Section 4.1

(s, T) = (E,B,T)

Configuration C=(s,T)

Definition 1:

A partial trace or a sub trace is a sequence of conﬁgurations (s0, T0),...,(sn,Tn) such that for any two successive conﬁgurations we have (sj,Tj) (sj+1,Tj+1).

A complete trace ends with done , error or deadlock

Definiton 2:

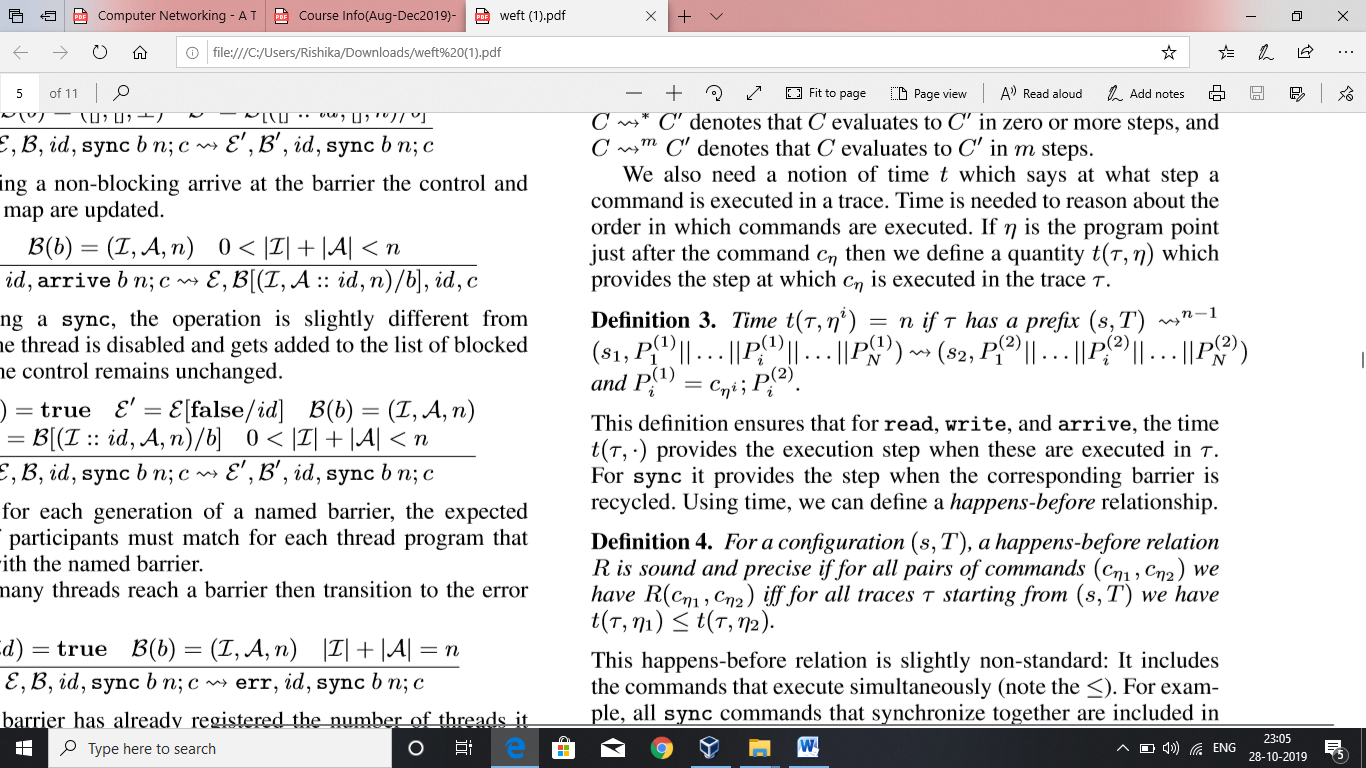
A (complete) trace τ starting from a conﬁguration (s,T)is a subtrace (s,T),...,(s0,done),or(s,T),...,(err,T0), or (s,T),...,(s0,T0) where T0 6= done and no rule is applicable (deadlock).

