Linguistics 106, lecture notes
The limits of Finite State Automata

25 July 2002

1 Introduction: Are there non-Regular Languages?

- We set out to find a general format for stating the grammatical rules of
  a language—rules accounting for the distribution of morphemes in the
  language.

  Our first idea was the rules could be “word chains”, in Pinker’s sense.
  We implemented the word-chain model with Finite State Automata and,
  equivalently, with Regular grammars.

- So far we have seen what FSAs and RGs can do. Now we should ask the
  question: Is there anything FSAs/RGs can’t do?

- This question will have two independent parts:
  1. Are there sets of strings (languages) that cannot be generated by any
     FSA, or any RG? And if so, then what defines the difference between
     Regular Languages and non-Regular languages?
  2. Are FSAs/RGs adequate for the task of describing the structures we
     find in natural language sentences, e.g. in sentences of English?

2 FSAs and unbounded dependencies

- The expressive power of FSAs is limited by the following property:
  The transitions of an FSA are defined solely in terms of the single state
  the machine is in and the single symbol read; and the transitions may go
  to just a single state, not a sequence of them:

  \[ \delta([symbol],[state]) = [state]. \]

- Because of this property:
It is impossible to state explicitly in a single transition rule that the occurrence of \( \sigma_j \) in an input string depends on the occurrence of a symbol \( \sigma_i \) elsewhere in the string, whether before \( \sigma_j \) or after.

- **Regular Grammars** have the following related property:
  In the rules of a RG, the left-hand side of the rule may refer to just a single non-terminal AND the right-hand side may introduce just a single terminal; also, the right-hand side may introduce just a single non-terminal, not a sequence of them.\(^1\):
  \[ A \rightarrow \sigma(B) \]

- Because of this property:
  It is impossible to state explicitly in a single rewrite rule that the occurrence of \( \sigma_j \) in an output string depends on the occurrence of a symbol \( \sigma_i \) elsewhere in the string, whether before \( \sigma_j \) or after.

- Yet there are languages, even Regular Languages, where the occurrence of a symbol (or substring) \( \sigma_j \) depends on the occurrence of a symbol (or substring) \( \sigma_i \) elsewhere. For example:
  \[ RL_{dep} = \{a1z, a2z, a3z, \ldots, b1y, b2y, b3y, \ldots\} \]
  \[ RL_{a \leq p \leq z} = \{\varepsilon, ab, aabb, aaabbb, aaaaabbbb, aaaaabbbbb\} \]

- **Question:** How can such dependencies be recognized by an FSA (RG), if the transitions (rewrite rules) in an FSA (RG) can refer neither what symbols have already read (written), nor to what symbols will be read (written)?

- **Answer:** An FSA (RG) can model such dependencies only by multiplying states (non-terminals), in order to multiply the number of distinguishable paths (derivations) in the machine (grammar).
  This way, relevantly different histories of symbols read (or terminals written) lead to different states (non-terminals).

---

\(^1\) In a rule \( A \rightarrow \sigma(B) \), the left-hand side is \( A \), and the right-hand side is \( \sigma(B) \). The left-hand side is the specification of what is to be rewritten; the right-hand side is what that is to be rewritten as.
Examples:
1. FSA and RG for $RL_{dep}$:

   $$S \rightarrow aA \quad A \rightarrow 1B \quad B \rightarrow z$$
   $$A \rightarrow 2B$$
   $$\vdots$$
   $$S \rightarrow bC \quad C \rightarrow 1D \quad D \rightarrow y$$
   $$\vdots$$

2. $\{ \omega \mid a^nb^n, 0 \leq n \leq i \}$ for progressively higher finite values of $i$.

