Eric Crosson, Nhan Do, William Mauldin, and Daniel Officewala

Professor Vijay Garg

EE 360P

May 3, 2016

***Abstract**: This report presents the analysis of several popular distributed algorithms through the use of the Promela modeling language. It was written as the final project for Professor Vijay Garg's Concurrent and Distributed Computing course at the University of Texas at Austin.*

1 Introduction

Spin is a multi-threaded software verification tool that supports the high-level language Promela to specify meta-level models of systems [1]. Given a Promela model, Spin either performs random simulations of the system's execution or generates a C program that exhaustively verifies that system's correctness properties, including deadlocks, unspecified receptions, and unexecutable blocks, as well as system invariants and other properties defined by the user.

2 Project Description

The intent of this project is to verify four models of distributed algorithms written in Promela. We chose to model the Dining Philosophers Algorithm [8], the Token Ring Algorithm [9], Szymanski's Mutual Exclusion Algorithm [7], and Chandy and Lamport's Global Snapshot Algorithm [3]. We based our implementation heavily upon existing implementations of these algorithms, modified to suit our specific uses.

Each of the four team members was responsible for researching one algorithm, finding a suitable model, and re-implementing it for use within our directory structure and build system. Eric created the majority of the build system and set up Docker [6] to allow for our research to be reproducible, all of which is available on Github [5]. William assisted with this process by

helping to write some of the CMake files necessary to build the project. Eric focused his research on Szymanski's Algorithm, Daniel on the Token Ring Algorithm, William on Chandy and Lamport's Algorithm, and Nhan on the Dining Philosophers Algorithm.

3 Performance Results

We were able to successfully build all of the algorithms and run them in Spin, but only three out of the four passed the verification tests that we used. We modified all of the existing Promela models such that any defined constants are now able to be passed as parameters and that any assertions are extracted to separate test files.

3.1 Dining Philosophers Algorithm

The Dining Philosophers Algorithm is a widely used algorithm for guaranteeing mutual exclusion between multiple processes sharing resources. We used a model created by oflynned [8] for verification. We used one test file for this model to ensure deadlock resolution. When run in Spin, we had no assertion failures, so deadlock was verified as impossible. However, we determined that the algorithm was not free from starvation, since it is theoretically possible for the special process (the process that surrenders a held resource when a second resource is unavailable) to constantly surrender its held resource if another, faster process constantly gets access to the second resource prior to the special process.

3.2 Token Ring Algorithm

In the Token Ring Algorithm, a ring of processes is constructed, and each process is assigned a specific position in the ring. A token for critical section (CS) access is passed around the ring, and a process can either use it to access the CS or pass it along. We used a model created by [9] in order to model the algorithm, and a set of mutual exclusion test cases created by Fumiyoshi Kobayashi [4]. As expected, when we ran the tests against the model in Spin, safet and liveness

were confirmed, since a process is only allowed to enter the CS if it holds the sole token, and the token is continually passed either immediately or after exiting the CS. However, these tests do not account for faults (lost token). In order to implement fault-tolerance, the algorithm could be modified by having processes send an acknowledgement upon receiving a token from a previous process.

3.3 Szymanski's Mutual Exclusion Algorithm

Szymanski solved Lamport's open problem whether there exists an algorithm with a constant number of communication bits per process that satisfies every reasonable fairness and fault-tolerance requirement that Lamport's solution to the mutual exclusion problem satisfies. Szymanski realized that a 5 state model is sufficient to keep track of processes as they prepare for and exit the critical section; running these states through a Karnaugh map yields a mutual exclusion solution that requires only three boolean variables per process. The lack of global variables means a fault tolerant version of this algorithm is achievable, since there is no danger of an alive process reading a stale (dead) process's global state variable.

Szymanski's algorithm does satisfy the proofs listed in [10], namely safety, liveness, and absence of deadlocks. The absence of global variables in Szymanski's implementation yielded a fault-tolerant version of the algorithm that did not exhibit the liveness property as asserted in the proof's publication. Szymanski was close though; Promela shows that transposing the order of the epilogue of the critical section is enough to prove liveness and thus solve Lamport's problem.

3.4 Chandy and Lamport's Global Snapshot Algorithm

Chandy and Lamport's Global Snapshot Algorithm defines a method to take a global snapshot of a distributed system, which has proven to be a challenge due to the absence of shared memory and a shared clock. In this algorithm, the global snapshot is determined by collecting all process states (messages recorded by a given process prior to receiving a marker) and channel states (messages recorded by a processes after receiving a marker). All channels are assumed to be

unidirectional and FIFO. The algorithm theoretically guarantees consistency (all recorded states are concurrent in the happened-before model). To test consistency, we used a model created by Mordecai Ben-Ari [2] that includes a sender process and a receiver process communicating over a channel. The two tests we extracted theoretically should have confirmed that $i)$ the last message sent was always equal to the message at the marker (correct channel states), and $ii)$ any message recorded was always less than the message at the marker (correct process states). Unfortunately, due to our previous unfamiliarity with the Promela language, both of our extracted tests continually resulted in assertion failures. We attribute this to our lack of experience with Promela due to the fact that this model has previously been verified in its original state without separate test files and parameter passing.

4 Conclusions

Although not all of our Promela models were able to pass their respective verification tests, we were able to build and execute them using Spin. Given more time to learn the grammar and syntax of Promela, we believe that we could devise more robust tests that produce more accurate results.

References

[1] *Basic Spin Manual*. `http://spinroot.com/spin/Man/Manual.html`. Accessed: 2016-04-28.

[2] M. Ben-Ari. "Case Studies". In: *Principles of the Spin Model Checker* (2008), pp. 197–200.

[3] K. Mani Chandy and Leslie Lamport. *Distributed Snapshots: Determining Global States of Distributed Systems*. Vol. 3. 1. New York, NY, USA: ACM, Feb. 1985, pp. 63–75. DOI: `10.1145/214451.214456`. URL: `http://doi.acm.org/10.1145/214451.214456`.

[4] Promela Source Code. *Camelot*. 2008. URL: www.ueda.info.waseda.ac.jp/~kobayashi/Promela/benchmark/index.html (visited on 04/28/2016).

[5] Eric Crosson. *Camelot*. 2016. URL: https://github.com/ericcrosson/camelot (visited on 04/28/2016).

[6] Docker. *Getting Started with Docker*. 2016. URL: https://www.docker.com/ (visited on 04/28/2016).

[7] Mateusz Machalica. *Promela Szymanski*. 2014. URL: https://github.com/stupaq/promela-szymanski.

[8] oflynned. "Dining Philosophers Promela". In: (2008). URL: https://github.com/oflynned/DiningPhilosophersPromela.

[9] Dr. Joe Pfeiffer. *A TOKEN RING ALGORITHM*. url. http://www.cs.nmsu.edu/ arao/courses/cs574/mutex/tokenring.html.

[10] B. K. Szymanski. "A simple solution to Lamport's concurrent programming problem with linear wait". In: *ICS '88 Proceedings of the 2nd international conference on Supercomputing* (1988), pp. 621–626.