

# Regular Languages: The Pumping Lemma

LING 106

March 18, 2009

## 1. REGULAR LANGUAGES AND FSAS

Recall that by definition:

- A **REGULAR LANGUAGE** is a language that can be modeled with a FSA.

Suppose we have a language  $L$ . Then:

- If we want to prove that  $L$  is regular, we construct a FSA that models it.
- If we want to prove that  $L$  is not regular, we...don't construct a FSA?
  
- If we can construct a FSA that models  $L$ , then  $L$  is regular.
- If we can't figure out how to construct a FSA that models  $L$ , then either  $L$  is not regular or we're not clever enough.

We need a better method. Ergo:

To prove that a language **IS REGULAR**, construct a FSA.  
To prove that a language **IS NOT REGULAR**, use the Pumping Lemma.

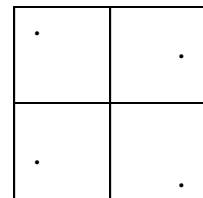
## 2. BACKGROUND: THE PIGEONHOLE PRINCIPLE

**Definition:** The **PIGEONHOLE PRINCIPLE** states that if you put  $x$  objects into  $y$  holes and  $x > y$ , then some hole must have more than one object in it.

### 2.1. Examples

- Prove that, if you place five dots at random into a  $1'' \times 1''$  square, there will be some pair less than  $.71''$  apart.

*Proof:* Divide the square into quadrants. The diagonal of each quadrant is  $(\sqrt{2})/2 \approx .707''$  long, so that's the farthest apart two dots within a single quadrant can be.



Now place the dots into the square. By the Pigeonhole Principle, because there are five dots and four quadrants, some quadrant must have more than one dot. Therefore, the pair of dots in the same quadrant must be less than  $.71''$  apart.

- If you pick five cards from a normal deck, two of them will be the same suit.

*Proof:* There are four “holes” (clubs, diamonds, hearts, spades). You can put one object in each hole for the first four objects—i.e., the first four cards can each be a different suit—but the fifth object will have to go into an occupied hole—i.e., it’ll have to be the same suit as one of the other cards.

## 2.2. *Applying the principle to regular languages and FSAs*

- Partee et al., p. 468: “Consider an infinite [regular language]  $L$ . By definition, it is accepted by some FSA,  $M$ , which, again by definition, has a finite number of states. But since  $L$  is infinite, there are strings in  $L$  which are as long as we please, and certainly  $L$  contains strings with more symbols than the number of states in  $M$ . Thus, since  $M$  accepts every string in  $L$ , there must be a loop in  $M$ ...”
- That is: suppose we have a FSA with  $y$  states. Accepting a string of length  $z$  requires passing through (at least)  $z$  states, because each symbol in the string corresponds to a transition, and thus to another state entered.<sup>1</sup> Thus, any string of length  $x > y$  that it accepts will have to pass through at least  $x$  states, and thus by the Pigeonhole Principle, it must pass through some state more than once.
- Thus, there must be a loop: if  $q_k$  can be entered more than once, then a possible pathway is:

$$q_0, q_1, \dots, q_k, \dots, q_k, \dots, q_{n-1}, q_n$$

Consequently, there is a substring  $s$  read by the sequence  $q_k, \dots, q_k$ . (Note that the ellipses may be empty, in which case the substring is of length 1.)

- And because there is a loop, we can construct longer strings of the language by repeating the loop—that is, by repeating substring  $s$ , aka “pumping” it.

---

<sup>1</sup> “At least”, because in fact the number of states entered = the number of symbols in the string + the number of  $\epsilon$  transitions taken.

### 3. THE PUMPING LEMMA

$A$  is a regular language if there is a number  $p$ , the *pumping length*, such that:  
 if  $s$  is any string in  $A$  of length at least  $p$ , then  $s$  can be divided into three pieces,  $s = xyz$ , such that:

1. for each  $i \geq 0$ ,  $xy^iz \in A$  ( $y^i =$  “string  $y$ , repeated  $i$  times”)
2.  $y \neq \epsilon$
3.  $|xy| \leq p$

#### 3.1. Explanation of the Pumping Lemma

- 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’s “long enough” (i.e., longer than the pumping length)
- That property is that the string has a (non-null) substring that can be pumped and still produce strings of the language. We’ll call that substring  $y$ ; anything before it we’ll call  $x$ , and anything after it we’ll call  $z$ . So the string is  $x \circ y \circ z$ .
- “The substring can be pumped” means that we can repeat it zero or more times:

$$x \circ z, x \circ y \circ z, x \circ yy \circ z, x \circ yyy \circ z, \dots, x \circ yyyyyyyyyyyyyyy \circ z, \dots$$

e.g. if  $x = \mathbf{0001}$ ,  $y = \mathbf{10}$ , and  $z = \mathbf{00}$ , then we would have the strings

$x$	$z$	$x$	$y$	$z$	$x$	$yy$	$z$
<b>000100, 00011000, 0001101000, ...</b>							

What the Pumping Lemma tells us is that, if  $A$  is regular...

