University of Basrah

Faculty of Education for Pure Sciences

Computer Department

LANGUAGES

In English we distinguish the three different entities: letters, words, and sentences. There is a certain parallelism between the fact that groups of letters make up words and the fact that groups of words make up sentences. Not all collections of letters form a valid word, and not all collections of words form a valid sentence. The analogy can be continued. Certain groups of sentences make up coherent paragraphs, certain groups of paragraphs make up coherent stories, and so on.

This situation also exists with computer languages. Certain character strings are recognizable words (GOTO, END ...). Certain strings of words are recognizable commands. Certain sets of commands become a program (with or without data).

To construct a general theory that unifies all these examples, it is necessary for us to adopt a definition of a “most universal language structure,” that is, a structure in which the decision of whether a given string of units constitutes a valid larger unit is not a matter of guesswork but is based on explicitly stated rules.

It is very hard to state all the rules for the language “spoken English,” since many seemingly incoherent strings of words are actually understandable utterances. This is due to slang, idiom, dialect, and our ability to interpret poetic metaphor and to correct unintentional grammatical errors in the sentences

However, as a first step to defining a general theory of abstract languages, it is right for us to insist on precise rules, especially since computers are not quite as forgiving about imperfect input commands as listeners are about informal speech.

When we call our study the Theory of Formal Languages, the word “for- mal” refers to the fact that all the rules for the language are explicitly stated in terms of what strings of symbols can occur. No liberties are tolerated, and no reference to any “deeper understanding” is required. Language will be considered solely as symbols on paper and not as expressions of ideas in the minds of humans. In this basic model, language is not communication among intellects, but a game of symbols with formal rules. The term “formal” used here emphasizes that it is the form of the string of symbols we are interested in, not the meaning.

We begin with only one finite set of fundamental units out of which we build structures. We shall call this the alphabet. A certain specified set of strings of characters from the alphabet will be called the language. Those strings that are permissible in the language we call words. The symbols in the alphabet do not have to be Latin letters, and the sole universal requirement for a possible string is that it have only finitely many symbols in it.

We shall wish to allow a string to have no letters. This we call the empty string or null string, and we shall denote it by the symbol Λ or ε. No matter what language we are considering, the null string is always Λ. Two words are considered the same if all their letters are the same and in the same order so there is only one possible word of no letters. For clarity, we do not allow the symbol Λ to be part of the alphabet for any language.

The most familiar example of a language for us is English. The alphabet is the usual set of letters plus the apostrophe and hyphen. Let us denote the whole alphabet by the Greek letter capital sigma (Σ).

$$\Sigma = \{ a\ b\ c\ d\ \ldots\ldots z\ '\ -\ \}$$

Sometimes we shall list a set of elements separated by spaces and sometimes by commas. If we wished to be super meticulous, we would also include in Σ the uppercase letters and the seldom used diacritical marks.

We can now specify which strings of these letters are valid words in our language by listing them all, as is done in a dictionary. It is a long list, but a finite list, and it makes a perfectly good definition of the language. If we call this language ENGLISH-WORDS we may write

ENGLISH-WORDS = (all the words (main entries) in a standard dictionary}

In the line above, we have intentionally mixed mathematical notation (the equal sign, the braces denoting sets) and a prose phrase. This results in perfectly understandable communication; we take this liberty throughout. All of our investigations will be agglomerates of informal discussion and precise symbolism.

Of course, the language ENGLISH-WORDS, as we have specified it, does not have any grammar. If we wish to make a formal definition of the language of the sentences in English, we must begin by saying that this time our basic alphabet is the entries in the dictionary. Let us call this alphabet Γ, the capital gamma.

Γ ={the entries in a standard dictionary, plus a blank space, plus the usual punctuation marks}

In order to specify which strings of elements from Γ produce valid words in the language ENGLISH-SENTENCES, we must rely on the grammatical rules of English. This is because we could never produce a complete list of all possible words in this language; that would have to be a list of all valid English sentences. Theoretically, there are infinitely many different words in the language ENGLISH-SENTENCES. For example:

I ate one apple.

I ate two apples.