   (a) Finite State Automata:

   $$\{ \omega \mid \omega = a^nb^n, 0 \leq n \leq 1 \}$$

   $$\{ \omega \mid \omega = a^nb^n, 0 \leq n \leq 2 \}$$

   $$\{ \omega \mid \omega = a^nb^n, 0 \leq n \leq 3 \}$$

   $$\{ \omega \mid \omega = a^nb^n, 0 \leq n \leq 4 \}$$

   $$\{ \omega \mid \omega = a^nb^n, 0 \leq n \leq i \}$$

   $$\vdots$$
(b) Regular Grammars:

<table>
<thead>
<tr>
<th></th>
<th>$a^n b^n, 0 \leq n \leq 1$</th>
<th>$a^n b^n, 0 \leq n \leq 2$</th>
<th>$a^n b^n, 0 \leq n \leq 3$</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \epsilon$</td>
<td>$S \rightarrow \epsilon$</td>
<td>$S \rightarrow \epsilon$</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>$S \rightarrow aB$</td>
<td>$S \rightarrow aB$</td>
<td>$S \rightarrow aB$</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>$B \rightarrow b$</td>
<td>$B \rightarrow b$</td>
<td>$B \rightarrow b$</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>$S \rightarrow aC$</td>
<td>$S \rightarrow aC$</td>
<td>:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C \rightarrow aD$</td>
<td>$C \rightarrow aD$</td>
<td>:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D \rightarrow bB$</td>
<td>$D \rightarrow bB$</td>
<td>:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S \rightarrow aE$</td>
<td>$E \rightarrow aF$</td>
<td>$F \rightarrow aG$</td>
<td>$G \rightarrow bD$</td>
<td></td>
</tr>
</tbody>
</table>

- Let’s think of the ability to keep track of such dependencies as a kind of memory.

- **Fact we just established**: The memory available to FSAs relies entirely on the multiplication of states.

- **Question**: What limit does this fact put on the kind of languages that a **Finite State Automaton** can generate?

- **Answer**: Since an FSA can have only a **finite** number of states, the memory available to an FSA can have only a bounded (finite) depth.

- To spell it out somewhat more:
  Because the number of states is finite, there may be only a finite number of distinct paths connecting states.
  The number of distinct paths is exactly what gives an FSA its memory, because each distinct path constitutes a distinct history of symbols read.
  Consequently the number of ‘histories’ of symbols read that any FSA can distinguish is finite, and bounded by the number of its states.

As a result, any language whose generation requires a memory of unbounded depth cannot be Regular.
• For example the following languages are not Regular:

\[ L_{a^n b^n} = \{ \omega \mid \omega = a^n b^n, n \geq 0 \} \]
\[ L = \{ \omega \mid \omega \text{ is a palindrome} \} \]

3 The Pumping Lemma

• **Question**: Are there ways to tell whether a language is Regular?

• One important diagnostic tool is known as *The Pumping Lemma*.

3.1 Informal derivation of the PL

• Any FSA has only a finite number of states.
  Consequently, there is a bound on how long a path in an FSA can be
  without repeating any states (i.e. without including a loop).
  For instance, if FSA \( M \) has three states, there could not possibly be a
  path in \( M \) that was more than two ‘transition arrows’ long, unless that
  path repeated at least one state (i.e. included a loop).

• Since there is a bound on non-looping path length, there is a bound on
  the length of non-looping paths that lead to accept states.

• A path to an accept state of course represents a string recognized by the
  FSA. And the length of a path to an accept state is equal to the length of
  the string accepted.

\[
\text{Therefore, for any FSA, there is a maximal length } l_{\text{max}} \text{ such that the} \\
\text{FSA could not possibly recognize strings longer than } l_{\text{max}} \text{ without running} \\
\text{through some state(s) more than once.}
\]

• Suppose that \( \omega \) is a string in \( L \) whose length is greater than \( l_{\text{max}} \), the
  crucial ‘pumping length’ for \( M \).

• Then the path in \( M \) defined by \( \omega, \Pi_\omega \), must include at least one pass
  through at least one loop.
  Call one loop in \( \Pi_\omega \lambda \), and suppose that \( \omega \) defines \( m \) passes through loop \( \lambda \).
• There must be a path to an accept state in $M$ which differs from $\Pi_\omega$ just in that there are $m + 1$ runs through $\lambda$.