- Condition 1: all of these strings are in  $A$ .
- Condition 2: while  $x$  or  $z$  might be empty strings,  $y$  is not. (If  $y$  were empty, the lemma would be trivially true:  $xy^iz = xz$  for all  $i$ .)
- Condition 3: we can find the substring  $y$  within the first  $p$  symbols—the combined length of  $x$  and  $y$  is at most  $p$ .

### 3.2. *Caveat! What the pumping lemma cannot do*

The pumping lemma says “If  $p$  is true, then  $q$  is true”—that is, if it’s true that a language is regular, then it’s true that there’s some pumping length. The lemma does not...

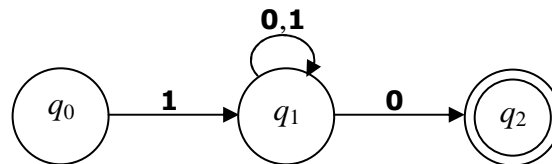
- Tell you how to find that length.
- Allow you to prove that a language is regular. It states that “If  $p$ , then  $q$ ”, but you can’t use that to conclude “If  $q$ , then  $p$ ”. For example, it’s certainly true that if I have a brother, I’m not an only child; but you can’t conclude from that statement that if I’m not an only child, I have a brother. (I could have a sister.)

In this case, you can’t reason as follows: Language  $Z$  has a “pumping length” for which you can find a string that, when repeated, gives you more strings in the language. Therefore, Language  $Z$  must be regular.

- Work for every non-regular language—see above. You also can’t use “If  $p$  is true, then  $q$  is true” to conclude “If  $p$  is false, then  $q$  is false”. Using the example above, you can’t conclude that if I don’t have a brother, I am an only child. (Again, I could have a sister.)

### 3.3. *Examples with regular languages*

- $B = \{w \mid w \text{ begins with } \mathbf{1} \text{ and ends with } \mathbf{0}\}$



As it happens,  $p = 3$  is a valid pumping length.<sup>2</sup> Then any string of length 3 or greater will work; let’s use **10100010**. We can break it into

$$x = \mathbf{1}, y = \mathbf{01}, z = \mathbf{00010}$$

in which case we can pump  $y$ , and the Pumping Lemma tells us that all of the strings at the top of the next page....

---

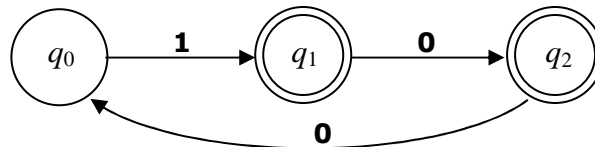
<sup>2</sup> Because I said so. Again, the Lemma doesn’t tell you how to find the length; I spent some quality time analyzing the language to reach this conclusion.

$$\begin{aligned}
 xy^0z &= \mathbf{1}[\ ]\mathbf{00010} \\
 xy^1z &= \mathbf{1}[\mathbf{01}]\mathbf{00010} \\
 xy^2z &= \mathbf{1}[\mathbf{0101}]\mathbf{00010} \\
 xy^3z &= \mathbf{1}[\mathbf{010101}]\mathbf{00010} \\
 &\dots
 \end{aligned}$$

are strings in  $B$ . (Which they are, since they start with  $\mathbf{1}$  and end with  $\mathbf{0}$ .)

Note that this isn't a proof that  $B$  is regular! Even knowing that  $B$  obeys the pumping lemma doesn't prove that it's regular (see previous section). This is just a demonstration of what the lemma means.

- **Question:** Show that the strings  $\mathbf{100}$  and  $\mathbf{1100}$  can also be divided in a way that complies with the Pumping Lemma.
- **Question:** Apply the Pumping Lemma to language  $C$ , modeled by the following FSA, using the strings below. Assume the pumping length is 3.



