Regular Expressions, Pumping Lemma, Right Linear Grammars
Ling 106
March 25, 2002

1 Regular Expressions

A regular expression describes or generates a language: it is a kind of shorthand for listing the members of a language. We say that the value of a regular expression is a language. Regular languages are those languages that may be generated from a regular expression.

1.1 Formal Definition of a Regular Expression

R is a regular expression if R is:

1. a for some a in the alphabet Σ,
   (This corresponds to the language \{a\}.)
2. \(\epsilon\)
   (This corresponds to the language containing only the empty string \(\epsilon\): i.e., \{\epsilon\}.)
3. \(\emptyset\)
   (This corresponds to the empty language.)
4. \((R_1 \cup R_2)\), where \(R_1\) and \(R_2\) are regular expressions.
5. \((R_1 \circ R_2)\), where \(R_1\) and \(R_2\) are regular expressions.
6. \((R_1^*)\), where \(R_1\) is a regular expression.

In this definition, regular expressions are defined in terms of smaller regular expressions. A definition of this type is called an inductive definition: It tells you how to build up bigger regular expressions from smaller ones.

1.2 Some Examples of Regular Expressions

The value of a regular expression is a language.

Given \(\Sigma = \{0, 1\}\):

1. \(0^*10^* = \{w \mid w\ has\ exactly\ a\ single\ 1\}\)
2. \(\Sigma^*1\Sigma^* = \{w \mid w\ has\ at\ least\ one\ 1\}\)
3. \(1\Sigma^* = \{w \mid w\ begins\ with\ 1\}\)
4. \(\Sigma^*001\Sigma^* = \{w \mid w\ contains\ the\ string\ 001\ as\ a\ substring\}\)
5. \((\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}\)

6. \(11 \cup 00 = \{11,00\}\)

7. \((11 \cup 00)^* = \{w \mid w \text{ begins with either 11 or 00}\}\)

8. \((0 \cup \varepsilon)^* = 01^* \cup 1^*\)

9. \(1^*\emptyset = \emptyset\)
   Concatenating the empty set to any set yields the empty set.

10. \(\emptyset^* = \{\varepsilon\}\)
    The result of the star operation on an empty language is the empty string.

11. Question
    \((\Sigma\Sigma\Sigma)^* = \)

12. Question
    \(0^*0 \cup 1^*1 \cup 0 \cup 1 = \)

1.3 Some Properties of Regular Expressions

Let \(R\) be any regular expression:

1. \(R \cup \emptyset = R\)
   Adding the empty language to any other language leaves that language unchanged.

2. \(R \circ \varepsilon = R\)
   Concatenating the empty string to any string will not change the string.

3. \(R \cup \varepsilon\) may not be the same as \(R\).
   If \(R = a\), then \(L(R) = \{a\}\) and \(L(R \cup \varepsilon) = \{a, \varepsilon\}\).

4. \(R \circ \emptyset\) may not be the same as \(R\).
   If \(R = a\), then \(L(R) = \{a\}\) but \(L(R \circ \emptyset) = \emptyset\).

1.4 Equivalence with Finite Automata

Regular expressions and finite automata are equivalent in their descriptive power. That is, any regular expression can be converted into a finite automaton that recognizes the language it describes, and vice versa.

**Theorem (Sipser’s Theorem 1.28)**
A language is regular if and only if some regular expression describes it.
2 The Pumping Lemma

lemma \(= \text{def} \) an auxiliary proposition (theorem) used in the demonstration of another proposition (theorem)

2.1 What is the Pumping Lemma useful for?

- We know that a language is regular if we can construct a finite state automaton for it.

- Not all languages are regular. How do we know if a language is not regular? Can we simply conclude that the language is not regular if we cannot construct an FSA for it? Not really. Perhaps we were unable to construct an FSA for the language because we hadn’t tried hard enough or because we were unlucky.

- We need some systematic method for showing that a language is not regular and, therefore, that an FSA cannot be constructed for it.

- The Pumping Lemma states a deep property that all regular languages share. By showing that a language does not have the property stated by the Pumping Lemma, we can guarantee that it is not regular.

2.2 The Pigeon Hole Principle

- If \( p \) number of pigeons are placed into fewer than \( p \) holes, some hole has to have more than one pigeon in it.

- Similarly, if an FSA has \( n \) number of states, and this machine accepts strings of length \( n \) or greater, it will have to pass through at least one state more than once in order to accept such strings.

