Mark Shirazaki
Cpts 350
HW 10

1^ Design an algorithm that decides where there is an W-path on which □(yellow ∨ ◇blue) holds.

□(yellow ∨ ◇ blue) G = (V, E) is of vertices

E ⊆ V × V is the set of edges

1^B. Algorithm for <>[](yellow ∨ blue)

Let S be a stack (for DFS)
or queue (for BFS) initialized with
V_0.

Let visited be an empty set
While S is not empty do
  V = S.pop() (for DFS) or S.dequeue() (for BFS)
  If V not in Visited then
  Add V to visited
  If C(V) is yellow or blue then
  Add V to P
  For each U in Adj(V) do
  If U not in visited then
  S.push(U) (for DFS) or S.enqueue(U)
  (For BFS)
  else if U in P and C(U) is yellow
  or blue then
  return W-path on which <>[] yellow ∨ blue
  holds exists.

Algorithm Always Yellow or Blue (G, V_0): Pseudocode

let S be a stack (for DFS) or queue (for BFS)
Initialized with V_0
let visited be an empty set
while S is not empty do
  V = S.pop() (for DFS) or S.dequeue() (for BFS)
  if V not visited then
    Add V to visited
    If C(v) is neither yellow nor blue then
      Continue to the next iteration of the loop
    for each U in Adj(V) do
      If U not in visited then
        S.push(U) (for dFS) or S.enqueue(U)
                                for BFS
      else if C(U) is yellow or blue and there
      exists a path P from V_0 to U then
        return that W-path on which
        [](yellow ∨ blue) holds exists

return // no path on which [](yellow ∨ blue) holds

2. Design an algorithm to decide whether
there is an ω-path on which it passes red nodes
for infinitely many times and passes blue nodes
for only finitely many times.

RedSCCs = {SCC | SCC is a strongly connected component of
G composed only of red nodes}

If no RedSCCs, return False

let S be a stack (for DFS) or S.dequeue() (for BFS)
If V not in visited then
    Add V to visited
    if C(V) is blue and V is part of a cycle then
        Add V to blueCycleNodes and continue to the next
        iteration
    For each v in Adj(V) do
        if not in visited and v not in blueNodes cycle
        then S.push(v) (for DFS) or S.enqueue(v) (for BFS)
            if V is part of RedSCCs then
                return True

3. Design an algorithm to decide
whether there is a good ω-path

Algorithm FindGoodOmegaPath(G, V_0)
    Identify all cycles C_b where R(C_b) = B(C_b)
    For each cycle C_b:
        If P(V_0, C_b) exists:
            return true, indicating a good path exists

4. Design an algorithm to decide whether there is a bad w-path.

Identify all cycles C_r where R(C_r) mod 5 = 0

For each cycle C_r:

If P(v_0, C_r) exists:

Return true, indicating a bad path exists

computing paths and cycles with the needed properties, would be part of a complex implementation.