CSC238S **Problem Set 1 Solutions (Partial)**

1. a) A 1-place error detecting code $C \subseteq \{0, 1, \ldots, k-1\}^n$ with $|\, C \,| = k^{n-1}$ can be defined as follows:

$$C = \{(x_1, \ldots, x_n) \mid x_n = (\sum_{i=1}^{n-1} x_i) \; mod \; k\}$$

Claim: Any two distinct code words in C differ in at least two places.

b) This is a direct modification of the proof given on page 37 (example 3.6). By induction on $n \geq 1$, we show that *any* 1-place error detecting code $C \subseteq \{0, 1, \ldots, k-1\}^n$ has $|\, C \,| \leq k^{n-1}$.

$n = 1$ is obvious

$n > 1$ Partition C into $C_0 \cup C_1 \cup \cdots \cup C_{k-1}$ where $C_i = \{(x_1, \ldots, x_n) \mid x_1 = i\}$.

Let $D_i \subseteq \{0, 1, \ldots, k-1\}^{n-1}$ be defined as $D_i = \{(x_2, \ldots, x_n) \mid (x_1, \ldots, x_n) \in C_i\}$.

Claim: $D_i \subseteq \{0, 1, \ldots, k-1\}^{n-1}$ is a 1-place error detecting code.

By the induction hypothesis, $|\, D_i \,| \leq k^{n-2}$. Then

$|\, C \,| = \sum_{i=0}^{k-1} |\, C_i \,| = \sum_{i=0}^{k-1} |\, D_i \,| \leq k \cdot k^{n-2} = k^{n-1}$. $\square$

2. Let E be an expression using only binary operators. We want to show that for every expression $|\, E \,|$ is odd. That is, $\forall k \geq 1 \; [\exists E(k = |\, E \,|) \rightarrow k \text{ odd}]$.

$k = 1$ is trivial.

$k > 1$ Let E be such that $|\, E \,| = k$. (If no such E exists, then we are done.)

Else since $|\, E \,| > 1$, $E = (E_1 \# E_2)$, where $\#$ is a binary operator.

Clearly $k_1 = |\, E_1 \,| < k_1$ and $k_2 = |\, E_2 \,| < k$. Therefore, k_1 and k_2 are odd and thus $(k_1 + k_2)$ is even.

$$\begin{aligned} |\, E \,| &= |\, E_1 \,| + |\, E_2 \,| + 3 \quad (\text{since } |\, (\,| = |\,) \,| = |\, \# \,| = 1) \\ &= k_1 + k_2 + 3 \;\; \text{which is odd} \end{aligned}$$

$\square$

Note: We are assumming a particular inductive definition of an expression. Another possible definition is as follows: if $|\, E \,| > 1$ then E has one of the following two forms:

- $E = E_1 \# E_2$
- $E = (E_1)$

A proof of the the same theorem would now require two cases for the induction step.

3. b)

```
procedure bitonicmax(
            list: array 1.. * of recordType,
            size: int, var location: int)

% Assumes that size is a positive integer, and the records
% in list are sorted in nondecreasing order by key.

var first, last, middle: int
first := 1
last := size
loop
    exit when first = last %only one element left
    middle := (first + last) div 2
    if list(middle).key < list(middle+1).key then
    last := middle
    else
    first := middle + 1
    end if
end loop
    location := first
end bitonicmax
```