- a.  $\mathbf{1001}$
  - b.  $\mathbf{10010}$
  - c.  $\mathbf{1001001}$
  - d.  $\mathbf{100}$
  - e.  $\mathbf{10}$
- **Exercise:** Apply the Pumping Lemma to language  $E = \{w \mid w \text{ contains the substring } \mathbf{101} \text{ exactly once}\}$ , with pumping length 4. Try at least three different strings. Draw the FSA for the language if you think it would help.

#### 4. USING THE PUMPING LEMMA TO PROVE THAT A LANGUAGE IS NOT REGULAR

Again: the Pumping Lemma states that, if a language  $L$  is regular, then all strings in  $L$  that are at least  $p$  symbols long have a valid cutting based on  $p$ . The contrapositive<sup>3</sup> of this statement is:

If language  $L$  has at least one string that is at least  $p$  symbols long, and that has no valid cutting based on  $p$ , then language  $L$  is not a regular language.

<sup>3</sup> Technical term. Given *If p, then q*, the converse (*If q, then p*) and the inverse (*If not p, then not q*) aren't necessarily true, but the contrapositive (*If not q, then not p*) is. Example of a contrapositive: from "If I have a brother, I am not an only child", it does follow that "If I am an only child, then I don't have a brother".

So we can prove that a language is not regular by contradiction:

- Assume that  $L$  is regular.
- Use the Pumping Lemma to guarantee the existence of a pumping length  $p$ .
- Find a string  $s$  in  $L$  that has length  $p$  or greater, but that cannot be pumped. Demonstrate that it cannot be pumped by considering all ways of cutting  $s$  into  $x$ ,  $y$ ,  $z$  and show that none of them are valid cuttings.

If  $L$  were regular as assumed, the existence of  $s$  would contradict the pumping lemma. Therefore,  $L$  is not regular.

#### 4.1. Example of a non-regular language

We've seen a lot of languages that "count"—languages whose strings must be  $n$  symbols long, or have at least  $n$  of some symbol, or whose  $n$ th character or  $n$ th-to-last character must be a certain symbol, etc. But not all counting is possible...

Let  $G = \{0^n 1^n \mid n \geq 0\}$ . ( $x^n = "x, \text{repeated } n \text{ times}"$ .) To show that  $G$  is not regular...

##### 4.1.1. Proof #1

- Assume that  $G$  is regular. Then there is some pumping length  $p$ .
- Take  $s = 0^p 1^p$ . (e.g., if  $p = 3$ , then  $s = 000111$ .)
- The Pumping Lemma tells us that there's a  $y$  in the first  $p$  symbols that can be pumped. But: if  $y$  is in the first  $p$  symbols, it contains only **0**s; so when we pump it, we end up with more **0**s than **1**s, i.e. we end up with a string that's not in  $G$ .
- So it's not the case that every string in  $G$  has a valid cutting, for *any* value of  $p$ ; i.e., there is no pumping length  $p$ , in contradiction to what we had before.
- Therefore, the assumption was wrong, and  $G$  is not regular.

##### 4.1.2. Proof #2

- Assume that  $G$  is regular. Then there is some pumping length  $p$ .
- Take any string whose length is greater than  $p$ . The Pumping Lemma tells us that we can split  $s$  into three pieces,  $xyz$ , such that the conditions of the lemma are met. How can we cut  $s$ ?
  - We could cut it so that  $y$  contains only **0**s. In that case,  $xyyz$  will contain more **0**s than **1**s, and is not a member of language  $G$ . (And string  $xz$  will contain fewer **0**s than **1**s.)

- We could cut it so that  $y$  contains only **1**s. But again,  $xyyz$  will contain more **1**s than **0**s, and is not a member of language  $G$ . (And string  $xz$  will contain fewer **1**s than **0**s.)
- We could cut it so that  $y$  contains both **0**s and **1**s. Now  $xyyz$  may contain the same number of **1**s and **0**s—but they won't be in the right order. That is,  $y$  must be of the form **0...1**, so  $yy$  is of the form **0...10...1**, but in the strings of  $G$ , the **0**s all precede the **1**s. So once again, this string is not a member of language  $G$ .
- And of course if we cut it so that  $y$  contains neither **0**s nor **1**s, then  $y = \epsilon$ , which is disallowed by the second condition.
- Therefore, we have a string  $s$  that cannot be split, contrary to the Pumping Lemma. Therefore, the original assumption is wrong, and  $G$  is not regular.