I ate three apples.

The trick of defining the language ENGLISH-SENTENCES by listing all the rules of English grammar allows us to give a finite description of an infinite language.

If we go by the rules of grammar only, many strings of alphabet letters seem to be valid words, for example, “I ate three Tuesdays.” In a formal language we must allow this string. It is grammatically correct; only its meaning reveals that it is ridiculous. Meaning is something we do not refer to in formal languages. we are primarily interested in syntax alone, not semantics or diction. We shall be like the bad teacher who is interested only in the correct spelling, not the ideas in a home- work composition.

In general, the abstract languages we treat will be defined in one of two ways. Either they will be presented as an alphabet and the exhaustive list of all valid words, or else they will be presented as an alphabet and a set of rules defining the acceptable words.

Earlier we mentioned that we could define a language by presenting the alphabet and then specifying which strings are words. The word “specify” is trickier than we may at first suppose. Consider this example of the language called MY-PET. The alphabet for this language is

{a c d g o t}

There is only one word in this language, and for our own perverse reasons we wish to specify it by this sentence:

If the Earth and the Moon ever collide, then MY-PET = {cat}

but, if the Earth and the Moon never collide, then MY-PET = {dog}

One or the other of these two events will occur, but at this point in the history of the universe it is impossible to be certain whether the word dog is or is not in the language MY-PET.

This sentence is not an adequate specification of the language MY-PET because it is not useful. To be an acceptable specification of a language, a set of rules must enable us to decide, in a finite amount of time, whether a given string of alphabet letters is or is not a word in the language.

The set of rules can be of two kinds. They can either tell us how to test a string of alphabet letters that we might be presented with, to see if it is a valid word; or they can tell us how to construct all the words in the language by some clear procedures.

Let us consider some simple examples of languages. If we start with an alphabet having only one letter, the letter x, $\Sigma = \{x\}$

we can define a language by saying that any nonempty string of alphabet characters is a word.

$$L1 = \{x \ \ xx \ \ xxx \ xxxx \ \ ...\}$$

or to write this in an alternate form $L1 = \{x^n \text{ for } n = 1\ 2\ 3\ \ldots\}$

In this language, as in any other, we can define the operation of concatenation, in which two strings are written down side by side to form a new longer string. In this example, when we concatenate the word xxx with the word xx, we obtain the word xxxxx. The words in this language are clearly analogous to the positive integers, and the operation of concatenation is analogous to addition:

x^n concatenated with x^m is the word x^{n+m}

It will often be convenient for us to designate the words in a given language by new symbols, that is, other than the ones in the alphabet. For example, we could say that the word xxx is called a and that the word xx is b. Then to denote the word formed by concatenating a and b we write the letters side by side:

ab = xxxxx

It is not always true that when two words are concatenated, they produce another word in the language. For example, if the language is

$$L_2 = \{ x \;\; xxx \;\; xxxxx \;\; xxxxxxx \ldots\}$$
$$= \{ x^{odd} \}$$
$$= (x^{2n+1} \quad \text{for } n = 0\ 1\ 2\ 3\ .\ .\ \}$$

Then a = xxx and b = xxxxx are both words in L_2, but their concatenation ab = xxxxxxxx is not in L_2 Notice that the alphabet for L_2 is the same as the alphabet for L_1. Notice also the liberty we took with the middle definition. In these simple examples, when we concatenate a with b we get the same word as when we concatenate b with a. We can depict this by writing: ab = ba

But this relationship does not hold for all languages. In English when we concatenate “house” and “boat” we get “houseboat,” which is indeed a word but distinct from “boathouse,” which is a different thing—not because they have different meanings but because they are different words. “Merry-go-round” and “carousel” mean the same thing, but they are different words.

EXAMPLE

Consider another language. Let us begin with the alphabet:

Σ= {0 1 2 3 4 5 6 7 8 9} and define the set of words:

L_3={ any finite string of alphabet letters that does not start with the letter zero }

This language L_3, then looks like the set of all positive integers written in base 10.

