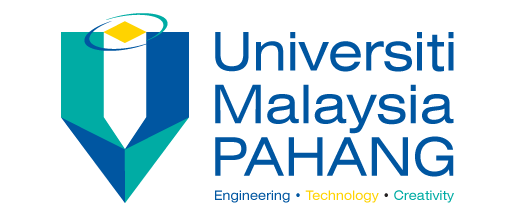
****

**GROUP PROJECT DEVELOPEMENT**

**PRODUCER-CONSUMER PROBLEM**

**BCS2213 FORMAL METHOD**

**SEMESTER 2 (2013/2014)**

**Documented by**

CHUA KWAN LIANG CB11035

ELVIRA ARIAPUTRA CB13125

KEVIN WONG LIANG JI CB13016

LIM SENG KEE CB11049

CHEAH JIT CHOON CB11091

1. Introduction

In the course BCS2213, Formal Method, we learned about Temporal Logic of Actions (TLA) and its applications.

TLA is a specification language combination of temporal logic with actions logic that is used to express behaviors of concurrent systems. Not only that, TLA is also good in describing asynchronous systems. TLA allows writing a precise and formal description of almost any kind of discrete system by a single formula.

In order to fully understand TLA, we apply TLA in modeling producer-consumer problem which is also known as the bounded-buffer problem. Producer-consumer problem is a classic example of a multi-purpose synchronization problem that involves two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. Thus, we implement TLA specification for the multiple producers and consumers.

1. Objective

The main objective of this project is to help students in having a better understanding on TLA specification. The Modeling Producer-Consumer problem is a problem that occurs in our daily routine. The multitasking computer may have the deadlock problem if there is an inadequate where both processes are idle and active at the same time. Another objective of this project is to allow students understand the usefulness of formal method in computer programs in our daily life. Formal methods provided a solution with reliable, effective and secure system. It uses the logic, finite state machine and discrete math in developing codes. Last but not least, this project will enhance the relationship among the group members because of working together to solve the TLA problems with our ability, rational, tolerance and team working. Our aim is to solve the task of Modeling Producer-Consumer problem with a solution in TLA format.

1. Project Description

Title: Modeling Producer-Consumer Problem

Producer- consumer problem, (also known as the bounded-buffer problem) was a classic example of multi process synchronization problem. In this model, there will have two process, consumer and producer sharing a same buffer in a queue. The producer is to generate a piece of data and send into buffer, at the same time the consumer will consume the data from the buffer a piece of data per time. The main goal is to make sure that the buffer state is fully occupy by data and will not empty.

When the buffer is fully loaded, producer will stop sending data into buffer, it will wait until the consumer retrieves the data from buffer. In the same way, if buffer is empty then the consumer will waiting until the buffer is filling by producer. The solution can be reached by means of inter-process communication, typically using smartphone. An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