C->C’ – c evaluates to C’ in one step

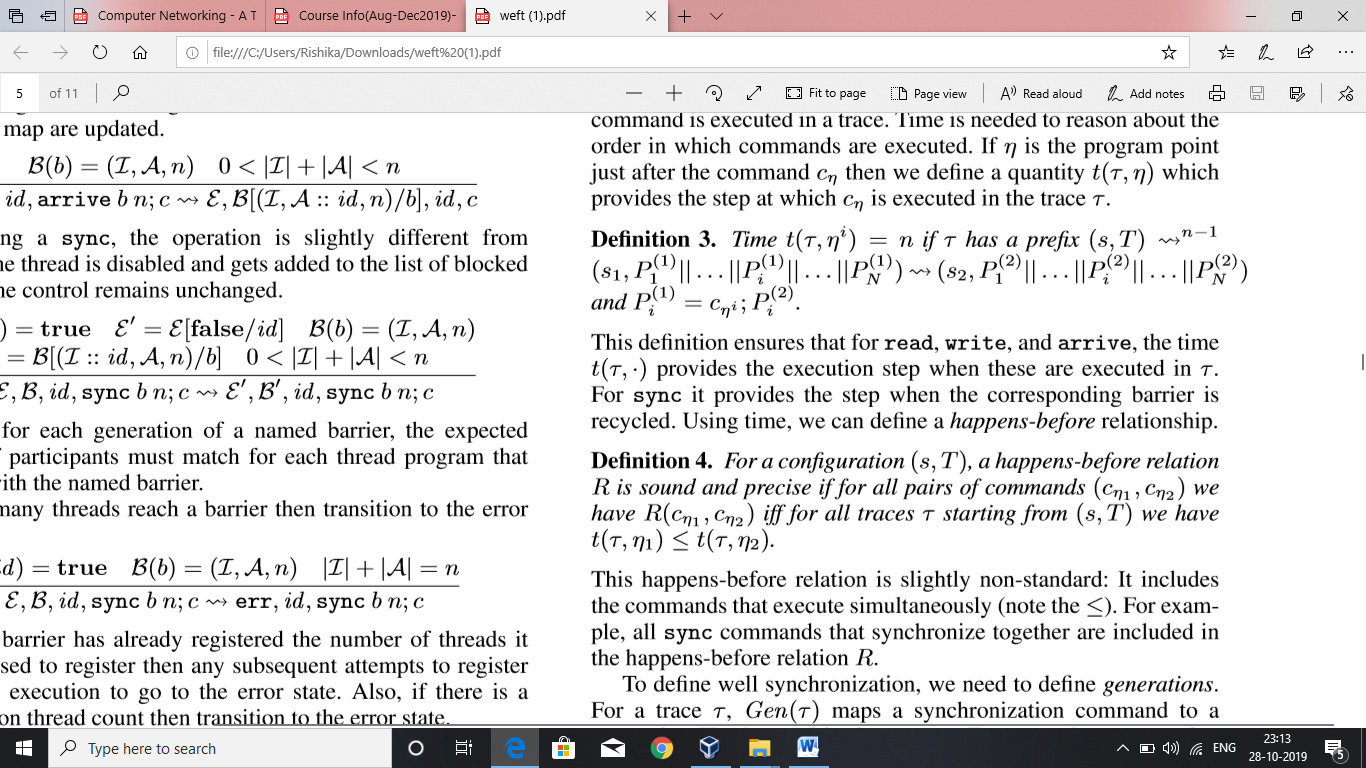
C->\*C’ – C evaluates to C’ in zero or more steps

C->mC\* C evaluates to C’ in m steps

If η is the program point just after the command cη then we deﬁne a quantity t(τ,η) which provides the step at which cη is executed in the trace τ.



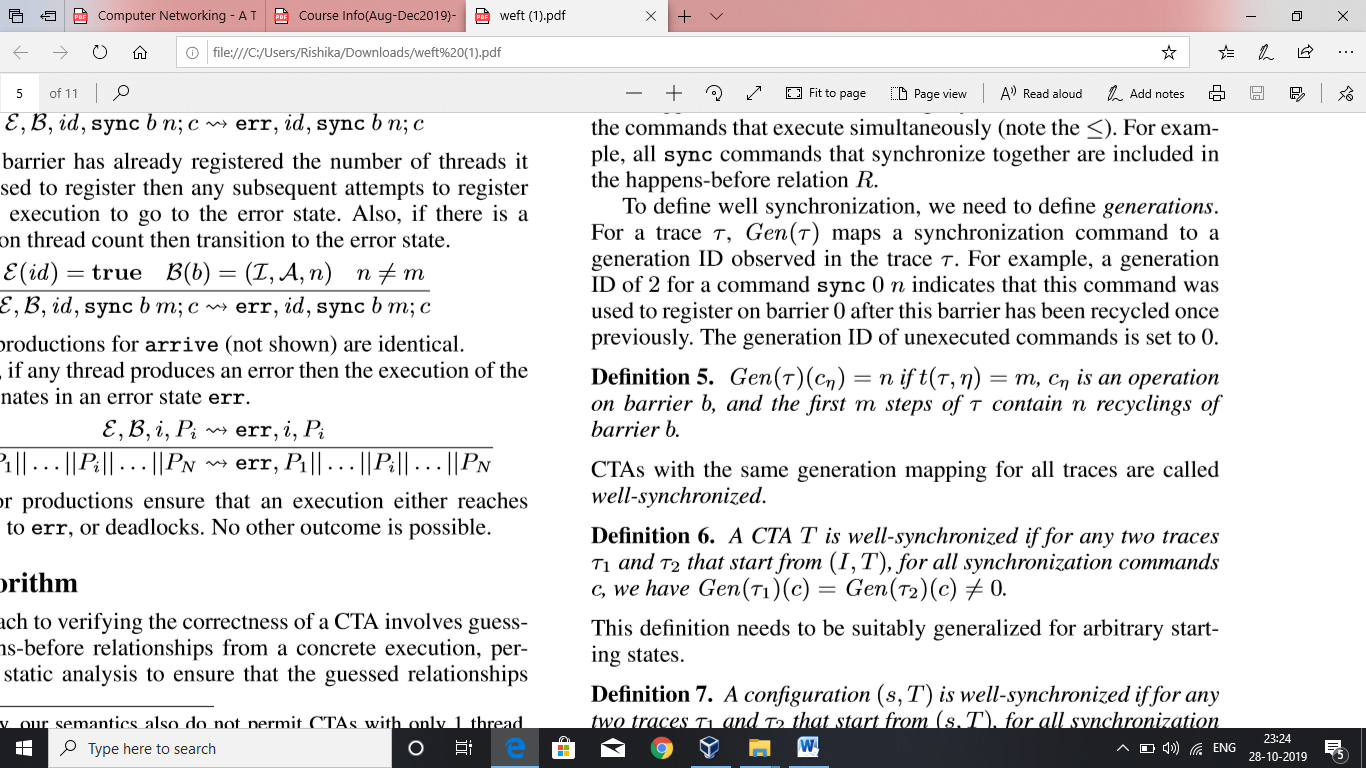
Time t(T,.) provides the execution step when read , write and arrive are executed in T. Sync , the step is the corresponding barrier recycled . Time helps us too figure out happens before relationships.