## 4.2. Another example of a non-regular language

Let  $H = \{w \mid w \text{ has an equal number of } \mathbf{0}\text{s and } \mathbf{1}\text{s, in any order}\}$ .

### 4.2.1. Proof #1

- Assume that  $H$  is regular. Let  $p$  be the pumping length. We have three options, much like the ones above:
  - $y$  contains more **0**s than **1**s;
  - $y$  contains more **1**s than **0**s;
  - $y$  contains the same number of **0**s and **1**s.
- If  $y$  contains more **0**s than **1**s, then  $xyyz$  will contain more **0**s than **1**s; similarly if  $y$  contains more **1**s than **0**s.
- If  $y$  contains the same number of **1**s and **0**s...the step from the previous problem won't work here, because  $xyyz$  will indeed be in  $H$ . How can we stop this?

Answer: take the string  $s = \mathbf{0}^p\mathbf{1}^p$ .  $s$  has no **1**s in the first  $p$  symbols, so the only way for  $y$  to contain the same number of **0**s and **1**s is...

- ...if  $y$  contains none; but then  $y = \epsilon$ , which is disallowed;
- ...if  $y$  crosses the midpoint of  $s$ , but then  $|xy| > p$ , which is also disallowed.
- Therefore, we have shown that, no matter what  $p$  is, there is some string  $s \in H$  whose length is greater than  $p$ , and which cannot be cut according to the Pumping Lemma. (The Pumping Lemma guarantees that all strings of that sort can be cut.) Therefore,  $H$  is not regular.

### 4.2.2. Proof #2

- $I = \{0^*1^*\}$  is a regular language.
- Suppose  $H$  is a regular language. Then because regular languages are closed under intersection,  $H \cap I$  is a regular language.
- $H \cap I = \{w \mid w \text{ has the same number of } 0\text{s and } 1\text{s, and the } 0\text{s all precede the } 1\text{s}\}$ , which must be regular. But in fact: that's exactly language  $G = \{0^n1^n\}$ , and we have shown that  $G$  is not regular, so our assumption has led us to a contradiction.
- Therefore,  $H$  is not a regular language.

### 4.3. Another example, using “pumping down”

Let  $J = \{0^a1^b \mid a > b\}$ . Show that  $J$  is not regular.

- Once again, assume  $J$  is regular, in which case it has pumping length  $p$ .
- Let  $s = 0^{p+1}1^p$ .  $s \in J$ , and  $|s| > p$ , so it can be cut, in which case  $y$ ...
  - ...contains at least one **1**. In this case,  $|xy| > p$ , because no **1**s in the string occur in the first  $p$  symbols, and this is disallowed.
  - ...contains only **0**s (and at least one **0**). In this case,  $y \neq \epsilon$  and it's possible that  $|xy| \leq p$ . And  $xyz \in J$ , because adding more iterations of  $y$  increases the number of **0**s, so the string will have more **0**s than **1**s. So now...?

Condition 1 of the Pumping Lemma states that  $xy^iz \in J$  even when  $i = 0$ . So consider  $xy^0z = xz$ . Removing  $y$  decreases the number of **0**s in  $s$ . But  $s$  has only more **0** than **1**, so removing even a single **0** will give a string  $0^a1^b$  where  $a \leq b$ . So  $xz \notin J$ , so this is also not a valid cutting.

- Therefore, there is no way to cut  $s$ , etc. etc.,  $J$  is not regular.

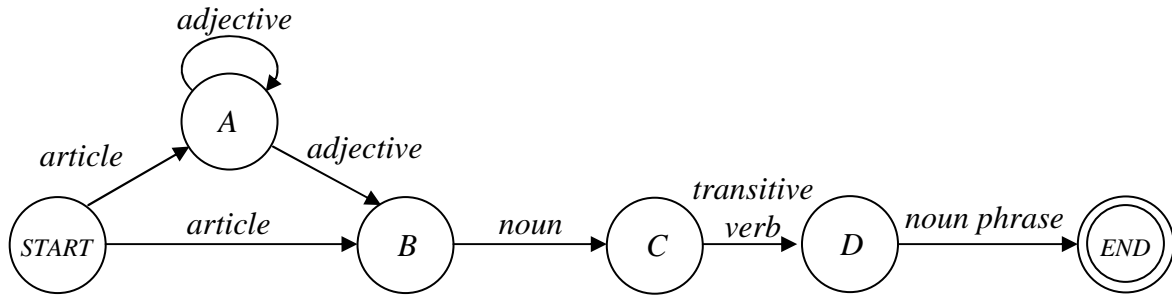
**Exercise:** Show that languages  $K, L, M$  are not regular, using the Pumping Lemma and reasoning abstractly about  $p$ , as above:

- $K = \{001^n01^n \mid n \geq 0\}$
- $L = \{01^n001^{n+2} \mid n \geq 0\}$
- $M = \{0^n1^m \mid n = m + 3\}$

## 5. IS ENGLISH A REGULAR LANGUAGE?

We can now prove that languages aren't regular, and that languages are regular. We still have the question: what about English?