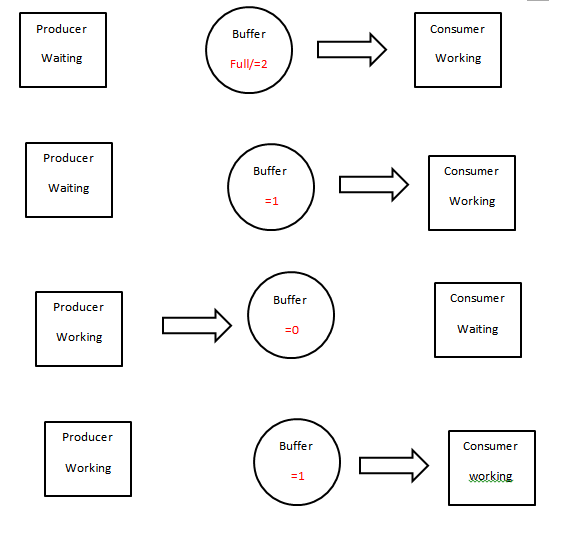


Figure 3.1 The Process Producer-Consumer

1. TLA Module

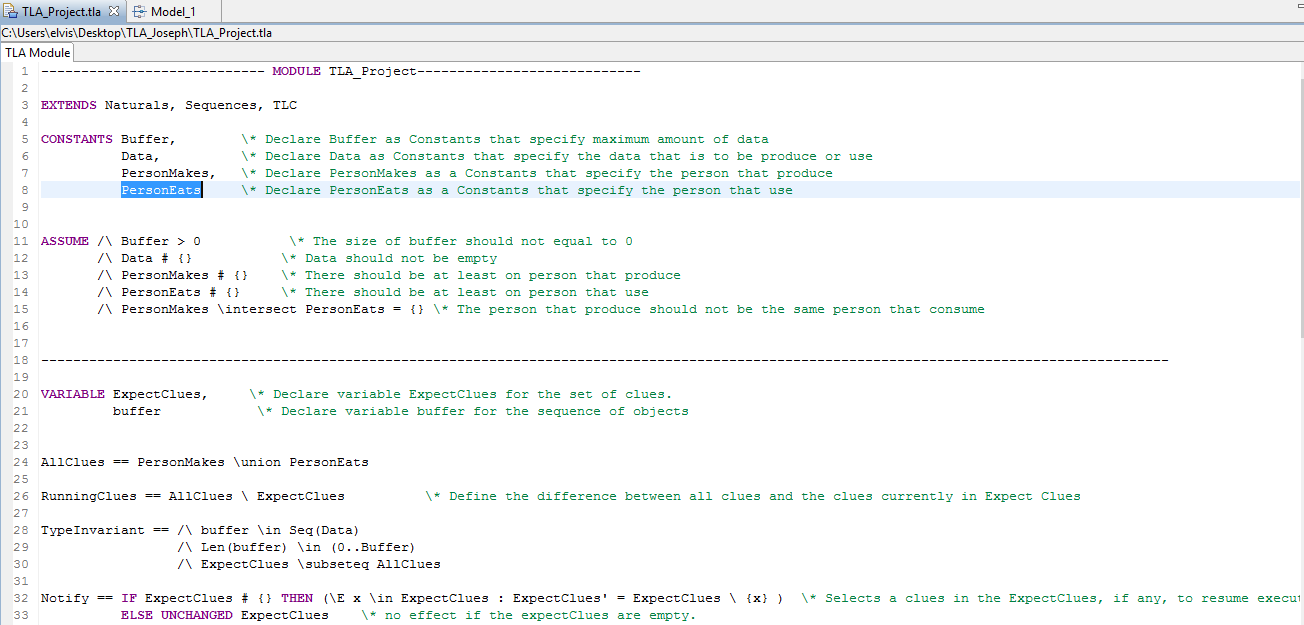


Figure 4.1 TLA module

Based on this producer consumer module has four constants which act as parameters. First is the PersonMakes clues, follow by a set of PersonEats clues, a buffer size and a set of data value. In this module, we assume that the sets of clues are all disjoint and also are non-empty set, the buffer size is also non-empty and there is at least one piece of data that can be sent through the buffer.

Secondly, declared ExpectClues variable and buffer variable. The ExpectClues is a set of clues and the buffer is a sequence of data elements. AllClues is defined as all the PersonMakes clues union with all the PersonEats clues and RunningClues is defined as a set difference between all the clues and the clues currently running. For the Notify, it’s defined as selects a clue in the ExpectClues set, if any, to resume the execution, else remain unchanged for the ExpectClues.   The Expect is defined as suspended the clue and placed the clue into the ExpectClues. Init is to initialize the buffer as empty sequence and also the ExpectClues as an empty set. Method Get suspends the calling clue until the buffer is nonempty. The clue then removes one object from the buffer and does a single call to Notify to (potentially) unlock a clue blocked on Expect. Method Put is symmetric. Both methods are synchronized in order that their execution appears atomic to all clues. In particular, a clue cannot see intermediate states while the buffer is being modified by another clue. This is also why the call to Notify can take place before the buffer is actually modified. The basic mechanism to suspend a clue by call to Expect which suspends a clue unconditionally.

