

**BCS2213 FORMAL METHODS**

**SEMESTER 1 SESSION 2013/2014**

**TLA Group Project**

TITLE: MODELLING PRODUCER–CONSUMER PROBLEM

**PREPARED FOR:**

**Prof. Dr. Vitaliy Mezhuyev**

**PREPARED BY:**

|  |  |
| --- | --- |
| Group Members Name | ID Number |
| GOH WEN SEN | **CB11046** |
| CHAI WEI CHING | **CB11056** |
| REWATHI KARUDAN | **CD11060** |
| MOHAMAD SAIFUL ANUAR BIN YUSOF | **CB12013** |
| MUHAMMAD TAUFIQ BIN MOHD SAFWAN CHIN | **CB11096** |
| AREEF AMEER BIN ZAMMERI | **CB10049** |

1. **Introduction**

Temporal Logic of Actions (TLA) is one of the specifications which used in computer programming. TLA is a specification of modern computer systems needs expression of its temporal properties. In year 1977, Amir Pnueli introduced the use of temporal logic for describing system behaviors. In the late 1980's, Lesly Lamport invented the Temporal Logic of Actions (TLA), a variant of Pnueli's original logic. TLA is applicable for specifying a wide class of software systems from simple programs to large distributed and concurrent systems. In addition, TLA is good for describing asynchronous systems. TLA allows writing a precise and formal description of almost any kind of discrete system by a single formula. It uses first order logic and set theory for expressing ordinary mathematics. TLA expands ordinary mathematics by temporal operators such as next, always, eventually.

We are students of Formal Methods (BCS2213) was given a TLA project which we are have to solve a given case study by using TLA.We were selected a problem involving producer and consumer. Our title is Modelling Producer–consumer problem. The Modelling Producer–Consumer Problem (MP-CP) is a famous case study which is classical example of a multi-process synchronization problem. It was build up in scheduling problems with consumable resources such as raw materials or money in particular for batch based processing.

1. **Objective of TLA project**

The main objective of our project is to better understanding of students on TLA specification. The Modelling Producer-Consumer Problem is a common problem we always face n our daily life such as multitasking computer which may show deadlock if there is an inadequate where both processes are waiting to be awakened. Another objective of this TLA project is to allow student understand the benefits of formal methods in computer programmes even in their daily life as well. Formal methods is a method which giving a solution with highly reliable, more effective and safe secure systems. In addition the Formal Methods are using the logic, finite state machine, discrete math in developing codes. In the nutshell, this given project will enhance the relationship the group members. We all work together to solve one TLA problem with tolerance, rational and helping each other. Here our aim is to solve the given task Modelling Producer – Consumer problem with a better solution in TLA format.

