Lesson 2

Too many languages, too few grammars

2.1 Grammars and Computability

Grammars and formal languages are another way to look at what is computable and what is not. Given a set of symbols Σ, a formal language is a set of strings from Σ^*. We write $L \subseteq \Sigma^*$.

In this case, "knowing" a formal language means being able to decide for any string, whether it belongs to the language or not. This problem is called the membership problem. To be clear, knowing a natural language obviously entails much more than this, but people have argued it at least includes this sort of knowledge.

A grammar G can be thought of as a kind of algorithm which solves the membership problem. What we will see is that most formal languages have no grammars. To put it more strongly, for most formal languages, there are no mechanisms that can decide for every string whether it belongs to the language or not. We will see that, in a sense, there are just too few grammars and too many languages.

But before we consider this question in earnest, let us make clear the membership problem.

2.2 The Membership Problem

The **membership problem** is the problem of deciding whether a string belongs to a set. The problem can be stated thusly: Given a set of strings S and *any* string $s \in \Sigma^*$, output whether $s \in S$. Is there an algorithm that solves this problem for a given S?

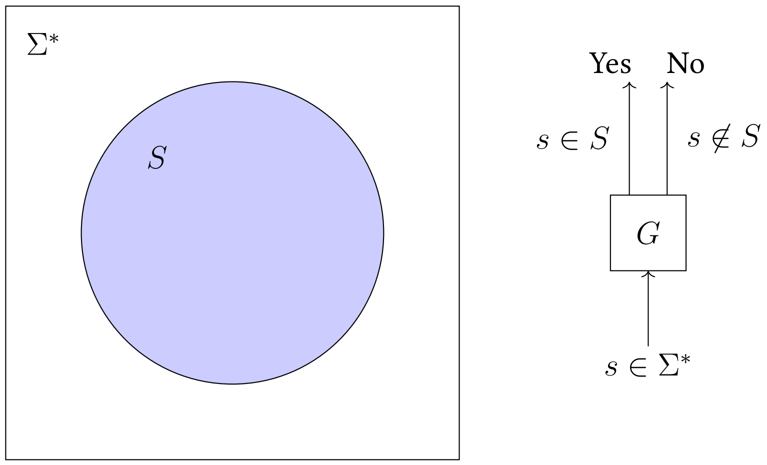

Figure 2.1: The membership problem

Example 1. *A string belongs to S if it does not contain aa as a substring.*

$s \in S$	$s \notin S$
abba	*baab*
abccba	*aaccbb*
babababa	*ccaaccaacc*
...	...

Example 2. *A string belongs to S if it does not contain aa as a subsequence.*

$s \in S$	$s \notin S$
cabb	*baab*
babccbc	*babccba*
bbbbbb	*bbacccccccccaccc*
...	...

Exercise 1. *Write finite state acceptors that solves the membership problem for the above two languages.*

If a grammar G solves the membership problem for language L, we often say that G *recognizes* L.

2.3 One-to-One Correspondence and Enumeration

So how many grammars are there? We write down grammars. Even if they take many pages of gigabytes of space, they are ultimately finite in size. A grammar, like a computer program, is just a string of finite length. Clearly there are infinitely many logically possible grammars.

How many languages are there? If formal languages are subsets of Σ^* then the powerset of Σ^* is the set of all formal languages. How can we count these? Can we put them in one-to-one correspondence with the grammars?

The number of elements in a set is called its cardinality. One way to see whether two sets have the same size, that is the same cardinality, is to see if it is possible if the members of each set can be put in one-to-one correspondence with the members of the other set, without leaving any elements out. If so, there is a way to do this, it seems reasonable to say that the two sets have the same cardinality. Basically, we match up or pair up elements of each set and if we "run out" of elements "at the same time" then we can say they have the same cardinality. While the phrases in quotes can be understood when comparing the sizes of finite sets, when we compare the sizes of infinite sets, these casual notions are wanting.

For example the natural numbers $\{0, 1, 2, \ldots\}$ is an infinite set. Any set we can put in one-to-one correspondence with it will have the same cardinality.

For example, consider these lists:

$$0, 2, 4, 6, \ldots$$

$$\lambda, a, aa, aaa, \ldots$$

In the above lists, every element can be put in one-to-one with the natural numbers. In the first example, the function can assign the value $2n$ to the nth element in the list. In the second example, the function can assign the value a^n to the nth element in the list.

A set is *enumerable*, or *countable* if it can be put in one-to-one correspondence with the natural numbers. Another way of thinking about it is that there is way to arrange its members in an ordered list where each member sooner or later will be encountered. So each item must appear sooner or later as the nth entry, for some finite n. All we need is one such list to establish the enumerability of such a set.