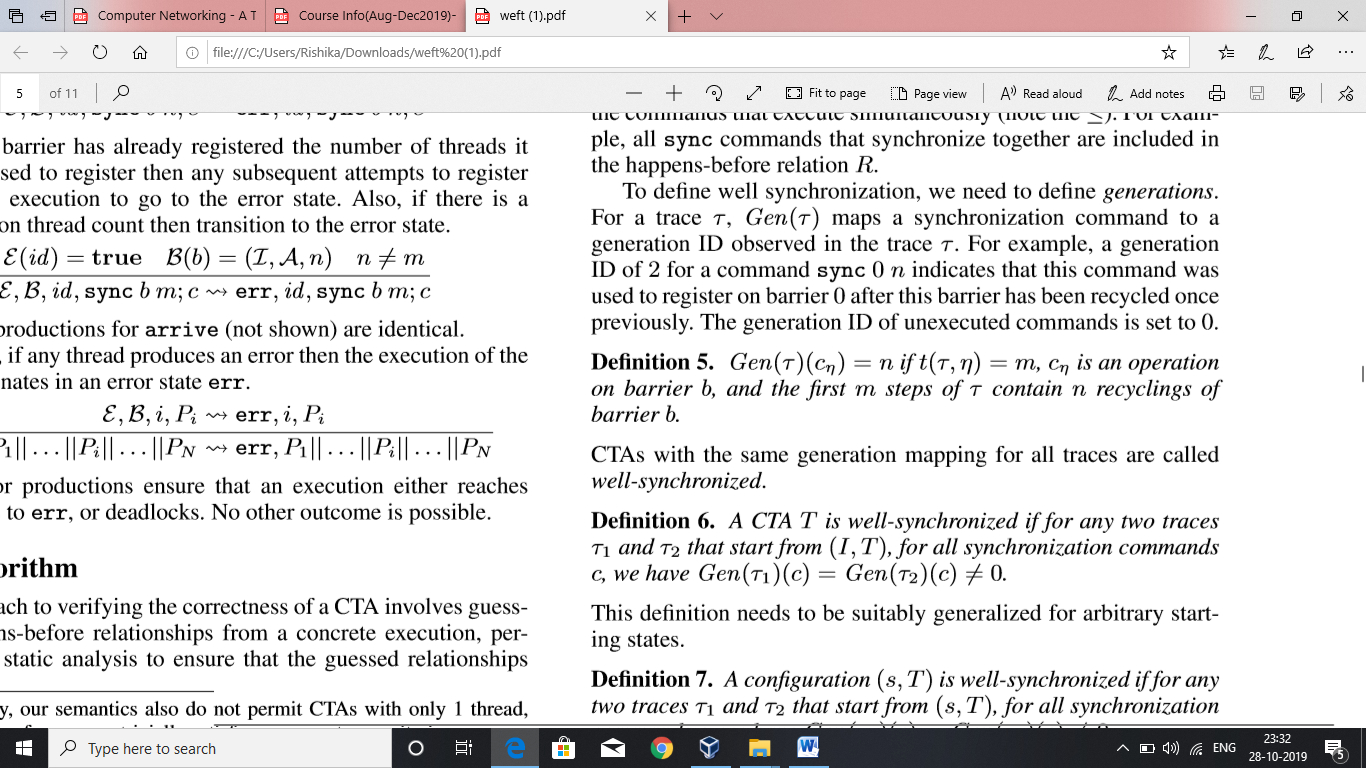
Happens before relationship includes commands that execute simultaneously i. e sync commands are included in happens before relationship