Perhaps it's regular. There was a regular language we saw a month ago...



...and it does seem to model the basic approach, i.e. our knowledge of grammar involves knowing that nouns follow articles, verbs follow subjects, objects follow transitive verbs, etc. It needs to be larger—we'd want  $\langle C, \textit{intransitive-verb} \rangle \rightarrow \text{END}$  as another transition, for instance, but that's just a question of scale, not possibility.

And yet...

English is *not* a regular language. (Chomsky, 1950s)

Why not?

### 5.1. Relative Clause Dependency

Relative clauses involve *center embedding*, which cannot be regular.

- Step 1: consider Language  $G$ , with brackets added for ease of reading.

$G = \{$  **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 saw] bit] chased] died, ...}**

i.e.,  $G = \{(\mathbf{the\ noun})^n (\mathit{transitive-verb})^{n-1} \mathit{intransitive-verb}\}$

After a while, sentences in  $G$  become impossible to process; but in theory we recognize them as legitimate sentences of English.

It's not hard to show that this is not regular: we showed that  $\{0^n 1^n \mid n \geq 0\}$  is not regular, and this works out much the same way.

- Step 2: consider the language  $H = \{(\text{the noun})^* (\text{transitive-verb})^* \text{died}\}$ .  $H$  is regular.
- Step 3:  $G = \text{English} \cap H$ , so if English is regular,  $G$  is regular. But  $G$  is not regular; therefore, English is not regular.

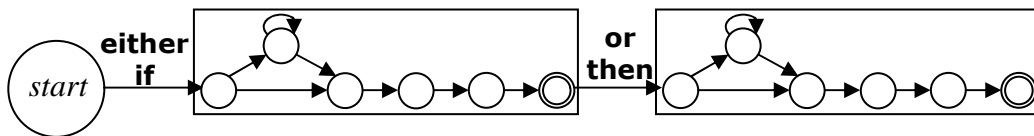
**Exercise:** show that  $G$  is not regular and that  $H$  is regular.

## 5.2. Long Distance Dependency

Finite state automata have no memory—which is why they can't "count" in ways that require remembering how many 0s there were and then putting in that many 1s. It also means they can't keep track of items that depend on each other across arbitrarily long strings:

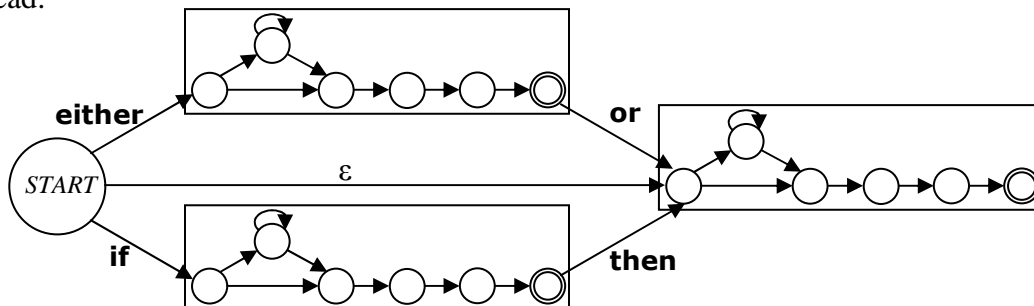
- (1) a. **Either the girl eats ice cream or the girl eats candy.**
- b. **If the girl eats ice cream, then the boy eats hot dogs.**
- c. \***Either the girl eats ice cream then the girl eats candy.**
- d. \***If the girl eats ice cream, or the boy eats hot dogs.**

Let's extend Language  $E$ —the box here is  $E$  itself:



**Problem:** when we're at the **or/then** transition, the FSA has no memory of what came before, so it'll generate sentences like (1c) and (1d).