That is, there will be a loop in the machine.

\[ q_1, q_2, q_3, \ldots, q_k, \ldots q_{n-1}, q_n \]

- This means that there is some substring that is read by the sequence of states: \( q_k, \ldots, q_k \)

- Given a string with length \( n \) or greater, which has a substring read by looping through \( q_k \), we can construct even longer strings of the language by repeating (pumping) that substring over and over again.

- See also the discussion in Partee pages 468-471.
2.3 What does the Pumping Lemma say?

2.3.1 Sipser’s Theorem 1.37: Pumping Lemma

If $A$ is a regular language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xy^i z$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

2.3.2 Translation

- The Pumping Lemma says that if a language $A$ is regular, then any string in the language will have a certain property, provided that it is ‘long enough’ (that is, longer than some length $p$, which is the pumping length).

Inside any string in $A$ that’s longer than $p$, we will be able to find a piece that can be repeated (pumped) as many times as we want, and the result will always be a string in $A$. Moreover, this piece can be found within the first $p$ letters of our string.

- So, given any string $s$ in $A$ longer than $p$, we can find a substring in $s$ that can be pumped. We’ll call this substring $y$. Then anything before $y$ we’ll call $x$, and anything after $y$ we’ll call $z$.

The whole string can thus be rewritten as $x - y - z$. (Remember that $x$, $y$, and $z$ correspond to strings and not symbols of the alphabet.)

By repeating $y$ zero or more times, we get:

$$xz, xy^2z, xyy^2z, xyy^3z, \ldots, xyyyyyyyyyyyyzz, \ldots$$

What the Pumping Lemma says is that each of these strings must be in $A$.

- Condition 1: “for each $i \geq 0$, $xy^i z \in A$”

$xy^2z$ is the same as $xyyz$, etc. So this says that putting in multiple copies of $y$ (i.e., pumping $y$) will give you strings that are still in the language. For $i = 0$, you get no copies of $y$, i.e., the string $xz$.

- Condition 2: “$|y| > 0$”

While $x$ or $z$ may have length zero, the length of $y$ cannot be zero. That is, $y$ is not the empty string. If you allowed $y$ to be the empty string, the theorem would be trivially true. This is because if $y$ was the empty string, you would end up with $xz$, which is just $s$, the original string you started with, no matter how many times you pump $y$.

- Condition 3: “$|xy| \leq p$”

Since $x$ is the piece before $y$, this says that all of $y$ must come from the first $p$ letters of our string $s$, so that the combined length of $x$ and $y$ is at most $p$.  

4
2.4 Example

- Let’s apply the Pumping Lemma to the following language \( B \).
  \( B = \{w \mid w \text{ begins with } 1 \text{ and ends with } 0, \text{ with anything in between}\} \).

Let’s assume that the pumping length \( p \) is 3. Let’s take some string longer than 3, say, 10100010. We can break this string down as follows:

\[ x = 1, \ y = 01, \ z = 00010 \]

By pumping \( y \), we get:

\[ xy^0z = 1-00010, \ xy^1z = 1-01-00010, \ xy^2z = 1-0101-00010, \ xy^3z = 1-010101-00010, \]

All of these strings begin with 1 and end with 0. So, the pumping lemma works for this language and this string.

- Question: What happens if we apply the Pumping Lemma to the following language \( C \), assuming that the pumping length \( p \) is 4?
  \( C = \{01\} \)

2.5 How to use the Pumping Lemma

2.5.1 ...to prove that a language is not regular

The pumping lemma is most useful when we want to prove that a language is not regular. We do this by using a proof by contradiction.

To prove that language \( B \) is not regular:

1. Assume that \( B \) is regular.

2. Use the pumping lemma to guarantee the existence of a pumping length \( p \) such that all strings of length \( p \) or greater in \( B \) can be pumped.

3. Find a string \( s \) in \( B \) that has length \( p \) or greater but that cannot be pumped.

4. Demonstrate that \( s \) cannot be pumped by considering all ways of dividing \( s \) into \( x, y, \) and \( z \), and for each division, finding a value \( i \) where \( xy^i z \notin B \).

\[ \Rightarrow \text{The existence of } s \text{ contradicts the pumping lemma if } B \text{ were regular. Hence } B \text{ cannot be regular.} \]
2.5.2 Example 1 (Sipser’s Example 1.38)

(See also Partee pages 470-471 for a discussion of this example.)

Let $B$ be the language $\{0^n1^n \mid n \geq 0\}$. Show that $B$ is not regular, using the pumping lemma.

We will do this by assuming that $B$ is regular and showing that a contradiction follows. (Therefore, the assumption we started out with must have been wrong, and thus $B$ is not regular.)