Indeed there must be any number of paths to accept states which differ from $\Pi_\omega$ just in there being $m + x$ runs through $\lambda$, for any positive integer $x$.

Moreover, there must be a path which differs from $\Pi_\omega$ just in that it skips $\lambda$ altogether.

• The class of languages accepted by FSAs is the class of Regular Languages. Therefore, any Regular Language will have the properties just described. If $L$ is regular, then any string in $L$ over a certain length will have to include some indefinitely repeatable substring.

These conclusions constitute the main idea of “The Pumping Lemma”.

3.2 Formal statement of the PL

The Pumping Lemma

If $L$ is a Regular Language over alphabet $\Sigma$,

Then there is a number $n$ such that any string $\omega$ in $L$ whose length is at least $n$ can be divided into three pieces, $\omega = xyz$, where all the following conditions hold:

1. $y \neq \epsilon$
2. for any number $i \geq 0$, $xy^iz \in L$
3. $|xy| \leq n$

Four important details of the Pumping Lemma

1. The PL reads, “If $L$ is regular, then...”. So the PL says that any Regular Language has a certain property. It does not say that any language with the ‘pumping property’ is Regular.

Consequently, it would be invalid to argue that a language is regular because it has this property. We can only argue that, because a language lacks the pumping property, it is not regular.
2. The PL says that, in a Regular Language, every string over a certain length has a certain property. So, to prove that a language is not Regular, it is sufficient to demonstrate that a single string over the crucial length does not have the 'pumping property.'

3. The PL says that, if $L$ is Regular, then every string in $L$ over length $n$ must meet certain conditions. Notice, if $L$ has zero strings over length $n$, then these conditions are met trivially.

4. The PL says that any string in a regular language $L$ over length $n$ must be segmentable into $xyz$, such that $xy^iz$ is also in $L$, for any $i$ greater that or equal to zero. Thus $xy^0z$ must be in $L$, meaning that $xz$ must be in $L$.

### 3.3 Using the Pumping Lemma

To show that a language $L$ is not regular it suffices to show that $L$ does not have the ‘pumping property’. For if $L$ were regular, it would have this property.

1. $L_{ab^n} = \{ \omega \mid \omega = a^n b^n, n \geq 0 \}$

2. $L_{ab^n a^n} = \{ \omega \mid \omega = a^n b^n a^n, n \geq 0 \}$

3. $L_{\omega\omega} = \{ \omega \in \{a, b\}^* \mid \omega = \sigma\sigma \}$

4. $L_{n>m} = \{ \omega \mid \omega = a^n b^m, n > m \}$
4 Chomsky’s argument that English is not a RL

- In the 1950s, Chomsky presented an argument that English is not a Regular Language, relying on the results we just proved.

- Chomsky noted that English has many constructions involving discontinuous dependencies, where elements of one type must be matched by elements of another type elsewhere in the string:

    “Either$_1$ you are confused, or$_1$ I am a moron.”
    “If$_1$ you are confused, then$_1$ you should interrupt.”
    “The cat$_1$ meowed$_1$.”

- And he observed that there may be several such dependencies in a single string, one embedded inside the other.

    “If$_1$ either$_2$ you are confused or$_2$ I am a moron, then$_1$ someone should interrupt.”
    “The cat$_1$ the dog$_2$ chased$_2$ meowed$_1$.”

- Such constructions define sets of strings that are essentially similar to \( \{\omega \mid \omega = a^n b^n \} \). The number of \( X \)'s in one part of the string must match the number of \( Y \)'s elsewhere.

- We know that sets like this are Regular Languages if \( n \) is bounded, but not Regular if \( n \) can be any number.

  **Question:** Is there a limit on how many dependencies of the type Chomsky observed there may be in a sentence of English?

- Chomsky insisted that there is no *grammatical* limit. It’s just that, the more there are, the harder the sentence is to understand.

