Numerical Representation of Strings

2.) $g(u, v) = \text{CONCAT}_n(u, v)$
This function is primitive recursive because it is defined by the following equation:
$\text{CONCAT}_n(u, v) = u \cdot n^{|v|} + v$

Example: Alphabet $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, X\}$:
$\text{CONCAT}_{10}(13, 478) = 13 \cdot 1000 + 478 = 13478$

3.) $\text{CONCAT}^{(m)}(u_1, \ldots, u_m)$
This function is primitive recursive for each $m, n \geq 1$.
It follows at once from the previous example using composition.

Example:
Alphabet $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, X\}$, $m = 3$:
$\text{CONCAT}_{10}(u, v, w) = u \cdot n^{|v| + |w|} + v \cdot n^{|w|} + w$
$\text{CONCAT}_{10}(13, 478, 9) = 13 \cdot 10000 + 478 \cdot 10 + 9 = 134789$

4.) $\text{RTEND}_n(w) = h(0, n, w)$
Remember:
$i_m = h(m, n, x), \ m = 0, \ldots, k.$
$\text{RTEND}_n$ gives the rightmost symbol of a given word.

We know that $h$ is primitive recursive, so $\text{RTEND}_n$ is also primitive recursive.

5.) $\text{LTEND}_n(w) = h(|w| - 1, n, w)$
Corresponding to $\text{RTEND}_n$, $\text{LTEND}_n$ gives the leftmost symbol of a given word.

Example:
Alphabet $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, X\}$:
$\text{LTRUNC}_{10}(3478) = 3478 - 3 \cdot 100 = 478.$

6.) $\text{RTRUNC}_n(w) = g(1, n, w)$
Remember:
$g(m, n, x) = u_m$
$u_0 = i_0 \cdot n^m + i_1 \cdot n^{m-1} + \ldots + i_m \cdot n + i_0$
$u_1 = i_0 \cdot n^{m-1} + i_1 \cdot n^{m-2} + \ldots + i_1$
$\text{RTRUNC}_n$ gives the result of removing the rightmost symbol from a given nonempty string.
When we can omit the reference to the base $n$, we often write $w^-$ for $\text{RTRUNC}_n(w)$.
Note that $0^- = 0$.

7.) $\text{LTRUNC}_n(w) = w^- \cdot (\text{LTEND}_n(w) \cdot n^{|w| - 1})$
$\text{LTRUNC}_n$ gives the result of removing the leftmost symbol from a given nonempty string.

Example: Alphabet $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, X\}$:
$\text{LTRUNC}_{10}(3478) = 3478 - 3 \cdot 1000 = 478.$
Numerical Representation of Strings

We will use these newly introduced primitive recursive functions to prove the computability of a pair of functions that can be used in changing base.

Let \( 1 \leq n < l \).
Let \( A \subseteq \bar{A} \), where \( A \) is an alphabet of \( n \) symbols and \( \bar{A} \) is an alphabet of \( l \) symbols.
So whenever a string belongs to \( A^* \), it also belongs to \( \bar{A}^* \).

April 25, 2019
Theory of Computation
Lecture 20: Calculations on Strings V

Numerical Representation of Strings

For any \( x \in \mathbb{N} \), let \( w \) be the word in \( A^* \) that represents \( x \) in base \( n \).
Then we write \( \text{UPCHANGE}_{n,l}(x) \) for the number which \( w \) represents in base \( l \).

Examples:
\( \text{UPCHANGE}_{2,6}(5) = 13 \)
The representation of 5 in base 2 is \( s_2s_1 \). In base 6, \( s_2s_1 \) represents the number 13.
\( \text{UPCHANGE}_{1,5}(3) = 31 \)
The representation of 3 in base 1 is \( s_1s_1s_1 \). In base 5, \( s_1s_1s_1 \) represents the number 31.

April 25, 2019
Theory of Computation
Lecture 20: Calculations on Strings V

Numerical Representation of Strings

Correspondingly, we define the following:
For \( x \in \mathbb{N} \), let \( w \) be the string in \( \bar{A}^* \) which represents \( x \) in base \( l \).
Let \( w' \) be obtained from \( w \) by crossing out all of the symbols that belong to \( \bar{A} - A \), so \( w' \in A^* \).
We write \( \text{DOWNCHANGE}_{n,l}(x) \) for the number which \( w' \) represents in base \( n \).

Example:
\( \text{DOWNCHANGE}_{2,6}(109) = 5 \)
The representation of 109 in base 6 is \( s_2s_6s_1 \). We cross out the \( s_6 \) and get \( s_2s_1 \). In base 2, \( s_2s_1 \) represents the number 5.