The case of Notify is the most interesting because it is modelled non-deterministically: some clue is removed from the wait set, but we don't specify which one. For the Put (clue, data): if the buffer is not full, add data at the end and notify; otherwise, the clue waits and the buffer is left unchanged. The definition of Next reads as follows: for the system to transition to its next state, some clue, currently running, performs an operation; either clue is a PersonMakes and it attempts to put some piece of data in the buffer; or clue is a PersonEats and it attempts to retrieve some piece of data from the buffer. The deadlock that is being investigated here happens in some of these behaviours, but not all and as we have seen before, there are many behaviours in which it doesn't happen. The NoDeadlock defined as there is always at least one clue running (the ◻ makes it apply to all the states of the system) and the theorem expresses that the system is type-correct and satisfies the NoDeadlock property.

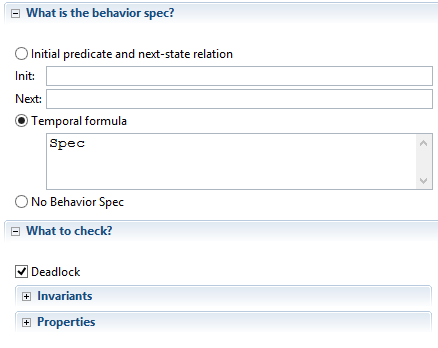


Figure 4.2 TLA Specification

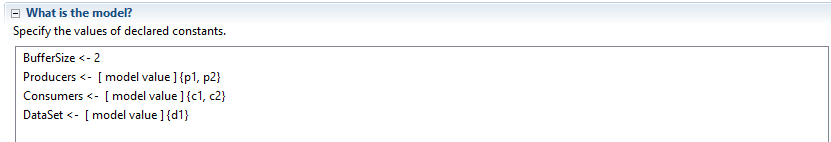


Figure 4.3 TLA Model Checker

We need to instantiate a module by assigning values to all its constants before we check its states. Since what is put into the buffer is irrelevant, so we use only one data to reduce the number of states to explore. Below is the result.

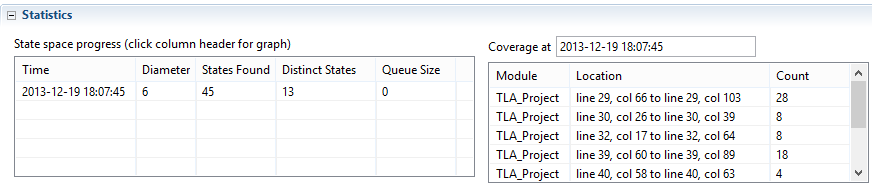


Figure 4.4 TLA results

1. Conclusion

In conclusion, we have completed the TLA specification for producer-consumer problem. We can’t deny that to specify this system, it require a high expertise and knowledge due to its complexity that present, but because of our teamwork, our caring towards each other, we have completed this specification. Besides that, we also learned how each variable interacts with each other and an error that hidden in a line of specification will render the entire specification not executable. On top of that, we also have learnt how to use the model checking tools that is a sub module of the TLA toolbox which we can feed the constant with not only an ordinary assignment but also model value. The one that stand out from another is that we have learnt that formal method can help the development of the system and reduce the hidden bug that is present in the developing system especially for that safety critical system which upon failure may consume life. At last not least we also thank our lecturer Prof. Dr. Vitaliy Mezhuyev who has guided us throughout this whole semester which we are able to produce this specification. Once again, thank You Prof.Dr. Vitaliy Mezhuyev.

1. References
2. Temporal logic of actions, Wikipedia, <http://en.wikipedia.org/wiki/Temporal_Logic_of_Actions>
3. Producer-consumer problem, Wikipedia,
4. <http://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem>
5. Producer Consumer Design Pattern with Blocking Queue Example in Java, Javarevisited, <http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html>
6. Java Examples - Producer Consumer Problem, Tutorial Spoint, <http://www.tutorialspoint.com/javaexamples/thread_procon.htm>
7. The Producer Consumer Problem, InterProcess Communication, <http://cs.gmu.edu/cne/modules/ipc/aqua/producer.html>
8. Producer Consumer Problem with Wait and Notify Example, Java 67, 30th December 2012, <http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html>
9. <http://cs.wellesley.edu/~ecom/lecture/ConsumerProducerRight.html>