# Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

# What is this course about?

- What can formal models of language teach us, if anything, about human language?
- Can we use information theoretic concepts to describe aspects of human language?

This course will:

- extend your knowledge of formal languages
- extend your knowledge of parsing
- introduce some ideas from information theory
- tell you something about human language processing and acquisition

# Study and Supervisions

- Technical handouts: Grammars, Information Theory
- Formal Language vs. Natural Language handouts
- Lecture Slides
- Two supervision worksheets

# Study and Supervisions

## Supervision content

- coding exercises
- some short proofs
- short written answers

## Useful Textbooks

- Jurafsky, D. and Martin, J. *Speech and Language Processing*
- Manning, C. and Schutze, H. *Foundations of Statistical Natural Language Processing*
- Ruslan M. *The Oxford Handbook of Computational Linguistics*
- Clark, A., Fox, C, and Lappin, S. *The Handbook of Computational Linguistics and Natural Language Processing*
- Kozen, D. *Automata and Computability*

# A **natural language** is a human communication system

- A natural language can be thought of as a mutually understandable communication system that is used between members of some population.
- When communicating, speakers of a natural language are tacitly agreeing on what strings are allowed (i.e. which strings are **grammatical**).
- Dialects and specialised languages (including e.g. the language used on social media) are all natural languages in their own right.
- Note that named languages that you are familiar with, such as *French*, *Chinese*, *English* etc, are usually historically, politically or geographically derived labels for populations of speakers rather than linguistic ones.

# A **natural language** has high ambiguity

I made her duck

1. I cooked waterfowl for her
2. I cooked waterfowl belonging to her
3. I created the (plaster?) duck she owns
4. I caused her to quickly lower her head
5. I turned her into a duck

Several types of ambiguity combine to cause many meanings:

- morphological (*her* can be a dative pronoun or possessive pronoun and *duck* can be a noun or a verb)
- syntactic (*make* can behave both transitively and ditransitively; *make* can select a direct object or a verb)
- semantic (*make* can mean *create*, *cause*, *cook* ...)

# A **formal language** is a set of **strings** over an **alphabet**

Alphabet

An alphabet is specified by a **finite** set, $\Sigma$, whose elements are called symbols. Some examples are shown below:

- $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ the 10-element set of decimal digits.
- $\{a, b, c, ..., x, y, z\}$ the 26-element set of lower case characters of written English.
- $\{aardvark, ..., zebra\}$ the 250,000-element set of *words* in the Oxford English Dictionary.[1]

Note that e.g. the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, ...\}$ cannot be an alphabet because it is infinite.

[1] Note that the term *alphabet* is overloaded

# A **formal language** is a set of **strings** over an **alphabet**

## Strings

A string of length $n$ over an alphabet $\Sigma$ is an ordered $n$-tuple of elements of $\Sigma$.

$\Sigma^*$ denotes the set of all strings over $\Sigma$ of finite length.

- If $\Sigma = \{a, b\}$ then $\epsilon$, *ba*, *bab*, *aab* are examples of strings over $\Sigma$.
- If $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, ...\}$
- If $\Sigma = \{cats, dogs, eat\}$ then $\Sigma^* = \{\epsilon, cats, cats\ \ eat, cats\ \ eat\ \ dogs, ...\}$[2]

## Languages

Given an alphabet $\Sigma$ any subset of $\Sigma^*$ is a **formal language** over alphabet $\Sigma$.

[2] The spaces here are for readable delimitation of the symbols of the alphabet.

# Reminder: languages can be defined using **rule induction**

### Axioms

Axioms specify elements of $\Sigma$ that exist in $\mathcal{L}$.

$$\frac{}{a}\ \text{(a1)}$$

### Induction Rules

Rules show **hypotheses** above the line and **conclusions** below the line (also referred to as **children** and **parents** respectively). The following is a unary rule where $u$ indicates some string in $\Sigma^*$:

$$\frac{u}{ub}\ \text{(r1)}$$

# Reminder: languages can be defined using **rule induction**

### Derivations

Given a set of axioms and rules for inductively defining a subset, $\mathcal{L}$, of $\Sigma^*$, a derivation of a string $u$ in $\mathcal{L}$ is a finite rooted tree with nodes which are elements of $\mathcal{L}$ such that:

- the root of the tree (towards the bottom of the page) is $u$ itself;
- each vertex of the tree is the conclusion of a rule whose hypotheses are its children;
- each leaf of the tree is an axiom.