Some lists are insufficient to establish enumerability. For example, the natural numbers themselves cannot be enumerated by the list below.

$$1, 3, 5, \ldots, 2, 4, 6, \ldots$$

This is because it will not be possible to assign an index to any even numbers: they will never be encountered after finitely many steps traversing the list, no matter how large that finite number is. But as long as we have one list of the elements of some set S in one-to-one correspondence with the natural numbers that is enough to say S is enumerable.

When an infinite set can be enumerated, it is also said to be *countably infinite.*

There are some surprising consequences to defining cardinality in terms of one-to-one correspondence. We have already seen one: the set of non-negative even integers is the same as the set of all non-negative integers! It is strange to think that an infinite set A, which is a proper subset of another infinite set B, can have the same size and cardinality as B. Nonetheless, all that really been said is that "same cardinality" means in "one-to-one correspondence." If both sets can be put in one-to-one correspondence with the natural numbers then they can be put in one-to-one correspondence with each other, and all those sets have the same cardinality; they are countably infinite.

Exercise 2.

1. *Show that the set PAIRS =* $\{(x, y) : x, y \in \mathbb{N}\}$ *is countably infinite.*
2. *Show that the set TRIPLES =* $\{(x, y, z) : x, y, z \in \mathbb{N}\}$ *is countably infinite.*

2.4 Enumerating Σ^*

The usual way to enumerate strings in Σ^* is to order them first by their length and then within strings of the same length to order them in dictionary order, as shown below. Consequently, for

0	λ	1	a	4	aa	13 aaa
		2	b	5	ab	…
		3	c	6	ac	
				7	ba	
				8	bb	
				9	bc	
				10	ca	
				11	cb	
				12	cc	

Figure 2.2: Enumerating Σ^* with $\Sigma = \{a, b, c\}$.

any finite alphabet Σ, the cardinality of Σ^* is countably infinite.

Exercise 3. *Show that the set STRINGPAIRS =* $\{(x, y) : x, y \in \Sigma^*\}$ *is countably infinite.*

2.5 The powerset of Σ^*

Can we also enumerate the powerset of Σ^*? After all, it is also an infinite set. The answer is No, and the result is ultimately due to a theorem by the German mathematician Cantor.

Theorem 1. *Fix a finite alphabet Σ. The powerset of Σ^* is not enumerable.*

Proof. Suppose there was an enumeration of the subsets of Σ^*.

$$S_1, S_2, \ldots S_m, \ldots$$

We will construct a subset of Σ^* and show that it is not any one of the elements in the above list.

The construction uses the fact that Σ^* is enumerable. Let w_n denote the nth string in the enumeration of Σ^*.

$$S' = \{w_n : n \in \mathbb{N}, w_n \notin S_n\}$$

The claim is that S' cannot belong to the list above. To see why this is true, consider S_1. Could S' be the same as S_1? Well if w_1 belongs to S_1 then it does not belong to S'. On the other hand if if $w_1 \notin S_1$ then w_1 does belong to S'! So $S' \neq S_1$. The same logic will show that for any $m \in \mathbb{N}$, S' cannot be the same as S_m! Since m could be any number, it follows that S' does not belong to any element of the enumeration.

We may wonder what happens if we make a new enumeration with S', such as the one below.

$$S', S_1, S_2, \ldots S_m, \ldots$$

But we can just repeat the argument above and create a new set S'' which does not occur. So for any enumeration of the powerset of Σ^*, there is always going to be subsets of Σ^* that have not been included. $\square$

It follows that the powerset of Σ^* cannot be placed in one-to-one correspondence with $\mathbb{N}$. An infinite set that is not enumerable is called *uncountably infinite*.

2.6 Countably many grammars but uncountably many languages

Consequently, we have established that there are countably many grammars but uncountably many languages.

Each program (or grammar) is ultimately a finite string, that is an element of Σ^*. Not all elements of Σ^* will be interpretable, just like not any text file is a well-formed program of Python. Nonetheless we can enumerate these programs. The programs which takes strings as input and output Yes or No solve the membership problem for some language. The set of languages that have grammars are called *computable*. All the others are *uncomputable*.

This is directly analogous to to Turing's result on real numbers. Most real numbers are not computable in the sense that there are real numbers r for which no algorithm (Turing Machine) exist which can you tell what the nth digit of r is. The original paper (Turing, 1937) is well worth reading, and I have also enjoyed Chaitin (2004) and Chaitin (2006), who gets as close as is probably possible to "naming" an uncomputable real number.

What does this mean for learning? Well it is the first of many hardness results on learning. There can be no algorithm that can learn *every* formal language because most of these languages are *uncomputable*. That is, there is no grammar (and so no output of any algorithm) which recognizes them.

Bibliography

Chaitin, Gregory. 2004. How real are real numbers? Https://arxiv.org/pdf/math/0411418.pdf.

Chaitin, Gregory. 2006. *Meta Math! The Quest for Omega.* Vintage.

Turing, Alan. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2:230–265.