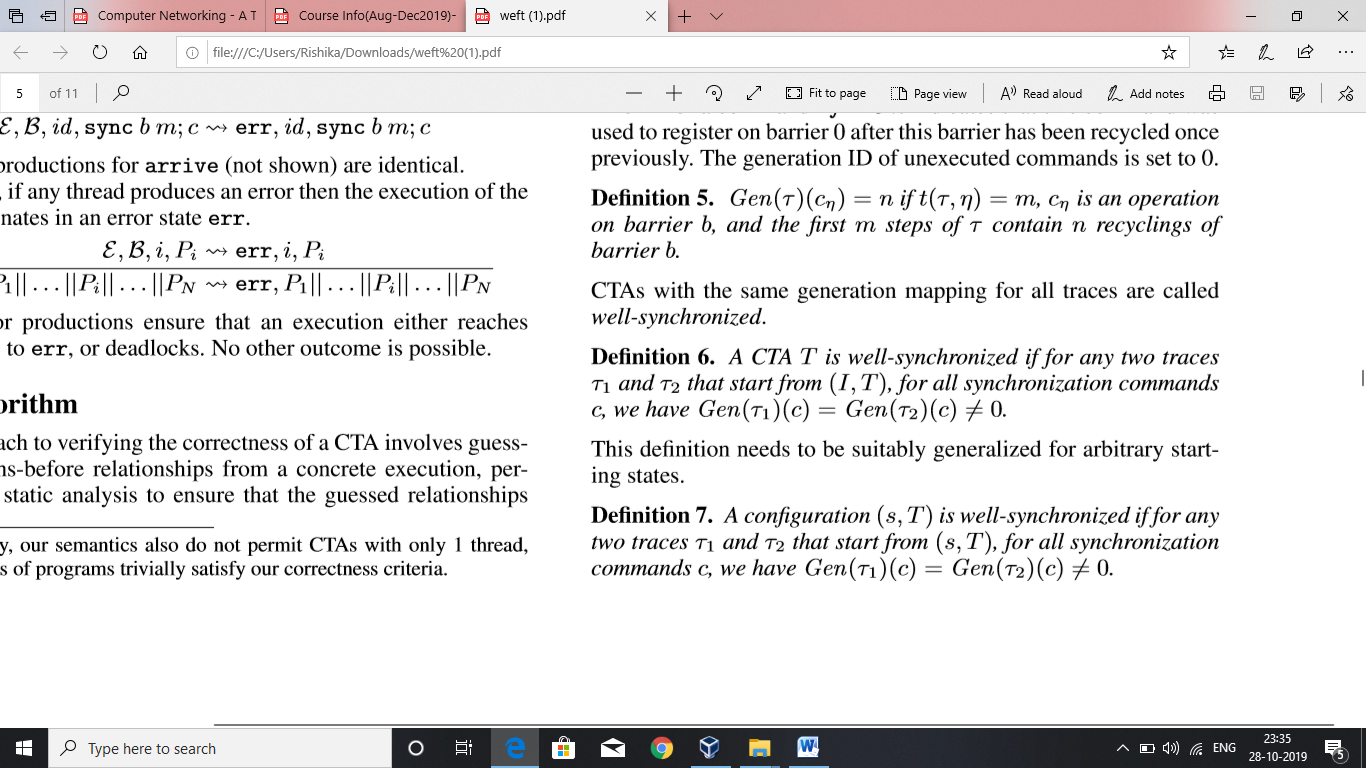
Gen(T) maps a synchronization command to a generation ID . Example : A gen Id of 2 for command sync 0 n indicates that the command was used to register on barrier 0 after the barrier is recycled.

The gen ID of unexecuted commands is set to 0.



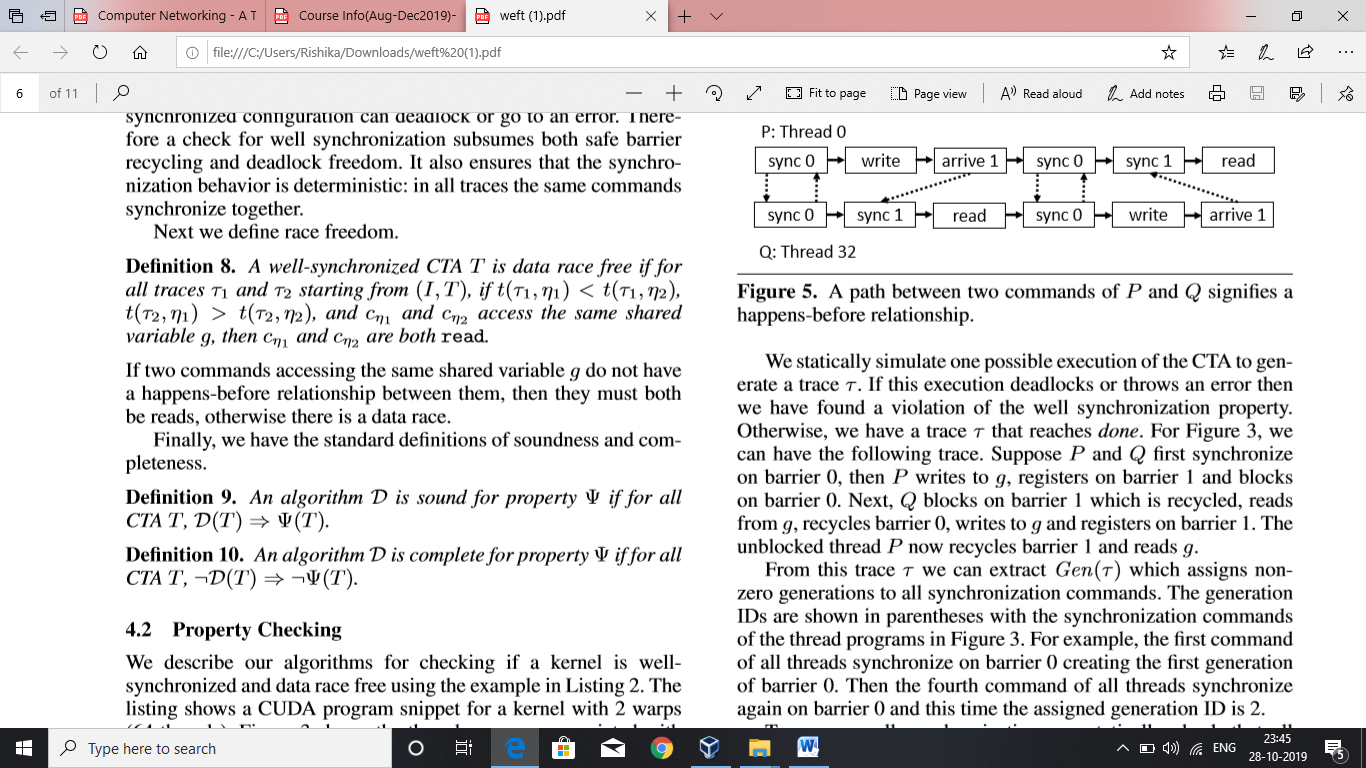
If the generation mapping for all traces in the CTA are same , then it is said to be well-synchronized.



