Theory of Computation Assignment no. 1

Goktug Saatcioglu

(1) If $x = True$ and $y = True$, then the LHS equals $False$ since $\neg(True \land True) = \neg True = False$. The RHS equals $(\neg True \lor \neg True) = False \lor False = False$, which is equal to the LHS. Thus, when both x and y are $True$, the LHS = RHS.
When $x = True$ or $y = True$, but not $x = True$ and $y = True$, we see that the LHS equals $True$ since $\neg(True \land False) = \neg False = True$ and $\neg(False \land True) = \neg False = True$. The RHS in both cases equals $True$ because $(\neg True \lor \neg False) = False \lor True = True$ and $(\neg False \lor \neg True) = True \lor False = True$. Thus, when either x or y is $True$ (but not both), the LHS = RHS.
Finally, if $x = False$ and $y = False$, then the LHS equals $True$ since $\neg(False \land False) = \neg False = True$. The RHS equals $(\neg False \lor \neg False) = True \lor True = True$, which is equal to the LHS. Thus, when both x and y are $False$, the LHS = RHS.
We see that the for every setting for the binary variables x and y, the LHS and RHS of the equation evaluate the same truth value. $\therefore \neg(x \land y) = (\neg x \lor \neg y)$.

(2) Suppose there are $2n + 1$ beads on a necklace. Assign a color to every 2 beads, this gives us $2n$ beads colored with n colors. By the pigeonhole principle, there is a remaining bead that forms a match with one of the n colors we have. Thus, there is at least a set of at least 3 beads on the necklace that will have the same color.

(3) Let $T(n)$ be the function over natural numbers n, defined as follows: $T(1) = 2$ and $T(2) = 4$ and for any other n:

$$T(n) = 2 + \min_{i=1\ldots n-2} \{T(i) + T(n - i - 1)\}.$$

Using strong induction we prove that $T(n) = 2n$ for all n.
Base case. $n = 3$.
$T(3) = 2 + \min_{i=1\ldots1}\{T(i) + T(2 - i)\} = 2 + T(1) + T(1) = 2 + 2 + 2 = 6 = 2 \times 3 = 2n$ $\therefore$ holds true for base case.
Inductive step. We show for $k \geq 3$, that if $T(h) = 2h$ for all $h \leq k$ then $T(k + 1)$ is also true.

$$\begin{aligned} T(k+1) &= 2 + \min_{i=1\ldots k+1-2}\{T(i) + T(k + 1 - i - 1)\} \\ &= 2 + \min_{i=1\ldots k-1}\{T(i) + T(k - i)\} \end{aligned} \tag{1}$$

In (1) we see that the min operator attempts to find the minimum between the range $i = 1$ to $k - 1$ of $T(i) + T(k - i)$. For all i in that range, each iteration of the min operator will evaluate the value of $T(k)$ since by the induction hypothesis, $T(i) + T(k - i) = 2i + 2(k - i) = 2i + 2k - 2i = 2k = T(k)$ for all i, k. Thus, the min operator in (1) will always evaluate to the value of $T(k)$. From the induction hypothesis, we see that $T(k) = 2k$ and thus equation (1) evaluates to:

$$\begin{aligned} T(k+1) &= 2 + \min_{i=1\ldots k-1}\{T(i) + T(k - i)\} \\ &= 2 + T(k) && \text{[by evaluating the min operator]} \\ &= 2 + 2k && \text{[by the induction hypothesis]} \\ &= 2(k + 1). \end{aligned} \tag{2}$$

Conclusion. By the principle of strong induction, we see that for $n \geq 3$, $T(n) = 2n$. Furthermore, by definition, $T(1) = 2$ and $T(2) = 4$. $\therefore T(n) = 2n$ for all n.

(4) Consider the algorithm E, given below, which uses H as a subroutine.

function E(x)
  $A \leftarrow H(x, xx)$
  if $A = $ "Yes" **then** loop forever
  else ($A = $ "No" and) $D(x)$ outputs "Yes"
  end if
end function

If x is not a valid program or x is not a double then $H(x,x)$ evaluates to "Yes". However, to consider valid programs with inputs that are doubles, we must modify H used to determine A such that H gets a valid program x, which may or not be a double, and a string representation of x which is a double, making the second input xx. Here if the program string were to be a double, such as $x = yy$, then $xx = yyyy$ is still a double and the program logic is held (H still attempts to solve the 2HALT problem).
Now consider what happens if we run E on its own code ($E(E)$). We then get the following contradiction:

$$\begin{aligned} E(E) \text{ never halts} &\implies H(E, EE) = \text{"}Yes\text{"} \\ &\implies E(E) \text{ halts} \implies H(E, EE) = \text{"}No\text{"} \\ &\implies E(E) \text{ never halts} \implies \ldots \end{aligned}$$

From the contradiction we see that there is no algorithm $H(P, x)$ that solves 2HALT. (Note: if we change $H(x, xx)$ to $H(x, x)$ we will obtain the same contradiction. However, now the algorithm E will not be attempting to solve the 2HALT problem.)