Instead:



**Problem:** these can nest arbitrarily deeply.

- (2) a. **If [either the girl eats ice cream or the girl eats candy], then the boy eats hot dogs.**
- b. **Either [if the girl eats ice cream, then the boy eats hot dogs], or [if the girl eats candy, then the boy eats cheesesteaks].**

Thus, we'd need to keep duplicating the above machine arbitrarily deep...at which point it no longer has a finite number of states.

## 6. RIGHT LINEAR GRAMMARS

A new way of modeling a language:

**Definition:** A **RIGHT LINEAR GRAMMAR** is a 4-tuple  $\langle T, N, S, R \rangle$  where:

- $T$  is a finite set of terminal symbols, including the empty string;
- $N$  is a finite set of nonterminal symbols;
- $S$  is the start symbol;
- $R$  is a set of rewrite rules of the form  $A \rightarrow xB$  or  $A \rightarrow x$ , where  $A$  and  $B$  are nonterminals and  $x$  is a terminal.

Example:

- $G_1 = \langle T, N, S, R \rangle$ , where  $T = \{\mathbf{a}, \mathbf{b}\}$ ,  $N = \{X, A, B\}$ ,  $X$  is the start symbol, and

$$R = \left\{ \begin{array}{l} X \rightarrow \mathbf{aA} \\ A \rightarrow \mathbf{aA} \\ A \rightarrow \mathbf{bB} \\ B \rightarrow \mathbf{bB} \\ B \rightarrow \mathbf{b} \end{array} \right\}$$

- For instance, using parentheses to mark rewrite rules, **aaabb** is a string of  $G_1$  because:  $X \Rightarrow \mathbf{aA} \Rightarrow \mathbf{a(aA)} \Rightarrow \mathbf{a(a(aA))} \Rightarrow \mathbf{a(a(a(bB)))} \Rightarrow \mathbf{a(a(a(b(b))))}$ .

## 7. RIGHT LINEAR GRAMMARS AND FSAS

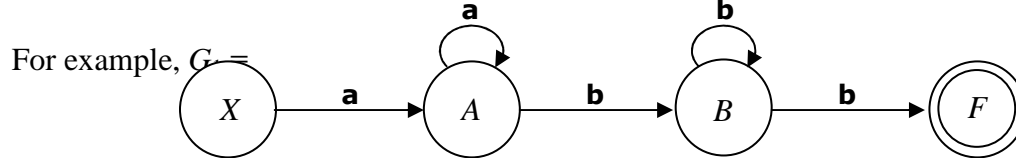
Are RLGs more powerful than FSAs? No—in fact:

**Proposition:** RLGs and FSAs are equivalent: they model exactly the same languages.

### 7.1. Converting a RLG to a FSA

If  $G = \langle T, N, S, R \rangle$  is a right linear grammar, then  $D = \langle Q, \Sigma, \delta, s, F \rangle$  is an equivalent FSA, where:

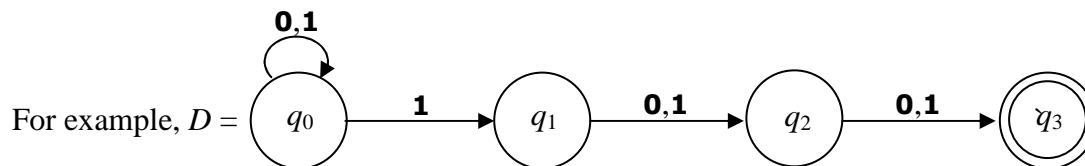
- $Q = N$  plus a final state  $F$
- $\Sigma = T$
- $s = S$
- $F =$  the final state  $F$ , and
- For each rule of the form  $A \rightarrow xB$ ,  $\delta(\langle A, x \rangle) = B$ , and  
 For each rule of the form  $A \rightarrow x$ ,  $\delta(\langle A, x \rangle) = F$



### 7.2. Converting a FSA to a RLG

If  $D = \langle Q, \Sigma, \delta, s, F \rangle$  is a FSA, then  $G = \langle T, N, S, R \rangle$  is an equivalent right linear grammar, where:

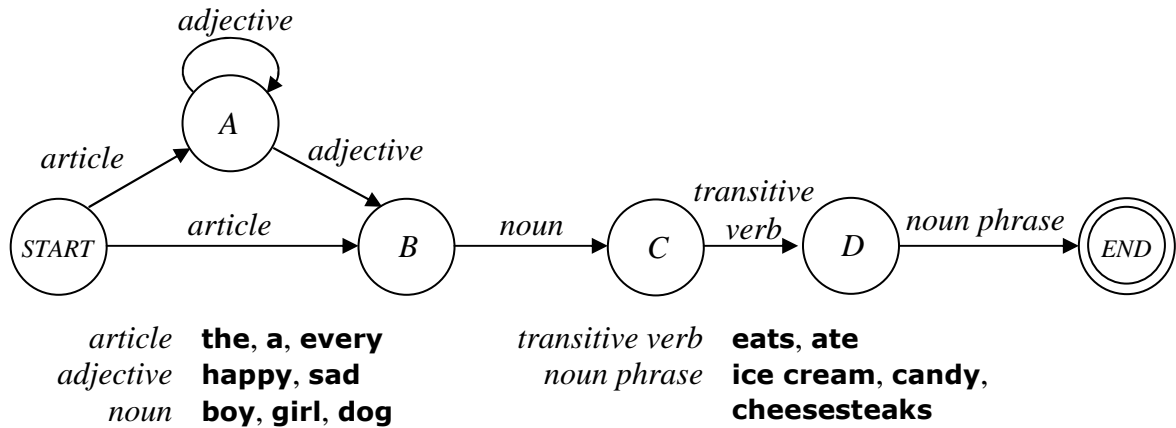
- $T = \Sigma$
- $N = Q$
- $S = s$
- For each transition  $\delta(\langle q_i, x \rangle) = q_j$ :  $[q_i \rightarrow xq_j] \in R$ , and  
 For each transition  $\delta(\langle q_i, x \rangle) = q_j$  where  $q_j$  is a final state:  $[q_i \rightarrow x] \in R$ , and  
 If  $s \in F$ , then  $[S \rightarrow \epsilon] \in R$



gives  $G$ , where  $T = \{0, 1\}$ ,  $N = \{q_0, q_1, q_2, q_3\}$ ,  $S = q_0$ , and  $R =$

$$\left\{ \begin{array}{l} q_0 \rightarrow 0q_0 \\ q_0 \rightarrow 1q_0 \\ q_0 \rightarrow 1q_1 \\ q_1 \rightarrow 0q_2 \\ q_1 \rightarrow 1q_2 \\ q_2 \rightarrow 0 \\ q_2 \rightarrow 1 \end{array} \right\}.$$

Another example, Language *E*:



$T = \{\mathbf{the, a, every, happy, sad, boy, girl, dog, eats, ate, ice cream, candy, cheesesteaks}\}$

$N = \{START, A, B, C, D, \textit{article}, \textit{adjective}, \textit{noun}, \textit{transitive-verb}, \textit{noun-phrase}\}$

$R = \{ S \rightarrow \textit{article} A, \quad S \rightarrow \textit{article} B, \quad A \rightarrow \textit{adjective} A, \quad A \rightarrow \textit{adjective} B,$   
 $B \rightarrow \textit{noun} C, \quad C \rightarrow \textit{transitive-verb} D, \quad D \rightarrow \textit{noun-phrase},$   
 $\textit{article} \rightarrow \mathbf{the} / \mathbf{a} / \mathbf{every},^4$        $\textit{adjective} \rightarrow \mathbf{happy} / \mathbf{sad},$   
 $\textit{noun} \rightarrow \mathbf{boy} / \mathbf{girl} / \mathbf{dog},$        $\textit{transitive-verb} \rightarrow \mathbf{eats} / \mathbf{ate},$   
 $\textit{noun-phrase} \rightarrow \mathbf{ice cream} / \mathbf{candy} / \mathbf{cheesesteaks}\}$

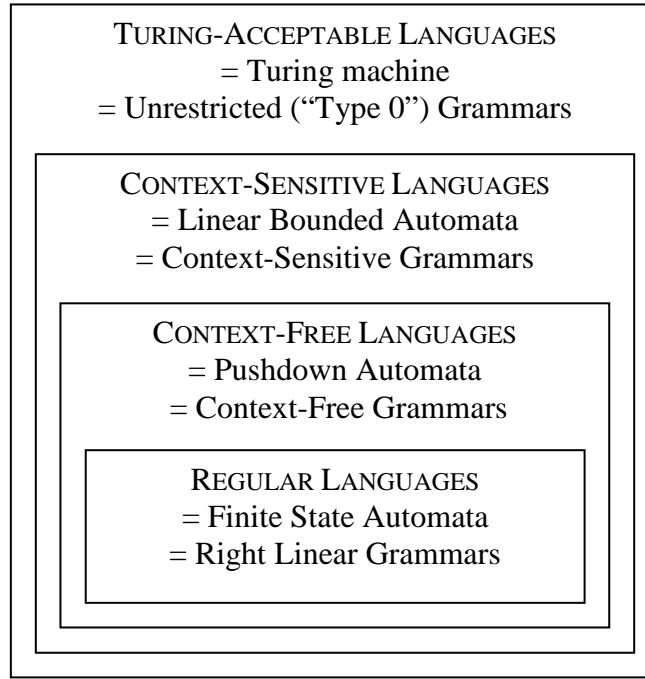
**Exercise:** For each of the following language, construct a FSA and its corresponding RLG.

