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, Σ, 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 Σ is an ordered n-tuple of elements of Σ.

Σ^* denotes the set of all strings over Σ of finite length.

- If $\Sigma = \{a, b\}$ then ϵ, ba, bab, aab are examples of strings over Σ.
- 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 Σ any subset of Σ^* is a **formal language** over alphabet Σ.

[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 Σ 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 Σ^*:

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

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

DERIVATIONS

Given a set of axioms and rules for inductively defining a subset, $\mathcal{L}$, of Σ^*, 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:

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

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

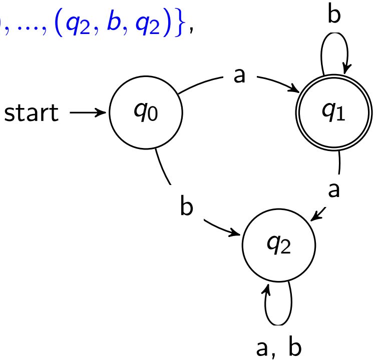
Recall that a language is regular if it is equal to the set of strings accepted by some deterministic finite-state automaton (DFA).
A DFA is defined as $M = (\mathcal{Q}, \Sigma, \Delta, s, \mathcal{F})$ where:

- $\mathcal{Q} = \{q_0, q_1, q_2 ...\}$ is a finite set of states.
- Σ 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 δ. 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

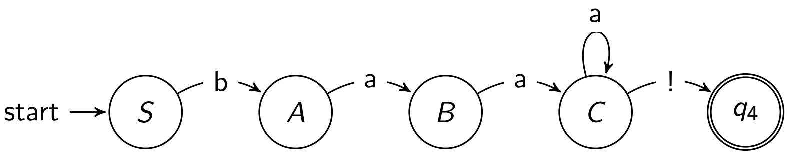
Reminder: **regular languages** are accepted by **DFAs**

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

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



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



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

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

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:

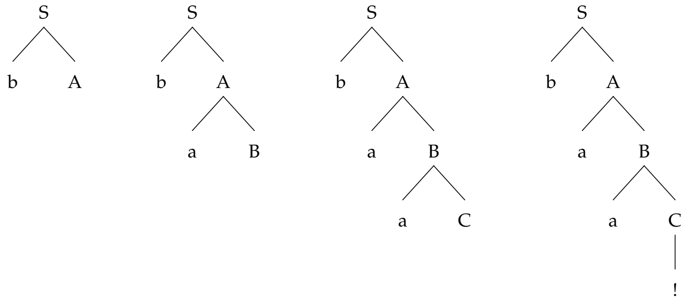
- $\mathcal{N} = \{\mathcal{Q}\}$ the non-terminals are the states of M
- $\Sigma = \Sigma$ the terminals are the set of transition symbols of M
- $S = s$ the starting symbol is the starting state of M
- $\mathcal{P} = q_i \rightarrow aq_j$ when $\delta(q_i, a) = q_j \in \Delta$
  or $q_i \rightarrow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

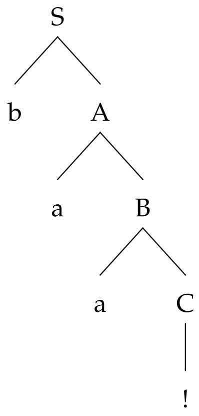
$$\begin{aligned} X &\rightarrow aY \\ X &\rightarrow a \end{aligned}$$

or alternatively, they can all be expressed as:

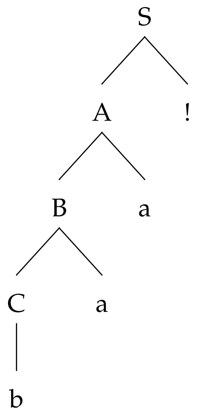
$$\begin{aligned} X &\rightarrow Ya \\ X &\rightarrow a \end{aligned}$$

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



S	→	bA
A	→	aB
B	→	aC
C	→	aC
C	→	!



S	→	A!
A	→	Ba
B	→	Ca
C	→	Ca
C	→	b

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

A phrase structure grammar over an alphabet Σ 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 Σ: Terminal symbols (often lowercase letters) are elements of Σ 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 $\alpha \rightarrow \beta \in \mathcal{P}$
>
> This is written $w \underset{G}{\Rightarrow} v$

A string in the language $\mathcal{L}(G)$ is a member of Σ^* 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 Σ^* 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?

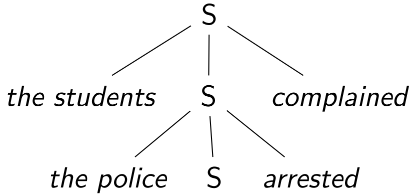
- We'd like fast algorithms for natural language processing applications.
- Potentially tells us something about human processing and acquisition (more in later lectures).

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 $/\textit{the } a \, (\textit{that the } a)^{n-1} b^n/$ 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

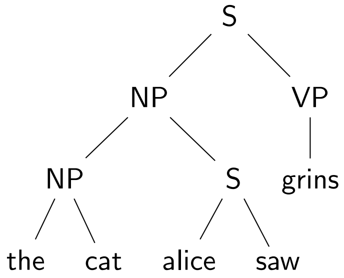
But for finite n we can still model English using a DFA—we can design the states to capture finite levels of embedding.

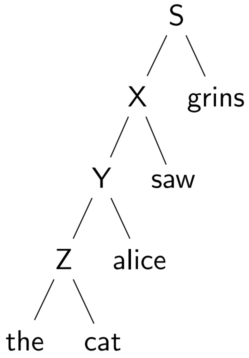
So are there any other reasons not to just use a regular grammar?

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



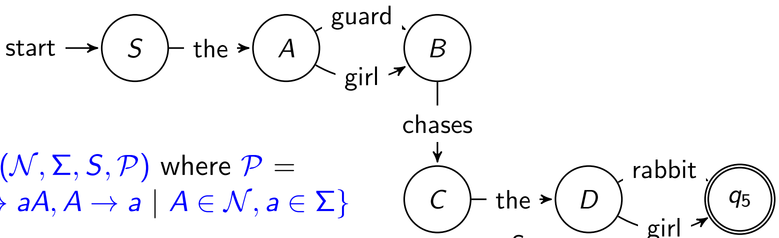


Formal Models of Language

Paula Buttery

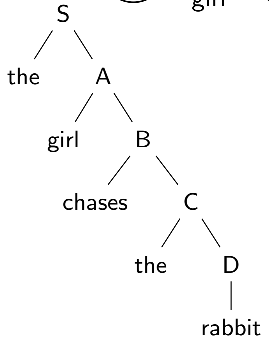
Dept of Computer Science & Technology, University of Cambridge

Regular grammars give us **linear** trees

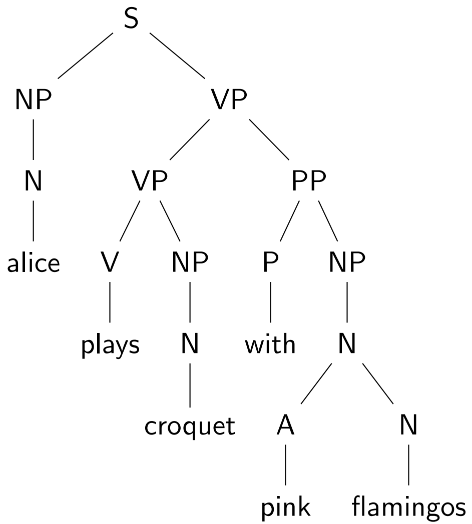


$G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where $\mathcal{P} = \{A \rightarrow aA, A \rightarrow a \mid A \in \mathcal{N}, a \in \Sigma\}$

- $\mathcal{N} = \{S, A, B, C, D, q_5\}$
- $\Sigma = \{the, girl, guard, ...\}$
- $\mathcal{S} = S$
- $\mathcal{P} = \{S \rightarrow the\ \ A,$
  $A \rightarrow guard\ \ B \mid girl\ \ B,$
  $B \rightarrow chases\ \ C,$
  $C \rightarrow the\ \ D,$
  $D \rightarrow girl \mid rabbit\}$



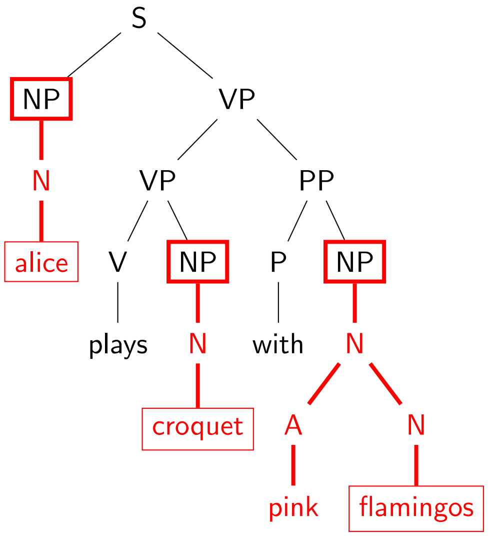
Context-free grammars capture **phrase structure**



$G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where
$\mathcal{P} = \{A \rightarrow \alpha \mid$
$A \in \mathcal{N}, \alpha \in (\mathcal{N} \cup \Sigma)^*\}$

A brief excursion into linguistic terminology...

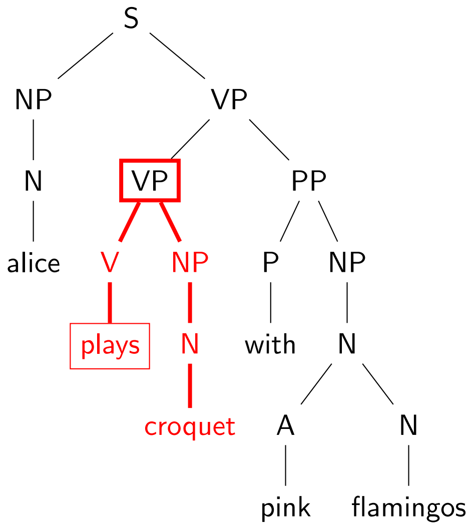
Context-free grammars capture **phrase structure**



When modelling natural language, linguists label the non-terminal symbols with names that encode the most *influential* word in the phrase. They call this influential word the **head**.

