Another example:

Let $\text{TOT}$ be the set of all numbers $p$ such that $p$ is the number of a program that computes a total function $f(x)$ of one variable:

$\text{TOT} = \{ z \in \mathbb{N} \mid (\forall x) \Phi(x, z) \downarrow \}$

Since $\Phi(x, z) \downarrow \iff x \in W_z$, $\text{TOT}$ is simply the set of numbers $z$ such that $W_z$ is the set of all nonnegative integers.

**Theorem 6.1**: $\text{TOT}$ is not recursively enumerable.

**Proof:**

Suppose that $\text{TOT}$ were r.e.

Since $\text{TOT} \neq \emptyset$ (we know that there are total unary functions), by Theorem 4.9 there is a computable function $g(x)$ such that $\text{TOT} = \{ g(0), g(1), g(2), \ldots \}$.

Let $h(x) = \Phi(x, g(x)) + 1$.

Since any $g(x)$ is the number of a program that computes a total function, $\Phi(x, g(x))$ is defined for all $x$, and $h(x)$ is a computable function.

Let $h(x)$ be computed by a program with number $p$.

Then $p \in \text{TOT}$, which means that $p = g(i)$ for some $i$. Then

$h(i) = \Phi(i, g(i)) + 1$ by definition of $h$

$= \Phi(i, p) + 1$ since $p = g(i)$

$= h(i) + 1$ since $h$ is computed by $p$. Contradiction!

The elements on the diagonal make it impossible for the function $h$ to be computed by any of the programs $g(x)$.

**Theorem 6.1** gives a reason why we base our studies of computability on partial rather than total functions:

By Church's Thesis, Theorem 6.1 shows that there is no algorithm to determine whether an $L$ program computes a total function.

Another important technique for determining nonrecursive sets is the reducibility method.

Once some set (such as the set $K$) has been shown to be nonrecursive, we can use that set to give other examples of nonrecursive sets.
Reducibility

**Theorem 6.2:** Suppose $A \leq^m B$.
1. If $B$ is recursive, then $A$ is recursive.
2. If $B$ is r.e., then $A$ is r.e.

**Proof:** Let $A = \{ x \in \mathbb{N} | f(x) \in B \}$, where $f$ is computable, and let $P_B(x)$ be the characteristic function of $B$.

If $B$ is recursive, then $P_B(x)$, the characteristic function of $A$, is computable, so $A$ is recursive.

If $B$ is r.e., then $B = \{ x \in \mathbb{N} | g(x) \downarrow \}$ for some partially computable function $g$.

Then $A = \{ x \in \mathbb{N} | g(f(x)) \downarrow \}$, and since $g(f(x))$ is partially computable, $A$ is r.e.

We will often use Theorem 6.2 in the following form:
If $A$ is not recursive (r.e.), then $B$ is not recursive (r.e.).

**Example:**
$K_0 = \{ z \in \mathbb{N} | \Phi_r(z)(l(z)) \downarrow \} = \{ \langle x, y \rangle | \Phi_y(x) \downarrow \}$

Obviously, $K_0$ is r.e. However, we can show that $K_0$ is not recursive by reducing $K$ to $K_0$.

$K = \{ n \in \mathbb{N} | n \in W_n \}$.

Now $x \in K$ if and only if $\langle x, x \rangle \in K_0$, and the function $f(x) = \langle x, x \rangle$ is computable.

Therefore, $K \leq^m K_0$, and $K_0$ is not recursive.

Numerical Representation of Strings

So far, our programs in the language $\mathcal{L}$ have been using natural numbers as their inputs and output.

For many applications, however, we would prefer to perform computations on strings on some alphabet instead.

You remember that we introduced a numbering of $\mathcal{L}$ programs so that $\mathcal{L}$ programs could be used as input and output of another (or the same) $\mathcal{L}$ program.

With regard to strings, we will use the same approach:
We will associate numbers with strings on $A$ in a one-one manner.

For example, if we have the string $w = 372$, then $k = 2$, $i_2 = 3$, $i_1 = 7$, $i_0 = 2$.

To find the number associated with this string, we use exactly the following formula:

$x = i_k n^k + i_{k-1} n^{k-1} + \ldots + i_1 n^1 + i_0 n^0$

$x = 3 \cdot 10^2 + 7 \cdot 10^1 + 2 = 372$.

If $w = 372$ is an octal representation of an integer, then we would have $n = 8$ and therefore:

$x = 3 \cdot 8^2 + 7 \cdot 8^1 + 2 = 192 + 56 + 2 = 250$.
Numerical Representation of Strings
Then we use exactly the same formula as before to associate \( w \) with an integer \( x \):
\[
x = i_k n^k + i_{k-1} n^{k-1} + \ldots + i_1 n^1 + i_0 n^0.
\]
With \( w = 0 \) we associate the number \( x = 0 \).
For example, consider the alphabet \( A = \{a, b, c\} \) and the string \( w = caba \).
Then \( x = 3 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^1 + 1 = 81 + 9 + 6 + 1 = 97 \).
Now why is this representation unique?
We can prove this by showing how to retrieve the subscripts \( i_0, i_1, \ldots, i_k \) from \( x \) for any \( x > 0 \).