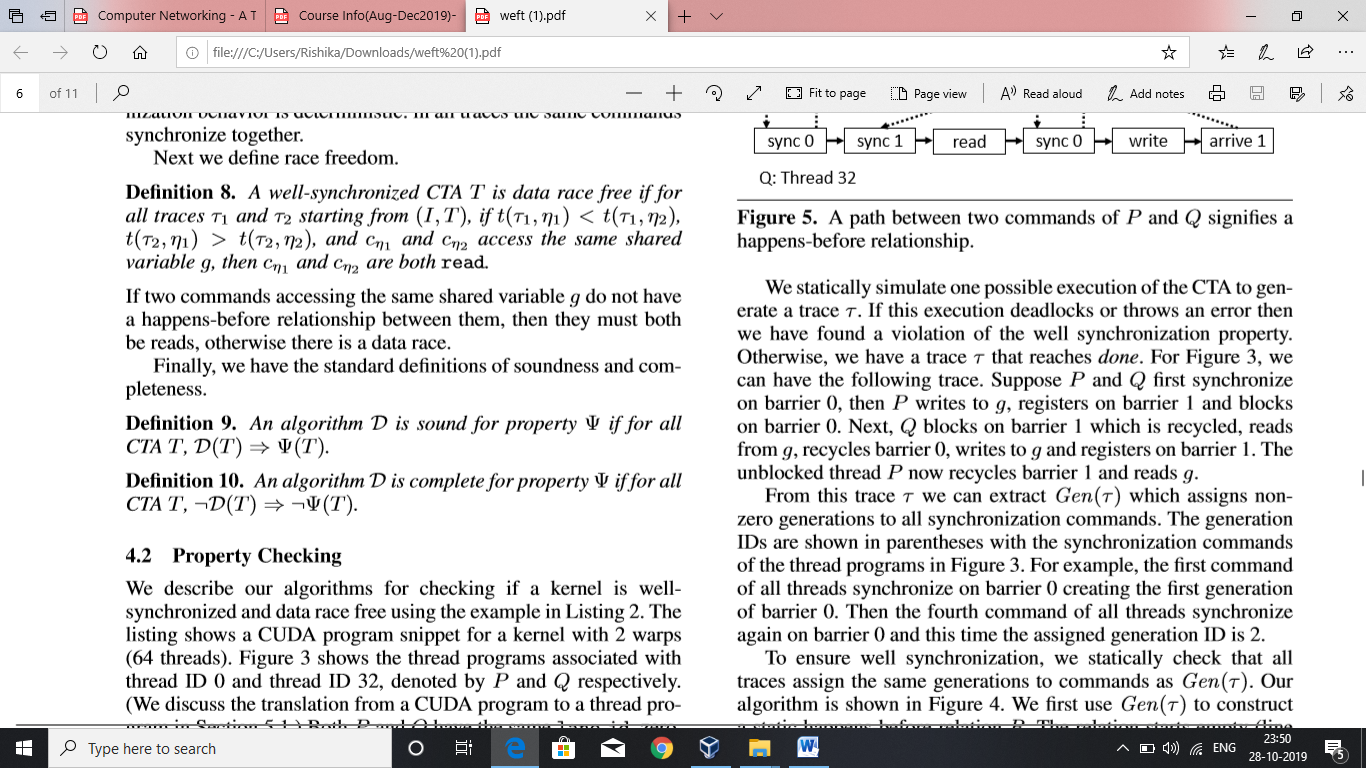


The non-zero check takes care that no trace of a well-synchronized configuration can deadlock or go to an error .The check also considers safe barrier recycling and deadlock freedom. It also ensures that same commands in traces synchronize together.

Race freedom :