- Let $p$ be the pumping length given by the pumping lemma. Choose $s$ to be the string $0^{p-1}1^{p-1}$.
- Because $s \in B$, and $s$ has length greater than $p$, the pumping lemma guarantees that we can split $s$ into three pieces, $s = xyz$ in such a way that for any $i \geq 0$, the string $xy^iz$ is in $B$. We consider three cases to show that this result is impossible.

1. The string $y$ contains only 0s. In this case, the string $xyyz$ has more 0s than 1s and so is not a member of $B$, violating condition 1 of the pumping lemma. This is a contradiction.

2. The string $y$ contains only 1s. In this case, the string $xyyz$ has more 1s than 0s and so is not a member of $B$. This is another contradiction.

3. The string $y$ contains both 0s and 1s. In this case, the string $xyyz$ may have the same number of 0s and 1s, but they will be out of order with some 1s before 0s. But in our language $B$, all the 0s must precede the 1s. Thus, $xyyz$ is not in our language. This is another contradiction.

- There is no other way to split up the string $s$, so a contradiction is unavoidable if we make the assumption that $B$ is regular, and so $B$ is not regular.

3 Right Linear Grammars

Right linear grammars are also called Type 3 grammars.

3.1 Formal Definition of Right Linear Grammars

- A right linear grammar is a 4-tuple $< T, N, S, R >$ where:

  1. $T$ is a finite set of terminals, including the empty string.
  2. $N$ is a finite set of non-terminals.
  3. $S$ is the start symbol.
  4. $R$ is a finite set of rewrite rules of the form $A \rightarrow xB$ or $A \rightarrow x$, where $A$ and $B$ stand for non-terminals and $x$ stands for a terminal.
3.2 Finite State Automata and Right Linear Grammars

Every FSA has a corresponding right linear grammar and vice versa. That is, for every FSA, there is an equivalent right linear grammar that accepts the same language, and vice versa.

3.2.1 Converting a right linear grammar to an equivalent FSA

1. $S$ is the start state.

2. Associate with each rule of the form $A \to xB$ a transition in a FSA from state $A$ to state $B$ reading $x$.

3. Associate each rule of the form $A \to x$ with a transition from state $A$ reading $x$ to a final state, $F$.

- Example: Representing $G1$ above as a FSA:
3.2.2 Converting a FSA to a right linear grammar

1. The states of the FSA become the non-terminals of the grammar, and the symbols of the alphabet of the FSA become the terminals of the grammar.

2. $q_0$ becomes the start symbol $S$.

3. For each transition $\delta(q_i, x) = q_j$, we put in the grammar a rule $q_i \rightarrow xq_j$. (E.g., the transition $\delta(A, 0) = B$ becomes $A \rightarrow 0B$.)

4. For each transition $\delta(q_i, x) = q_j$, Where $q_j$ is a final state, we add to the grammar the rule $q_i \rightarrow x$. (E.g., the transition $\delta(D, 1) = E$ where $E \in F$, becomes $D \rightarrow 1$.)

- Example:

![Diagram](attachment:image.png)

$G_2 = \langle T, N, q_0, R \rangle$ where $T = \{a, b\}; \ N = \{q_0, q_1\};$ and

$$R = \{\begin{array}{l}
q_0 \rightarrow aq_0 \\
q_0 \rightarrow bq_1 \\
qu_1 \rightarrow aq_1 \\
qu_1 \rightarrow bq_0 \\
q_0 \rightarrow b \\
qu_1 \rightarrow a
\end{array}\}$$

3.3 English is not a regular language.

- Example 1

Let $L = \{The \ cat \ died, \ The \ cat \ the \ dog \ chased \ died, \ The \ cat \ the \ dog \ the \ rat \ bit \ chased \ died, \ The \ cat \ the \ dog \ the \ rat \ the \ elephant \ admired \ bit \ chased \ died \ ...\}$

The language $L$ can basically be described as: $(\text{noun})^n \ (\text{verb})^n$. This is a nested dependency structure, and it corresponds to $\{0^n1^n \mid n \geq 0\}$. We have already shown above that this language is not regular using the pumping lemma.

- Example 2

Let $L_2 = \{\text{John and Mary like to eat and sleep, respectively; John, Mary, and Sue like to eat, sleep, and dance, respectively; John, Mary, Sue, and Bob like to eat, sleep, dance, and cook, respectively...}\}$
This is a cross-serial dependency structure, also described by \((\text{noun})^n (\text{verb})^n\), which again corresponds to \(\{0^n1^n \mid n \geq 0\}\). Of course, we have already seen that this language is not regular using the pumping lemma.

- Other examples:
  subject-verb agreement
  either...or
  if...then

See the discussion in Partee pages 477-479 and of course Pinker pages 92-97.