Question 1 *Process algebra[5 points]*

We saw a large number of algebraic laws of CCS this Wednesday. Here are some more proposed laws. Briefly discuss the merits of each proposed law, and whether you think it should be accepted. If you think it should be accepted, give an informal intution for why; if you think not, give a concrete example where it has absurd consequences.

1.1 $P \mid P = P$

Answer:

I think so. $\mid$ means parallel. Suppose $Clock = P \mid P$, $Clock = P.P + P.P + P$, This is true when the starting point and the key point are the same, that is, P is self-cycling.

1.2 $P \setminus b \setminus b = P \setminus b$

Answer:

I think so, because $P \setminus b$ means the action b and $\bar{b}$may not be executes in P, $P \setminus b \setminus b$ means the action b and $\bar{b}$may not be executes in $P \setminus b$, both of them have same meanings.

1.3 $(a.P) \mid (a.Q) = a.(P \mid Q)$

Answer:

I think it is not. Because, $a.(P \mid Q)$ means that process a must finish before process P or Q, but $(a.P) \mid (a.Q)$ can existed that $a.P.a.Q$, which process a do after P or Q.

Question 2 *Ricart-Agrawala Algorithm[4 points]*

Pseudocode of the Ricart-Agrawala Algorithm is:

Algorithm 10.2: Ricart-Agrawala algorithm

```
                integer myNum ← 0
                set of node IDs deferred ← empty set
                integer highestNum ← 0
                boolean requestCS ← false
     Main
     loop forever
p1:    non-critical section
p2:    requestCS ← true
p3:    myNum ← highestNum + 1
p4:    for all other nodes N
p5:       send(request, N, myID, myNum)
p6:    await reply's from all other nodes
p7:    critical section
p8:    requestCS ← false
p9:    for all nodes N in deferred
p10:      remove N from deferred
p11:      send(reply, N, myID)
     Receive
     integer source, requestedNum
     loop forever
p1:    receive(request, source, requestedNum)
p2:    highestNum ← max(highestNum, requestedNum)
p3:    if not requestCS or requestedNum ≪ myNum
p4:       send(reply, source, myID)
p5:    else add source to deferred
```

2.1 Suppose that we exchanged the lines $p8$ and $p9 - p11$ in Main, i.e. request $CS \leftarrow false$ now executes after the for loop (instead of executing before the for loop). Suppose furthermore that it's possible for **Receive** to preempt **Main** at these locations. Provide an example that illustrates why the modified algorithm is no longer correct.

Answer:

Suppose the exchanging $p8$ and $p9 - p11$, when a process p is finish and get out from critical section, it send message to all node N in deferred, during this period, the Receive part of P check the requestCS of itself is true, and it will record the source into defer. Suppose the receiver accept the request and put them in defer, the loop may go on forever, and causing that process P may not apply to enter the Critical section again.

2.2 In **Receive**, can the statement

p2: highestNum ← max(highestNum, requestNum) be replaced by

p2: highestNum ← requestNum? Why? Justify your answer and provide an example.

Answer:

Suppose there is a request message sent to P which be delay by some ways, during this time, more request which have high marks had been accepted by P. If the grogram only copy request number as the highestNum, it may choose a less number, and it may appear two or more request message with same requestNum which send by different process, receive part $p3$ find the requestNum is not smaller than my number, and record them, it may lead that some process cannot get reply from all node and cannot go into critical section.

Question 3 *Token-Passing[4 points]*

Pseudocode of the Ricart-Agrawala Token-passing Algorithm is:

Algorithm 10.3: Ricart-Agrawala token-passing algorithm

```
        boolean haveToken ← true in node 0, false in others
        integer array[NODES] requested ← [0,...,0]
        integer array[NODES] granted ← [0,...,0]
        integer myNum ← 0
        boolean inCS ← false

sendToken
    if exists N such that requested[N] > granted[N]
        for some such N
            send(token, N, granted)
            haveToken ← false
```

Main

```
    loop forever
p1:     non-critical section
p2:     if not haveToken
p3:         myNum ← myNum + 1
p4:         for all other nodes N
p5:             send(request, N, myID, myNum)
p6:         receive(token, granted)
p7:         haveToken ← true
p8:     inCS ← true
p9:     critical section
p10:    granted[myID] ← myNum
p11:    inCS ← false
p12:    sendToken
```

Receive

```
    integer source, reqNum
    loop forever
p13:    receive(request, source, reqNum)
p14:    requested[source] ← max(requested[source], reqNum)
p15:    if haveToken and not inCS
p16:        sendToken
```

3.1 In node i, can the value of **requested**$[j]$ be less than the value of **granted**$[j]$ for $j \neq i$? Why? Justify your answer.

Answer:

Yes. Suppose a process P send a message for all of the node and get the token, one of the other process Q have not get the request message or message had been lost, when this process Q get the token, it will appear that the request times of the P is less than the granted of P.

3.2 In node i, can the value of **requested**$[j]$ be greater than the value of **granted**$[j]$ for $j \neq i$? Why? Justify your answer.

Answer:

Yes. When there are more than one process p, q... which send request message, the process which had token need to choose one of it, suppose it is p and send the token. During this time, the process q send the second or more request and get by the process p which get the token, the request times of q will larger than the granted time of p.