Outline

1 Recapitulation

2 Numerical representation of Strings (Words)

3 A List of Primitive Recursive Functions
We seek to extend computations from numbers to words on certain alphabets.

- An **alphabet** is a finite non-empty set of **symbols**.
- A **word** is an $n$-tuple of symbols $w = (a_1, a_2, \ldots, a_n)$ written as $a_1 a_2 \cdots a_n$. Here $n$ is the **length** of $w$ denoted by $n = |w|$.
- If $|A| = m$, there are $m^n$ words of length $n$.
- There is a unique word of length 0 denoted by 0.
The set of words over the alphabet $A$ is denoted by $A^*$. 

A *language* over the alphabet $A$ is any subset of $A^*$. 

We do not distinguish between the symbol $a$ and the word $a$. 

If $u$, $v$ are words, we write $uv$ for the word obtained by placing $v$ after $u$. 

**Example**

If $A = \{a, b, c\}$, $u = bab$, $v = caba$, then

$$uv = babcaba \text{ and } vu = cababab.$$ 

We have $u0 = 0u = u$ for every $u \in A^*$. 
The set of words over the alphabet $A$ is denoted by $A^*$.

A *language* over the alphabet $A$ is any subset of $A^*$.

We do not distinguish between the symbol $a$ and the word $a$.

If $u$, $v$ are words, we write $uv$ for the word obtained by placing $v$ after $u$.

**Example**

If $A = \{a, b, c\}$, $u = bab$, $v = caba$, then

$$uv = babcaba \text{ and } vu = cababab.$$
Word product is **associative**, that is,

\[ u(vw) = (uv)w \]

for \( u, v, w \in A^* \).

If either \( uv = uw \) or \( vu = wu \), then \( v = w \).

If \( u \) is a word and \( n > 0 \) we write

\[ u^n = uu \cdots u \]

and \( u^0 = \lambda \).
Let $A = \{s_1, \ldots, s_n\}$ be an alphabet that consists of $n$ symbols and let

$$w = s_{i_k} s_{i_{k-1}} \cdots s_{i_1} s_{i_0}$$

be a word in $A^*$. The integer associated with $w$ is

$$x = i_k \cdot n^k + i_{k-1} \cdot n^{k-1} + \cdots + i_1 \cdot n + i_0.$$ 

The integer associated with the null word 0 (the word without symbols) is 0.
Example

Let $A = \{s_1, s_2, s_3\}$ be an alphabet that consists of 3 symbols. The number associated with the word $s_2s_1s_1s_3s_1$ is

\[
x = 2 \cdot 3^4 + 1 \cdot 3^3 + 1 \cdot 3^2 + 3 \cdot 3^1 + 1 \\
= 2 \cdot 81 + 1 \cdot 27 + 1 \cdot 9 + 3 \cdot 3 + 1 = 208.
\]
When an alphabet, say $A = \{a, b, c\}$ is used, we assume that the symbols $a, b, c$ correspond to $s_1, s_2, s_3$. Then, the number that represents the word $w = baacb$ (which corresponds to $s_2 s_1 s_1 s_3 s_2$) is

$$2 \cdot 3^4 + 1 \cdot 3^3 + 1 \cdot 3^2 + 3 \cdot 3^1 + 2 = 209.$$
The representation of a word by a number is unique. This follows from the fact that we can retrieve the subscripts of the symbols from the numerical equivalent of the word.

Recall that:

- \( R(x, y) \) is the remainder when \( x \) is divided by \( y \).
- \( y \mid x \) is the predicate which is TRUE when \( y \) is a divisor of \( x \).
Define the **primitive recursive** functions

\[
R^+(x, y) = \begin{cases} 
R(x, y) & \text{if } \sim (y|x) \\
y & \text{otherwise,}
\end{cases}
\]

\[
Q^+(x, y) = \begin{cases} 
\lfloor x/y \rfloor & \text{if } \sim (y|x) \\
\lfloor x/y \rfloor - 1 & \text{otherwise.}
\end{cases}
\]

**Theorem**

*We have*

\[x = Q^+(x, y) \cdot y + R^+(x, y)\]

*and* \(0 < R^+(x, y) \leq y\).
Proof

The equality clearly holds as long as \( y \) is not a divisor of \( x \). If \( y \) divides \( x \) we have:

\[
\frac{x}{y} = \left\lfloor \frac{x}{y} \right\rfloor = \left( \left\lfloor \frac{x}{y} \right\rfloor \div 1 \right) + \frac{y}{y} = Q^+(x, y) + \frac{R^+(x, y)}{y}.
\]

