Logic for Linguists:
Lecture 5

Gregory Wilsenach

University of Cambridge

13th November 2019

Some History

In *Syntactic Structures* Chomsky presented a very formal approach to syntax.

The idea was to start with a set of symbols and set of formal rules for generating new parts of a sentence from old.

The hope was to construct a system along these lines that could generate all and only the grammatical sentences of a given natural language.

This recursive, we might now say algorithmic, approach to the subject lead Chomsky and others to the study of progressively more powerful (and complex) formal systems.

The Chomsky Hierarchy

In particular, it lead to the following hierarchy of formal languages:

- Type-0: Recursively Enumerable Languages,
- Type-1: Context Sensitive Languages,
- Type-2: Context-Free Languages, and
- Type-3: Regular Languages.

This hierarchy is presented above in decreasing order of strength. In other words, any class of languages on this list also contains those that appear lower down on the list.

We will concern ourselves today with the lowest rung on this ladder, regular languages. We will begin by discussing a formalisation of the notion of an algorithm.

Deterministic Finite State Automata

A deterministic finite state automaton (DFA) consists of set of states and transition arrows.

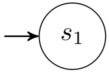
Each arrow is labelled by a symbol from some finite set, which we call the vocabulary.

We designate one state to be starting state and designate a set of states to be the accepting or final states.

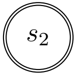
Diagrams

We usually represent a DFA using a diagram.

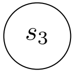
Start state:



Final state:



Other state:



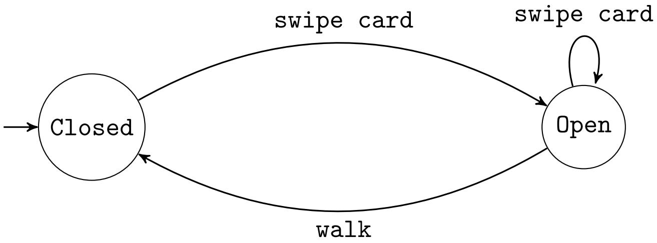
Generic Example

Here is a generic example:

$$\begin{tikzcd}
{} \arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, "1"] & q_2 \arrow[loop above, "1"] \arrow[r, bend left, "0"] & q_3 \arrow[l, bend left, "{0, 1}"]
\end{tikzcd}$$

Example 1

Let's model these automatic doors we see around the university.



The vocabulary

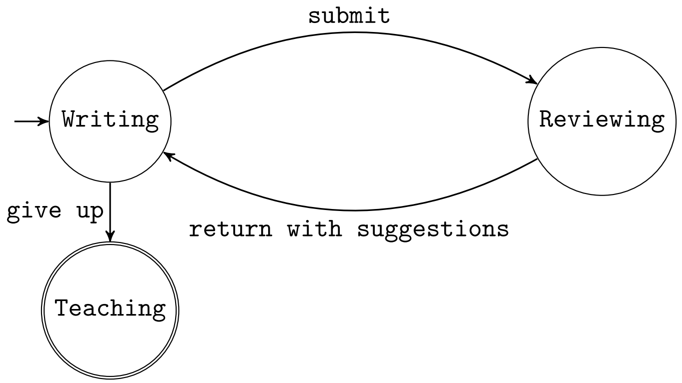
$$\Sigma := \{\text{walk}, \text{swipe card}\}$$

The set of states

$$Q := \{\text{Closed}, \text{Open}\}$$

Example 2

The life of an academic.



The vocabulary is $\Sigma := \{\text{submit}, \text{return with suggestions}, \text{give up}\}$

The set of states is $Q := \{\text{Writing}, \text{Teaching}, \text{Reviewing}\}$.

Formal Definition

A deterministic finite-state automaton A is a tuple

$$A := (\Sigma, Q, q_0, \delta, F),$$

where:

- Σ is a finite set, which we call the alphabet or vocabulary;
- Q is a finite (non-empty) set, which we call the set of states;
- $q_0 \in Q$ is the initial state;
- $\delta : Q \times \Sigma \to Q$ is the state transition function; and
- $F \subseteq Q$ is the set of final states.

