Algorithm:

Now how do we verify whether or not the CTA is veracious?.
It involves static analysis to ensure that the guessed relationships are reserved for all possible executions and using the same to prove CTA's correctness.

Now Understanding those meanings of those important terms i.e correctness properties.

Definition:

Definition 1(Execution traces) says that partial or subtrace made up of configurations such that for any two successive configurations.Complete trace of the configuration leads to either done or err, or a deadlock.

Definition 2

A (complete) trace τ starting from a configuration (s, T)is a subtrace (s, T), . . . , (s 0 , done), or(s, T), . . . ,(err, T0), or (s, T), . . . ,(s 0 , T0) where T 0 6= done and no rule is applicable (deadlock).
C->C' – C -->C' in one step
C->*C' – C--> C' in zero or more steps
C->^{m}C* C--> C' in m steps
η is the program point just after the command cη
so we then define a quantity which provides the step at which cη is executed while trace I.e t(τ,η).
Trace is τ.

Definition 3:

Time t(τ, ηi) = n if τ has a prefix (s, T) n−1 (s1, P(1) 1 || . . . ||P (1) i || . . . ||P (1) N) (s2, P(2) 1 || . . . ||P (2) i || . . . ||P (2) N) and P (1) i = cηi ; P (2) i .

This is for making sure that that read,write,and arrive ,the time facilitate execution step while a trace.

Definition 4:

For a configuration (s, T), a happens-before relation R is sound and precise if for all pairs of commands (cη1 , cη2) we have R(cη1 , cη2) iff for all traces τ starting from (s, T) we have t(τ, η1) ≤ t(τ, η2).

When relation is slightly non-standard,this happens before it . Now consider Commands that execute simultaneously (note the ≤). For example, all sync commands that synchronize together are included in the happens-before relation R.
For example, a generation ID of 2 for a command sync 0 n indicates that this command was used to register on barrier 0 after this barrier has been recycled once previously.

Generation ID -->unexecuted commands is set to 0.

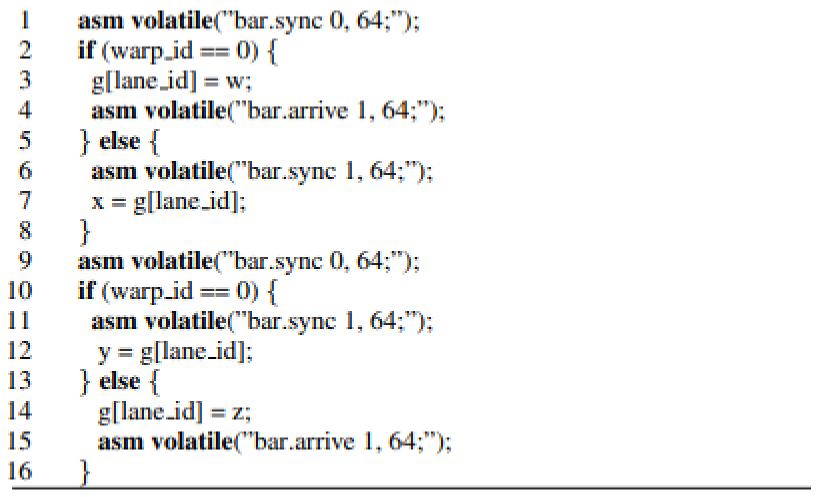
<u>Definition 5:</u>

Gen(τ)(cη) = n if t(τ, η) = m, cη is an operation on barrier b, and the first m steps of τ contain n recyclings of barrier b.
well-synchronized: CTAs with the same generation mapping for all traces .

<u>Definition 6:</u>

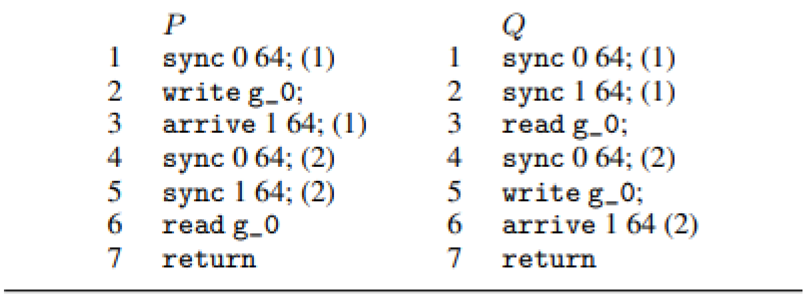
A CTA T is well-synchronized if for any two traces τ1 and τ2 that start from (I, T), for all synchronization commands c, we have Gen(τ1)(c) = Gen(τ2)(c) 6= 0.