  The following sentences are grammatical, even if they are hard to understand:

    “If$_1$ it’s true either$_2$ that you suspect that if$_3$ either$_4$ I continue to claim that ... or$_4$ if anyway Lawrence does, then$_3$ we might be here forever, or$_2$ that you are simply bored, then$_1$ perhaps the point has been made.

    “The cat$_1$ the dog$_2$ the hunter$_3$ the police$_4$ ... arrested$_4$ trained$_3$ chased$_2$ meowed$_1$.”

- So Chomsky concluded that English is not a Regular Language.
5 Context Free Grammars

- **Fact we just learned:** FSAs and RGs cannot generate languages permitting an unbounded number of discontinuous dependencies.

- **Question:** What sorts of devices can?

- Today we will discuss one type of grammar that can: Context Free Grammars.\(^2\)

- Recall the definition of a *Grammar*:

  **Grammar** A set \( \{V_T, V_N, S, R\} \), where:
  
  - \( V_T \) is a set of symbols, the *terminal alphabet*;
  - \( V_N \) is a set of symbols, the *non-terminal alphabet*;
  - \( S \) is the unique *start symbol*, \( S \in V_N \); and
  - \( R \) is a set of rewrite rules, \( \Phi \rightarrow \Psi \), where: \( \Phi, \Psi \in (V_T \cup V_N)^* \); \( \Phi \) always contains at least one non-terminal; and in at least one rule, \( \Phi = S \).

- A *Context Free Grammar* is a special type of Grammar:

  **Context Free Grammar** A grammar all of whose rules are of the form
  
  \[ A \rightarrow \ldots \]

  where \( A \) is a single non-terminal symbol \( (A \in V_T) \) and \( \ldots \) is any string over the union of terminal and non-terminal alphabets \( (\ldots \in (V_T \cup V_N)^*) \).

  

  **CFG rules** \( A \rightarrow \ldots \)
  
  **RG rules** \( A \rightarrow \sigma(B) \)

  **Possible CFG rules** \( A \rightarrow BC \quad A \rightarrow xBy \quad A \rightarrow BxCyD \)

  \[ A \rightarrow xB \quad A \rightarrow xyz \quad A \rightarrow \epsilon \]

  **Impossible CFG rules** \( AB \rightarrow xB \quad xA \rightarrow xBC \quad xAy \rightarrow xBCy \quad x \rightarrow y \quad x \rightarrow yA \quad x \rightarrow \epsilon \)

  \[ \text{N.B.: Every RG is a CFG, but not vice versa.} \]

\(^2\)There is an automata-theoretic equivalent of Context Free Grammars, called Pushdown Automata. I'm not sure how much time we will have to explore these.
Context Free Language  A language generable by a CFG.

Example CFGs:

1. \( G_1 = \{ V_{1T}, V_{1N}, S_1, R_1 \} \), where:

   \[
   V_{1T} = \{ a, b \} \quad V_{1N} = \{ S, A, B \} \quad S_1 = S
   \]

   \[
   R_1 = \begin{cases}
   S \rightarrow aaRb \\
   S \rightarrow \epsilon \\
   R \rightarrow bbSa \\
   R \rightarrow \epsilon
   \end{cases}
   \]

2. \( G_2 = \{ V_{2T}, V_{2N}, S_2, R_2 \} \), where:

   \[
   V_{2T} = \{ a, b \} \quad V_{1N} = \{ a, b \} \quad S_2 = S
   \]

   \[
   R_2 = \begin{cases}
   S \rightarrow SS \\
   S \rightarrow AB \\
   A \rightarrow aAb \\
   B \rightarrow bbA \\
   A \rightarrow \epsilon \\
   B \rightarrow \epsilon
   \end{cases}
   \]

Exercises

1. Give a CFG for the language \( L_4 \):

   \[
   L_4 = \{ \omega \mid \omega = a^n b^n, \text{n is even, m is odd} \}
   \]

2. Give a CFG for the language \( L_4 \):

   \[
   L_4 = \{ \omega \mid \omega = \sigma^{reverse} \}
   \]