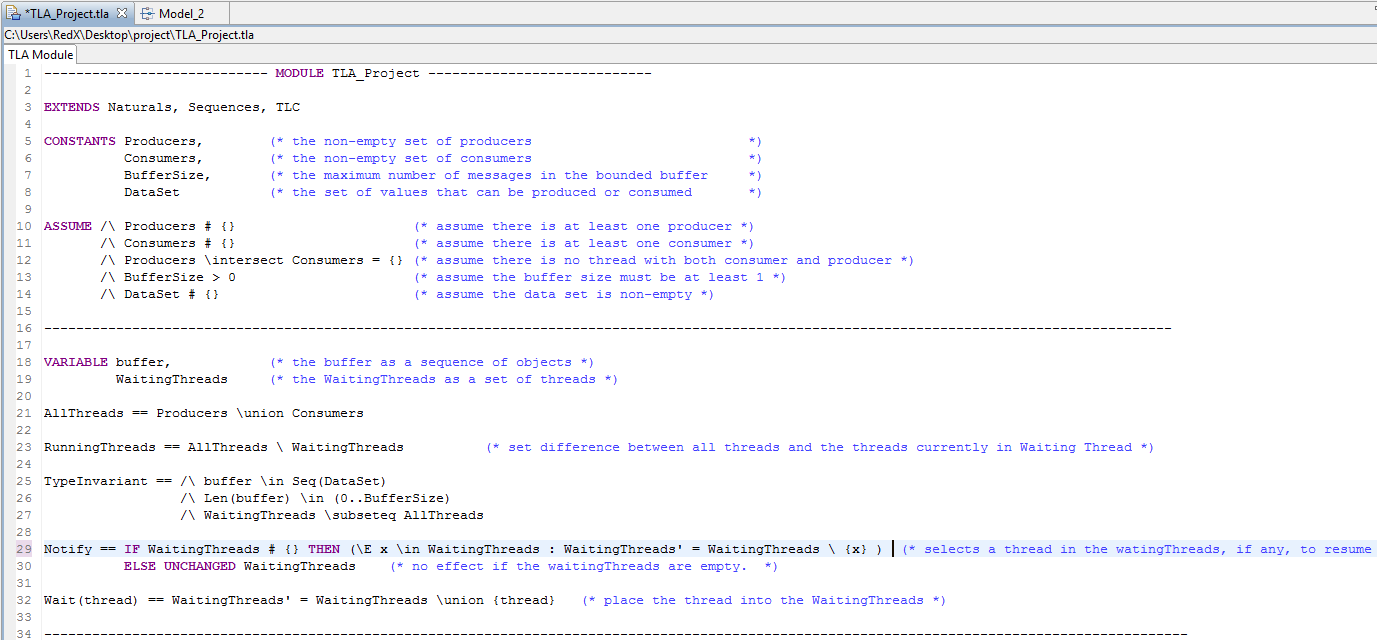
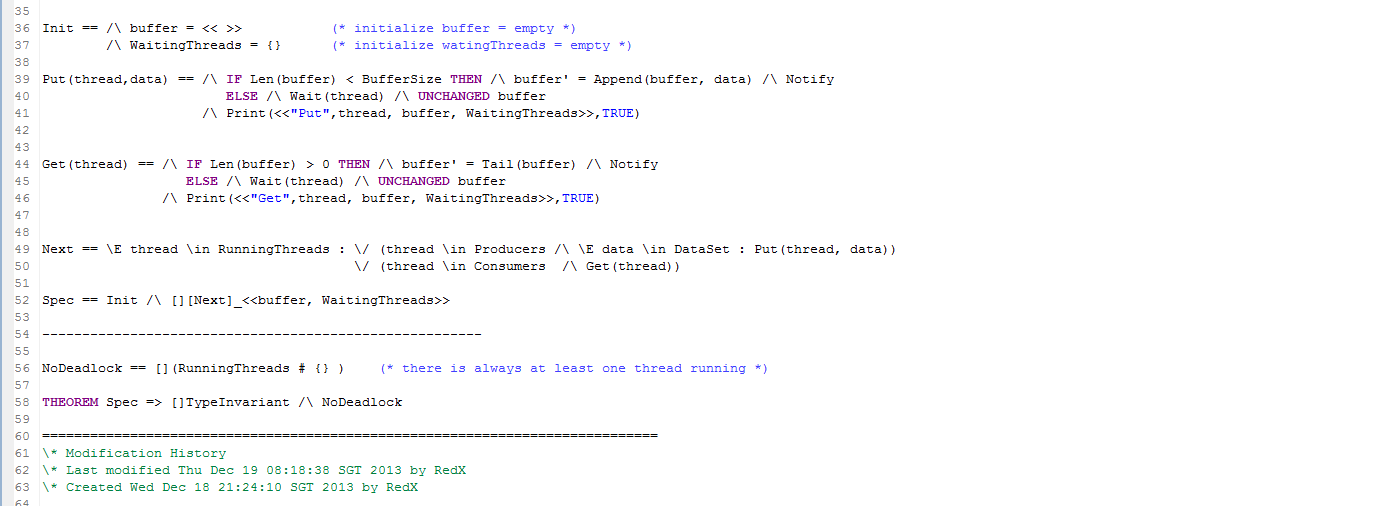
1. **Project Description**