This is generalized for arbitrary starting states.
<u>Definition 7:</u>
A configuration (s, T) is well-synchronized if for any two traces τ1 and τ2 that start from (s, T), for all synchronization commands c, we have Gen(τ1)(c) = Gen(τ2)(c) 6= 0.



```
asm volatile("bar.sync 0, 64;");
if (warp_id == 0) {
  g[lane_id] = w;
  asm volatile("bar.arrive 1, 64;");
} else {
  asm volatile("bar.sync 1, 64;");
  x = g[lane_id];
}
asm volatile("bar.sync 0, 64;");
if (warp_id == 0) {
  asm volatile("bar.sync 1, 64;");
   y = g[lane_id];
} else {
   g[lane_id] = z;
   asm volatile("bar.arrive 1, 64;");
}
```

Listing 2. CUDA snippet for the working example.



```
P                        Q
sync 0 64; (1)           sync 0 64; (1)
write g_0;               sync 1 64; (1)
arrive 1 64; (1)         read g_0;
sync 0 64; (2)           sync 0 64; (2)
sync 1 64; (2)           write g_0;
read g_0                 arrive 1 64 (2)
return                   return
```

Figure 3. Thread programs generated from Listing 2; P has thread ID 0 and Q has 32.

The non-zero makes sure that no trace of a wellsynchronized configuration can deadlock or go to an error. Therefore a check for well synchronization considers or includes both safe barrier recycling and deadlock freedom. It also ensures that the synchronization behavior is deterministic: in all traces the same commands synchronize together.

Definition 8:

A well-synchronized CTA T is data race free if for all traces τ1 and τ2 starting from (I, T), if t(τ1, η1) < t(τ1, η2), t(τ2, η1) > t(τ2, η2), and cη1 and cη2 access the same shared variable g, then cη1 and cη2 are both read.

This is for race freedom.let us understand how this condition works.

For this condition to be happened two commands accessing the same shared variable g do not have a happens-before relationship between them, then they must both be reads, otherwise there is a data race.

Definition 9:

An algorithm D is sound for property Ψ if for all CTA T, D(T) ⇒ Ψ(T).

this is for soundness,which means the logical terms and equations introduced here are valid and is guaranteed to be safe.

Definition 10:

An algorithm D is complete for property Ψ if for all CTA T, ¬D(T) ⇒ ¬Ψ(T).

This is a Standard definition for completeness which means

It accepts everything, it rejects nothing, so all programs it rejects (i.e., none) have errors. This follows because from a false hypothesis ("a program is rejected") we can logically infer anything at all (including "the program has errors").

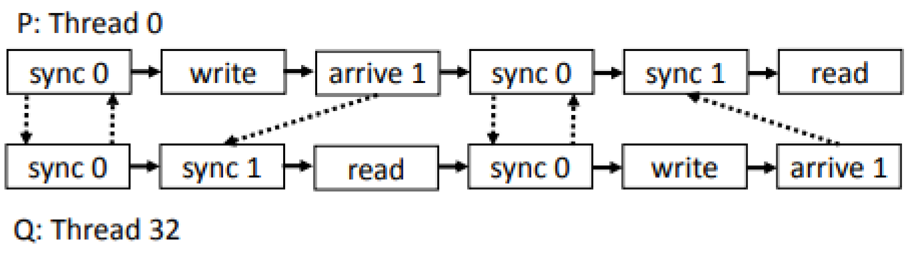
Property Checking:
Using algorithms we are checking whether or not a kernel is wellsynchronized and data race free using the similar example as before.

```
P                          Q
sync 0 64; (1)             sync 0 64; (1)
write g_0;                 sync 1 64; (1)
arrive 1 64; (1)           read g_0;
sync 0 64; (2)             sync 0 64; (2)
sync 1 64; (2)             write g_0;
read g_0                   arrive 1 64 (2)
return                     return
```

