs are greater than its own ID. If there is no reply from the processes to which the message has been sent, the process that started the election becomes the leader. Conversely, if a process whose ID is greater than the process that initiated the current election replies to the message, the election is replaced. Eventually, there will be no processes left that reply to such a message, and the one remaining process will become the leader. In this report, we formalize and specify the Bully algorithm, which performs the above operations, as a state transition system with eight observable components, and 11 rewriting rules for state transitions. We then used model checking to verify that the algorithm enjoys desired properties based on the formal specification created. In the verification experiments, the number of processes targeted by the Bully algorithm was set to five, and model checking was performed. The results of the

validation experiments confirm that the Bully algorithm satisfies the properties of a leader election algorithm.

In the Bully algorithm, all processes perform the selection process step by step in synchronized timing. Therefore, it is difficult to perform model checking in Maude, which inherently performs the process asynchronously. However, in this report, we decided to perform model checking of the Bully algorithm by performing three synchronous measures to the rewriting rules of the formal specification that we created. As a result of the measures, it was confirmed that the properties of a leader election algorithm were satisfied, so the measures are considered to be effective.

In the Chang-Roberts algorithm, when a non-candidate process detects the absence of a leader, it starts the election process. The process that detects the absence of the leader becomes a candidate as a starting process and sends a message including its own process ID to the neighboring process. If the process that received the message is a non-candidate, the message is relayed to the neighboring process, and if the process that received the message is a candidate, it compares its own process ID with the process ID in the received message. If its own process ID is higher, it sends the received message to its neighbor. Conversely, if its own process ID is lower, it discards the received message. This process is continued until only the message containing the lowest process ID has gone around the ring. In this report, we formalize and specify the Chang-Roberts algorithm, which operates as described above, as a state transition system with four observable components, and nine rewriting rules for state transitions. The five properties that the algorithm must satisfy are described as LTL formulas. We then used model checking to verify that the algorithm enjoys the five properties based on the formal specification created. In the verification experiments, the number of processes targeted by the Chang-Roberts algorithm was set to five, two initial states were defined that considered the sorting order, and model checking was performed. The results of the verification experiments confirmed that the Chang-Roberts algorithm satisfies the properties of a leader election algorithm, regardless of the order of processes arranged in a ring. Additionally, it was confirmed that the three properties specific to the Chang-Roberts algorithm were also satisfied.

In the Franklin algorithm, a non-initiator process starts the election process when it detects the absence of a leader. The process that detects the absence of the leader becomes the initiator and sends a message including its own process ID to both neighboring processes. If a process receiving the message is a non-initiator, it relays the message to the neighboring process as a passive process. However, if a process that received the message is an initiator, it compares its own process ID with the higher of the process IDs of the messages received from

both neighboring processes. If its own process ID is higher, it sends a message including its own process ID to both neighboring processes again. If its own process ID is lower, it becomes a passive process. If it is equal to its own process ID, it becomes a leader. In this report, we formalize and specify the Franklin algorithm, which operates as described above, as a state transition system with four observable components, and 12 rewriting rules for state transitions. The two properties that the algorithm must satisfy are described as LTL formulas. We then used model checking to verify that the algorithm enjoys the two properties based on the formal specification created. In the verification experiments, the number of processes targeted by the Franklin algorithm was set to five, two initial states were defined that considered the sorting order, and model checking was performed. The results of the verification experiments confirmed that the Franklin algorithm satisfies the properties of a leader election algorithm, regardless of the order of processes arranged in a ring.

keywords: model checking, distributed algorithm, leader election, Maude, safety property, liveness property, state transition system