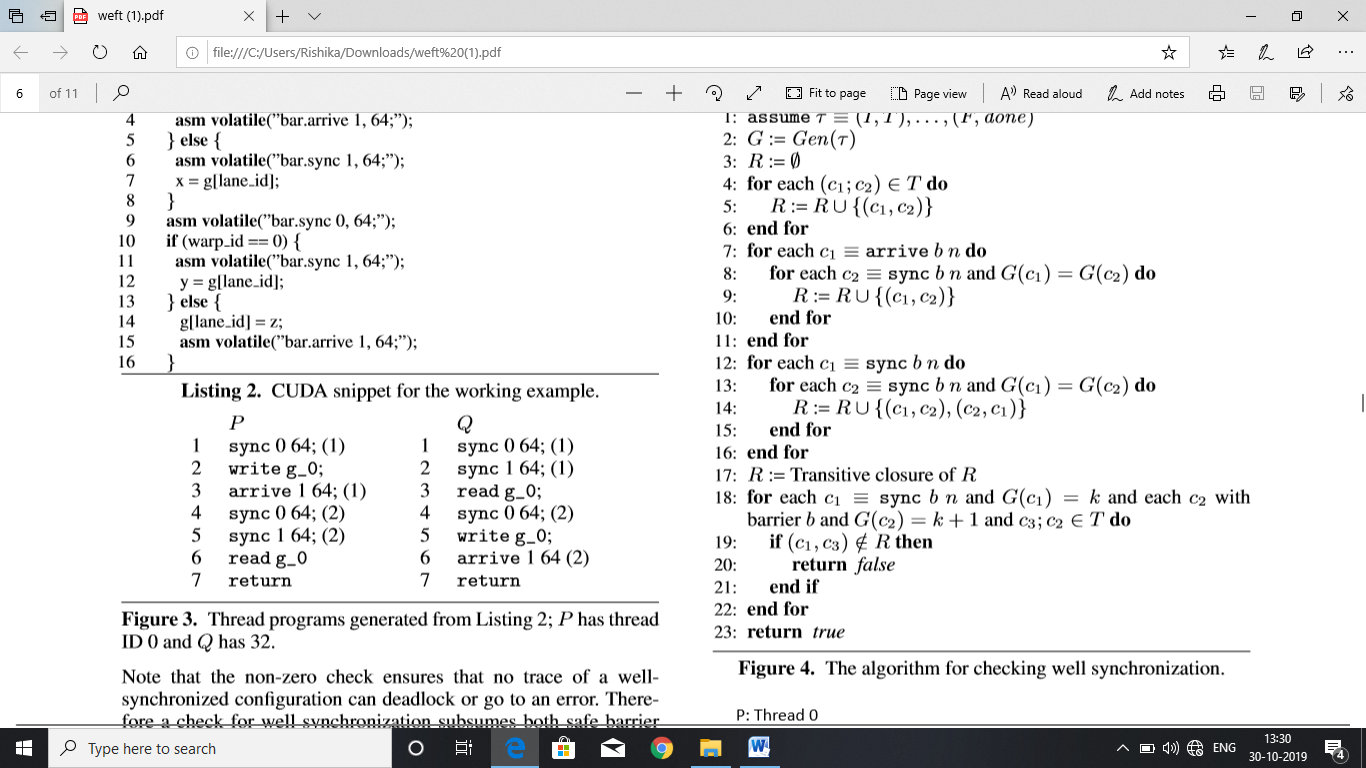


If Two commands access the same shared variable g and they do not have a happens before relationship between them , then both should be reads .If there is any write , they get into dataraces.



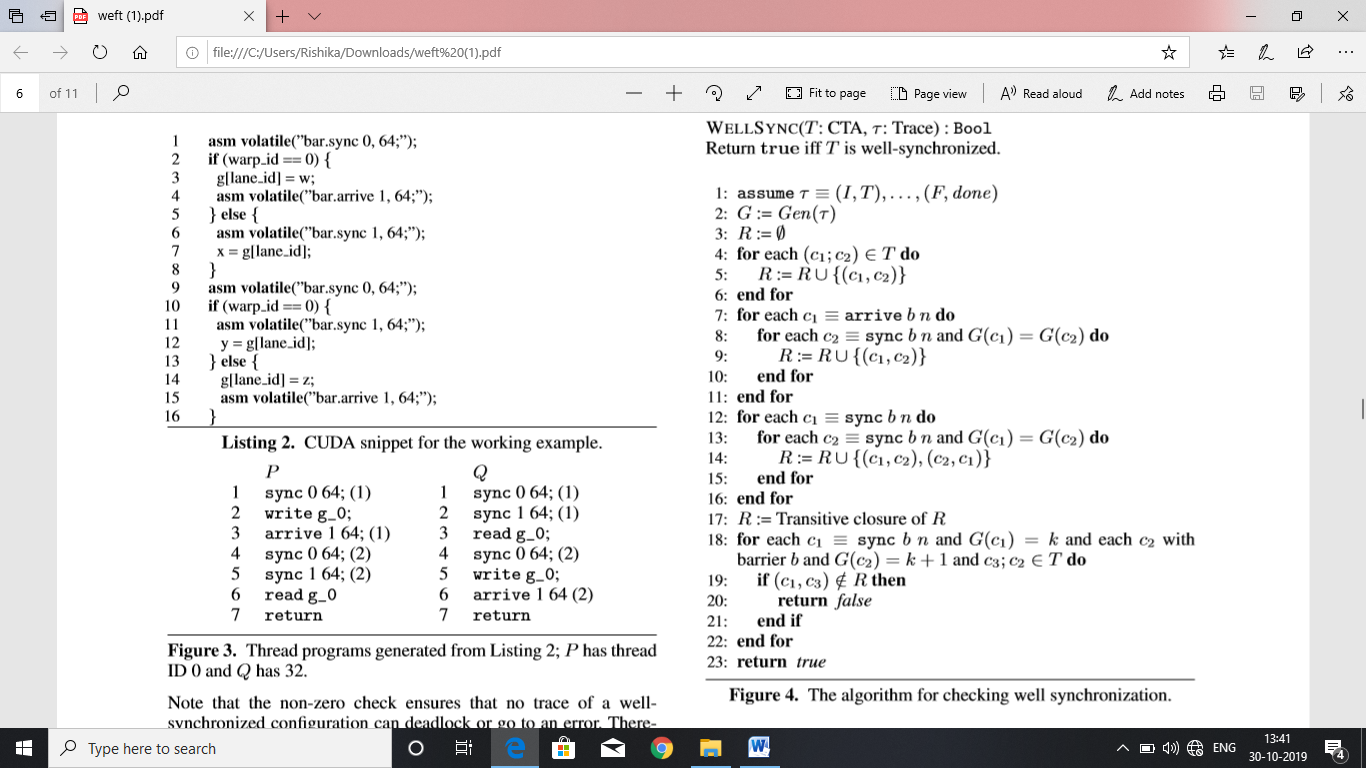
Section 4.2 – Property Checking

We consider 2 threads with Id’s as 0 and 32 denoted by P and Q. The lane\_id is given by threadId % 32 which is 0 , same for both hence may involve in data races.If j is constant , each location g[j] is treated as a separate variable in thread programs.