Using our axiom and rule, the derivation for the string $abb$ is:

$$\frac{}{a}\ (\mathrm{a1}) \qquad\qquad \frac{u}{ub}\ (\mathrm{r1}) \qquad\qquad \cfrac{\cfrac{\cfrac{}{a}\ (\mathrm{a1})}{ab}\ (\mathrm{r1})}{abb}\ (\mathrm{r1})$$

# Reminder: languages can also be defined using **automata**

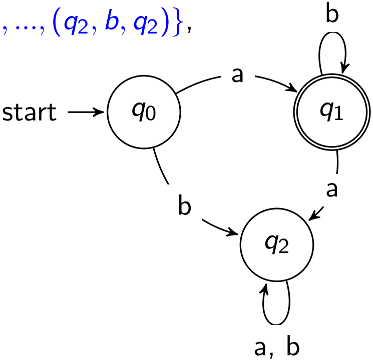
Recall that a language is regular if it is equal to the set of strings accepted by some deterministic finite-state automaton $(\mathrm{DFA})$.
A DFA is defined as $M = (\mathcal{Q}, \Sigma, \Delta, s, \mathcal{F})$ where:

- $\mathcal{Q} = \{q_0, q_1, q_2 ...\}$ is a finite set of states.
- $\Sigma$ is the alphabet: a finite set of transition symbols.
- $\Delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a function $\mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ which we write as $\delta$. Given $q \in \mathcal{Q}$ and $i \in \Sigma$ then $\delta(q, i)$ returns a new state $q' \in \mathcal{Q}$
- $s$ is a starting state
- $\mathcal{F}$ is the set of all end states

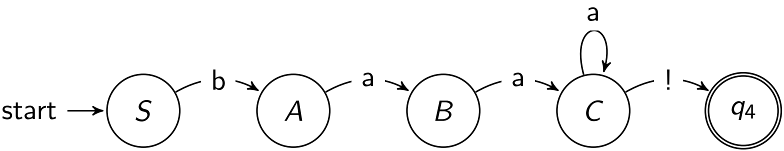
# Reminder: **regular languages** are accepted by **DFAs**

For $\mathcal{L}(M) = \{a, ab, abb, ...\}$:

$$
\begin{aligned}
M=(\ & \mathcal{Q} = \{q_0, q_1, q_2\}, \\
& \Sigma = \{a, b\}, \\
& \Delta = \{(q_0, a, q_1), (q_0, b, q_2), ..., (q_2, b, q_2)\}, \\
& s = q_0, \\
& \mathcal{F} = \{q_1\}\ )
\end{aligned}
$$



# Simple relationship between a DFA and **production rules**



$$
\begin{aligned}
Q &= \{S, A, B, C, q_4\} \\
\Sigma &= \{b, a, !\} \\
q_0 &= S \\
F &= \{q_4\}
\end{aligned}
$$

$$
\begin{aligned}
S &\rightarrow bA \\
A &\rightarrow aB \\
B &\rightarrow aC \\
C &\rightarrow aC \\
C &\rightarrow !
\end{aligned}
$$

# **Regular grammars** generate regular languages

Given a DFA $M = (\mathcal{Q}, \Sigma, \Delta, s, \mathcal{F})$ the language, $\mathcal{L}(M)$, of strings accepted by $M$ can be generated by the regular grammar $G_{reg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where:

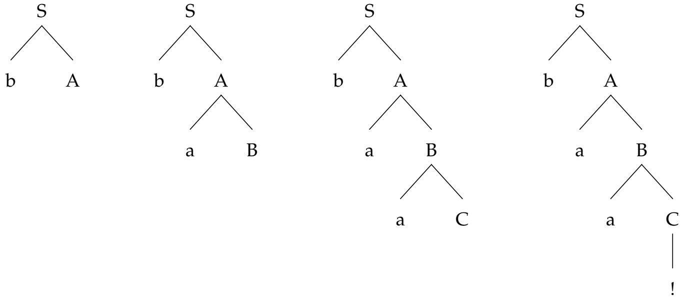
- $\mathcal{N} = \{\mathcal{Q}\}$ the non-terminals are the states of $M$
- $\Sigma = \Sigma$ the terminals are the set of transition symbols of $M$
- $S = s$ the starting symbol is the starting state of $M$
- $\mathcal{P} = q_i \to aq_j$ when $\delta(q_i, a) = q_j \in \Delta$
  or $q_i \to \epsilon$ when $q \in \mathcal{F}$ (i.e. when $q$ is an end state)

# Strings are **derived** from production rules

In order to derive a string from a grammar

- start with the designated starting symbol
- then non-terminal symbols are repeatedly expanded using the rewrite rules until there is nothing further left to expand.

The rewrite rules derive the members of a language from their internal structure (or **phrase structure**)



$S \rightarrow bA$ $A \rightarrow aB$ $B \rightarrow aC$ $C \rightarrow !$

# A regular language has a **left**- and **right-linear** grammar

For every regular grammar the rewrite rules of the grammar can all be expressed in the form:

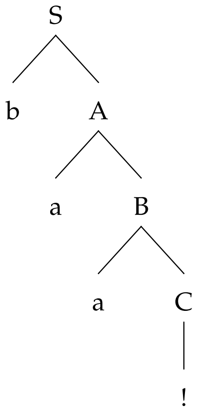
$$\begin{aligned} X &\rightarrow aY \\ X &\rightarrow a \end{aligned}$$

or alternatively, they can all be expressed as:

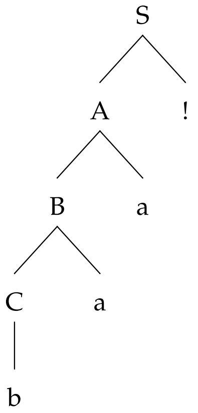
$$\begin{aligned} X &\rightarrow Ya \\ X &\rightarrow a \end{aligned}$$

The two grammars are **weakly-equivalent** since they generate the same strings.
But not **strongly-equivalent** because they do not generate the same structure to strings

# A regular language has a **left**- and **right-linear** grammar



$$\begin{aligned} S &\rightarrow bA \\ A &\rightarrow aB \\ B &\rightarrow aC \\ C &\rightarrow aC \\ C &\rightarrow ! \end{aligned}$$



$$\begin{aligned} S &\rightarrow A! \\ A &\rightarrow Ba \\ B &\rightarrow Ca \\ C &\rightarrow Ca \\ C &\rightarrow b \end{aligned}$$

# A regular grammar is a **phrase structure grammar**

A phrase structure grammar over an alphabet $\Sigma$ is defined by a tuple $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$. The language generated by grammar $G$ is $\mathcal{L}(G)$:

NON-TERMINALS $\mathcal{N}$: Non-terminal symbols (often uppercase letters) may be **rewritten** using the rules of the grammar.

TERMINALS $\Sigma$: Terminal symbols (often lowercase letters) are elements of $\Sigma$ and **cannot be rewritten**. Note $\mathcal{N} \cap \Sigma = \emptyset$.

START SYMBOL $S$: A **distinguished non-terminal symbol** $S \in \mathcal{N}$. This non-terminal provides the starting point for derivations.[3]

PHRASE STRUCTURE RULES $\mathcal{P}$: Phrase structure rules are pairs of the form $(w, v)$ usually written:
$w \rightarrow v$, where $w \in (\Sigma \cup \mathcal{N})^*\mathcal{N}(\Sigma \cup \mathcal{N})^*$ and $v \in (\Sigma \cup \mathcal{N})^*$

[3] $S$ is sometimes referred to as the axiom but note that, whereas in the inductively defined sets above the axioms denoted the smallest members of the set, here the axioms denote the existence of particular derivable structures.