**Title**: Modelling Producer-Consumer Problem

The producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

**Buffer**

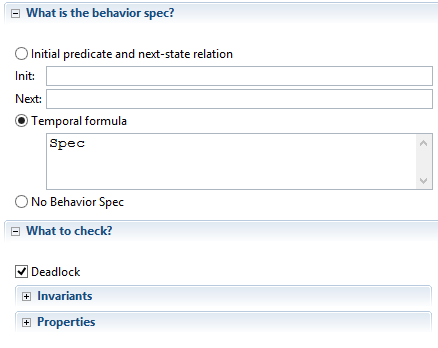
1. **TLA Module**

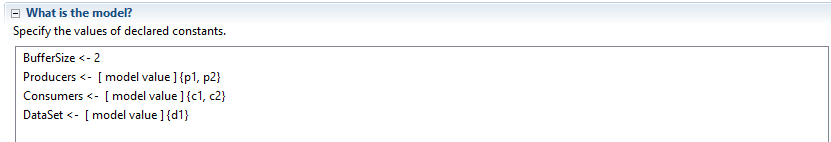
****

The producer consumer module has four constants which act as parameters. The first one is a set of producer threads, follow by a set of consuming threads, a buffer size and a data value set. A very important tip here is we need to assume that the sets of threads 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.

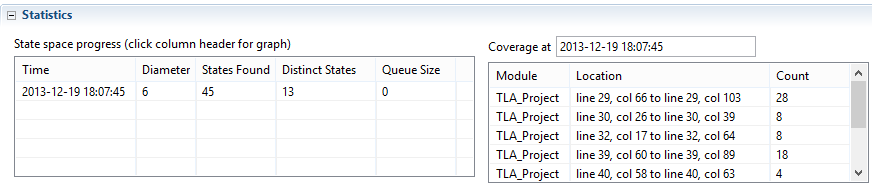
Next, we declared buffer variable and WaitingThreads variable. The buffer is a sequence of data elements while the WaitingThreads is a set of threads. AllThreads is defined as all the producer threads union with all the consumer threads and RunningThreads is defined as a set difference between all the threads and the threads currently running. For the Nofity, it’s defined as selects a thread in the WaitingThreads set, if any, to resume the execution, else remain unchanged for the WatingThreads.   The Wait is defined as suspended the thread and placed the thread into the WaitingThreads. Init is to initialize the buffer as empty sequence and also the WaitingThreads as an empty set. Method get suspends the calling thread until the buffer is nonempty. The thread then removes one object from the buffer and does a single call to notify to (potentially) unlock a thread blocked on wait. Method put is symmetric. Both methods are synchronized in order that their execution appears atomic to all threads. In particular, a thread cannot see intermediate states while the buffer is being modified by another thread. This is also why the call to notify can take place before the buffer is actually modified. The basic mechanism to suspend a thread by call to Wait which suspends a thread unconditionally.

The case of notify is the most interesting because it is modelled non-deterministically: some thread is removed from the wait set, but we don't specify which one. For the Put (thread, data): if the buffer is not full, add data at the end and notify; otherwise, the thread waits and the buffer is left unchanged. The definition of Next reads as follows: for the system to transition to its next state, some thread, currently running, performs an operation; either thread is a producer and it attempts to put some piece of data in the buffer; or thread is a consumer 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 thread 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.





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.



1. **Conclusion**

As a conclusion, after a lot of struggling and effort finally we had done our TLA project successfully. Along our journey of this project we had gained more experiences, important of TLA specification and moral values too. At the same time we also had a great chance to know well about the important of the subject Formal Methods (BCS2213) for computer students. This subject allows us to improve our knowledge in computer programme development. We also learned some vital value of system such as the accuracy, correctness, robustness of the problem using a specific specification. We also had experiences on the model checking technique after we done our system in TLA software. The function of TLA model checking is to display the possible distinct states developed system like Producer- Consumer Problem. The most important lesson here is that a formal model can be used for many purposes and simple model-checking can be a way to discover bugs that result from non-determinism, for which testing can be very tricky and requires patience, computing resources and, what is much more problematic, luck. At last not least we also thank our lecturer Prof. Dr. Vitaliy Mezhuyev who shared his ideas regarding our task Producer- Consumer Problem. He also gave motivations to us and helps in complete our task successfully.

1. **References**

* <http://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem>
* <http://cs.gmu.edu/cne/modules/ipc/aqua/producer.html>
* [https://www.google.com/search?q=producer+consumer+problem&biw=1366&bih=62](https://www.google.com/search?q=producer+consumer+problem&biw=1366&bih=622&tbm=isch&tbo=u&source=univ&sa=X&ei=SZitUue-FMnJrAelr4DwCw&ved=0CD4QsAQ)
* <http://www.tutorialspoint.com/javaexamples/thread_procon.htm>
* <http://cs.wellesley.edu/~ecom/lecture/ConsumerProducerRight.html>
* <http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html>
* <http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html>