4. **Problem:** Show that any alphabet $\Sigma$ with at least two symbols is comparable to the binary alphabet $\Gamma = \{0, 1\}$, in the sense that strings over $\Sigma$ can be easily encoded into strings over $\Gamma$ and vice versa. How much can the length of a string change in your encoding? (I.e., if the length of a string $w \in \Sigma^*$ is $|w| = n$ symbols, what is the length of the corresponding string $w' \in \Gamma^*$?) Could you design a similar encoding if the target alphabet consisted of only one symbol, e.g. $\Gamma = \{1\}$?

**Solution:** Let $\Sigma$ be some alphabet with $k$ symbols, $k > 1$. The strings of $\Sigma$ can be coded as strings of $\Gamma = \{0, 1\}$ in the following manner.

- Set the symbols of $\Sigma$ to equal integers $\{1, \ldots, k\}$.
- These numbers (the symbols of $\Sigma$) can be represented with binary numbers of length $\lceil \log_2 k \rceil$.
- Every string in $\Sigma^*$ can therefore be represented as a string of $\Gamma$ by replacing the symbols of $\Sigma$ with their binary encoding.

The decoding from $\Gamma^*$ to $\Sigma^*$ can be done in a similar fashion by taking strings of length $\lceil \log_2 k \rceil$ from a string and interpreting them as symbols of $\Sigma$.

If the length of a string $w \in \Sigma^*$ is $|w| = n$ symbols, the length of its counterpart $w' \in \Gamma^*$ is $|w'| = n \cdot \lceil \log_2 k \rceil$. This is because the number of symbols needed to encode any symbol in $\Sigma$ is $\lceil \log_2 k \rceil$.

For an example, consider the alphabet $\Sigma = \{a, b, c, d, e, f\}$ and the string $aacfd$. As $|\Sigma| = 6$, $\lceil \log_2 6 \rceil = \lceil 2.58 \rceil = 3$ bits are needed to represent the symbols of $\Sigma$. One possible encoding is

$$
\begin{align*}
a &\mapsto 001 \\
b &\mapsto 010 \\
c &\mapsto 011 \\
d &\mapsto 100 \\
e &\mapsto 101 \\
f &\mapsto 110
\end{align*}
$$

With this encoding, the representation of $aacfd$ is $001001011110100$.

A similar coding scheme cannot be constructed if $\Gamma = \{1\}$. A unary presentation of the form $1 \mapsto 1, 2 \mapsto 11, 3 \mapsto 111, \ldots$ can of course be defined, but the code obtained in this way can no longer be decoded unambiguously. For an example, the encodings of $111, 12, 21$ and $3$ are all the string $111$.

5. **Problem:** Design finite automata that recognise the following languages:

(a) $\{a^m b^n \mid m = n \mod 3\}$;

(b) $\{w \in \{a, b\}^* \mid w \text{ contains equally many } a\text{'s and } b\text{'s, modulo } 3\}$.  

(The notation $\equiv m \mod 3$ means that the numbers $m$ and $n$ yield the same remainder when divided by three.)

**Solution:**
a) The language $L = \{a^m n^n \mid m = n \mod 3\}$ can be recognized by the finite automaton:

$$M = (Q, \Sigma, \delta, q_0, F)$$

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$\Sigma = \{a, b\}$$

$$F = \{q_0, q_5\}$$

The state transition function $\delta$ is:

<table>
<thead>
<tr>
<th>$q$</th>
<th>$\delta(q, a)$</th>
<th>$\delta(q, b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_2$</td>
<td>$q_5$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_0$</td>
<td>$q_4$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_6$</td>
<td>$q_4$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$q_6$</td>
<td>$q_5$</td>
</tr>
<tr>
<td>$q_5$</td>
<td>$q_6$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_6$</td>
<td>$q_6$</td>
<td>$q_6$</td>
</tr>
</tbody>
</table>

The state $q_6$ is used as a rejecting state. It means that the automaton moves into that state as soon as it is clear that the word cannot belong to the language (when $ba$ substring is found). The machine then stays in that state until the end of the string. These states are often left out when an automaton is represented as a state diagram. This is also the case for the diagram below:

![State Diagram](image)

b) Language $L = \{w \in \{a, b\}^* \mid w$ contains as many $a$ and $b$ letters modulo 3$\}$ is recognized by the following finite automaton:

![State Diagram](image)

6. **Problem**: Design a finite automaton that recognizes sequences of integers separated by plus and minus signs (e.g. $11 + 20 - 9$, $-5 + 8$). Implement your automaton as a computer program that also calculates the numerical value of the input expression.

**Solution**: The plus and minus operations can be recognized with the following automaton:

![State Diagram](image)
Here $d$ is a shorthand notation that means any number from set $\{0, \ldots, 9\}$.
Below is a C-program that reads user input and does the required addition. It has been modified from the C-program presented in the lecture that can identify a number and it’s size.

```c
#include <stdio.h>
#include <ctype.h>

int main (void)
{
    int q;  /* state */
    int c;  /* input symbol */
    int sgn, val, sum;
    sgn=1; val = 0, sum = 0;
    q = 0;

    while ((c = getchar()) !='\n') {
        switch (q) {
        case 0:
            if (c == '+') q = 1;
            else if (c == '-') {
                sgn = -1;
                q = 1;
            }
            else if (isdigit(c)) {
                val = c - '0';
                q = 2;
            }
            else q = 99;
            break;

        case 1:
            if (isdigit(c)) {
                val = c - '0';
                q = 2;
            }
            else q = 99;
            break;

        case 2:
            if (isdigit(c)) {
                val = 10 * val + (c - '0');
                q = 2;
            }
            else if (c == '+')
            {
                sum = sum + val*sgn;
                val = 0;
                sgn = 1;
                q = 1;
            }
            else if (c == '-')
            {
                sum = sum + val*sgn;
            }
            break;
        }
    }
    return 0;
}
```
val = 0;
sgn = -1;
q = 1;
}
else q = 99;
break;

case 99:
    break;
}
}
sum = sum + sgn*val;
if (q == 2)
    printf("SUM IS %d.\n", sum);
else
    printf("INVALID INPUT.\n");
return;