Suggested Solutions to HW #2

1. (2.24) We can define **anti-Gray codes** in the following way. Instead of minimizing the difference between two consecutive strings, we can try to maximize it. Is it possible to design an encoding for any even number of objects such that each pair of two consecutive strings differ by k bits (where k is the number of bits in each string)? How about $k-1$ bits (or $k-2$, $k-3$, etc.)? If it is possible, find an efficient construction.

Solution. (Chih-Pin Tai) It is impossible to design an encoding for an even number of objects such that each pair of two consecutive strings differ by k bits (where k is the number of bits in each string). The reason is that if we want to make each pair of two consecutive strings differ by k bits, we will find that we can only construct two-object anti-Gray codes because the third string will be the same as the first.

Therefore, we try to make each pair of two consecutive strings differ by $k-1$ bits. We can construct anti-Gray codes in the following way: (1) Construct Gray codes for the $n(=2^k)$ objects. (2) Reverse each bit of the code for every $2i$-th object. (3) For $(2i+1)$-th objects, add an additional bit 0 to the left of the most significant bit and, for $2i$-th objects, add an additional bit 1.

The construction is correct if each pair of two consecutive strings differ by $k-1$ bits which is the difference we can maximize and there is no collision for all objects. To see that each pair of two consecutive strings differ by $k-1$ bits, we consider the property of Gray codes. Because each pair of consecutive strings differ by 1 bit in Gray codes and we reverse each bit for $2i$-th objects, each pair of consecutive strings must differ by $k-1$ bits. To see that there is no collision for all objects, we consider the p-th object in our construction. If p is odd, it is impossible that the *p-th* object will conflict with other $(2i+1)$-th objects because it does not conflict with other $(2i+1)$-th objects in Gray codes. Besides, it is impossible that the p-th object will conflict with $2i$-th objects because relative to $2i$-th objects, we add a different bit 0 to the left of the most significant bit of the p-th object. Similarly, we can prove the case when p is even. Therefore, there is no collision for all objects in our construction.

□

4 (2.39) Design an algorithm to convert a binary number to a decimal number. The algorithm should be the opposite of algorithm *Convert_to_Binary* (see Fig. 1). The input is an array of bits b of length k, and the output is a number n. Prove the correctness of your algorithm by using a loop invariant.

Solution. (Modified by Yi-Wen Chang)

```
***Algorithm Convert_to_Binary***$(n)$:
**Input:** $n$ (a positive integer).
**Output:** $b$ (an array of bits corresponding to the binary
representation of n).

***begin***
    $t := n$ ;
    $k := 0$ ;
    ***while*** $t > 0$ ***do***
        $k := k + 1$ ;
        $b[k] := t\ mod\ 2$ ;
        $t := t\ div\ 2$ ;
***end***
```

Figure 1: Algorithm *Convert_to_Binary*.

```
**Algorithm Convert_to_Decimal** $(b, k)$;
**Input:** $b$ (an array of bits containing a binary number), $k$ (the array size of $b$)
**Output:** $dec$ (a positive integer corresponding to the decimal number of array $b$)
**begin**
    $dec := 0$;
    $i := k$;
    **while** $i > 0$ **do**
            $dec\ :=\ dec \times 2 + b[i]$;
            $i\ :=\ i - 1$;
**end**
```

Let $Inv(n, dec, k, b)$ denote the following assertion:

$n = dec \times 2^i + m$ and $i \geq 0$,

where n is the number represented by b, i.e.,

$$n = \begin{cases} 0 & \text{if } k = 0 \\ b[k] \times 2^{k-1} + b[k-1] \times 2^{k-2} + \cdots + b[1] \times 2^0 & \text{if } k \geq 1 \end{cases}$$

and m is the number represented by b from position i to 0, i.e.,

$$m = \begin{cases} 0 & \text{if } i = 0 \\ b[i] \times 2^{i-1} + b[i-1] \times 2^{i-2} + \cdots + b[1] \times 2^0 & \text{if } i \geq 1 \end{cases}$$

Claim: $Inv(n, dec, i, b)$ is a loop invariant of the while loop, assuming that n is non-negative. (The invariant is sufficient to deduce that, when the program terminates, $i = 0$ and so dec stores the decimal n represented by b.)

Proof: The proof is by induction on the number of times the loop is executed. More specifically, we show that (1) the assertion is true when the flow of control reaches the loop for the first time and (2) given that the assertion is true and the loop condition holds, the assertion will remain true after the next iteration (i.e., after the loop body is executed once more).

(1) When the flow of control reaches the loop for the first time, $dec = 0$ and $i = k$. With m denoting the binary number represented by b from position k to 0, $dec \times 2^i + m = 0 \times 2^k + m = 0 + n = n$ (when $i = k$, m equals n, trivially) and $i \geq 0$. Therefore, the assertion $Inv(n, dec, i, b)$ holds.

(2) Assume that $Inv(n, dec, i, b)$ is true at the start of the next iteration and the loop condition $(i > 0)$ holds. Let n', dec', i', and b' denote respectively the values of n, t, i, and b after the next iteration. We need to show that $Inv(n', t', i', b')$ also holds.

From the loop body, we deduce the following relationship:

$$\begin{aligned}
&i' = i - 1 \\
&b'[j] = b[j] \text{ for all } j \leq k \\
&dec' = dec \times 2 + b[i] \\
&m' = m - b[i] \times 2^{i-1} \\
&n' = n \text{ (the value of } n \text{ never changes)}
\end{aligned}$$

Thus, we have:

$$\begin{aligned}
dec' \times 2^{i'} + m' &= (dec \times 2 + b[i]) \times 2^{i-1} + m' \\
&= dec \times 2 \times 2^{i-1} + b[i] \times 2^{i-1} + m' \\
&= dec \times 2^i + b[i] \times 2^{i-1} + m - b[i] \times 2^{i-1} \\
&= dec \times 2^i + m = n = n'
\end{aligned}$$

In addition, since $i > 0$ (given that the loop condition holds), $i' = i - 1 \geq 0$.Therefore, $Inv(n', t', k', b')$ holds after the next iteration. $\square$