# Definition of a phrase structure grammar **derivation**

Given $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ and $w, v \in (\mathcal{N} \cup \Sigma)^*$ a **derivation step** is possible to transform $w$ into $v$ if:

> $u_1, u_2 \in (\mathcal{N} \cup \Sigma)^*$ exist such that $w = u_1 \alpha u_2$, and $v = u_1 \beta u_2$ and $\alpha \to \beta \in \mathcal{P}$
>
> This is written $w \underset{G}{\Rightarrow} v$

A string in the language $\mathcal{L}(G)$ is a member of $\Sigma^*$ that can be derived in a **finite number of derivation steps** from the starting symbol $S$.

We use $\underset{G^*}{\Longrightarrow}$ to denote the reflexive, transitive closure of derivation steps, consequently $\mathcal{L}(G) = \{w \in \Sigma^* | S \underset{G^*}{\Longrightarrow} w\}$.

# PSGs may be grouped by production rule properties

Chomsky suggested that phrase structure grammars may be grouped together by the properties of their production rules.

| NAME | FORM OF RULES |
|---|---|
| regular | $(A \rightarrow Aa$ or $A \rightarrow aA)$ and $A \rightarrow a \mid A \in \mathcal{N}$and $a \in \Sigma$ |
| context-free | $A \rightarrow \alpha \mid A \in \mathcal{N}$ and $\alpha \in (\mathcal{N} \cup \Sigma)^*$ |
| context-sensitive | $\alpha A \beta \rightarrow \alpha \gamma \beta \mid A \in \mathcal{N}$ and $\alpha, \beta, \gamma \in (\mathcal{N} \cup \Sigma)^*$and $\gamma \neq \epsilon$ |
| recursively enum | $\alpha \rightarrow \beta \mid \alpha, \beta \in (\mathcal{N} \cup \Sigma)^*$ and $\alpha \neq \epsilon$ |

A **class** of languages (e.g. the class of regular languages) is all the languages that can be generated by a particular TYPE of grammar.

The term **power** is used to describe the **expressivity** of each type of grammar in the hierarchy (measured in terms of the number of subsets of $\Sigma^*$ that the type can generate)

# We can reason about properties of language classes

**All Chomsky languages classes are closed under union.**

$\mathcal{L}(G_1) \cup \mathcal{L}(G_2) = \mathcal{L}(G_3)$ where $G_1, G_2, G_3$ are all grammars of the same type

e.g. the union of a context-free language with another context-free language will yield a context-free language.

**All Chomsky language classes are closed under intersection with a regular language.**

$\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \mathcal{L}(G_3)$ where $G_1$ is a regular grammar and $G_2, G_3$ are grammars of the same type

e.g. the intersection of a regular language with a context-free language will yield another context-free language.

# We can define the **complexity** of language classes

The **complexity** of a language class is defined in terms of the **recognition problem**.

| Type | Language Class | Complexity |
|---|---|---|
| 3 | regular | $O(n)$ |
| 2 | context-free | $O(n^c)$ |
| 1 | context-sensitive | $O(c^n)$ |
| 0 | recursively enumerable | *undecidable* |

# Can regular grammars model natural language?

Why do we care about the answer to this question?

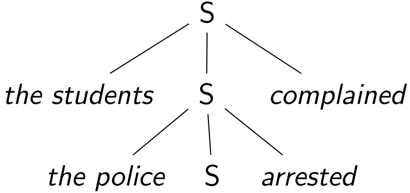
- We'd like fast algorithms for natural language processing applications.
- Potentially tells us something about human processing and acquisition (more in later lectures).

# Can regular grammars model natural language?

Centre Embedding

Infinitely recursive structures described by the rule, $A \rightarrow \alpha A \beta$, which generate language examples of the form, $a^n b^n$.

- *The students the police arrested complained*



- *The luggage that the passengers checked arrived*
- *The luggage that the passengers that the storm delayed checked arrived*

In general $/the\ a\ (that\ the\ a)^{n-1} b^n/$ where *nouns* are mapped to $a$ and *verbs* to $b$

# Reminder: use the **pumping lemma** to prove **not** regular

The **pumping lemma** for regular languages is used to prove that a language is **not** regular. The pumping lemma property is:

> All $w \in \mathcal{L}$ with $|w| \geq l$ can be expressed as a concatenation of three strings, $w = u_1 v u_2$, where $u_1, v$ and $u_2$ satisfy:

- $|v| \geq 1$ (i.e. $v \neq \epsilon$)
- $|u_1 v| \leq l$
- for all $n \geq 0$, $u_1 v^n u_2 \in \mathcal{L}$ (i.e. $u_1 u_2 \in \mathcal{L}$, $u_1 v u_2 \in \mathcal{L}$, $u_1 vv u_2 \in \mathcal{L}$, $u_1 vvv u_2 \in \mathcal{L}$, etc.)