L_3={ 1 2 3 4 5 6 7 8 9 10 11 12 ...}

We say “look like” strings of instead of “is” because L_3 is only a formal collection of symbols. The integers have other mathematical properties. If we wanted to define the language L_3 so that it includes the string (word) 0, we could say:

L_3(any finite string of alphabet letters that, if it starts with a 0, has no more letters after the first}

DEFINITION

We define the function "length of a string" to be the number of letters in the string. We write this function using the word "length." For example, if a = xxxx in the language L1 above, then length(a) = 4, If c = 428 in the language L_3, then length(c) = 3 ,or we could write directly that in L_1, length(xxxx) = 4 and in L^3 length(428) = 3 . In any language that includes the empty string Λ we have: length(Λ) = 0**,** for any word w in any language, if length(w) = 0 then w = Λ. We can now present yet another definition of L_3.
L_3={any finite string of alphabet letters that, if it has length more than one, does not start with a zero}

This is not necessarily a better definition of L_3, but it does illustrate that there are often different ways of specifying the same language.

There is some inherent ambiguity in the phrase "any finite string," since it is not clear whether we intend to include the null string (**Λ**, the string of no letters). To avoid this ambiguity, we shall always be more careful. The language L_3 above does not include **Λ**, since we intended that that language should look like the integers, and there is no such thing as an integer with no digits. On the other hand, we may wish to define a language like L_1 but that does contain **Λ**.

L_4 = {**Λ** x xx xxx xxxx ...}

= (x^n for n = 0 12 3 …. } Here we have said that X^0 = **Λ**, not x^0 =1 as in algebra.

In this way x^n is always the string of n x's. This may seem like belaboring a trivial point, but the significance of being careful about this distinction will emerge over and over again.

In L_3 it is very important not to confuse 0, which is a string of length 1, with **Λ**. Remember, even when **Λ** is a word in the language, it is not a letter in the alphabet.

DEFINITION

Let us introduce the function **reverse**. If **a** is a word in some language **L**, then reverse(a) is the same string of letters spelled backward, called the reverse of a, even if this backward string is not a word in L.

EXAMPLE

reverse(xxx) = xxx

reverse(xxxxx) = xxxxx

reverse(145) = 541

But let us also note that in L_3, reverse(140) = 041 which is not **a** word in L_3.

DEFINITION

Let us define a new language called PALINDROME over the alphabet Σ = {a, b}

PALINDROME ={Λ, and all strings x such that reverse(x) = x }, If we begin listing the elements in PALINDROME we find PALINDROME = {Λ, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba ... }

The language PALINDROME has interesting properties that we shall examine later. Sometimes when we concatenate two words in PALINDROME we obtain another word in PALINDROME such as when abba is concatenated with abbaabba. More often, the concatenation is not itself a word in PALINDROME, as when aa is concatenated with aba..

DEFINITION

Given an alphabet Σ, we wish to define a language in which any string of letters from Σ is a word, even the null string. This language we shall call the closure of the alphabet. It is denoted by writing a star (an asterisk) after the name of the alphabet as a superscript Σ* . This notation is sometimes known as the **Kleene star** after the logician who was one of the founders of this subject.

EXAMPLE

If Σ = (x }, then Σ* = L_4 = {Λ x xx xxx …}

EXAMPLE

If Σ = {0,1 }, then Σ *={Λ 0 1 00 01 10 11 000 001…}.

EXAMPLE

If Σ = { a,b,c }, then Σ* ={Λ a b c aa ab ac ba bb bc ca cb cc aaa... }

We can think of the Kleene star as an operation that makes an infinite language of strings of letters out of an alphabet. When we say “infinite language” we mean infinitely many words each of finite length.

Notice that when we wrote out the first several words in the language, we put them in size order (words of shortest length first) and then listed all the words of the same length alphabetically. We shall usually follow this method of sequencing a language.

We shall now generalize the use of the star operator to sets of words, not just sets of alphabet letters.

DEFINITION

If S is a set of words, then by S* we mean the set of all finite strings formed by concatenating words from S, where any word may be used as often as we like, and where the null string is also included.

