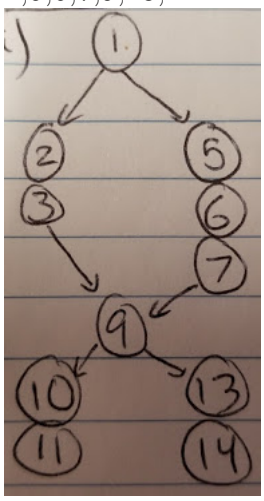
CSC410 A5

Joshua Prier,

November 22, 2019

1. (a) **All Paths:**
1,2,3,9,10,11
1,2,3,9,13,14
1,5,6,7,9,10,11
1,5,6,7,9,13,14



(b)

Infeasible Path		
Line No.	Assignment	Path Conditions
1,2,3	$x \leftarrow X - 5$ $y \leftarrow Y + 5$	$X + Y > 10$
9,10,11	$x \leftarrow -(X - 5)$ $y \leftarrow -(Y + 5)$	$X + Y > 10$ AND $(X - 5) + (Y + 5) < 0$

This path is infeasible because of the conflicting path conditions.
$10 < (X + Y) = (X - 5) + (Y + 5) \not< 0$.

(c)

Assertion Violation Path		
Line No.	Assignment	Path Conditions
1,5,6,7	$x \leftarrow Y$ $y \leftarrow X$	$X + Y \leq 10$
9,13,14	$x \leftarrow Y - 1$ $y \leftarrow X - 1$	$X + Y \leq 10$ AND $X + Y \geq 0$

This path will cause an assertion violation in the case that $X+Y \leq 2$ since the assignments from 9,13,14 subtract 2 from the sum, $(X-1)+(Y-1) = X+Y-2$

Why the other 2 paths never cause an assertion violation:

Assertion Violation Path		
Line No.	Assignment	Path Conditions
1,5,6,7	$x \leftarrow Y$ $y \leftarrow X$	$X + Y \leq 10$
9,10,11	$x \leftarrow -Y$ $y \leftarrow -X$	$X + Y \leq 10$ AND $X + Y < 0$

Since $x \leftarrow -Y$ and $y \leftarrow -X$ and $X + Y < 0$ then $(X-1) + (Y-1) = -(X+Y) > 0$

Assertion Violation Path		
Line No.	Assignment	Path Conditions
1,2,3	$x \leftarrow X - 5$ $y \leftarrow Y + 5$	$X + Y > 10$
9,13,14	$x \leftarrow (X-5) - 1 = X - 6$ $y \leftarrow (Y+5) - 1 = Y + 4$	$X + Y > 10$ AND $X + Y \geq 0$

Since $X+Y > 10$ then $(X-6)+(Y+4) = X+Y-2 > 10-2 = 8$ therefore $x + y > 0$

2. (a) $\Box b \rightarrow b \cup o$
   (b) $\Box \neg o \rightarrow ((r \rightarrow \bigcirc w) \wedge (w \rightarrow \bigcirc r)$
   (c) $\Box \neg o \rightarrow r \cup (b \cup (w \cup r))$
   (d) $\Box(r \wedge b \wedge g) \rightarrow \bigcirc w$
   (e) $\Box r \rightarrow r \cup \neg r$

3. (a) $\varphi\ U\ \neg\varphi\ \equiv\ true$
Let $t = A_1A_2\ldots$ be some trace of some transition system T:

By definition of U
$t \models \varphi\, U \neg\varphi = (\forall i 0 \leq i \leq j : A_i \models \varphi = true) \wedge (\forall j : i \leq j : A_j \models \neg\varphi = true)$
By Distributivity of $\wedge$ over $\forall$:
$= \ (\forall i 0 \leq i \leq j : A_i \models \varphi = true\ \wedge\ A_j \models \neg\varphi = true)$
$= \ (\forall i 0 \leq k : A_k \models t\ r\ u\ e)$
$= t \models\ true$

