Lecture 7: 4/21/2009

Announcements: Ps4 out today,
Ps3 solutions out tomorrow;

Formal Languages (Chapter 12): This topic is discussed in much greater detail in ECS120.

0 is both a character and a string: $0 \in \Sigma$
ε is only a string it is not a character.
01 is likewise a string, it is not a character.

Language – A set of strings, all over the same alphabet.

$\Sigma = \{0,1\}$

This is a language. It is also an alphabet.
All of the following are examples of languages:

$L = \{\varepsilon, 00, 01, 10, 11, 0000, 0001 \cdots\} = \{x \in \Sigma^* : |x| \text{ is even}\}$

$L = \{1^P : \text{p is prime}\} = \{11, 111, 11111, 1111111, \cdots\}$

$L = \{x \in \{0,1\}^* :$ x represents a prime number, no leading zeroes, written in binary$\}$

$= \{10, 11, 101, 111, 1011\}$

L={dog, cat, fish}

Languages can be finite or infinite.

Languages are sets (of strings), so set operators apply to languages. For example:

$L_1 = \{\text{dog,cat,fish}\}$

$L_2 = \{\text{dog,frog}\}$

$|L_1 \cup L_2| = 4$

$|L_1 \setminus L_2| = 2$

$|L_1 \oplus L_2| = 3$

The concatenate function can be lifted to apply to languages; when L_1 and L_2 are languages, $L_1 \ L_2$ is defined as all the combinations of a string from L_1followed by a string from L_2. That is,

$$L_1 L_2 = L_1 \circ L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

With the set we have written above,
$L_1 L_2 = \{\text{dogdog,dogfrog,catdog,catfrog,fishdog,fishfrog}\}$

True or False: $\varnothing$ is a language.
True – all its elements are strings (that is, all 0 of them).

$L\varnothing = \varnothing L = \varnothing$

$\{\varepsilon\}$ is also a language – the singleton language containing just the empty string. Clearly

$L\{\varepsilon\} = L$ as
$x\varepsilon = \varepsilon x = x$

Student question:: is ε in every language?
Answer, No, we can choose to have it in a given language or not.

$L = \{1^i : i \text{ is even}\} = \{\varepsilon, 11, 1111, 111111\}$

$x^0 = \varepsilon$
We can write L^2 for $L\ L$, and L^3 for $L\ L\ L$, etc. For the example just given, when L is all the even length strings of 1's, what is L^2 and L^3 . Just L.

True or False: if L contains the empty string, then LL' contains all of L'.
True.

Taking it a step further, we can represent all the strings formed by concatenating strings from the language as $L^* = \bigcup_{i \geq 0} L^i$

In order for that definition to make sense, we need to define L^0. $L^0 = \{\varepsilon\}$ is a convenient way to do this, since then, *L** always contains the empty sting. Also, then $L^{m+1} = \mathrm{L}^m\mathrm{L}$ works even when m=0.

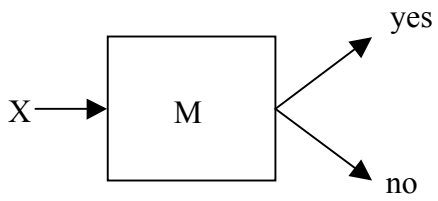
Say *L*={dog,cat}. Then *L** ={ ε, cat, dog, catcat, catdog, dogcat, dogdog, …}

$$\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000\cdots\}$$

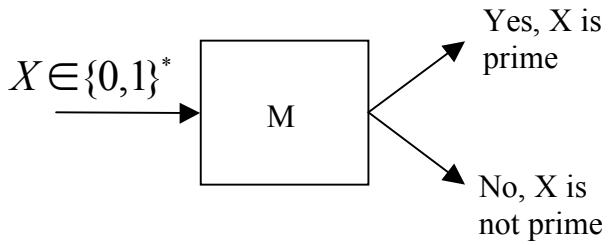
Just what we said earlier, but now seen in a more general light.

Why are computer scientists interested in languages?

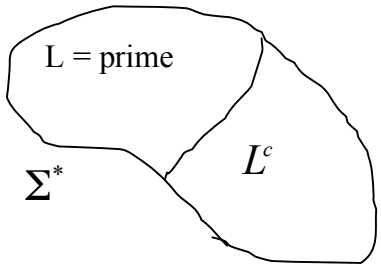
We can imagine a model for a computer program with an input of a string *X*. We define the language associated with the program by saying that $x \in L \Leftrightarrow M(X)$ says "yes"



That is to say, the computer program could decide if an input was part of the language. In this way, Languages correspond to decision questions. For example, a computer could take the input of a language X, and decide if the string represented by *X* is prime. It could also match a question more naturally about a language: is X a valid C program (then M is a compiler), or is X a valid sentence in English?



We can illustrate this as follows:



The program is splitting the universe into two sets: those it says "yes" to (they are in the language) and those that it says "no" to (they are not in the languge).

When the language *L* is finite, it is conceptually easy to write a program to decide it: just look it up in a table.

Other languages seem easy, too. For example,

$L_1 = \{1^i : i \text{ is even}\}$ is an "easy" language to make a machine to decide. One sense in which it is easy is that there's a short piece of notation that a machine could interpret that would describe that language. The short piece of notation I have in mind is this: $(11)^*$. This notation is supposed to mean "the string 11, repeated any number of times".

$L_2 = \{1^P : p \text{ is prime}\}$ is a much harder language to describe. It doesn't seem like we could describe it by a short string using symbols like concatenation and union and this star operator. The set of C programs or English sentences is even harder to describe.

Regular Languages (12.4)

We will consider the kinds of languages that can be described using the following special symbols:
( ) $\cup$ * $\circ$

We also allow symbols from some underlying alphabet, and the empty string symbol. What we have just described is the vocabulary we will use for **regular expression** of languages. The "meaningful" strings over these symbols denote language according to natural rules: For regular expressions α and β, we can define languages of more complicated regular expressions:

$$L((\alpha \cup \beta)) = L(\alpha) \cup L(\beta)$$
$$L((\alpha \circ \beta)) = L(\alpha)L(\beta)$$
$$L((\alpha^*)) = (L(\alpha))^*$$

Also, the language associated with a symbol from the alphabet denotes that singleton language, and the empty string symbol denotes that singleton language.

We can use this more specifically to describe some languages with nice compact expressions. In our earlier example, (11)* denotes the language:
$(\{1\} \circ \{1\})^* = \{11\}^* = L_1$

Here's another example:

$$0(0 \cup 1)^* 0 = \{x \in \{0,1\}^* : \text{ x starts and ends with a "0" and } |x| \geq 2\}$$

If we didn't write the part about the length of x then the right hand side would include the string 0 but the left-hand side would not.

What if we wanted to make a regular expression for a binary string that started and ended with the same character, or with no character at all? That would be

$$\varepsilon \cup 1 \cup 1(0 \cup 1)^* 1 \cup 0 \cup 0(0 \cup 1)^* 0$$

Formally, given an alphabet Sigma, the following are the regular expressions over Sigma:

-a, for all $a \in \sum$
- ε
-∅
-(x o y), (x u y), (x*) for any regular expressions x,y

Omit parenthesis with the understanding that o binds most strongly, then *, then u, and things group left-to-right,

L(R) - the language of a regular expression R - defined recursively in the natural way.

Exercise: write a regular expression for all strings over {0,1} that contain an even # of 0's and an even # of 1's.

Hmm..... sounds HARD!

Can it be done?!