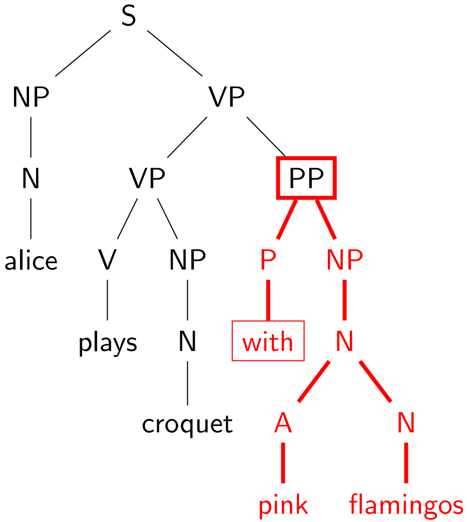
- noun phrases, *NP*, have a head noun

Context-free grammars capture **phrase structure**



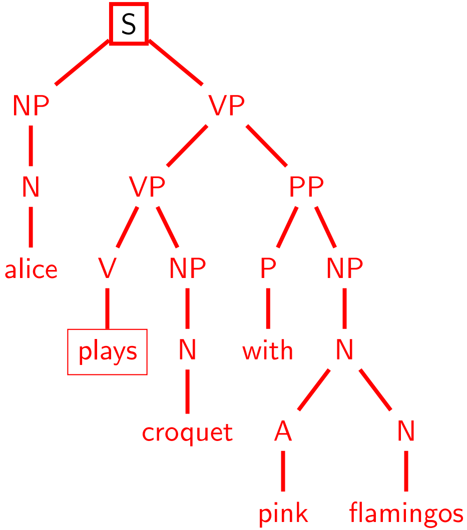
- verb phrases, *VP*, have a head verb

Context-free grammars capture **phrase structure**



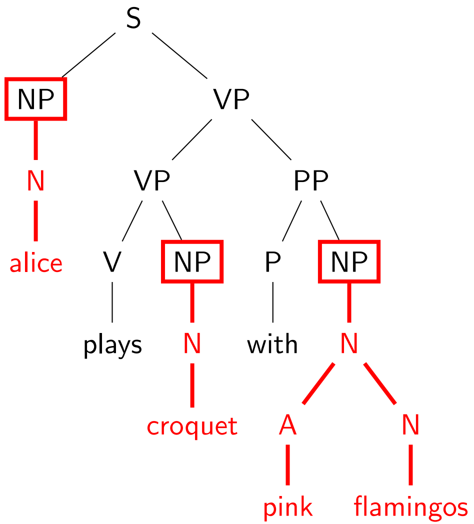
- prepositional phrases, *PP*, have a head preposition

Context-free grammars capture **phrase structure**



- the head of the whole string, S, is always the main verb

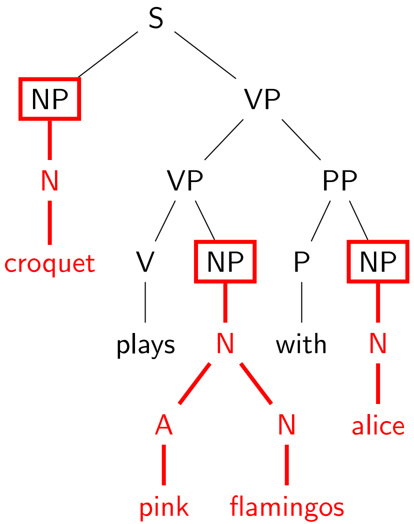
Context-free grammars capture **phrase structure**



Trees below nodes of the same type are interchangeable to yield another string in the language:

- $NP \rightarrow N$
- $N \rightarrow A\ N$
- $N \rightarrow alice | croquet | \ldots$

Context-free grammars capture **phrase structure**



Trees below nodes of the same type are interchangeable to yield another string in the language:

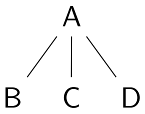
- $NP \rightarrow N$
- $N \rightarrow A\ N$
- $N \rightarrow alice | croquet | \ldots$

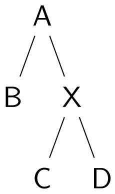
CFGs are often written in **Chomsky Normal Form**

Chomsky normal form: every production rule has the form, $A \rightarrow BC$, or, $A \rightarrow a$ where $A, B, C \in \mathcal{N}$, and, $a \in \Sigma$.

Conversion to Chomsky Normal Form

For every CFG there is a weakly equivalent CNF alternative.
$A \rightarrow BCD$ may be rewritten as the two rules, $A \rightarrow BX$, and, $X \rightarrow CD$.





CNF is a requirement for some parsing algorithms.

Context-free languages are accepted by **push down automata**

A PDA is defined as $M = (\mathcal{Q}, \Sigma, \Gamma, \Delta, s, \perp, \mathcal{F})$ where:

- $\mathcal{Q} = \{q_0, q_1, q_2 ...\}$ is a finite set of states.
- Σ is the input alphabet.
- Γ is the stack alphabet.
- $\Delta \subseteq (\mathcal{Q} \times (\Sigma \cup \epsilon) \times \Gamma) \times (\mathcal{Q} \times \Gamma^*)$ is a relation $(\mathcal{Q} \times (\Sigma \cup \epsilon) \times \Gamma) \rightarrow (\mathcal{Q} \times \Gamma^*)$ which we write as δ. Given $q \in \mathcal{Q}$, $i \in \Sigma$ and $A \in \Gamma$ then $\delta(q, i, A)$ returns (q', α), that is, a new state $q' \in \mathcal{Q}$ and replaces A at the top of the stack with $\alpha \in \Gamma^*$
- s is the starting state
- $\perp$ is the initial stack symbol
- $\mathcal{F}$ is the set of all end states

Moving from one state to the next we may **push** or **pop**

- in state q_x on encountering transition symbol a transition to state q_y popping A from the top of the stack and pushing B onto the stack

$$\begin{tikzcd} q_x \arrow[r, bend left, "a : A/B"] & q_y \end{tikzcd}$$

BEFORE	AFTER
A	B
z_0	z_0

- in state q_x transition to state q_y pushing A onto the stack

$$\begin{tikzcd} q_x \arrow[r, bend left, "\epsilon : \epsilon/A"] & q_y \end{tikzcd}$$

BEFORE	AFTER
z_0	A
	z_0

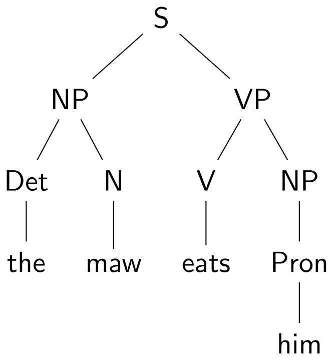
- in state q_x transition to state q_y popping A from the stack

$$\begin{tikzcd} q_x \arrow[r, bend left, "\epsilon : A/\epsilon"] & q_y \end{tikzcd}$$

BEFORE	AFTER
A	z_0
z_0	

A toy context-free grammar

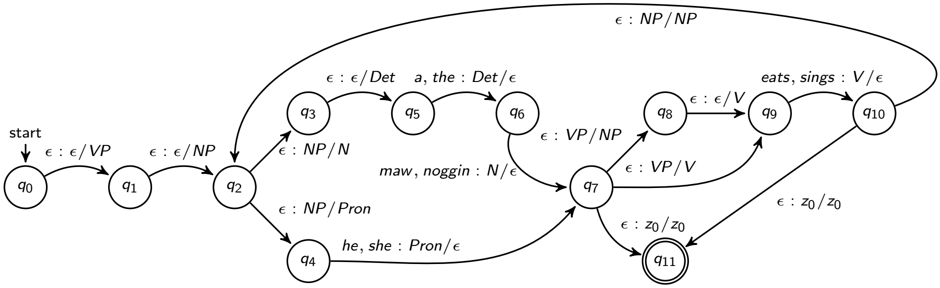
$$
\begin{array}{rcl}
\text{S} & \rightarrow & \text{NP VP} \\
\text{NP} & \rightarrow & \text{Pron} \\
\text{NP} & \rightarrow & \text{Det N} \\
\text{VP} & \rightarrow & \text{V} \\
\text{VP} & \rightarrow & \text{V NP} \\
\text{Det} & \rightarrow & \{\text{a, the}\} \\
\text{N} & \rightarrow & \{\text{maw, noggin, ...}\} \\
\text{Pron} & \rightarrow & \{\text{he, she, him, her}\} \\
\text{V} & \rightarrow & \{\text{eats, sings}\}
\end{array}
$$



Recognising a string with a push down automaton

S	→	NP VP
NP	→	Pron
NP	→	Det N
VP	→	V
VP	→	V NP

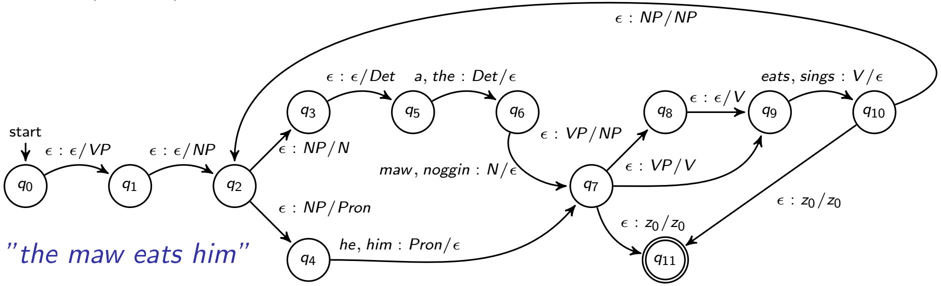
Det	→	{a,the}
N	→	{maw, noggin, ...}
Pron	→	{he, him, her}
V	→	{eats, sings}



Is *'the maw eats him'* a string in the language?

the	q_0	z_0			
the	q_0-q_1	VP	z_0		
the	q_1-q_2	NP	VP	z_0	
the	q_2-q_3	N	VP	z_0	
the	q_3-q_5	Det	N	VP	z_0
maw	q_5-q_6	N	VP	z_0	
eats	q_6-q_7	VP	z_0		
eats	q_7-q_8	NP	z_0		
eats	q_8-q_9	V	NP	z_0	
him	q_9-q_{10}	NP	z_0		
him	q_{10}-q_2	NP	z_0		
him	q_2-q_4	Pron	z_0		
him	q_4-q_7	z_0			
ϵ	q_7-q_{11}	z_0			

S	→	NP VP
NP	→	Pron
NP	→	Det N
VP	→	V
VP	→	V NP
Det	→	{a,the}
N	→	{maw, noggin, ...}
Pron	→	{he, him, her}
V	→	{eats, sings}

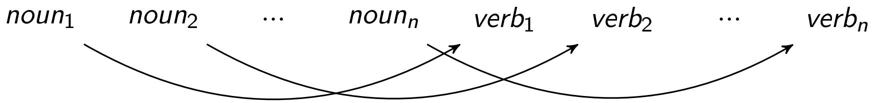


"the maw eats him"

Can context-free grammars model natural language?

