Concurrent Algorithms and Data Structures – Theory Assignment 1

Parosh Aziz Abdulla

November 24, 2024

Deadline: 2024–12–08.
Please, submit your solutions in .pdf format.

Problem 1 Consider the concurrent program P_1 shown in Fig 1. The program consists of two threads θ_1 and θ_2. The threads execute concurrently and share two global variables, x and y. Each thread is selected infinitely often to run during a given execution of the program. The threads have one local variable each, namely a and b, respectively. The values of all four variables are initially zero. Each of the threads executes an infinite loop. Let us consider the loop run by θ_1. During each iteration of the loop, the thread tries to enter its critical section $\texttt{CS}_1$. It will first set the shared variable x to one. Setting x to one indicates to θ_2 that θ_1 is about to enter its critical section. It then reads the value of the shared variable y to check whether θ_2 is also about to enter its critical section $\texttt{CS}_2$. It stores that value in a and then executes its `if`-statement. If θ_2 has not declared that it is about to enter its own critical section, thread θ_1 enters its critical section.

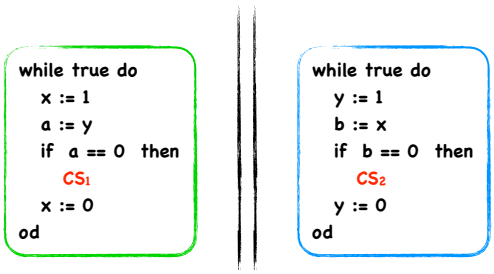

Figure 1: The concurrent program P_1.

Eventually, it leaves its critical section, sets the shared variable x to zero and repeats the loop.
Assume that the `if`-statement has a *spin-lock-like* behavior. For instance, θ_1 keeps waiting until `a` equals zero. If the condition is never satisfied, θ_1 will wait forever.
We can explain the behavior of θ_2 analogously.
Answer the following questions about P_1. For each of the three items, do the following: (i) Start with either the word *yes*, or the word *no*; then (ii) motivate your answer in at most five lines.

- *Safety property*: Is it possible to reach a state of P_1 where both θ_1 and θ_2 are in their critical sections, i.e., θ_1 is at $\texttt{CS}_1$ and θ_2 is at $\texttt{CS}_2$?
- *Deadlock*: Is it possible that P_1 enters a deadlock state, i.e., a state where neither of the threads can continue its execution?
- *Liveness property*: Is it possible that one of the threads starves, i.e., fails to enter its critical section, while the other thread enters its critical section repeatedly.
- Repeat the above three tasks assuming that the `if`-statement has non-spin-lock behavior, i.e., it jumps to the following statement if its condition is invalid.

Problem 2 We define a generalized version of the `Queue` abstract data type, which we call the *double-ended-queue*, denoted `DEQ`. As the name indicates, the queue has two active ends, *head* and *tail*.
The `enqueue` operation has an input value. The input value is the value to be inserted at the head or the tail of `DEQ`. The operation has a Boolean output value which tells us whether the operation was successful. An `enqueue` operation, with a key k, first checks the head. If the head's key does not equal k, it inserts k at the head. If *not*, it checks the tail. If the tail's key does not equal k, it inserts the element at the tail. We consider the operation successful in both scenarios and return the value `true`. If the keys at both the head and the tail equal k, then the operation fails, and we return the value `false`.
The `dequeue` operation also has an input value. The operation similarly checks the head and the tail in that order and fails if the searched key is neither at the head nor the tail. If it finds the key at the head, it removes it and outputs the value `head`. If the element is not at the head but at the

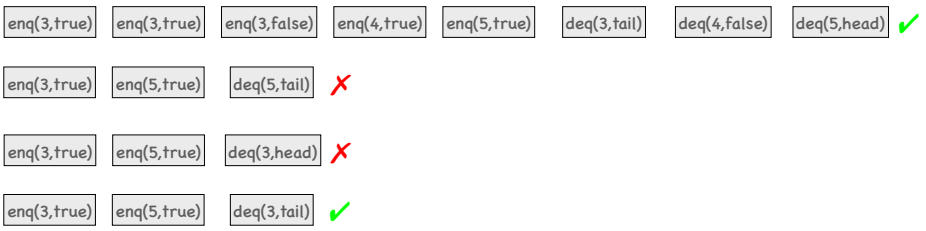

Figure 2: The generalized queue.

tail, it removes the key and outputs the value `tail`. Otherwise, it returns the value `false`.

Notice that (i) the possible output (return) values of an `enqueue` operation is either `true` or `false`; and (ii) the possible return values of an `dequeue` operation is either `head`, `tail`, or `false`.

Fig. 2 shows some sequences of operations that belong /do not belong to the abstract data type.

Your task is to define the abstract data type `DEQ`, i.e., the set S of states, the initial state s_{init}, the set M of method names, and the set R of rules.

Remark The terms "key" and "value" above are interchangeable.