This differs from ordinary division with reminders in that the “remainders” are permitted to take values between 1 and \( y \) rather than between 0 and \( y - 1 \).
Now, let $u_0 = x$ and $u_{m+1} = Q^+(u_m, n)$

Since we have

\[
\begin{align*}
  u_0 &= i_k \cdot n^k + i_{k-1} \cdot n^{k-1} + \cdots + i_1 \cdot n + i_0 \\
  u_1 &= i_k \cdot n^{k-1} + i_{k-1} \cdot n^{k-2} + \cdots + i_1 \\
  &\quad \vdots \\
  u_k &= i_k,
\end{align*}
\]

it follows that $i_m = R^+(u_m, n)$ for $0 \leq m \leq k$. 
To summarize the previous cases for computing $R^+(x, y)$ and $Q^+(x, y)$ we write:

<table>
<thead>
<tr>
<th>$y$ divides $x$</th>
<th>$y$ does not divide $x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^+(x, y) = y$</td>
<td>$R^+(x, y) = R(x, y)$</td>
</tr>
<tr>
<td>$Q^+(x, y) = \lfloor x/y \rfloor \div 1$</td>
<td>$Q^+(x, y) = \lfloor x/y \rfloor$.</td>
</tr>
</tbody>
</table>
Example

Let $S = \{s_1, s_2, s_3\}$ be an alphabet. Let us determine the word that has the numerical equivalent 208. We have $u_0 = 208$.

\[
i_0 = R^+(208, 3) = 1 \text{ since } \sim 3|208 \text{ and } u_1 = \lfloor 208/3 \rfloor = 69
\]
\[
i_1 = R^+(69, 3) = 3 \text{ since } 3|69 \text{ and } u_2 = \lfloor 69/3 \rfloor = 1 = 22
\]
\[
i_2 = R^+(22, 3) = 1 \text{ since } \sim 3|22 \text{ and } u_3 = \lfloor 22/3 \rfloor = 7
\]
\[
i_3 = R^+(7, 3) = 1 \text{ since } \sim 3|7 \text{ and } u_4 = \lfloor 7/3 \rfloor = 2
\]
\[
i_4 = R^+(2, 3) = 2
\]

Thus, the word we sought is $x = s_2s_1s_1s_3s_1$. 
To compute $u_{m+1}$ as $u_{m+1} = Q^+(u_m, n)$ we use the function $g(m, n, x) = u_m$. This function is primitive recursive because

$$g(0, n, x) = x,$$
$$g(m + 1, n, x) = Q^+(g(m, n, x), n).$$

If we let $h(m, n, x) = R^+(g(m, n, x), n)$, then $h$ is also primitive recursive and $i_m = h(m, n, x)$ for $0 \leq m \leq k$. 
Definition

Given the alphabet $A$ that consists of $s_1, \ldots, s_n$ in this order, the word $w = s_{i_k} s_{i_{k-1}} \cdots s_{i_1} s_{i_0}$ is the base $n$ notation for the number $x$, where

$$x = i_k \cdot n^k + i_{k-1} \cdot n^{k-1} + \cdots + i_1 \cdot n + i_0.$$ 

Note that 0 is the base $n$ notation for the null string for every $n$. This allows us to introduce the notion of $m$-ary partial function on $A^*$ with values in $A^*$ as being partially computable, or when is total, of being computable.
Subsets of $A^*$ are languages over the alphabet $A$. By associating numbers with the words of $A^*$ we can talk about recursive sets or $r.e.$ sets.

Let $A$ be an alphabet with $|A| = n$, say $A = \{s_1, \ldots, s_n\}$.

**Definition**

For $m \geq 1$ let $\text{CONCAT}_n^{(m)}$ be defined as

\[
\text{CONCAT}_n^{(1)}(u) = u,
\]
\[
\text{CONCAT}_n^{(m+1)}(u_1, \ldots, u_m, u_{m+1}) = z u_{m+1},
\]

where $z = \text{CONCAT}_n^{(m)}(u_1, \ldots, u_m)$.

Thus, for $u_1, \ldots, u_m$, $\text{CONCAT}_n^{(m)}(u_1, \ldots, u_m)$ is the string obtained by placing the strings $u_1, \ldots, u_m$ one after another. The superscript is usually omitted so we write

\[
\text{CONCAT}(s_2 s_1, s_1 s_1 s_2) = s_2 s_1 s_1 s_1 s_2.
\]
If we need to consider CONCAT as defining functions on $\mathbb{N}$, then note that:

- the string $s_2s_1$ in base 2 is $2 \cdot 2^1 + 1 = 5$;
- the string $s_1s_1s_2$ in base 2 is $1 \cdot 2^2 + 1 \cdot 2^1 + 2 = 8$;
- the string $s_2s_1s_1s_1s_2$ in base 2 is $2 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 2 = 48$.