Therefore, $\varphi\ U\ \neg\varphi\ \equiv\ true$

(b) $(\Diamond\Box\varphi_1) \wedge (\Diamond\Box\varphi_2) \equiv \Diamond(\Box\varphi_1 \wedge \Box\varphi_2)$

$\sigma \models (\Diamond\Box\varphi_1) \wedge (\Diamond\Box\varphi_2)$
$\sigma \models \Diamond\Box\varphi_1$ and $\sigma \models \Diamond\Box\varphi_2$
$\exists i \geq 0 \forall j \geq i \sigma[j\ldots] \models \varphi_1$ and $\exists i \geq 0 \forall j \geq i \sigma[j\ldots] \models \varphi_2$
$\exists i \geq 0 \forall j \geq i \sigma[j\ldots] \models \varphi_1$ and $\forall j \geq i \sigma[j\ldots] \models \varphi_2$
$\exists i \geq 0\ \sigma[i\ldots] \models (\Box\varphi_1 \wedge \Box\varphi_2)$
$\sigma \models \Diamond(\Box\varphi_1 \wedge \Box\varphi_2)$

Therefore $(\Diamond\Box\varphi_1) \wedge (\Diamond\Box\varphi_2) \equiv \Diamond(\Box\varphi_1 \wedge \Box\varphi_2)$

(c)

(d)

(e) $O\Diamond\varphi\ \equiv \Diamond O\varphi$
Let $t = A_0A_1A_2\ldots$ be some trace of some transition system T
$t' = B_0B_1B_2\ldots$ as the suffixe of t defined by $B_i = A_{i+1}$for every
$0 \leq\ i$

$t \models\ O\Diamond\varphi = \big(\forall i : i \geq 0 : A_{i+1}A_{i+2}A_{i+3}\ldots \models \Diamond\varphi\big)$
$= (\forall i : i \geq 0 : B_iB_{i+1}B_{i+2}\ldots \models \Diamond\varphi)$
$= (\forall i : i \geq 0 : B_iB_{i+1}B_{i+2}\ldots \models t\ r\ u\ e\ U\varphi)$
$= (\forall j : j \geq 0 : B_0B_1\ldots B_j \models t\ r\ u\ e \wedge \forall k : k > j : B_kB_{k+1}\ldots \models \varphi = true)$

By definition of B: $= (\forall j : j \geq 0 : A_1A_2\ldots A_{j-1} \models t\ r\ u\ e \wedge \forall k : k \geq j : A_kA_{k+1}\ldots \models \varphi$
$= trueUO\varphi$
$= \Diamond O\varphi$

Therefore, $O\Diamond\varphi\ \equiv \Diamond O\varphi$

4.

4. a) ♦b ⇒ (a U b)

π |= ♦b

∃j ≥ 0. π[j...] |= b

if a = true:

∃j ≥ 0. π[j...] |= b and ∀0 ≤ i < j. π[i...] |= true

∃j ≥ 0. π[j...] |= b and ∀0 ≤ i < j. π[i...] |= a

π |= a U b

π |= ♦b ⇒ (a U b)

∃π. π |= ♦b ⇒ (a U b)

if a = ¬true:

∃j ≥ 0. π[j...] |= b and ∀0 ≤ i < j. π[i...] |≠ ¬true

¬(∃j ≥ 0. π[j...] |= b and ∀0 ≤ i < j. π[i...] |= a)

π |≠ a U b

π |≠ ♦b ⇒ (a U b)

∃π. π |≠ ♦b ⇒ (a U b)

¬(∀π. π |= ♦b ⇒ (a U b))

So ♦b ⇒ (a U b) is satisfiable, but not valid.

b) O(a ∨ ♦a) ⇒ ♦a

π |= O(a ∨ ♦a)

π[1...] |= a ∨ ♦a

if π[1...] |= a:

∃j ≥ 0. π[j...] |= a

$\pi \models \Diamond a$

$\pi \models O(a \vee \Diamond a) \Rightarrow \Diamond a$

if $\pi[1\ldots] \models \Diamond a$:

$\exists j \geq 0.\ \pi[j+1\ldots] \models a$

$\exists j' \geq 0.\ \pi[j'\ldots] \models a$

$\pi \models \Diamond a$

$\pi \models O(a \vee \Diamond a) \Rightarrow \Diamond a$

So $O(a \vee \Diamond a) \Rightarrow \Diamond a$ is valid.