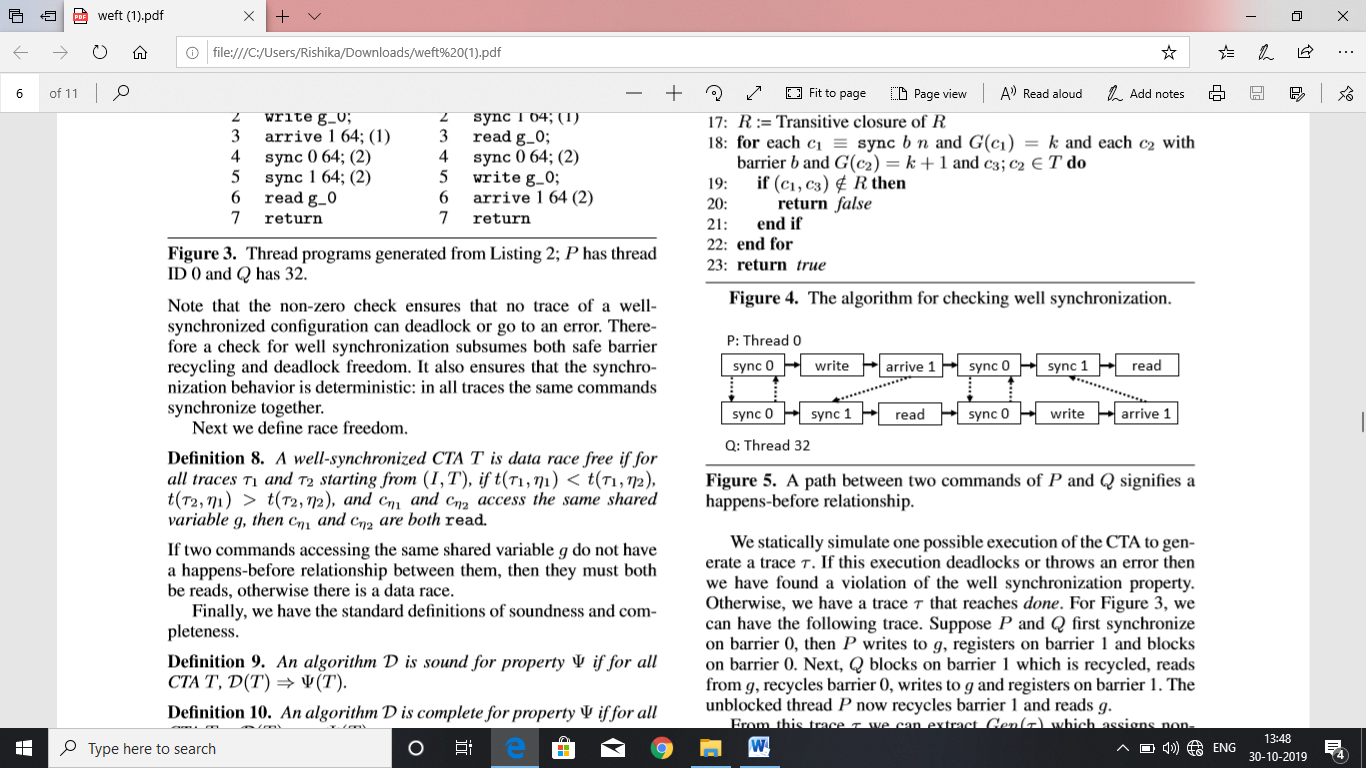


P and Q synchronize on barrier 0, then P writes to g, registers on barrier 1 and blocks on barrier 0. Next, Q blocks on barrier 1 which is recycled, reads from g, recycles barrier0,writes to g and registers on barrier1.The unblocked thread P now recycles barrier 1 and reads g.

In the first command P and Q sync on barrier 0 that is the first generation of barrier 0 , and similarly in the fourth command P and Q sync on barrier 0 creating the second generation of barrier 0.



Gen(T) is used to construct happens before relationship R , initially it is empty , we add successful commands to R.



The edges represent tuples in R.

In lines 7, 8 ,9 if c1 is an arrive and c2 is a sync and they are in the same generation , then add (c1,c2) to R .

In lines 12,13,14 . If c1 s sync and c2 is sync in the same generation then add (c1,c2) and (c2,c1) to R.

The cycles represent the happens before relationship includes command which execute simultaneously. The transitive closure of R is used to comput happens before relationship.

lines 18 to 23 check if there exists a relationship between successive generation of same barrier.

The successive generations are separated by a happens-before-relationship and therefore CTA is well synchronized.