CROSS SERIAL DEPENDENCIES

A small number of languages exhibit strings of the form



Zurich dialect of Swiss German

mer d'chind em Hans es huus haend wele laa hälfe aastriiche.

we the children Hans the house have wanted to let help paint.

we have wanted to let the children help Hans paint the house

Such expressions, i.e. of the form $/a^n b^m c^n d^m/$, may not be derivable by a context-free grammar.

mer d'chind^n em Hans^m es huus haend wele laa^n hälfe^m aastriiche.

$\rightarrow /wa^n b^m x c^n d^m y/$

Use the **pumping lemma** to prove **not** context-free

The pumping lemma for context-free languages (CFLs) is used to show that a language is not context-free. The pumping lemma property for CFLs is:

All $w \in \mathcal{L}$ with $|w| \geq k$ can be expressed as a concatenation of five strings, $w = u_1 y u_2 z u_3$, where u_1, y, u_2, z and u_2 satisfy:

- $|yz| \geq 1$ (i.e. we cannot have $y = \epsilon$ and $z = \epsilon$)
- $|yu_2z| \leq k$
- for all $n \geq 0$, $u_1 y^n u_2 z^n u_3 \in \mathcal{L}$
  (i.e. $u_1 u_2 u_3 \in \mathcal{L}$, $u_1 y u_2 z u_3 \in \mathcal{L}$, $u_1 yy u_2 zz u_3 \in \mathcal{L}$ etc.)

To prove that Swiss German is not context-free, similar proof as for **centre embeddings** (last lecture). Except that you need to remember that:
$\mathcal{L}_{reg1} \cap \mathcal{L}_{cfg1} = \mathcal{L}_{cfg2}$

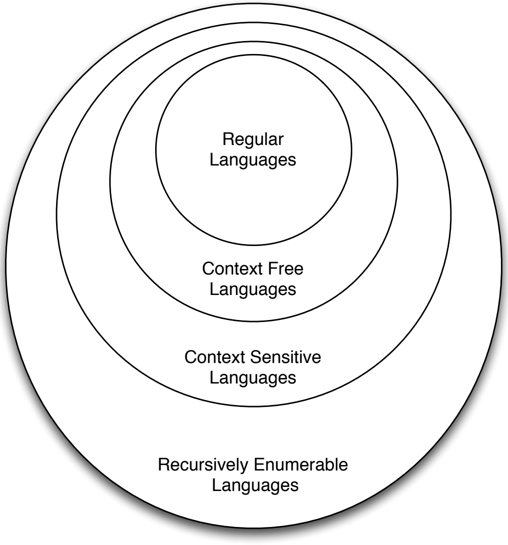
Are **CSGs** required to model natural languages?

Remember the **complexity** of a language class was defined in terms of the **recognition problem**.

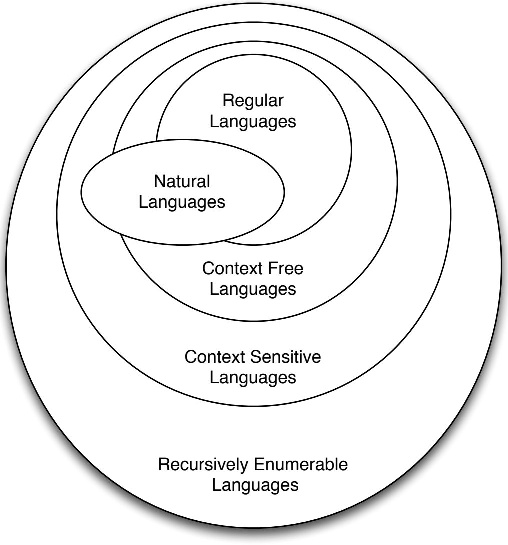
Type	Language Class	Complexity	machine
3	regular	$O(n)$	DFA
2	context-free	$O(n^c)$	PDA
1	context-sensitive	$O(c^n)$	LBA
0	recursively enumerable	*undecidable*	Turing

- Modelling natural languages using context-sensitive grammars is very expensive. In practice we don't have to because only very limited constructions are not captured by context-free grammars.
- However, it is still fun to place a limit on the complexity of natural languages — we are not limited to discussing language classes only in terms of the Chomsky hierarchy.

We are not limited to the **Chomsky hierarchy**



We are not limited to the **Chomsky hierarchy**



The mildly context-sensitive grammars

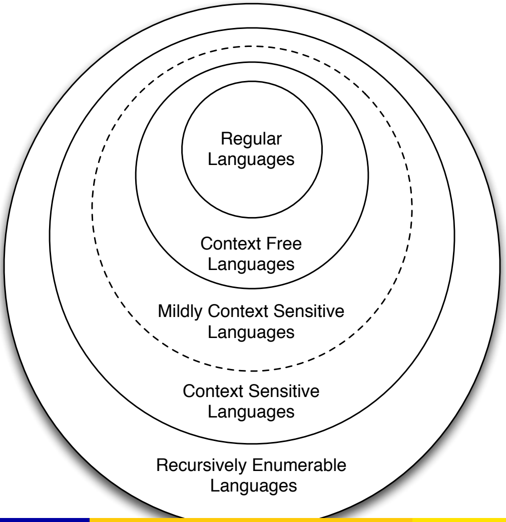
Joshi defined a class of languages that is more expressive than context-free languages, less expressive than context-sensitive languages and also sits neatly in the Chomsky hierarchy.

MILDLY CONTEXT-SENSITIVE languages

An abstract language class has the following properties:

- it includes all the context-free languages;
- members of the languages in the class may be recognised in polynomial time;
- the languages in the class account for all the constructions in natural language that context-free languages fail to account for (such as cross-serial dependencies).

Mildly CSGs are a **subset** of CSGs that account for natural language


Regular Languages
Context Free Languages
Mildly Context Sensitive Languages
Context Sensitive Languages
Recursively Enumerable Languages

In **Tree Adjoining Grammars** trees are rewritten as trees.

In phrase structure grammar symbols were rewritten with other symbols

In **Tree Adjoining Grammars** trees are rewritten as other trees.

The grammar consists of sets of two types of elementary tree:

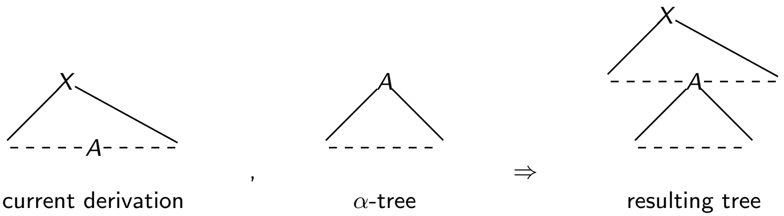
- **initial trees** or α trees
- **auxiliary trees** or β trees

A derivation is the result of recursive composition of elementary trees via one of two operations:

- **substitution**
- **adjunction**.

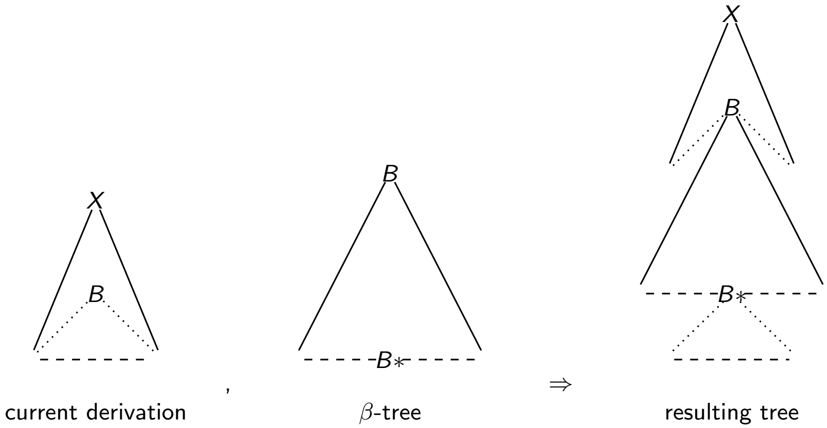
Tree adjoining grammars: the **substitution** operation

- SUBSTITUTION: a substitution may occur when a non-terminal leaf (that is, some $A \in \mathcal{N}$) of the current derivation tree is replaced by an α-tree that has A at its root.



Tree adjoining grammars: the **adjunction** operation

- ADJUNCTION:an adjunction may occur when an internal non-terminal node of the current derivation (some $B \in \mathcal{N}$) tree is replaced by a β tree that has a B at its root and *foot*.