# Reminder: use the **pumping lemma** to prove **not** regular

For each $l \geq 1$, find some $w \in \mathcal{L}$ of length $\geq l$ so that no matter how $w$ is split into three, $w = u_1 v u_2$, with $|u_1 v| \leq l$ and $|v| \geq 1$, there is some $n \geq 0$ for which $u_1 v^n u_2$ is not in $\mathcal{L}$.

To prove that $\mathcal{L} = \{a^n b^n | n \geq 0\}$ is not regular. For each $l \geq 1$, consider $w = a^l b^l \in \mathcal{L}$.

If $w = u_1 v u_2$ with $|u_1 v| \leq l$ & $|v| \geq 1$, then for some $r$ and $s$:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq l$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^l$

so $u_1 v^0 u_2 = a^r \epsilon a^{l-r-s} b^l = a^{l-s} b^l$

But $a^{l-s} b^l \notin \mathcal{L}$ so by the Pumping Lemma, $\mathcal{L}$ is not a regular language

# Complexity of sub-language is not complexity of language

Careful here though:

A regular grammar could generate constructions of the form $a^*b^*$ but not the more exclusive subset $a^nb^n$ which would represent centre embeddings.

More generally the complexity of a sub-language is not necessarily the complexity of a language.

If we show that the English subset $a^nb^n$ is not regular it does **not** follow that English itself is not regular.

# Can we prove English is not regular?

- If you intersect a regular language with another regular language you should get a third regular language. $\mathcal{L}_{reg1} \cap \mathcal{L}_{reg2} = \mathcal{L}_{reg3}$
- Also regular languages are closed under homomorphism (we can map all nouns to $a$ and all verbs to $b$)
- So if English is regular and we intersect it with another regular language (e.g. the one generated by $/the\ a\ (that\ the\ a)^* b^*/$) we should get another regular language.
  $if\ \mathcal{L}_{eng}\ then\ \mathcal{L}_{eng} \cap \mathcal{L}_{a*b*} = \mathcal{L}_{reg3}$
- However the intersection of an $a^* b^*$ with English is $a^n b^n$ ( in our example case specifically $/the\ a\ (that\ the\ a)^{n-1} b^n/$), which is not regular as it fails the pumping lemma property.
  $but\ \mathcal{L}_{eng} \cap \mathcal{L}_{a*b*} = \mathcal{L}_{a^n b^n}\ (which\ is\ not\ regular)$
- The assumption that English is regular must be incorrect.

# Problems using regular grammars for natural language

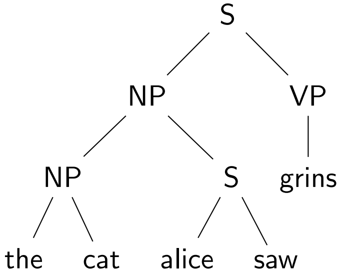
But for finite $n$ we can still model English using a DFA—we can design the states to capture finite levels of embedding.

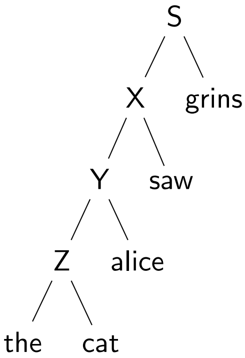
So are there any other reasons not to just use a regular grammar?

Redundancy Grammars written using finite state techniques alone are highly redundant: Regular grammars very difficult to build and maintain.

Useful internal structures The left-linear or right-linear internal structures derived by regular grammars are generally not very useful for higher level NLP applications. We need informative internal structure so that we can, for example, build up good semantic representations.

# Problems using regular grammars for natural language


S
NP
VP
NP
S
grins
the
cat
alice
saw


S
X
grins
Y
saw
Z
alice
the
cat