Language

Let Σ be a vocabulary. A string or word over Σ is simply a sequence of elements in our alphabet.

Example: if $\Sigma = \{a, b\}$ then $aaaabbaaba$ is a word in Σ.

Let Σ^* be the set of all words over Σ.

We call a subset $L \subseteq \Sigma^*$ a language over Σ.

Examples

Let $\Sigma = \{a, b\}$.

Example 1:

$$L_1 = \{a, aa, aaa, aaa, \ldots$$

Example 2:

$$L_2 = \{ab\}$$

We write a^n to denote a repeated n times.

Example 3:

$$L_3 = \{a^n b^n : n \in \mathbb{N}\}$$

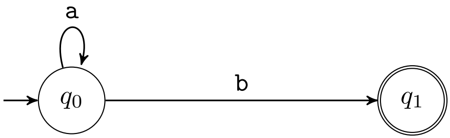
DFA's and Languages

Suppose we have some DFA called M over a vocabulary Σ. We can generate a string in Σ using M as follows:

- We start in the initial state and choose an arrow leaving that state;
- We write down the symbol labelling that arrow and now move along that arrow and transition to the new state;
- we choose an arrow leaving our current state;
- we write down the symbol labelling that arrow and transition to the new state; and
- we continue this process for as long as we want, with the caveat that we can only stop if we are in an accepting state.

Note: There are obviously lots of choices we can make, so a machine may be able to generate many different words.

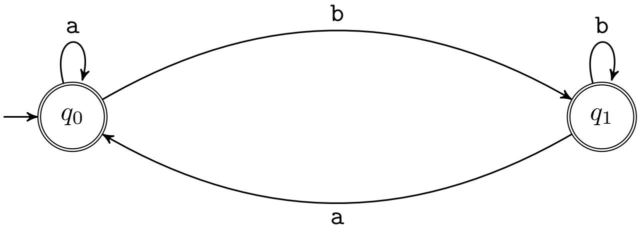
Example 1


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "\mathtt{a}"] \arrow[rr, "\mathtt{b}"] & & q_1 \end{tikzcd}$$

Can you give me an example of a word this machine can generate?

Can you describe the set of all words that can be generated by this machine?

Example 2


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "a"] \arrow[rr, bend left, "b"] & & q_1 \arrow[loop above, "b"] \arrow[ll, bend left, "a"]
\end{tikzcd}$$

Can you give me an example of a word this machine can generate?

Can you describe the set of all words that can be generated by this machine?

Languages and DFA's

We say that a DFA M over a vocabulary Σ accepts a string s over Σ if M can generate s. If M does not accept s we say M rejects s.

The language decided by M is the set of all strings that it accepts.

Returning to the Example

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[loop above, "\mathtt{a}"] \arrow[rrr, bend left, "\mathtt{b}"] & & & q_1 \arrow[loop above, "\mathtt{b}"] \arrow[lll, bend left, "\mathtt{a}"]
\end{tikzcd}$$

What is the is the language decided by this machine?

Operations on Languages: Concatenation

Let's spend some time speaking about languages and strings and some natural operations.

The concatenation of two strings w_1 and w_2 is just the string formed by writing w_1 and then writing w_2 right afterwards.

Example: aa concatenated with ab is $aaab$.

We write $w_1 \cdot w_2$ to denote the concatenation of two strings. The concatenation of two languages L_1 and L_2 is the language

$$L_1 \cdot L_2 = \{w_1 \cdot w_2 : w_1 \in L_1, w_2 \in L_2\}.$$

Example: Let $L_1 = \{a, ab\}$ and $L_2 = \{\epsilon, bb\}$ then

$$L_1 \cdot L_2 = \{a, abb, ab, abbb\}.$$

Kleene Star Operator

The Kleene Star of a language L is defined as:

$$L^* = \{\epsilon\} \cup L \cup L \cdot L \cup (L \cdot L) \cdot L \cup \ldots ...$$

In other words: the Keene star of a language is formed by concenating the language with itself $0, 1, 2, \ldots$ times and then taking the union of all of these languages.