- $\{\mathbf{ab, bb}\}$
- $\{\mathbf{a*ab*}\}$
- $\{w \mid w \text{ contains at least one occurrence of } \mathbf{a} \text{ and at least one occurrence of } \mathbf{b}\}$
- $\{w \mid w \text{ contains at most three } \mathbf{as}\}$

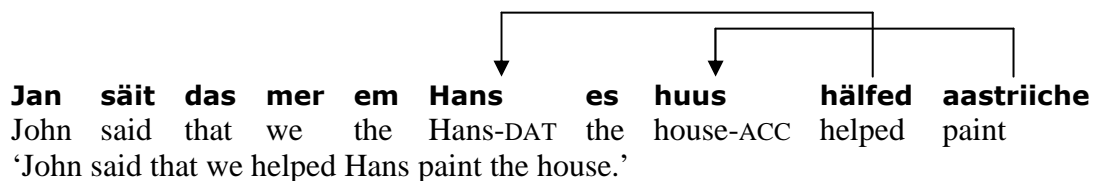
<sup>4</sup> Shorthand for the three rules  $\textit{article} \rightarrow \mathbf{the}$ ,  $\textit{article} \rightarrow \mathbf{a}$ ,  $\textit{article} \rightarrow \mathbf{every}$ .

## 8. THE “ONION” MODEL

- Chomsky (1963):



- Regular languages: e.g.  $\mathbf{a^*b^*}$ ,  $\mathbf{a^*bcd^*}$   
Right Linear Grammars:  $A \rightarrow xB$  or  $A \rightarrow x$ 
  - Characterized by having **no long-distance dependencies**
  - Chomsky (1957): natural language has long-distance dependencies (e.g. relative clauses) and thus cannot be a regular language.
- Context-free languages: e.g.  $\{\mathbf{a^n b^n} \mid n \geq 0\}$ ,  $\{\mathbf{a^n b^m c^m d^n} \mid m, n \geq 0\}$   
Context Free Grammars:  $A \rightarrow xBy$ 
  - Allow **nested long-distance dependencies**—e.g., in  $\mathbf{a^n b^m c^m d^n}$ , the lengths of the two inner strings are dependent, and the lengths of the two outer strings are dependent.
  - Shieber (1985): natural language has crossed long-distance dependencies, or at least Swiss German does...



The key fact: the dative-taking verb and its dative object *cross* the connection between the accusative-taking verb and its accusative object. Thus natural language is not context-free.

- Context-sensitive languages: e.g.  $\{a^m b^n c^m d^n \mid m, n \geq 0\}$ ,  $\{w \mid w \text{ contains an equal number of } \mathbf{a}\mathbf{s}, \mathbf{b}\mathbf{s}, \text{ and } \mathbf{c}\mathbf{s}, \text{ in any order}\}$   
Context-sensitive grammars:  $xAy \rightarrow xzy$  (that is, terminals can appear on the left side, as long as the right side is at least as long as the left)
  - Allow for **nested, crossed, shuffled long-distance dependencies**
- Turing-acceptable languages  
Unrestricted grammars:  $xAy \rightarrow x$  (that is, the left side must contain a non-terminal, but terminals can also be changed/removed).

**Question:** How powerful are natural languages?

Answer: natural language is not context-free, but much of it is.

1970s Transformational Grammar: Unrestricted, with some complete rewrite rules

- Movement rules: *subject verb+tense object*  $\rightarrow$  *object **be+tense verb+ed by** subject*  
(e.g. **john adores mary**  $\rightarrow$  **mary is adored by john**)
- Deletion rules: *noun **who is** gerund-clause*  $\rightarrow$  *noun gerund-clause*  
(e.g. **the boy who is studying linguistics**  $\rightarrow$  **the boy studying linguistics**)

More recent thought: Transformational Grammar is *too* powerful. The extra power beyond context-free grammar seems to be “mildly context sensitive”.