April 25, 2019
Theory of Computation
Lecture 20: Calculations on Strings V

Numerical Representation of Strings

UPCHANGE_{n,l} and DOWNCHANGE_{n,l} are actually primitive recursive functions, but now we will just show that they are computable.
This program computes UPCHANGE_{n,l}:

\[ [A] \]
\[
\text{IF } X = 0 \text{ GOTO E} \\
Z \leftarrow \text{LTEND}_{n}(X) \quad \text{// Z receives leftmost symbol} \\
X \leftarrow \text{LTRUNC}_{n}(X) \quad \text{// removes this symbol from x} \\
Y \leftarrow n \cdot Y + Z \quad \text{// add to output} \\
\text{GOTO A} \\
\]

April 25, 2019
Theory of Computation
Lecture 20: Calculations on Strings V

Numerical Representation of Strings

Let us look at the computation of \( \text{UPCHANGE}_{2,10}(12) \) iteration by iteration (string \( s_2s_1s_2 \)):

\[ [A] \]
\[
\text{IF } X = 0 \text{ GOTO E} \\
Z \leftarrow \text{LTEND}_{10}(X) \quad \text{// Z receives leftmost symbol} \\
X \leftarrow \text{LTRUNC}_{10}(X) \quad \text{// removes this symbol from x} \\
Y \leftarrow I \cdot Y + Z \\
\text{GOTO A} \\
\]

Result: 212

April 25, 2019
Theory of Computation
Lecture 20: Calculations on Strings V

Numerical Representation of Strings

This program computes DOWNCHANGE_{n,l}:

\[ [A] \]
\[
\text{IF } X = 0 \text{ GOTO E} \\
Z \leftarrow \text{LTEND}_{n}(X) \quad \text{// Z receives leftmost symbol} \\
X \leftarrow \text{LTRUNC}_{n}(X) \quad \text{// removes this symbol from x} \\
\text{IF } Z > n \text{ GOTO A} \quad \text{// check if we must cross out Z} \\
Y \leftarrow n \cdot Y + Z \\
\text{GOTO A} \\
\]

April 25, 2019
Theory of Computation
Lecture 20: Calculations on Strings V
A Progr. Language for String Computations

The instructions in our programming language $L$ do not seem well-designed for string computations. Let us consider the instruction type $V \leftarrow V + 1$. For example, if we use the alphabet $\{a, b, c, d\}$, and the variable $V$ contains the number corresponding to the string \texttt{cadd}, what will this instruction do? It will turn \texttt{cadd} into \texttt{cbaa}.

Why would we want to have this as one of our fundamental operations in the programming language?

To improve this situation, we will now introduce specific programming languages $L_n$ for computations on alphabets of size $n$ for all $n > 0$. All variables will be the same as in $L$, except that we now consider their values to be from $A^*$ with $|A| = n$.

The use of labels, formal rules of syntax, and macro expansions will also be identical to $L$.

An $m$-ary partial function on $A^*$ which is computed by a program in $L_n$ is said to be partially computable in $L_n$. If the function is total and partially computable in $L_n$, it is called computable in $L_n$.

A Progr. Language for String Computations

Instruction Interpretation

$V \leftarrow \sigma V$ (with $\sigma \in A$) Place the symbol $\sigma$ to the left of the string which is the value of $V$.

$V \leftarrow V -$ Delete the final symbol of the string which is the value of $V$.

If the value of $V$ is 0, leave it unchanged.

IF $V$ ENDS $\sigma$ GOTO L If the value of $V$ ends in the symbol $\sigma$, continue with the first instruction labeled L; otherwise proceed to the next instruction.

A Progr. Language for String Computations

Now let us look at some useful macros for use in $L_n$ and their expansions:

1. The macro IF $V\neq 0$ GOTO L has the expansion

   IF $V$ ENDS $s_1$ GOTO L
   IF $V$ ENDS $s_2$ GOTO L
   
   IF $V$ ENDS $s_n$ GOTO L

2. The macro $V \leftarrow 0$ has the expansion

   \[
   [A] \quad V \leftarrow V \\
   \text{IF } V\neq 0 \text{ GOTO A}
   \]

3. The macro GOTO L has the expansion

   $Z \leftarrow 0$
   $Z \leftarrow s_1 Z$
   \text{IF } Z \text{ ENDS } s_1 \text{ GOTO L}

Although the instructions of $L_n$ refer to strings, we can still think of them as referring to the numbers associated with the strings.

For example, given an alphabet with $n$ symbols, the instruction

$V \leftarrow s_i V$

has the effect of replacing the current value of the variable $V$ with the value $i \cdot n^{|V|} + V$.

So you see that the "natural" string operations in $L_n$ have complex numerical counterparts.