EXAMPLE

If S={ aa,b }, then S* ={∧ plus any word composed of factors of aa and b}

={∧ plus all strings of a's and b's in which the a's occur in even clumps }

={∧ b aa bb aab baa bbb aaaa aabb baab bbaa bbbb aaaab aabaa

Aabbb baaaa baabb bbaab bbbaa bbbbb ...}

The string aabaaab is not in S* since it has a clump of a's of length 3. The phrase "clump of a's" has not been precisely defined, but we know what it means anyway.

EXAMPLE

Let S ={ a, ab }. Then S*={(∧ plus any word composed of factors of a and ab]

={∧ plus all strings of a's and b's except those that start with b and those that contain a double b]

= {∧ a aa ab aaa aab aaaa aaab aaba abaa abab aaaaa aaaab aaaba aabaa aabab abaaa abaab ababa...}

By the phrase "double b" we mean the substring bb. For each word in S* every b must have an a immediately to its left. The substring bb is impossible, as is starting with a b. Any string without the substring bb that begins with an a can be factored into terms of (ab) and (a).

To prove that a certain word is in the closure language S*, we must show how it can be written as a concatenate of words from the base set S.

In the last example, to show that abaab is in S* we can factor it as follows: (ab)(a)(ab)

These three factors are all in the set S,' therefore their concatenation is in S*. This is the only way to factor this string into factors of (a) and (ab). When this happens, we say that the factoring is unique. Sometimes the factoring is not unique. For example, consider S={xx, xxx}.

Then:

S*={∧ and all strings of more than one x }

={ x^n for n= 0, 2, 3, 4, 5... }

={∧ xx xxx xxxx xxxxxx ...}

Notice that the word x is not in the language S*. The string xxxxxxx is in this closure for any of these three reasons. It is : (xx) (xx) (xxx) or (xx) (xxx) (xx) or (xxx) (xx) (xx)

Also, x^6 is either x^2 x^2 x^2 or else x^3x^3.

It is important to note here that the parentheses, ( ), are not letters in the alphabet but are used for the sole purpose of demarcating the ends of factors. So we can write xxxxx = (xx)(xxx). In cases where parentheses are letters of the alphabet, Σ= { x( ) }

length(xxxxx) = 5

but length((xx)(xxx)) = 9

Let us suppose that we wanted to prove mathematically that this set S* contains all for x^n for n≠1 Suppose that somebody did not believe this and needed convincing. We could proceed as follows.

First, we consider the possibility that there were some powers of x that we could not produce by concatenating factors of (xx) and (xxx).

Obviously, since we can produce x^4, x^5 , x^6, the examples of strings that we cannot produce must be large. Let us ask the question, “What is the smallest power of x (larger than 1) that we cannot form out of factors of xx and xxx?” Let us suppose that we start making a list of how to construct the various powers of x. On this list we write down how to form x^2, x^3, x^4,x^5 and so on. Let us say that we work our way successfully up to x^{373}, but then we cannot figure out how to form x^{374}. We become stuck, so a friend comes over to us and says, “Let me see your list. How did you form the word x^{372}?

Why don’t you just concatenate another factor of xx in front of this and then you will have the word x^{374} that you wanted.” Our friend is right, and this story shows that while writing this list out we can never really become stuck. This discussion can easily be generalized into a mathematical proof of the fact that S* contains all powers of x greater than 1.

We have just established a mathematical fact by a method of proof that we have rarely seen in other courses. It is a proof based on showing that something exists (the factoring) because we can describe how to create it (by adding xx to a previous case). What we have described can be formalized into an algorithm for producing all the powers of x from the factors xx and xxx. The method is to begin with xx and xxx and, when we want to produce x^n, we take the sequence of concatenations that we have already found will produce x^{n-2}, and we concatenate xx on to that.

The method of proving that something exists by showing how to create it is called **proof by constructive algorithm**. This is the most important tool in our whole study.. It is in general a very satisfying and useful method of proof, that is, providing that anybody is interested in the objects

we are constructing. We may have a difficult time selling powers of x broken into factors of xx and xxx.