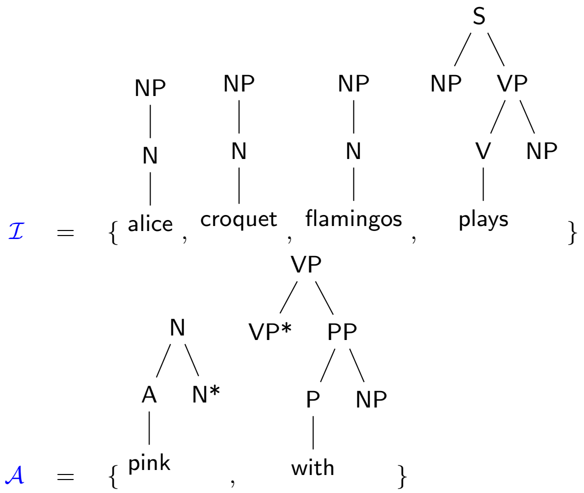


Tree adjoining grammars: definition

- $\mathcal{N}$ is the set of non-terminals
- Σ is the set of terminals
- S is a distinguished non-terminal $S \in \mathcal{N}$ that will be the root of complete derivations
- $\mathcal{I}$ is a set of **initial trees** (also known as α trees). Internal nodes of an α tree are drawn from $\mathcal{N}$ and the leaf nodes from $\Sigma \cup \mathcal{N} \cup \epsilon$.
- $\mathcal{A}$ is a set of **auxiliary trees** (also know as β trees). Internal nodes of an β-tree are drawn from $\mathcal{N}$ and the leaf nodes from $\Sigma \cup \mathcal{N} \cup \epsilon$. One leaf of a β-tree is distinguished as the **foot** and will be the same non-terminal as at its root (the foot is often indicated with an asterisk).

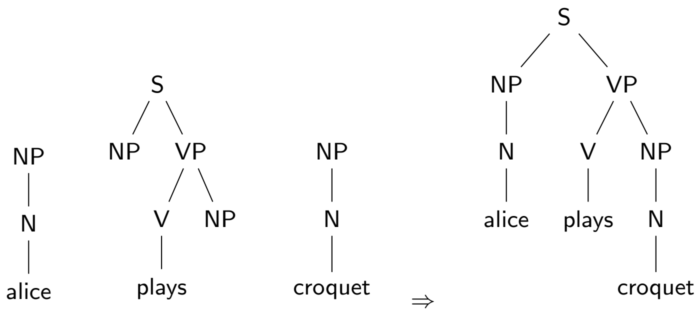
Tree adjoining grammars: natural language example

$G_{tag} = (\mathcal{N}, \Sigma, S, \mathcal{I}, \mathcal{A})$ where:



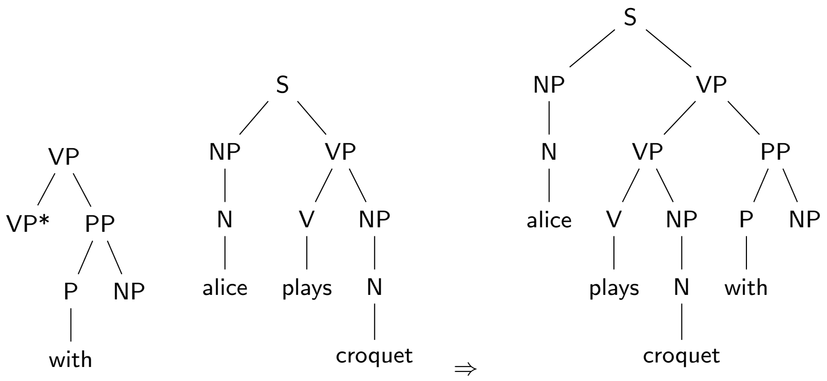
Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



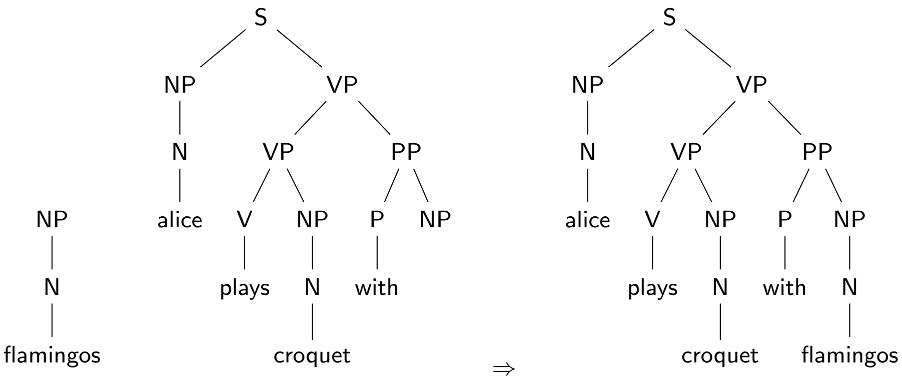
Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



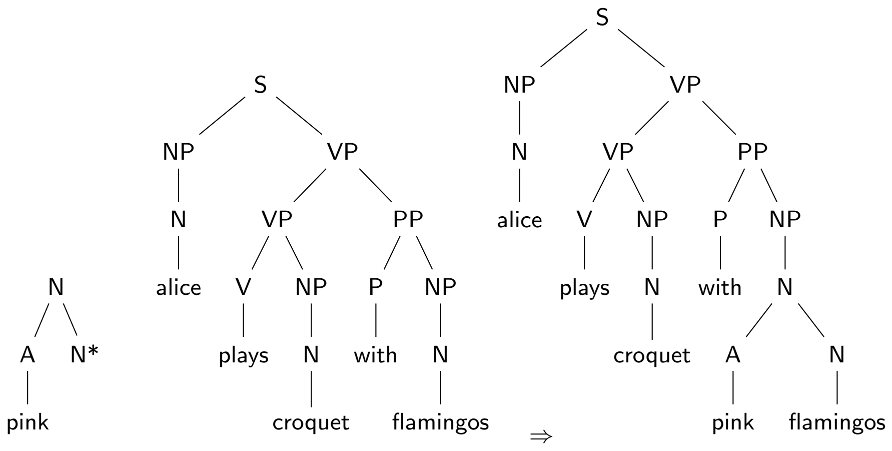
Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Shift-reduce parsers are useful for **deterministic** languages

LR(k) Shift-reduce parsers are most useful for recognising the strings of **deterministic** languages (languages where no string has more than one analysis) which have been described by an unambiguous grammar.

Quick reminder:

- The parsing algorithm has two actions: SHIFT and REDUCE
- Initially the input string is held in the buffer and the stack is empty.
- Symbols are **shifted** from the buffer to the stack
- When the top items of the stack match the RHS of a rule in the grammar then they are **reduced**, that is, they are replaced with the LHS of that rule.
- k refers to the look-ahead.

Reminder: shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string $abcd$ generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$\begin{aligned} \Sigma &= \{a, b, c\} \\ \mathcal{N} &= \{S, A, B, C, D\} \\ s &= \mathsf{S} \\ \mathcal{P} &= \{S \to A\ B, \\ &\quad A \to a, \\ &\quad B \to b\ C, \\ &\quad C \to c\ D, \\ &\quad D \to d\} \end{aligned}$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE
AbC		REDUCE
AB		REDUCE
S		

Reminder: properties of **Deterministic** CFLs

Deterministic context-free languages:

- are a proper subset of the context-free languages
- are accepted by deterministic push-down automata
- can be modelled by an unambiguous grammar
- can be parsed in linear time
- parser can be automatically generated from the grammar

CFGs used to model natural language are **not** deterministic

- Natural languages (with all their inherent ambiguity) are not well suited to shift-reduce parsers which operate deterministically recognising a single derivation without backtracking
- However, natural language parsing can be achieved deterministically by selecting parsing actions using a machine learning classifier (more on this next time).
- All CFGs (including those exhibiting ambiguity) can be recognised in polynomial time using chart parsing algorithms.

Ambiguous grammars derive a **parse forest**

Number of binary trees is proportional to the Catalan number

$$\texttt{Num of trees for sentence length n} = \prod_{k=2}^{n-1} \frac{(n-1)+k}{k}$$

sentence length	number of trees	sentence length	number of trees
3	2	8	429
4	5	9	1430
5	14	10	4862
6	42	11	16796
7	132	12	58786

We need parsing algorithms that can efficiently store the parse forest and not derive shared parts of tree more than once—chart parsers

The **Earley parser** is a **chart parsing** algorithm

The **Earley parser** is a dynamic programming algorithm that records partial derivations in a CHART (a table).

- Uses a top-down approach to explore the whole search space, recovering multiple derivations where they exist.
- The progress of the algorithm is encoded in something called a **dotted rule** or **progress rule**:

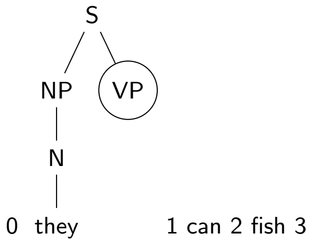
  $A \rightarrow {\bullet}\alpha\beta \mid \alpha{\bullet}\beta \mid \alpha\beta{\bullet}$ where $A \rightarrow \alpha\beta \in \mathcal{P}$.
- Rules of the form $A \rightarrow {\bullet}\alpha\beta$ have all symbols still to be explored;
- Rules of the form $A \rightarrow \alpha\beta{\bullet}$ have been completely *used up* deriving a portion of the string.

Partial derivations are recorded in a **chart**

- By convention, each row of the chart is referred to as an **edge**.
- An edge in the chart records a dotted rule, and its **span**.
- The span refers to the portion of the input string which is consistent with the partial tree.
- If we wish to discover the structure of a parse, an edge must also record the derivation **history** of the immediately previous partial tree(s) that made the current partial tree possible.