5.

6.

7. **Algorithm Explanation:**

   The algorithm that we have implemented finds the winning player by setting winning states on a grid of size m by n

   It starts at (0,0) the final state and sets it as the final state. Then we set all winning states that can get to the final state in one move. This means we fill the vertical (0,y), horizontal (x,0), and the diagonal (x,y) as winning states.

   In the next step we can check the lowest index of an empty/unset location on the grid, we then check if this state is a winning state. A winning state is a state that can reach the final state or put the other player in a loss state. A loss state is a state where all moves put the other player in a winning state. Then the current position's state is set, if the current state is a loss state we do the same as the final state and set all possible moves to get to that location as a winning state.

   This is repeated until the location (m,n) on the grid is filled

   Code and Runtime:

   To run the given code you will need python3 installed (seems to also work with python 2.7) and run ./game.sh 1 2
   This will run the script to call the Q7.py module
   Note: This script might have issues on unix systems (like cdf) because it was written on a dos system and the line endings will sometimes break the script. If this happens cdf will still run, python ./Q7.py 1 2

   The runtime complexity of the algorithm is roughly $O(\frac{m+n}{2})$ or under. This complexity runs any game within 100x100 games in well under a second.

   These times are done by looping through each row and only checking empty states in the grid, along with ending the algorithm as soon as

the (m,n) state is found

Additional time was gained through avoiding clean readable code and optimizing runtime, in python this means the code looks childish

Here is the code, it uses all builtin python3 so no external packages are needed to run this.
Code is also provided as a .py file on markus

```
import sys
import timeit
from itertools import repeat


def get_winning_player(m, n):
  # repeat avoids unnecessary i's being created
  grid = [['E' for i in repeat(None, m+1)] for j in
                              repeat(None, n+1)]
  grid = set_grid_loses(grid, 0, 0, m, n) #(0,0) is a
                               known state and can be
                              set
  i = 0
  while i <= n:
    # quit once result is found -- Can delete if this
                                is against the rules
    if grid[n][m] != 'E':
      return 1 if grid[n][m] == 'w' else 2

    while 'E' in grid[i]:
      j = grid[i].index('E')
      if is_winning_state(grid, i, j):
        grid[i][j] = 'w'
      else:
        grid = set_grid_loses(grid, i, j, m, n)
    i+=1

  return 1 if grid[n][m] == 'w' else 2
```

```
# if there is a move in which you can make the other
                              player lose you are in a
                              winning state
def is_winning_state(grid, x, y):
  if 'l' in grid[x][:y]:
    return True
  for i in range(x):
```

```
      if grid[i][y] == 'l':
        return True
    for i in range(min(x, y)-1):
      if grid[x-i-1][y-i-1] == 'l':
        return True
    return False
```

```
  def set_grid_loses(grid, x, y, m, n):
    # (0,0) is an exception to the rules as it is the
                                final state
    grid[x] = ['w'] * len(grid[x])
    for i in range(n+1):
      grid[i][y] = 'w'
      if i <= m:
        grid[i][i] = 'w'
    grid[x][y] = 'w' if x == 0 and y == 0 else 'l'
    return grid
```

```
  def main():
    start = timeit.default_timer()
    m, n = int(sys.argv[1]), int(sys.argv[2])
    print("The winning player for game ({},{}) is: {}".
                                format(m, n,
                                get_winning_player(m,n)))
    stop = timeit.default_timer()
    print("Algo took: " + str(stop-start))


  if __name__ == "__main__":
    main()
```