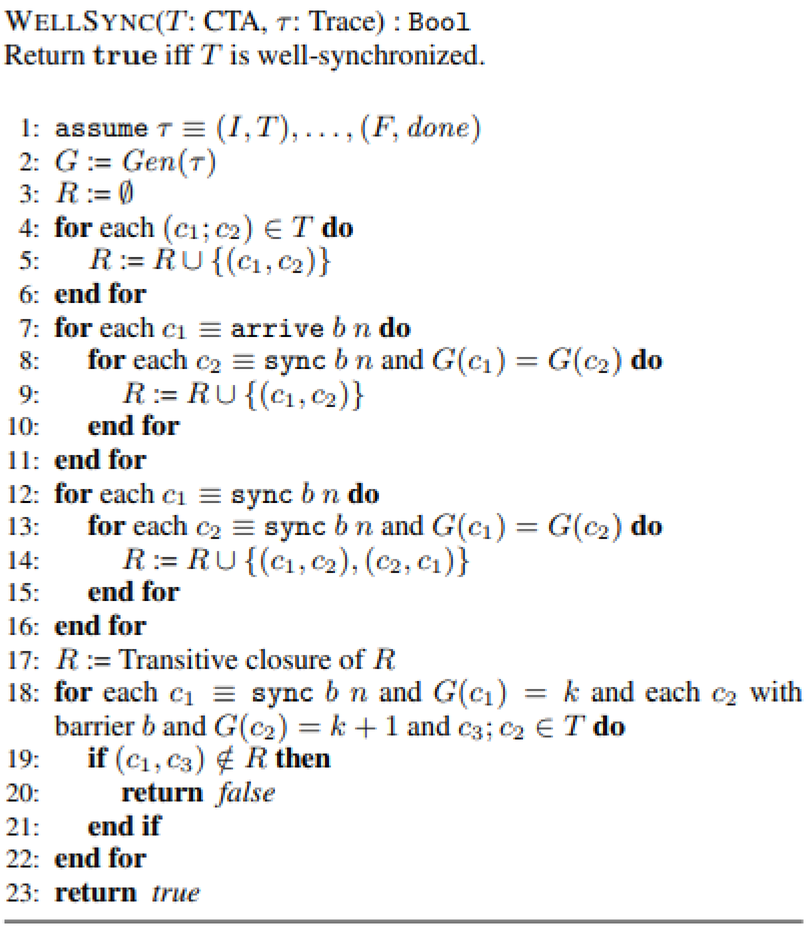
Figure 3. Thread programs generated from Listing 2; P has thread ID 0 and Q has 32.

This is the conclusion that P and Q are race free.

This shows CUDA program snippet for a kernel with 2 warps.
The thread programs associated with thread ID 0 and thread ID 32, denoted by P and Q respectively.Here P and Q with the same lane_id can have data races.So here If j is statistically known constant,then for each location g[j] (j is equal to lane_id)in shared memory is treated as a seperate variable g_j in the thread programs.



Here the path between two commands of P and Q signifies a happens-before relationship. From Figure 5, we observe that there is a path between any two accesses to the shared memory .



WELLSYNC(T: CTA, τ: Trace) : Bool
Return **true** iff T is well-synchronized.

```
assume $\tau \equiv (I, T), \ldots, (F, done)$
$G := Gen(\tau)$
$R := \emptyset$
for each $(c_1; c_2) \in T$ do
    $R := R \cup \{(c_1, c_2)\}$
end for
for each $c_1 \equiv$ arrive $b\ n$ do
    for each $c_2 \equiv$ sync $b\ n$ and $G(c_1) = G(c_2)$ do
        $R := R \cup \{(c_1, c_2)\}$
    end for
end for
for each $c_1 \equiv$ sync $b\ n$ do
    for each $c_2 \equiv$ sync $b\ n$ and $G(c_1) = G(c_2)$ do
        $R := R \cup \{(c_1, c_2), (c_2, c_1)\}$
    end for
end for
$R :=$ Transitive closure of $R$
for each $c_1 \equiv$ sync $b\ n$ and $G(c_1) = k$ and each $c_2$ with
barrier $b$ and $G(c_2) = k + 1$ and $c_3; c_2 \in T$ do
    if $(c_1, c_3) \notin R$ then
        return false
    end if
end for
return true
```

First we need to stimulate the one execution of CTA to generate a trace τ.

The violation of the well-synchronization property happens when the 1 st execution leads to deadlocks or errors,otherwise τ is done.
Suppose P and Q first 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 barrier 0, writes to g and registers on barrier 1. The unblocked thread P now recycles barrier 1 and reads g.
and later we will ensure all the synchronization.

If c1 and c2 are in the same generation ,then adding those to R.these result in the dashed edges and these commands are executed simultaneously.

After constructing happens-before relation for all program commands and checking,we need to establish a relation between P and Q.There happens to be a path between the two commands in the latest diagram.

<u>For lemma 1:</u>

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

So it totally says,The soundness of R is direct because there will be no out-of-order executions.It just needs to make sure that the precise part requires an induction on the size of the programs and the observation that the tuples in R are the only restrictions on the ordering in the execution imposed by the semantics.

R ensures that kernel is well-synchronized and generations are ordered.

We shall look into the Following Soundness results based on a different approaches.

<u>Lemma 2:</u>

If τ ≡ (I, T) ∗ (F, done) and WELLSYNC(T,τ) returns true then T is well-synchronized.

For the executions that reach done,it means it is well synchronized. WELLSYNC ensure that the new generation introduced has a happens-before-relationship with generations well-synchronized.

Well synchronized programs have precise R
Time-Complexity: For a CTA with n commands can be O(n 3)
worst case though
As a side effect of this procedure,Lemma provides sound and precise static R.
For two commands c1 and c2 accessing same shared memory location with atleast one write,we report a race if (c1, c2) (c2,c1)dont belong to R.

we need to check that there are happens-before relationships between the write in P and the write in Q, between the write in P and the read in Q and vice versa.