Partial derivations are recorded in a **chart**

For an illustration, consider the partial tree below which has been derived when attempting to parse the sentence *they can fish*:



ID	RULE	[start, end]	HIST
⋮			
e_i	$S \rightarrow NP_{\bullet}VP$	$[0, 1]$	h_k

Partial derivations are recorded in a **chart**

For input string $u = a_1 ... a_n$ and grammar $G_{cfg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$:

An edge $A \rightarrow \alpha_{\bullet}\beta\ [i, j]$ is added if

$S \underset{G^*}{\Longrightarrow} a_1 ... a_i A\gamma$ where γ are symbols in u yet to be parsed

and $\alpha \underset{G^*}{\Longrightarrow} a_{i+1} ... a_j$

- The chart is **initialised** with the edge $S \rightarrow {}_{\bullet}\alpha\beta\ [0, 0]$;
- The input string $u = a_1 ... a_n$ is **recognised** when we add the edge $S \rightarrow \alpha\beta_{\bullet}\ [0, n]$.

Today's toy grammar

We will parse the sentence *they can fish* using $G_{cfg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where:

$$\begin{array}{rcl}
\mathcal{N} & = & \{S, NP, VP, PP, N, V, P\} \\
\Sigma & = & \{can, fish, in, rivers, they \ldots\} \\
S & = & S \\
\mathcal{P} & = & \{S \rightarrow NP\ VP \\
 & & NP \rightarrow N\ PP \mid N \\
 & & PP \rightarrow P\ NP \\
 & & VP \rightarrow VP\ PP \mid V\ VP \mid V\ NP \mid V \\
 & & N \rightarrow can \mid fish \mid rivers \mid \ldots \\
 & & P \rightarrow in \mid \ldots \\
 & & V \rightarrow can \mid fish \mid \ldots\ \}
\end{array}$$

0 they 1 can 2 fish 3

Initialise the chart

The chart is initialised with $S \rightarrow {}_\bullet \alpha\beta \; [0,0]$.

ID	RULE	[start, end]	HIST
e_0	$S \rightarrow {}_\bullet \; NP \;\; VP$	$[0,0]$	

In rule induction notation:

$$\frac{}{S \rightarrow {}_\bullet \alpha\beta \; [0,0]} \; \text{(induction step)}$$

Three steps of the Earley parser: **predict** step

This step adds new edges to the chart and can be thought of as expanding tree nodes in the top-down derivation.

ID	RULE	[start, end]	HIST
e_0	$S \rightarrow {\bullet}\ NP\ VP$	$[0,0]$	
e_1	$NP \rightarrow {\bullet}\ N$	$[0,0]$	
e_2	$NP \rightarrow {\bullet}\ N\ PP$	$[0,0]$	

In rule induction notation:

$$\frac{A \rightarrow \alpha_{\bullet} B\beta\ [i,j]}{B \rightarrow {}_{\bullet}\gamma\ [j,j]} \text{ (predict step) where } B \rightarrow \gamma \in \mathcal{P}$$

Three steps of the Earley parser: **scan** step

This step allows us to check if we have a node that is consistent with the input sentence. If the input sentence is $u = a_1 \ldots a_n$ we can add a new edge if $A \rightarrow {\bullet} a\ [i, j-1]$ and $a = aj$.

ID	RULE	[start, end]	HIST
e_0	$S \rightarrow \bullet\ NP\ VP$	$[0,0]$	
e_1	$NP \rightarrow \bullet\ N$	$[0,0]$	
e_2	$NP \rightarrow \bullet\ N\ PP$	$[0,0]$	
e_3	$N \rightarrow they\ \bullet$	$[0,1]$	

In rule induction notation:

$$\frac{A \rightarrow {\bullet} a\ [i, j-1]}{A \rightarrow a {\bullet}\ [i, j]} \text{ (scan step) when } a = a_j$$

Three steps of the Earley parser: **scan** step

- For natural language sentence parsing tasks, Σ can be the finite set of words in the language (a very large set).
- when carrying out the predict step from a rule like $NP \rightarrow \bullet \mathcal{N}$ we would end up adding a new edge for every *noun* in the language.
- To save us from creating all these edges we can privilege a set of the non-terminals and perform a forward look-up of the next a_j to see whether it will be consistent.
- In our example this set would be $\mathcal{N}_{PofS} = \{N, V, P\}$, that is, all the non-terminal symbols that represent the **parts-of-speech** of the language (such as *nouns*, *verbs*, *adjectives*...).
- During the scanning step, we find edges containing non-terminals in $\mathcal{N}_{PofS}$ with a dot on their LHS and check if the upcoming word is consistent with the part-of-speech. Iff it is consistent then we add an edge to the chart.

Three steps of the Earley parser: **complete** step

This step propagates fully explored tree nodes in the chart.

ID	RULE	[start, end]	HIST
e_0	$S \rightarrow \bullet\ NP\ VP$	$[0,0]$	
e_1	$NP \rightarrow \bullet\ N$	$[0,0]$	
e_2	$NP \rightarrow \bullet\ N\ PP$	$[0,0]$	
e_3	$N \rightarrow they\ \bullet$	$[0,1]$	
e_4	$NP \rightarrow N\ \bullet$	$[0,1]$	e_3
e_5	$NP \rightarrow N\ \bullet\ PP$	$[0,1]$	e_3
e_6	$S \rightarrow NP\ \bullet\ VP$	$[0,1]$	e_4

In rule induction notation:

$$\frac{A \rightarrow \alpha_\bullet B\beta\ [i,k] \qquad B \rightarrow \gamma_\bullet\ [k,j]}{A \rightarrow \alpha B_\bullet \beta\ [i,j]}\ \text{(complete step)}$$

ID	RULE	[start, end]	HIST	word n
e_0	$S \rightarrow \bullet\ NP\ VP$	$[0,0]$		**word 0**
e_1	$NP \rightarrow \bullet\ N$	$[0,0]$		**word 1**
e_2	$NP \rightarrow \bullet\ N\ PP$	$[0,0]$		
e_3	$N \rightarrow they\ \bullet$	$[0,1]$		
e_4	$NP \rightarrow N\ \bullet$	$[0,1]$	(e_3)	
e_5	$NP \rightarrow N\ \bullet\ PP$	$[0,1]$	(e_3)	
e_6	$S \rightarrow NP\ \bullet\ VP$	$[0,1]$	(e_4)	
e_7	$PP \rightarrow \bullet\ P\ NP$	$[1,1]$		**word 2**
e_8	$VP \rightarrow \bullet\ V$	$[1,1]$		
e_9	$VP \rightarrow \bullet\ V\ NP$	$[1,1]$		
e_{10}	$VP \rightarrow \bullet\ V\ VP$	$[1,1]$		
e_{11}	$VP \rightarrow \bullet\ VP\ PP$	$[1,1]$		
e_{12}	$V \rightarrow can\ \bullet$	$[1,2]$		
e_{13}	$VP \rightarrow V\ \bullet$	$[1,2]$	(e_{12})	
e_{14}	$VP \rightarrow V\ \bullet\ NP$	$[1,2]$	(e_{12})	
e_{15}	$VP \rightarrow V\ \bullet\ VP$	$[1,2]$	(e_{12})	
e_{16}	$S \rightarrow NP\ VP\ \bullet$	$[0,2]$	(e_4, e_{13})	
e_{17}	$VP \rightarrow VP\ \bullet\ PP$	$[1,2]$	(e_{13})	
e_{18}	$NP \rightarrow \bullet\ N$	$[2,2]$		**word 3**
e_{19}	$NP \rightarrow \bullet\ N\ PP$	$[2,2]$		
e_{20}	$VP \rightarrow \bullet\ V$	$[2,2]$		
e_{21}	$VP \rightarrow \bullet\ V\ NP$	$[2,2]$		

The run time of the Earley parser is **polynominal**

- The **complete** step dominates run time $O(n^2)$
- Running time of the Earley parser is $O(n^3)$
- Run time is reduced in various scenarios, e.g. when the grammar is unambiguous or left-recursive .[1]

So what makes a sentence **complex** for a human to process?

[1] See `https://homepages.cwi.nl/~jve/lm2005/earley.pdf` for a full discussion

The term **complexity** can be used to describe human processing difficulty

The term **complexity** is also used to describe the perceived human processing difficulty of a sentence: work in this area is generally referred to as **computational psycholinguistics**.

Complexity within this domain can refer to:

- the **time and space requirements** of the algorithm that your brain is posited to require while processing a sentence.
- the **information theoretic content** of the sentence itself in isolation from the human processor (more in later lectures on this on)

The term **complexity** can be used to describe human processing difficulty

Traditional work in this area has looked mainly at parsing algorithms to discover whether they exhibit properties that correlate with measurable predictors of complexity in human linguistic behaviour.

Two general assumptions are made in this work:

1) Sentences will take **longer to process** if they are more complicated for the human parser.
    - Processing time is usually measured as the time it takes to read a sentence.
    - This is can be done with eye-tracking machines which also identify whether the subject reread any parts of a sentence.
    - Researchers also use neuro-imaging techniques (MEG, fMRI)

The term **complexity** can be used to describe human processing difficulty

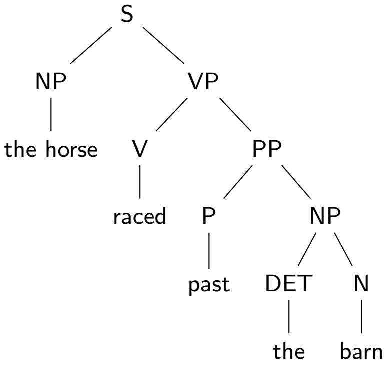
2) Sentences **will not occur frequently in the spoken language** if they are complicated to produce or comprehend.
   - Frequencies are calculated by counting constructions of interest in spoken language corpora.

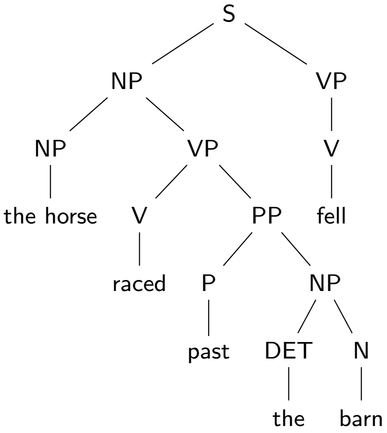
The assumption then is that one (or both) of the two measurements of perceived complexity above will correlate with time and space requirements of the parsing algorithm.

