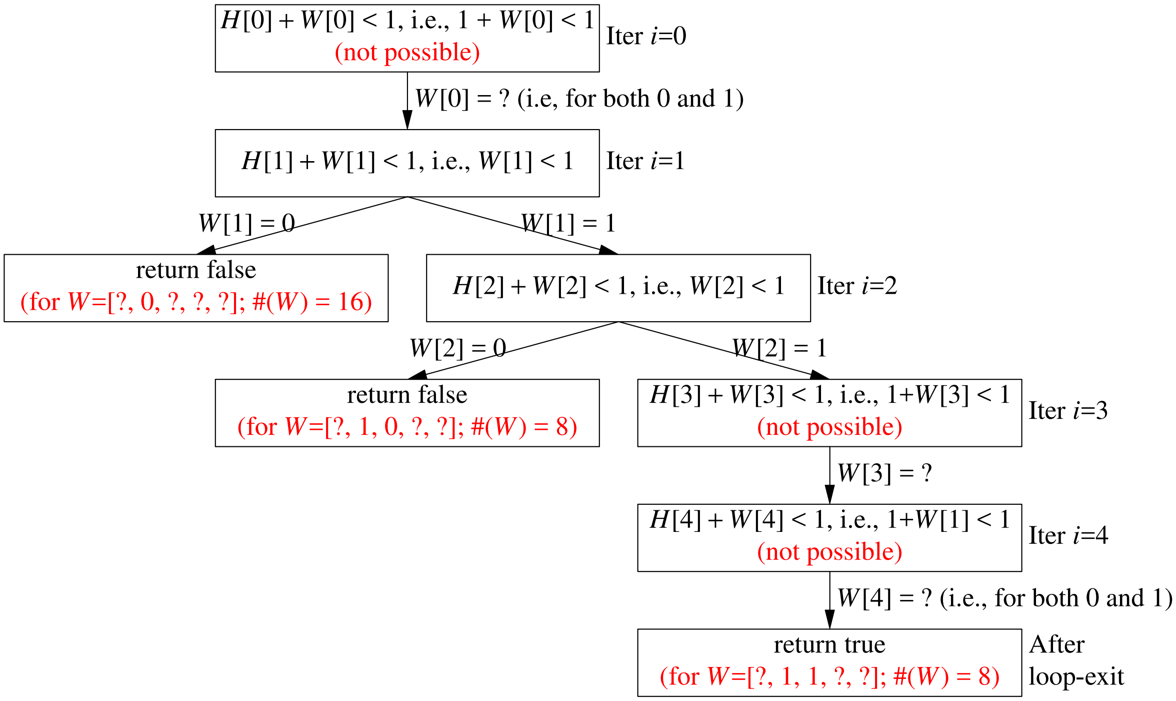
Shown below is a code for testing $H \cup W = \Omega$, when the sets H and W are represented as 0/1-arrays. We also show the "execution-tree" of the code for the given $H = [1, 0, 0, 1, 1]$ and different W's. The execution-tree shows the different true/false branches taken in line 2 of the for-loop for different W's. Finally, we show the performance analysis of the code in terms of the average number of loop-iterations for the code. (The shorter table format that was presented in previous lectures for the performance analysis is a "summary" of the information in the execution-tree.)

1. A code (for testing $H \cup W = \Omega$):

```
for (int i=0; i<H.length; i++)
    if (H[i] + W[i] < 1) return(false);
return(true);
```

2. Execution-tree for $H = [1, 0, 0, 1, 1]$:



H[0] + W[0] < 1, i.e., 1 + W[0] < 1
(not possible)
Iter i=0
W[0] = ? (i.e, for both 0 and 1)
H[1] + W[1] < 1, i.e., W[1] < 1
Iter i=1
W[1] = 0
W[1] = 1
return false
(for W=[?, 0, ?, ?, ?]; #(W) = 16)
H[2] + W[2] < 1, i.e., W[2] < 1
Iter i=2
W[2] = 0
W[2] = 1
return false
(for W=[?, 1, 0, ?, ?]; #(W) = 8)
H[3] + W[3] < 1, i.e., 1+W[3] < 1
(not possible)
Iter i=3
W[3] = ?
H[4] + W[4] < 1, i.e., 1+W[1] < 1
(not possible)
Iter i=4
W[4] = ? (i.e., for both 0 and 1)
return true
(for W=[?, 1, 1, ?, ?]; #(W) = 8)
After
loop-exit

3. Average #(loop-iterations) = $\dfrac{2{\times}16 + 3{\times}8 + 5{\times}8}{16 + 8 + 8}$ = 96/32 = 3.

4. This code returns true if and only if

   for all i, $H[i] + W[i] \geq 1$, i.e., for all i, $H[i] = 1$ or $W[i] = 1$ (or both)
   i.e., each item belongs to one of the sets H or W
   i.e., $H \cup W = \Omega$.

   Thus, we say the code tests whether $H \cup W = \Omega$ (i.e., $H \supseteq W^c$, i.e., $H^c \subseteq W$) or not.

5. Some properties of an execution-tree:

   (a) Each box (node) shows some computation without any branching. (We do not show here the assignments to i and the loop-test "$i < H$.length" to simplify the diagram.) The top node, called the *root* node, shows the common computation for all inputs (which in this case is the 1st iteration of the for-loop).

   (b) Each link is directed, indicating the progress of computation at the link's tail-node to that at the link's head-node. The label on the link shows the condition to be fulfilled for the computation to proceed in that direction.

   (c) There is a unique path from the top node in the tree to every other node.

   (d) Each *terminal node* (with no link from it) shows the end of computation for some input (here, W).

   (e) The and-combination of all link-labels along a path from the root node to a terminal node gives the complete input-condition for the computation to follow that path.