This allows us to write

$$\text{CONCAT}_2(5, 8) = 48.$$
Example

The length function $f(u) = |u|$ defined on $A^*$ and taking values in $\mathbb{N}$.

For each $x$, the number $\sum_{j=0}^{x} n^j$ has the base $n$ representation $s_1^{x+1}$; hence, this number is the smallest number whose base $n$ representation contains $x + 1$ symbols.
Example

The function $\text{CONCAT}_n(u, v)$ is primitive recursive because

$$\text{CONCAT}_n(u, v) = u \cdot n^{|v|} + v.$$
Example

The function $\text{CONCAT}^{(m)}_{n}(u, v)$ is primitive recursive for each $m, n \geq 1$. This follows from

\[
\begin{align*}
\text{CONCAT}^{(1)}_{n}(u) &= u, \\
\text{CONCAT}^{(m+1)}_{n}(u_1, \ldots, u_m, u_{m+1}) &= zu_{m+1},
\end{align*}
\]

where $z = \text{CONCAT}^{(m)}_{n}(u_1, \ldots, u_m)$. 
Example

The function $\text{RTEND}_n(w)$ which gives the rightmost symbol of a non-empty word $w$ is primitive recursive because

$$\text{RTEND}_n(w) = h(0, n, w),$$

where $h(0, n, x) = R^+(g(0, n, x))$, previously defined.
Example

The function $\text{LTEND}_n(w)$ which gives the leftmost symbol of a non-empty word $w$ is primitive recursive because

$$\text{LTEND}_n(w) = h(|w| - 1, n, w).$$
Example

The function \( \text{RTRUNC}_n(w) \) which gives the result of removing the rightmost symbol from a given non-empty word is primitive recursive because

\[
\text{RTRUNC}_n(w) = g(1, n, w).
\]

An alternative notation for \( \text{RTRUNC}_n(w) \) is \( w^- \).
Example

The function $\text{LTRUNC}_n(w)$ which gives the result of removing the leftmost symbol from a given non-empty word is primitive recursive because

$$\text{LTRUNC}_n(w) = w - i_k \cdot n^k.$$
Next, we discuss a pair of functions $\text{UPCHANGE}_{n,\ell}$ and $\text{DOWNCHANGE}_{n,\ell}$ that can be used to change base.

Let $A$ be an alphabet with $n$ symbols and $A'$ be an alphabet with $\ell$ symbols, where $1 \leq n < \ell$. A string that belongs to $A^*$ also belongs to $(A')^*$.

If $x \in \mathbb{N}$ and $w \in A^*$ is the word that represents $x$ in basis $n$, then $\text{UPCHANGE}_{n,\ell}(x)$ is the number which $w$ represents in basis $\ell$. 
Theorem

Let $0 < n < \ell$. Then the function $\text{UPCHANGE}_{n,\ell}$ and $\text{DOWNCHANGE}_{n,\ell}$ are computable.
Proof.

The next program computes $\text{UPCHANGE}_{n,\ell}$ by extracting the symbols of a word that the given number represents in basis $n$ and uses them to compute the number that the given word represents in basis $\ell$:

\[
\begin{align*}
[A] & \quad \text{IF } X = 0 \text{ GOTO E} \\
& \quad Z \leftarrow \text{LTEND}_n(X) \\
& \quad X \leftarrow \text{LTRUNC}_n(X) \\
& \quad Y \leftarrow \ell \cdot Y + Z \\
& \quad \text{GOTO A}
\end{align*}
\]
Proof cont’d

For $\text{DOWNCHANGE}_{n,\ell}$ the program will extract the symbols of the word that the given number represents in the base $\ell$. These symbols will be added only if they belong to the smaller alphabet:

[A] \hspace{1em} \text{IF } X = 0 \text{ GOTO } E
\hspace{1em} Z \leftarrow \text{LTEND}(X)
\hspace{1em} X \leftarrow \text{LTRUNC}(X)
\hspace{1em} \text{IF } Z > n \text{ GOTO } A
\hspace{1em} Y \leftarrow n \cdot Y + Z
\hspace{1em} \text{GOTO } A