Oblig 1 IN5170

By Finn Eivind Aasen, finneaa@ifi.uio.no

Task 1:

$\mathscr{V}$: statement → variable set: set of global variables in a statement (also for expressions)

$\mathscr{W}$: statement → variable set set of global write–variables

---

```
find (d) {
        i := head ;      # variable i is local to
                         # the current method instance
        while ( i ≠ null and i.val ≠ d ) { i := i.next ; }
        # return i
 }
```

$\mathscr{V}$ set for find:

{head, i.val, i.next}

$\mathscr{W}$ set for find:

Is empty

```
insert ( new ) {
        if ( tail = null ) { # empty list
                head := new ;
        } else {
                tail.next := new ;
        }
        tail := new
}
```

$\mathscr{V}$ set for insert:

{tail}

$\mathscr{W}$ set for insert:

{tail.next, head, tail}

```
bool del_lock := true # a global variable

delfront () {
        i f ( head ≠ null ) { # list not empty
                i f ( head = t a i l ) { t a i l := n u l l ; } # only 1 elem. in list
                        head := head.next ; }
}
```

$\mathscr{V}$ set for delfront:

{head.next, tail, head}

$\mathscr{W}$ set for delfront:

{del_lock, tail, head}

Task 2:

Now assume that several processes access the linked list. Consider all six pairs combining two of three routines given above.
(a) Which combinations of routines can be executed in parallel without interference?

- Two finds can be executed in parallel because they only write to local variables so no changes are done to the linked list that would impact the execution of the other routine.

- Find and insert can be executed in parallel because find makes no changes to shared variables and the execution of both routines will always be a result you could get from a sequential solution.

- Find and delfront can be executed in parallel because find makes no changes to shared variables and the execution of both routines will always be a result you could get from a sequential solution.

(b) Which combinations of routines must be executed one at a time?

- Two insert routines would need to be executed one at a time because if both are executed at the same time then one might overwrite the changes from one of the executions leading to a result we would not be able to get from an sequential execution. From sequential solutions would the list for example be ...AB or ...BA from two insert routines(one inserting element A and the other inserting element B), but if both inserts are done at the same time in a parallel execution we might end up with results like ...A or ...B as both did not know about each other and the last to complete overwrite the other routine.

- Two delfront routines would need to be executed one at a time because if both are executed at the same time then one might overlap the changes from one of the executions leading to a result we would not be able to get from an sequential execution. The sequential result of 2 delfront routines would lead to the removal of 2 elements from the front of the queue or an empty queue. But in parallel if 2 threads make changes to head at the same time then we can “lose” one of the delfronts because both change the same shared head to point at head.next leading to only removing 1 element from the queue.

- Insert and delfront would need to be executed one at a time because if this sequence happens to the shared variable during an parallel execution when we would get a result not obtainable from a sequential solution:
    - Insert: head := new
    - Delfront: head := head.next
    - Insert: tail := new

The result would be head would be null and tail would be new leading to that the find and delfront routine would be unusable and head would never be assigned a value.

Task 3:

Add await statements to program the synchronization code in the routines such that non-interference of concurrent executions is enforced. Try to make your atomic actions as small as possible, and do not delay a routine unnecessarily

---

```
bool insert_lock := false        # a global variable
bool del_lock := false           # a global variable

find (d) {
        i := head ;      # variable i is local to
                         # the current method instance
        while ( i ≠ null and i.val ≠ d ) { i := i.next ; }
        # return i
}

insert ( new ) {
                ⟨await (insert_lock = false and del_lock = false){ # await to check if we are
                                                                   #already running an insert and/or an
                                                                   #delfront
                insert_lock := true}⟩    #start CS
                if ( tail = null ) { # empty list
                        head := new ;
                } else {
                        tail.next := new ;
                }
                tail := new
                insert_lock := false  #end CS
}

delfront () {
        ⟨await (del_lock = false){       # await to check if we are already running an delfront
                del_lock := true}⟩       #start CS
        i f ( head ≠ null ) { # list not empty
                ⟨await (insert_lock = false)⟩# await to check if we are already running an insert
                i f ( head = t a i l ) { t a i l := n u l l ; } # only 1 elem. in list
                        head := head . next ; }
        del_lock := false;       #end CS
}
```