Example: $\{a\}^* = \{\epsilon, a, aa, aaa, \ldots\}$

Regular Languages (1)

We spoke earlier about Regular Languages, let's define this notion now.

A regular language L over Σ is one built up from a set of base languages using specified rules. The base languages are:

- the empty language $\{\}$;
- the language $\{\epsilon\}$ containing the empty word (ϵ is the empty word); and
- the singleton languages over Σ, for example if $\Sigma = \{a, b\}$ then $\{a\}$ and $\{b\}$ are the singleton languages.

Regular Languages (2)

We then have three operations for building new regular languages from known regular languages. They are as follows.

- The union of two regular languages is a regular language (i.e. if L_1 and L_2 are regular languages then the language containing all of the words in both L_1 and L_2 is regular).
- The concatenation of two regular languages is regular.
- The Keene star of a regular language is regular.

Example

Let

$$L = \{\epsilon\} \cup \{a^n b : n \in \{0, 1, 2, \ldots\}\}.$$

In other words L consists of all those strings of the form $a \ldots ab$. Is this language regular?

Yes! Here is the proof:

- $\{a\}$ is regular so $\{a\}^* = \{\epsilon, a, aa, \ldots\}$ is regular,
- $\{b\}$ is also regular and so $\{a\}^* \cdot \{b\}$ is regular,
- but $L = \{a\}^* \cdot \{b\}$ and so L is regular.

Bringing it All Together

It turns out that regular languages are exactly the languages recognised by deterministic finite-state automata!

Kleene's Theorem:

- For every regular language there is a DFA over the same vocabulary that decides that language.
- The language decided by any DFA is regular.

It follows that the bottom layer of the Chomsky hierarchy consists of all those languages decidable by DFA's.

Exercise

Let

$$L = \{\epsilon\} \cup \{a^n b : n \in \{0, 1, 2, \ldots\}\}.$$

Construct a DFA that decides L.

Limitations

We will not have time for this today, but we could define regular languages a third way, as exactly those languages which can be specified by a regular grammar.

A grammar in this context simply means a set of rules for generating strings.

A linguist might hope that the set of valid sentences in some natural language (say English) form a regular language.

Unfortunately this is not the case...

The Pumping Lemma For Regular Languages

Theorem (Pumping Lemma)

For every regular language L there exists an integer n such that for every $x \in L$ with $|x| \geq n$ there exists $u, v, w \in \Sigma^*$ such that $x = uvw$ and

- $|uv| \leq n$,
- $v \geq 1$, *and*
- *for all* $i \geq 0 : uv^i w \in L$.

In English: If we take a string $x \in L$ and insert v any number of times in the middle of it then the resultant string is still in L.

This result is what allows us to show that certain very simple languages are not regular. Example:

$$L = \{a^n b^n : n \in \{0, 1, \ldots\}\}$$

is not regular.

Informal Understanding

What you should really take away from the pumping lemma is that DFA's have no memory. We cannot implement something like this:

- first generate a string of a's of some length,
- store the number of a's generated, and
- generate a string of b's of the same length.

The DFA can't remember the number of a's that came earlier on.

Summary

In this lecture...

- We introduced the Chomsky hierarchy.
- We introduced DFA's and discussed what it means for a machine to recognise a language.
- We introduced regular languages and established the equivalence between regular languages and DFA's.
- We discussed the limitations of regular languages.

Reading

This is the most complete introduction to this area:

- `https://www.dpmms.cam.ac.uk/~tf/cam_only/crouchnotes.pdf`

In the document linked above see

- Section 3.4 for regular grammars (another characterisation of regular languages),
- Section 3.3 for a proof of Kleene's theorem, and
- Section 4.1 for the Pumping lemma and its proof.

I expect you can all find a copy of *Syntactic Structures* for more information on Chomsky's (initial) approach.