ECE 664
Lecture 10
Avi KAK

Calculation on Strings

- We want to extend our computability theory to strings on an alphabet A. One way to do this is to associate numbers with the words in A^*. In Lecture 2, we defined A^* as the set of all words that can be made with the symbols of the alphabet A.

- When we associate numbers with words, we must make sure that every word maps to a unique integer. This can be achieved by imposing an order on the elements of A:

$$A = \{a, b, c, d\}$$

order on A : $a \to 1$, $b \to 2$, $c \to 3$, $d \to 4$

If we MUST have 0 in A: $A = \{0, 1\}$, $0 \to s_1$, $1 \to s_2$
$w = 110 \Rightarrow 2\,2\,1$
$x = 2\times2^2 + 2\times2 + 1 = 13$

To express A in its ordered form, we write it as $A = \{s_1, s_2, s_3, s_4\}$ where it is understood that $s_1 = a$, $s_2 = b$, $s_3 = c$, and $s_4 = d$. Now given a word $w = s_{i_k} s_{i_{k-1}} \cdots s_{i_1} s_{i_0}$, we associate with it the number x given by

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

$n = |A|$

Example: $A = \{a, b, c, d\}$ $w = dbc \equiv s_4 s_2 s_3$. So $x = 4\times4^2 + 2\times4^1 + 3 = 75$

- To show the uniqueness of this representation, we must show how to retrieve the subscripts $i_0, i_1, \ldots$ from a given decimal integer x. The procedure consists of dividing x by n and setting i_0 to the remainder, with the caveat that if the remainder is 0, we set it to n and, at the same time, decrement the value of the quotient by 1. To see the need for decrementing the quotient when the remainder is 0, try finding the base-4 ($n=4$) string representation of the decimal integer 64. Your answer should be 334 assuming the alphabet is $A = \{1, 2, 3, 4\}$. You stop the procedure when the quotient is 0 either directly or after it's been decremented.

- The caveat mentioned above is needed because we do <u>not</u> use 0 in mapping the symbols of the alphabet to integers. Shown below is the difference between a base-4 representation using 0 and a base-4 representation that does NOT use zero:

Base-4 representation using zero $A = \{0, 1, 2, 3\}$																
N:	0	1	2	3	4	5	6	7	8	9	10	11	12	13		
$w \in A^*$:	0	1	2	3	10	11	12	13	20	21	22	23	30	31		

Base-4 representation of numbers <u>not</u> using 0 $A = \{1, 2, 3, 4\}$																
N:	~~0~~	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$w \in A^*$:		1	2	3	4	11	12	13	14	21	22	23	24	31	32	33	34

- When $n = |A|$, we can obviously think of $(i_k, i_{k-1}, \ldots, i_1, i_0)$ as the base-n representation of the word $w = s_{i_k} s_{i_{k-1}} \cdots s_{i_1} s_{i_0}$. The decimal (base-10) value of the base-n integer is x.

- Note that we could <u>not</u> have allowed the digit 0 in our ordering of the alphabet A. That is, we could <u>not</u> have done:

$$A = \{a, b, c, d\}$$

order on A : $a \to 0$, $b \to 1$, $c \to 2$, $d \to 3$

Now the decimal integer associated with the string 'ab' would be the same as with 'aab', or 'aaaab'. Thus the uniqueness in mapping strings to numbers or vice versa would be lost.

- Even though 0 is not ~~an ordering~~ in the mapping of the symbols of A to the ordering integers, we will use 0 to represent the empty string. We can even claim that the number associated with the <u>empty string</u> is 0.

- The upshot is that the set A^* of strings can be thought of as a set of numbers. We can think of these numbers either in the base-n representation with $n = |A|$ or as ordinary decimal integers.

- If the set A^* of strings is equivalently a set of numbers, we can use our programming language $\mathcal{S}$ to carry out all of the computing on A^*.
- We can therefore talk about computable and partially computable functions on A^*. We can also talk about primitive recursive functions on A^*. We can also talk about predicates on A^*. These map words to either s_1 (since s_1 represents 1 in base n) or to the distinguished symbol 0. We can also talk about a language $L \subseteq A^*$ being recursive or recursively enumerable.
- Here is using the programming language $\mathcal{S}$ to define computable (primitive recursive) functions on A^*:

① $f(u) = |u|$ (the length of the string u)

This function is defined on A^* but has values in N:

$$|u| = \min_{x \le u} \left[\sum_{j=0}^{x} n^j > u \right]$$

② $g(u,v) = \text{CONCAT}_n(u,v)$ (concatenation of the strings u and v)

$$= u \cdot n^{|v|} + v$$

③ $\text{RTEND}_n(w)$ returns the rightmost symbol of w

④ $\text{LTEND}_n(w)$ returns the leftmost symbol

⑤ $\text{RTRUNC}_n(w)$ gives the result of removing the rightmost symbol.

etc.

- Although we can use $\mathcal{S}$ to do all the computations on A^*, it is obviously not a convenient language for string processing. Let's therefore define languages that are more natural for string processing

---

A Programming Language for String Computations

With $n = |A|$, we define a language $\mathcal{S}_n$ as follows:

instruction set of S_n

$V \leftarrow \sigma V$	: place symbol σ to the left of the string value of V for each $\sigma \in A$
$V \leftarrow V^-$	: delete the rightmost symbol of the string value of V. If $V = 0$ (empty string), leave V unchanged.
IF V ENDS σ GOTO L	: for each $\sigma \in A$

In order to demonstrate that we can do <u>all</u> of the number-based computing (that we carried out previously with $\mathcal{S}$) with programs written in $\mathcal{S}_n$, it is convenient to first define the following macros for use with $\mathcal{S}_n$:

① $\boxed{\text{IF } V \neq 0 \text{ GOTO L}}$

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

② $\boxed{V \leftarrow 0}$

[A] $V \leftarrow V^-$
IF $V \neq 0$ GOTO A

③ $\boxed{\text{GOTO L}}$

$Z \leftarrow 0$
$Z \leftarrow s_1 Z$
IF Z ENDS s_1 GOTO L

④ $\boxed{V' \leftarrow V}$

⑤ $\boxed{V \leftarrow f(V_1, \ldots, V_m)}$

for any $f()$ that is known to be partially computable in $\mathcal{S}_n$

⑥ $\boxed{V \leftarrow V + 1}$

Increment by one the integer value of the string assigned to V.

⑦ $\boxed{V \leftarrow V \dot{-} 1}$

Decrement by one the integer value of the string assigned to V if that integer value is greater than 0. Otherwise, leave V unchanged.

On the strength of the macros ⑥ and ⑦ we can state the following theorem:

> If a function is partially computable (in $\mathcal{S}$), then it is also partially computable in $\mathcal{S}_n$ for every n.

Note the interesting aside that the languages $\mathcal{S}$ and $\mathcal{S}_1$ are one and the same. The numerical effect of the instructions $V \leftarrow s_1 V$ and $V \leftarrow V^-$ in $\mathcal{S}_1$ is the same as that of $V \leftarrow V+1$ and $V \leftarrow V-1$ in $\mathcal{S}$.