Let us observe that if the alphabet has no letters, then its closure is the language with the null string as its only word Symbolically, we write:

If $\Sigma = \phi$ (the empty set), then $Z^* = \{\Lambda\}$ This is not the same as

If $S = \{\Lambda\}$, then $S^* = \{\Lambda\}$

An alphabet may look like a set of one-letter words. If for some reason we wish to modify the concept of closure to refer to only the concatenation of some (not zero) strings from a set S, we use the notation + instead of *. For example,

If $\Sigma = \{x\}$,then $\Sigma^+ = \{x \ \ xx \ \ xxx \ ...\}$ which is the language L_1 that we discussed before.

If S={xx, xxx} then S+is the same as S* except for the word Λ, which is not in S^+ . This is not to say that S^+ cannot in general contain the word Λ. It can, but only on condition that S contains the word Λ. In this case, Λ is in S^+, since it is the concatenation of some (actually one) word from S (Λ itself). Anyone who does not think that the null string is confusing has missed something. It is already a problem, and it gets worse later.

If S is the set of three words $S = \{w_1 \ \ w_2 \ \ w_3\}$ then,

$S^+ = \{ w_1 \ \ w_2 \ \ w_3 \ w_1w_1 \ \ w_1w_2 \ \ w_1w_3 \ w_2w_1 \ \ w_2w_2 \ \ w_3w_3 \ w_3w_1 \ \ w_3w_2 \ \ w_3w_3 \ w_1w_1w_1 \ w_1w_1w_2 ....\}$

no matter what the words w_1,w_2, and w_3 are.

If w_1= aa, w_2 = bbb, $w_3 = \Lambda$, then S+={ aa bbb A aaaa aabbb...}

The words in the set S are listed above in the order corresponding to their w-sequencing, not in the usual size-alphabetical order.

What happens if we apply the closure operator twice? We start with a set of words S and look at its closure S*. Now suppose we start with the set S* and try to form its closure, which we denote as (S*)* or S**

If S is not the trivial empty set, then S* is infinite, so we are taking the closure of an infinite set. This should present no problem since every string in the closure of a set is a combination of only finitely many words from the set. Even if the set S has infinitely many words, we use only finitely many at a time. This is the same as with ordinary arithmetic expressions, which can be made up of only finitely many numbers at a time even though there are infinitely many numbers to choose from. From now on we shall let the closure operator apply to infinite sets as well as to finite sets.

THEOREM 1

For any set S of strings, we have S*= S**.

CONVINCING REMARKS

First let us illustrate what this theorem means. Say for example that S={a,b}. Then S* is clearly all strings of the two letters a and b of any finite length whatsoever. Now what would it mean to take strings from S* and concatenate them? Let us say we concatenated (aaba) and (baaa) and (aaba). The end result (aababaaaaaba) is no more than a concatenation of the letters a and b, just as with all elements of S*.

aababaaaaaba

= (aaba)(baaa)(aaba)

= [(a)(a)(b)(a)] [(b)(a)(a)(a)] [(a)(a)(b)(a)]

= (a)(a)(b)(a)(b)(a)(a)(a)(a)(a)(b)(a)

Let us consider one more illustration. If S={aa, bbb}, then S* is the set of all strings where the a's occur in even clumps and the b's in groups of 3, 6, 9... Some words in S* are

aabbbaaaa bbb bbbaa

If we concatenate these three elements of S*, we get one big word in S**, which is again in S*.

aabbbaaaabbbbbbaa

= [(aa)(bbb)(aa)(aa)] [(bbb)] [(bbb)(aa)]

This theorem expresses a trivial but subtle point. It is analogous to saying that if people are made up of molecules and molecules are made up of atoms, then people are made up of atoms.

PROOF

Every word in S** is made up of factors from S*. Every factor from S* is made up of factors from S. Therefore, every word in S** is made up of factors from S. Therefore, every word in S** is also a word in S*. We can write this as

$$S^{**} \subset S^{*}$$

using the symbol "⊂" from Set Theory, which means "is contained in or equal to " .

Now in general it is true that for any set A we know that A ⊂ A*, since in A* we can chose as a word any one factor from A. So if we consider A to be our set S*, we have

$$S^{*} \subset S^{**}$$