

Note

Problem 2 uses some terminology concerning proof rules (invariance induction rule, weak-fair rule, etc.), which are explained in [proofrules.pdf](#).

1. Distributed laxlock service

Informal description: A “laxlock” is like a lock except that it can be held simultaneously by upto N threads, where N is a positive integer parameter. So you can view it as a collection of N “tokens”. A thread calls `acq()` to acquire a token and `rel()` to release the token it holds. For convenience, we say a thread is “hungry” if it is in `acq()`, “eating” if it holds a permit, and “thinking” otherwise.

A distributed laxlock is one that can be accessed at different addresses. A thread can acquire a token from one address and release it at another.

Service program: Here is a distributed laxlock service with addresses 0 and 1. At each address j , there are input functions $v_j.acq()$ and $v_j.rel()$, where v_j is the sid of the local access system. A skeleton is provided below:

```
service Laxlock(int N) {
  ic {N ≥ 1}
  Set eating ← set()
  Map v ← map([0,sid()], [1,sid()])
  return v

  v[0..1 j].acq() {
    ic {mytid not in eating}
    oc {eating.size < N}
    eating.add(mytid)
    return
  }

  v[0..1 j].rel() {
    ic {mytid in eating}
    eating.remove(mytid)
    oc {true}
    return
  }

  progress assumption { // weak fairness at each address
    forall(j in 0..1:
      (thread t in v[j].acq) leads-to (t not in v[j].acq) or eating.size = N)
    }
}
```

To do:

Write down the inverse of the above service, say `LaxLockInverse(N,v)`.

2. Progress analysis of a distributed laxlock program

Program LaxDist(N) below is the same program as in hw2, except that eating is not present, adjust(.) is not hidden, and Lax assumes strong fairness. You have to decide whether or not it implements the distributed laxlock service Laxlock(N).

```

program LaxDist(N) {
  inputs {v[0].adjust(.), v[1].adjust(.)}
  Bag_eating ← []
  Map v
  v[0] ← startSystem(Lax(0))
  v[1] ← startSystem(Lax(1))
  w ← startSystem(Adjuster(v[0], v[1]))
} // LaxDist

```

```

program Adjuster(Sid v0, Sid v1) {
  int bal ← 0
  int y ← 0
  t ← startThread(f())

  function f():
    while (true)
      [bal, y] ← v0.adjust(bal, y)
      [bal, y] ← v1.adjust(bal, y)
      // sleep a bit

  atomicity assumption { }
  progress assumption {wfair for all threads}
} // Adjuster

```

```

program Lax(0..1 i) {
  int x ← if (i=0) N else 0
  return mysid

  input mysid.acq():
    ia {true}
    • await (x>0)
      x--
      return

  input mysid.rel():
    ia {true}
    • await (true)
      x++
      return

  input mysid.adjust(int bal, int y):
    • await (true)
      x ← x+bal
      if (x ≥ y+2)
        tmp ← (x-y)/2 // integer division
        x ← x-tmp
      return [tmp, x]
    else
      return [0, x]

  atomicity assumption {await}
  progress assumption {sfair for all threads}
} // Lax

```

```

program Z(int N) {
  ic {N ≥ 1}
  inputs(); outputs()
  Map v
  v[0] ← startSystem(Lax(0))
  v[1] ← startSystem(Lax(1))
  w ← startSystem(Adjuster(v[0], v[1]))
  si ← startSystem(LaxLockInverse(N, v))
}
// v ← startSystem(LaxDist(N)) expanded
// " " " " " "
// " " " " " "
// " " " " " "
// service inverse

```

To do:

Part a Write down invariant assertions for Z corresponding to the safety condition of $\text{LaxDist}(N)$ implements $\text{LaxLock}(N)$. (I want the assertions after accounting for Z 's effective atomicity.)

Part b Does Z satisfy these assertions, say $\text{Inv } P$.

If you answer yes, give a predicate that implies P and satisfies the invariance induction rule (give just the predicate; no need for details of how it satisfies the invariance rule).

If you answer no, give a finite evolution that ends in a state where P does not hold.

Part c Does Z satisfy the progress condition of the service inverse (si).

If you answer yes, give one or more leads-to assertions (each of the form X leads-to Y) such that

- each leads-to assertion holds via the weak-fair or strong-fair rule, and
- P leads-to Q follows from closure of the leads-to assertions.

If you answer no, give an allowed evolution of Z that satisfies Z 's progress assumption and does not satisfy P leads-to Q .