What makes a sentence expensive to process?

Example: long distance syntactic dependencies (e.g. garden-paths)

- The horse raced past the barn
- The horse raced past the barn fell—comparatively slow reading time





Hale — Earley parser as a model of sentence processing

Using predictability as a measure of difficulty

- The cognitive effort associated with a word in a sentence can be measured by the word's **surprisal** (negative log conditional probability): $\log \frac{1}{P(w_i|w_{1\cdots i-1})}$ (more on this in later lectures)
- The suggestion is that probabilistic context-free grammars (PCFGs) can be used to model human language processing.

  $G_{pcfg} = (\Sigma, \mathcal{N}, S, \mathcal{P}, q)$ where q is a mapping from rules in $\mathcal{P}$ to a probability and $\sum_{A\rightarrow\alpha\,\in\mathcal{P}} q(A \rightarrow \alpha) = 1$
- A probabilistic Earley parser is used as a model of online eager sentence processing.

Hale — Earley parser as a model of sentence processing

- The probabilistic Earley parser computes all parses of its input.
- As a psycholinguistic theory it is one of **total parallelism** (as opposed to a reanalysis theory)
- Calculate **prefix probabilities** i.e. probabilities of partially derived trees.
- Hypothesis is that the cognitive effort expended to parse a given prefix is **proportional to the total probability** of all the structural analyses which are not compatible with the prefix.
- Generates predictions about word-by-word reading times by comparing the total effort expended before some word to the total effort after.
- The explanation for garden-pathing is then **the reduction in the probability** of the new tree set compared with the previous tree set.
- The model accounts successfully for reading times.

Hale — Earley parser as a model of sentence processing

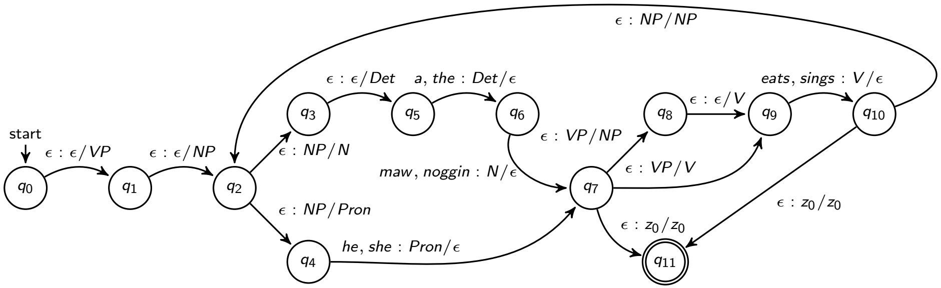
Toy grammar with probabilities

S	$\rightarrow$	NP VP	1
NP	$\rightarrow$	N PP	0.2
NP	$\rightarrow$	N	0.8
PP	$\rightarrow$	P NP	1
VP	$\rightarrow$	VP PP	0.1
VP	$\rightarrow$	V VP	0.2
VP	$\rightarrow$	V NP	0.4
VP	$\rightarrow$	V	0.3
N	$\rightarrow$	{it, fish, rivers, December, they}	0.2
P	$\rightarrow$	{in}	1
V	$\rightarrow$	{can, fish}	0.5

Hale — Earley parser as a model of sentence processing

$edge_n$	DOTTED RULE	[S, W]	HIST	**Prob**	**MaxProb**
e_0	S → ● NP VP	[0,0]			P(S → NP VP)=1
e_1	NP → ● N	[0,0]			P(e_0)P(NP → N)=1*0.8=0.8
e_2	NP → ● N PP	[0,0]			P(e_0)P(NP → N PP)=1*0.2=0.2
e_3	N → they ●	[0,1]		P(N → they)=0.2	
e_4	NP → N ●	[0,1]	(e_3)	P(e_3)P(NP → N) =0.2*0.8 =0.16	
e_5	NP → N ● PP	[0,1]	(e_3)		
e_6	S → NP ● VP	[0,1]	(e_4)		
e_7	PP → ● P NP	[1,1]			P(N → they)P(e_2)P(PP → P NP) =0.2*1*0.2*1=0.04
e_8	VP → ● V	[1,1]			P(N → they)P(e_1)P(VP → V) =0.2*1*0.8*0.3=0.048
e_9	VP → ● V NP	[1,1]			P(N → they)P(e_1)P(VP → V NP) =0.2*1*0.8*0.4=0.064
e_{10}	VP → ● V VP	[1,1]			P(N → they)P(e_1)P(VP → V VP) =0.2*1*0.8*0.2=0.032
e_{11}	VP → ● VP PP	[1,1]			P(N → they)P(e_1)P(VP → VP PP) =0.2*1*0.8*0.1=0.0016
e_{12}	V → can ●	[1,2]		P(V → can)=0.5	
e_{13}	VP → V ●	[1,2]	(e_{12})	P(e_{12})P(VP → V) =0.5*0.3 =0.15	
e_{14}	VP → V ● NP	[1,2]	(e_{12})		
e_{15}	VP → V ● VP	[1,2]	(e_{12})		
e_{16}	S → NP VP ●	[0,2]	(e_4,e_{13})	P(e_4)P(e_{13})P(S → NP VP) =0.2*0.8*0.5*0.3*1 =0.024	
e_{17}	VP → VP ● PP	[1,2]	(e_{13})		

Yngve—PDA as a model of sentence processing



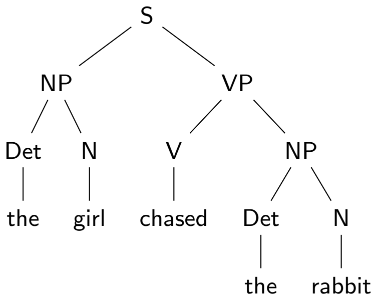
- **Hypothesis**: the size of the stack correlates with working memory load.
- **Prediction**: sentences which require many items to be placed on the stack will be difficult to process and also less frequent in the language.
- **Prediction**: when multiple parses are possible we should prefer the one with the minimised stack.

Yngve—PDA as a model of sentence processing

- Yngve formulated the problem as interaction between:
  - a **register** (which holds the current node) and
  - the **stack** (which contains all the nodes left to explore)
- Sentences are constructed top-down and left-to-right.
- Under these circumstances the size of the stack is hypothesised to correlate with working memory load.

Hypothesis: stack correlates with **working memory load**

S→NP VP
NP→Det N
VP→V NP
Det→the
N→girl
N→rabbit
V→chased



Register	Stack
S	
NP	VP
Det	N VP
the	N VP
N	VP
girl	VP
VP	
V	NP
chased	NP
NP	
Det	N
the	N
N	
rabbit	

Hypothesis: stack correlates with **working memory load**

Yngve's model makes **predictions** about centre embedding:

- Consider:

  This is the malt that the rat that the cat that the dog worried killed ate.

  STACK: N VP VP VP

- as opposed to:

  This is the malt that was eaten by the rat that was killed by the cat that was worried by the dog.

- Yngve evaluated his predictions by looking at frequencies of constructions in corpus data.

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Recap:

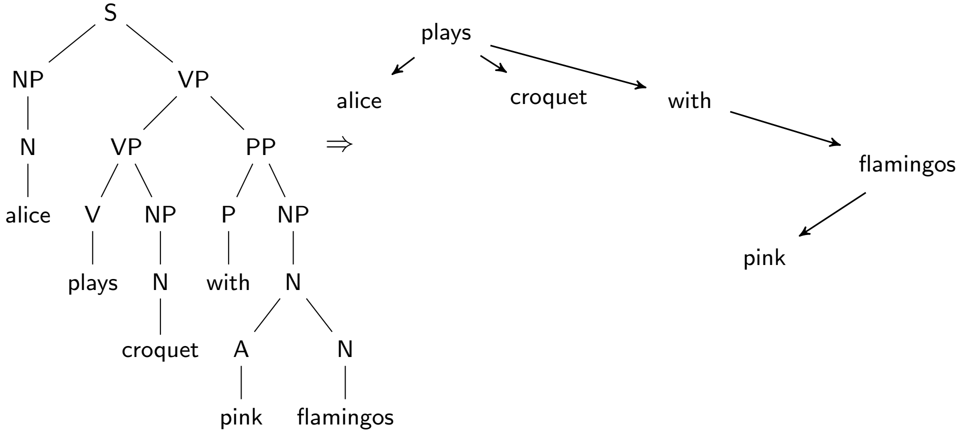
- We said LR shift-reduce parser wasn't a good fit for natural language because it proceeds deterministically and natural language is too ambiguous.
- We used the Earley parser to explore the whole tree-space, recording partial derivations in a chart.

However,

- We can use a modified version of the shift-reduce parser in order to parse natural language.
- First we're going to learn about **dependency grammars**.

A **dependency tree** is a directed graph

A **dependency tree** is a directed graph representation of a string—each edge represents a grammatical relationship between the symbols.



A **dependency grammar** derives **dependency trees**

Formally $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$ where:

- Σ is the finite set of alphabet symbols
- $\mathcal{D}$ is the set of symbols to indicate whether the dependent symbol (the one on the RHS of the rule) will be located on the left or right of the current item within the string $\mathcal{D} = \{\mathcal{L}, \mathcal{R}\}$
- s is the root symbol for the dependency tree (we will use $s \in \Sigma$ but sometimes a special extra symbol is used)
- $\perp$ is a symbol to indicate a halt in the generation process
- $\mathcal{P}$ is a set of rules for generating dependencies:
  $\mathcal{P} = \{(\alpha \to \beta,\ d) \mid \alpha \in (\Sigma\ \cup\ s),\ \beta \in (\Sigma \cup \perp),\ d \in \mathcal{D}\}$

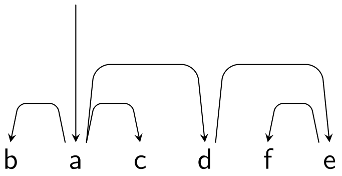
In dependency grammars we refer to the term on the LHS of a rule as the **head** and the RHS as the **dependent** (as opposed to *parents* and *children* in phrase structure grammars).