Numerical Representation of Strings
First, we define two primitive recursive functions
\[
\begin{align*}
R^+(x,y) &= \begin{cases} 
R(x,y) & \text{if } (y|x) \\
\left\lfloor \frac{x}{y} \right\rfloor & \text{otherwise}
\end{cases} \\
Q^+(x,y) &= \begin{cases} 
\left\lfloor \frac{x}{y} \right\rfloor & \text{if } (y|x) \\
\left\lfloor \frac{x}{y} \right\rfloor - 1 & \text{otherwise}
\end{cases}
\end{align*}
\]
where \( R(x,y) \) and \( \left\lfloor \frac{x}{y} \right\rfloor \) are defined as in Section 3.7.
Basically, \( R^+ \) and \( Q^+ \) are the "usual" remainder and quotient functions, except that remainders are now in the range between 1 and \( y \) instead of 0 and \( y-1 \).

Numerical Representation of Strings
So whenever \( y \) divides \( x \), we do not have a remainder of 0 but a remainder of \( y \), and accordingly the quotient is one number below the "actual" quotient.
Therefore, like with the usual quotient and remainder, it is still true that:
\[
x/y = Q^+(x,y) + R^+(x,y)/y,
\]
only that now we have \( 1 \leq R^+(x,y) \leq y \).
We will use the functions \( Q^+ \) and \( R^+ \) to show how to obtain the subscripts \( i_0, i_1, \ldots, i_k \) from any integer \( x > 0 \).

Numerical Representation of Strings
Let us define:
\[
\begin{align*}
\textbf{u}_0 &= x \\
\textbf{u}_{m+1} &= Q^+(\textbf{u}_m, n)
\end{align*}
\]
Then we have:
\[
\begin{align*}
\textbf{u}_0 &= i_k n^k + i_{k-1} n^{k-1} + \ldots + i_1 n^1 + i_0 \\
\textbf{u}_1 &= i_k n^{k-1} + i_{k-1} n^{k-2} + \ldots + i_1 \\
& \vdots \\
\textbf{u}_k &= i_k
\end{align*}
\]
The "remainders" \( R^+ \) are exactly the values of the \( i_m \): \( i_m = R^+(\textbf{u}_m, n), m = 0, \ldots, k \).

Numerical Representation of Strings
This is analogous to our usual base-\( n \) notation:
\[
\begin{align*}
\textbf{u}_0 &= x \\
\textbf{u}_{m+1} &= Q(\textbf{u}_m, n)
\end{align*}
\]
Then we have:
\[
\begin{align*}
\textbf{u}_0 &= i_k n^k + i_{k-1} n^{k-1} + \ldots + i_1 n^1 + i_0 \\
\textbf{u}_1 &= i_k n^{k-1} + i_{k-1} n^{k-2} + \ldots + i_1 \\
& \vdots \\
\textbf{u}_k &= i_k
\end{align*}
\]
The remainders \( R \) are exactly the values of the \( i_m \): \( i_m = R(\textbf{u}_m, n), m = 0, \ldots, k \).

Example: Find binary representation of number 13:
Then \( u_0 = x = 13; n = 2 \)
\[
\begin{align*}
u_1 &= Q(13, 2) = 6; i_0 = R(13, 2) = 1 \\
u_2 &= Q(6, 2) = 3; i_1 = R(6, 2) = 0 \\
u_3 &= Q(3, 2) = 1; i_2 = R(3, 2) = 1 \\
u_4 &= Q(1, 2) = 0; i_3 = R(1, 2) = 1
\end{align*}
\]
Then \( w = 1101 \).
Thus \( k = 3 \) and we have
\[
x = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0
\]
You certainly noticed that in our string representation we used symbols $s_1, \ldots, s_n$, while in the everyday number representation we use $s_0, \ldots, s_{n-1}$.

So what exactly are the analogies and differences between the two systems?

To find out about this, let us look at the following modified set of digits $D$:

$$D = \{s_1, \ldots, s_{10}\} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, X\}$$

Here, the $X$ stands for a digit with the value 10, while there is no digit with the value 0.

So what is the number $x$ associated with the string $w = 76$?

$$x = 7 \cdot 10 + 6 = 76$$

And what is the number for $w = 3X6$?

$$x = 3 \cdot 100 + 10 \cdot 10 + 6 = 406$$

Finally, what is the number for $w = XX$?

$$x = 10 \cdot 10 + 10 = 110$$

These examples already suggest that we can use this system in a way quite similar to our "usual" system.

Now let us turn this around: What is the string $w$ associated with the number $x = 39$?

$w = 39$ (as long as $x$ does not contain any 0s, $w$ is the "usual" decimal string representing $x$)

And what is the string for $x = 100$?

$w = 9X$

But what is the string for $x = 504$?

$w = 4X4$

Finally, what is the string for $x = 0$?

$w = 0$ (0 is the null string symbol)

We can even transfer our elementary arithmetic to the new system:

$$\begin{array}{c}
X 4 \\
+ 5 9 6 \\
\hline
6 9 X \\
\end{array}$$

corresponding to

$$\begin{array}{c}
1 0 4 \\
+ 5 9 6 \\
\hline
7 0 0 \\
\end{array}$$

$$\begin{array}{c}
X 2 3 \\
- X 1 X \\
\hline
3 \\
\end{array}$$

corresponding to

$$\begin{array}{c}
1 0 2 3 \\
- 1 0 2 0 \\
\hline
3 \\
\end{array}$$