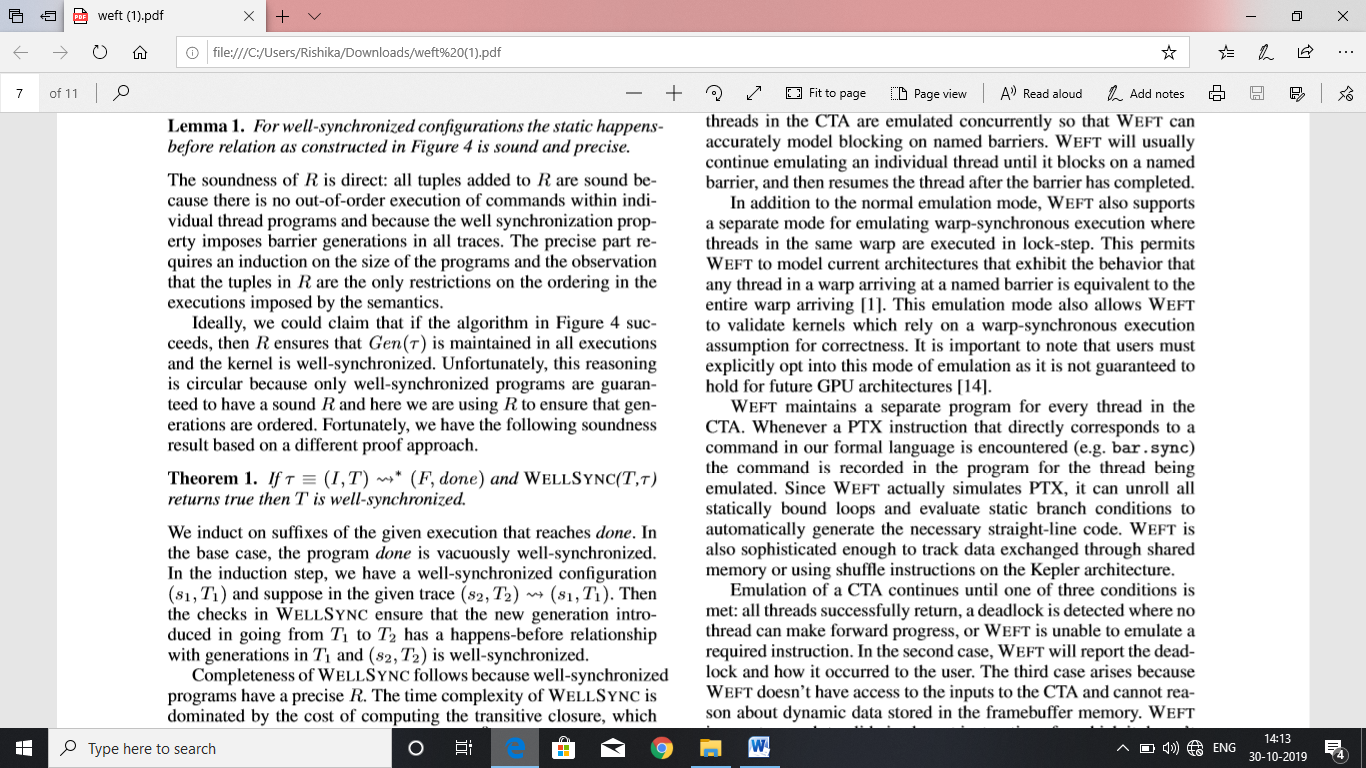
Lemma 1:

For well-synchronized conﬁgurations the static happens before relation as constructed in Figure 4 is sound and precise.

All tuples added to R is sound because all commands are executed in the order within thread programs.

R ensures that Gen(T) is maintained in all executions and kernel is well-synchronized.

Theorem 1:



we have a well-synchronized conﬁguration (s1,T1) and suppose in the given trace (s2,T2)-> (s1,T1). Then the checks in WELLSYNC ensure that the new generation introduced in going from T1 to T2 has a happens-before relationship with generations in T1 and (s2,T2) is well-synchronized.

The time complexity to compute transitive closure is O(n3) in worst case.Two commands c1 and c2 access the same shared memory with atleast one write , a race is occurred if (c1,c2) and (c2,c1) does not belong to R. We observe that there is a path between any two access to shared memory and P and Q are race free.

Section -5.1

Weft takes PTX assembly as input and translates it into formal language by emulating one CTA (probably the first one) until it terminates, deadlocks or encounters an error. Emulation is done by initializing CTA Id and thread IDs. All threads are emulated concurrently in CTA .Weft will continue emulating an individual thread until it is blocked on a named barrier, resumes after barrier is completed. Weft also has a code for warp-synchronous execution in which threads of same warp are executed in lock step. Through this the behaviour that any thread in warp arriving at named barrier is equivalent to enter warp arriving, can be modelled .It also helps to validate kernels which rely on warp synchronous execution assumption for correctness.

Weft has a separate program for every thread in CTA. After translation when a command is encountered, the command is recorded in the program for thread being emulated. Since WEFT simulates PTX, it can unroll all statically bound loops and evaluate static branch conditions to generate the straight line code. WEFT can track data exchanged through shared memory. Emulation of CTA continues until all threads successfully return, a deadlock is found where no thread can further progress or WEFT is unable to emulate a required instruction. When deadlock is found, in addition to reporting the deadlock it also tells how deadlock occurred to the user. WEFT doesn’t have access to the inputs of CTA and cannot reason about dynamic data stored in memory .If WEFT encounters a conditional statement ,shared memory access or barrier statement which cannot be emulated as the input depend on dynamic parameter of CTA , it gives an error saying that the kernel cannot be violated.