Dependency trees have **several representations**

Two diagrammatic representations of a dependency tree for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$ where:

$$
\begin{array}{lcl}
\Sigma & = & \{a...f\} \\
D & = & \{\mathcal{L}, \mathcal{R}\} \\
s & = & \text{a} \\
\mathcal{P} & = & \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
 & & (d \rightarrow e, \mathcal{R}) \\
 & & (e \rightarrow f, \mathcal{L}) \\
 & & (b \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R}) \\
 & & (c \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R}) \\
 & & (f \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R})\}
\end{array}
$$

$$\begin{tikzcd}
 & & \text{a} \arrow[dll] \arrow[d] \arrow[dr] & & \\
\text{b} & & \text{c} & \text{d} \arrow[dr] & \\
 & & & & \text{e} \arrow[dl] \\
 & & & \text{f} &
\end{tikzcd}$$



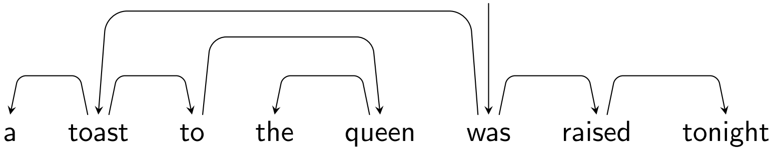
The same rules would have been used to generate the string *badfec*.

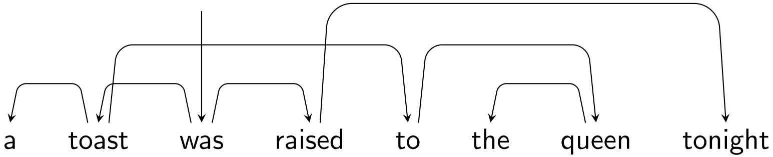
Useful when there is flexibility in the symbol order of grammatical strings.

Valid trees may be **projective** or **non-projective**

Valid derivation is one that is **rooted** in *s* and is **weakly connected**.

- Derivation trees may be **projective** or **non-projective**.
- Non-projective trees can be needed for long distance dependencies.





- The difference has implications for parsing complexity.

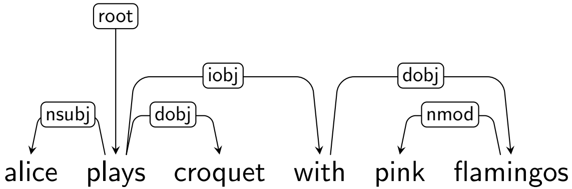
Labels can be added to the dependency edges

A label can be added to each generated dependency:

$$\mathcal{P} = \{(\alpha \to \beta : r,\ d) \mid \alpha \in (\Sigma \cup s),\ \beta \in (\Sigma \cup \bot),\ d \in \mathcal{D},\ r \in \mathcal{B}\}$$

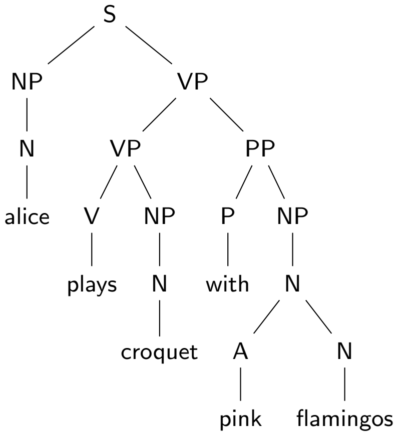
where $\mathcal{B}$ is the set of dependency labels.

When used for natural language parsing, dependency grammars will often label each dependency with the *grammatical function* (or the **grammatical relation**) between the words.

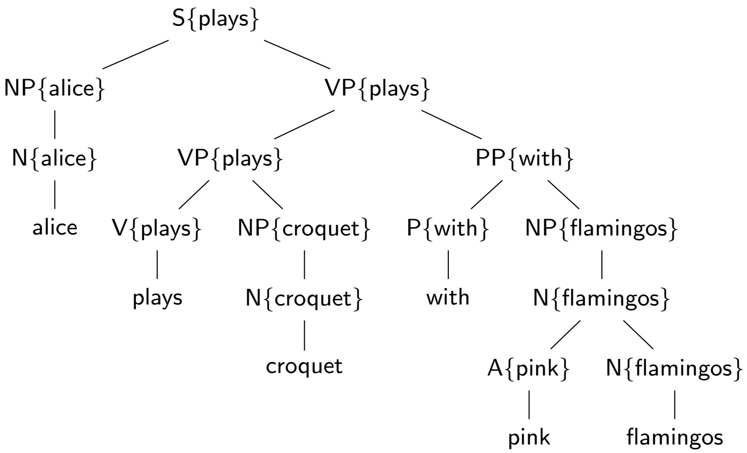


Dependency grammars can be **weakly equivalent** to CFGs

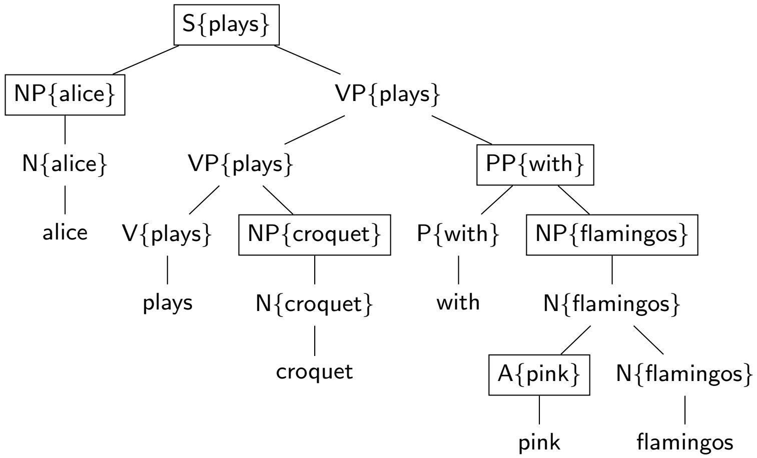
Projective dependency grammars can be shown to be **weakly equivalent** to context-free grammars.



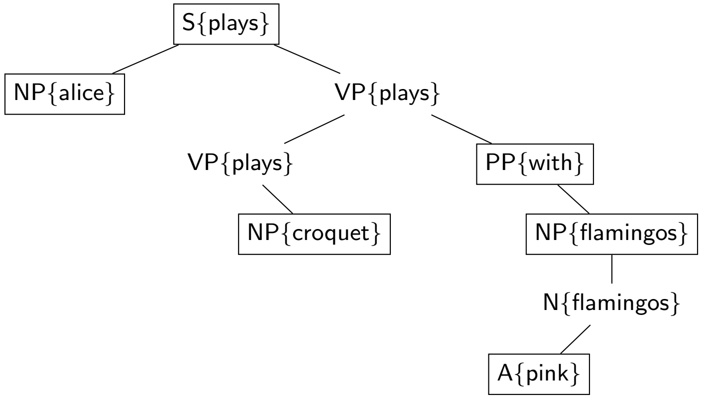
Dependency grammars can be **weakly equivalent** to CFGs



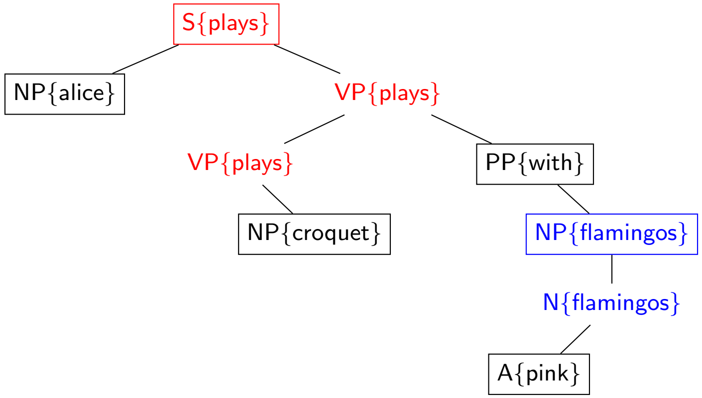
Dependency grammars can be **weakly equivalent** to CFGs



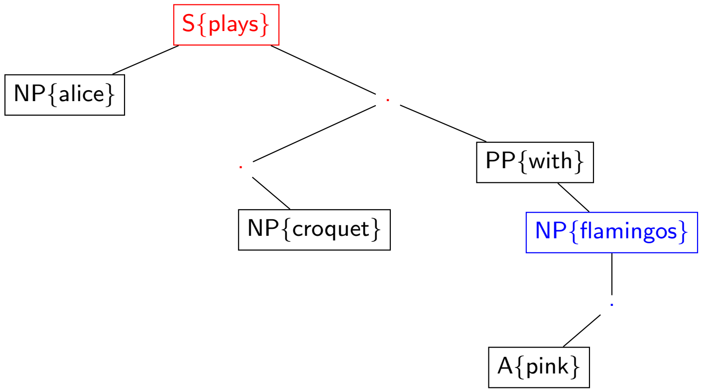
Dependency grammars can be **weakly equivalent** to CFGs



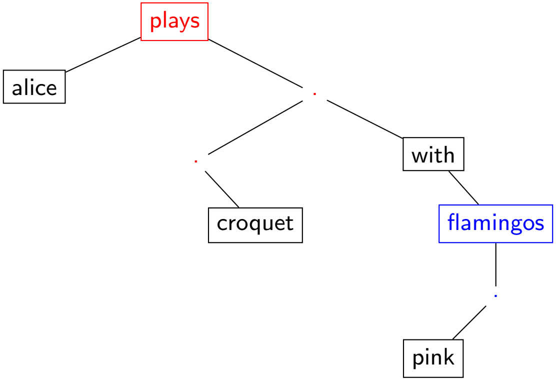
Dependency grammars can be **weakly equivalent** to CFGs



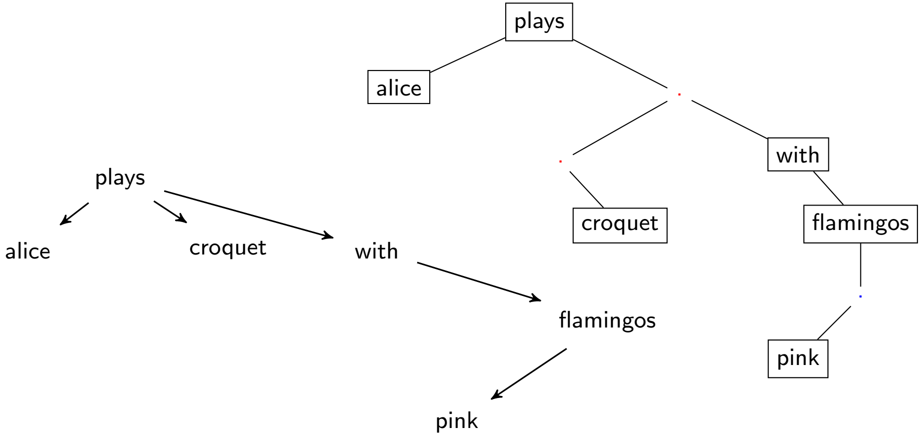
Dependency grammars can be **weakly equivalent** to CFGs



Dependency grammars can be **weakly equivalent** to CFGs



Dependency grammars can be **weakly equivalent** to CFGs



Projective dependency grammars can be shown to be **weakly equivalent** to context-free grammars.

Dependency parsers use a **modified** shift-reduce parser

- A common method for dependency parsing of natural language involves a modification of the LR shift-reduce parser
- The **shift** operator continues to move items of the input string from the buffer to the stack
- The **reduce** operator is replaced with the operations **left-arc** and **right-arc** which *reduce* the top two stack symbols leaving the *head* on the stack

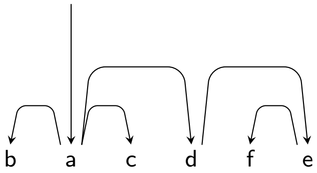
Consider $\mathcal{L}(G_{dep}) \subseteq \Sigma^*$, during parsing the stack may hold γab where $\gamma \in \Sigma*$ and $a, b \in \Sigma$, and b is at the top of the stack:

- LEFT-ARC reduces the stack to γb and records use of rule $b \rightarrow a$
- RIGHT-ARC reduces the stack to γa and records the use of rule $a \rightarrow b$

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned} \Sigma &= \{a...z\} \\ D &= \{\mathcal{L}, \mathcal{R}\} \\ s &= \mathrm{s} \\ \mathcal{P} &= \{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\ &\quad (d \to e, \mathcal{R}) \\ &\quad (e \to f, \mathcal{L})\} \end{aligned}$$



STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \to b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \to c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \to f$
ade		RIGHT-ARC	$d \to e$
ad		RIGHT-ARC	$a \to d$
a		TERMINATE	$root \to a$

Note that, for a deterministic parse here, a lookahead is needed

Data driven dependency parsing is **grammarless**

- For natural language there would be considerable effort in manually defining $\mathcal{P}$—this would involve determining the dependencies between all possible words in the language.
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items on the stack and in the buffer as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

We can use a **beam** search to record the parse forest

- The classifier can return a **probability** of an action.
- To avoid the problem of early incorrect resolution of an ambiguous parse, multiple competing parses can be recorded and a **beam search** used to keep track of the best alternative parses.
- Google's *Parsey McParseface* is an English language dependency parser that uses word-embeddings as features and a neural network to score parse actions. A beam search is used to compare competing parses.

Dependency parsers can be useful for parsing **speech**

The most obvious difference between spoken and written language is the mode of transmission:

- **Prosody** refers to the patterns of stress and intonation in a language.
- **Stress** refers to the relative emphasis or prominence given to a certain part of a word (e.g. CON-tent (the stuff included in something) vs. con-TENT (happy))
- **Intonation** refers to the way speakers' pitch rises and falls in line with words and phrases, to signal a question, for example.
- Co-speech gestures involve parts of the body which move in coordination with what a speaker is saying, to emphasise, disambiguate or otherwise.

We can use some of these extra features to help the parse-action-classifier when parsing spoken language.

Prosody has been used to resolve parsing ambiguity

- Briscoe suggested using a shift-reduce parser that favours shift over reduce wherever both are possible.
- In the absence of extra-linguistic information the parser delays resolution of the grammatical dependency.
- Extra features enable an override of the shift preference at the point where the ambiguity arises, including:
  - prosodic information (intonational phrase boundary)

  The model accounts for frequencies of certain syntactic constructions as attested in corpora.

Spoken language **lacks** string delimitation

- A fundamental issue that affects syntactic parsing of spoken language is the **lack of the sentence unit** (i.e string delimitation)—indicated in writing by a full-stop and capital letter.
- **Speech units** may be identified by pauses, intonation (e.g. rising for a question, falling for a full-stop), change of speaker.
- Speech units are not much like written sentences due to **speaker overlap**, **co-constructions**, **ellipsis**, **hesitation**, **repetitions** and **false starts**.
- Speech units often contain words and grammatical constructions that would not appear in the written form of the language.

Spoken language **lacks** string delimitation

Excerpt from the Spoken section of the British National Corpus

set your sights realistically haven't you and there's a lot of people unemployed and what are you going to do when you eventually leave college if you get there you're not gonna step straight into television mm right then let's see now what we're doing where's that recipe book for that chocolate and banana cake chocolate and banana cake which book was it oh right oh some of these chocolate cakes are absolutely mm mm mm right what's the topping what's that icing sugar cocoa powder and vanilla essence oh luckily I've got all those I think yes

Spoken language **lacks** string delimitation

Excerpt from the Spoken section of the British National Corpus

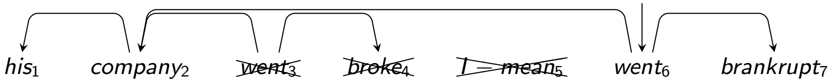
Set your sights realistically haven't you? And there's a lot of people unemployed. And what are you going to do when you eventually leave college? If you get there. You're not gonna step straight into television. Mm right then, let's see now what we're doing... Where's that recipe book for that chocolate and banana cake? Chocolate and banana cake which book was it? Oh right. Oh, some of these chocolate cakes are absolutely mm mm mm. Right, what's the topping? what's that? Icing sugar, cocoa powder and vanilla essence. Oh luckily I've got all those I think, yes!

Dependency parsers can be useful for parsing **speech**

- Spoken language can look noisy and somewhat *grammarless* but the disfluencies are predictable
- Honnibal & Johnson's Redshift parser introduces an **edit** action, to remove disfluent items from spoken language:
  - EDIT: on detection of disfluency, remove connected words and their dependencies.
- Parser uses extra classifier features to detect disfluency.

Example of dependency parser using an **edit** action

STACK	BUFFER	ACTION	RECORD
	his_1 … $bankrupt_7$	SHIFT	
his_1	$company_2$ … $bankrupt_7$	SHIFT	
his_1 $company_2$	$went_3$ … $bankrupt_7$	LEFT-ARC	$company_2 \rightarrow his_1$
$company_2$	$went_3$ … $bankrupt_7$	SHIFT	
$company_2$ $went_3$	$broke_4$ … $bankrupt_7$	LEFT-ARC	~~$went_3 \rightarrow company_2$~~
$went_3$	$broke_4$ … $bankrupt_7$	SHIFT	
$went_3$ $broke_4$	$I - mean_5$ … $bankrupt_7$	RIGHT-ARC	~~$went_3 \rightarrow broke_4$~~
$went_3$	$I - mean_5$ … $bankrupt_7$	SHIFT	
$went_3$ $I - mean_5$	$went_6$ $bankrupt_7$	EDIT	
$company_2$	$went_6$ $bankrupt_7$	SHIFT	
$company_2$ $went_6$	$bankrupt_7$	LEFT-ARC	$went_6 \rightarrow company_2$
$went_6$	$bankrupt_7$	SHIFT	
$went_3$ $bankrupt_7$		RIGHT-ARC	$went_6 \rightarrow bankrupt_7$
$went_3$		TERMINATE	$root \rightarrow went_6$



Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

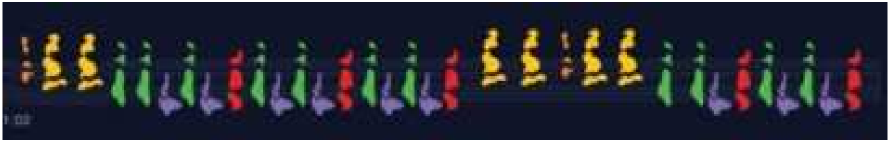
- Last time we looked at ways to parse without ever building a grammar
- But what if we want to know what a grammar is for a set of strings?

- Today we will look at grammar induction.

  ...we'll start with an example

CFGs may be inferred using recursive **byte-pair encoding**

The following is a *speech unit* of whale song:



b a a c c d c d e c d c d e c d c d e a a b a a c c d e c d c d e

We are going to infer some rules for this string using the following algorithm:

- count the frequency of all adjacent pairs in the string
- reduce the most frequent pair to a non-terminal
- repeat until there are no pairs left with a frequency > 1

This is used for compression—once we have removed all the repeated strings we have less to transmit or store (we have to keep the grammar to decompress)

CFGs may be inferred using recursive **byte-pair encoding**

b a a c c d c d e c d c d e c d c d e a a b a a c c d e c d c d e

$F \rightarrow c\ d$

b a a c F F e F F e F F e a a b a a c F e F F e

$G \rightarrow F\ e$

b a a c F G F G F G a a b a a c G F G

$H \rightarrow F\ G$

b a a c H H H a a b a a c G H

$I \rightarrow a\ a$

b I c H H H I b I c G H

$J \rightarrow b\ I$

J c H H H I J c G H

$K \rightarrow J\ c$

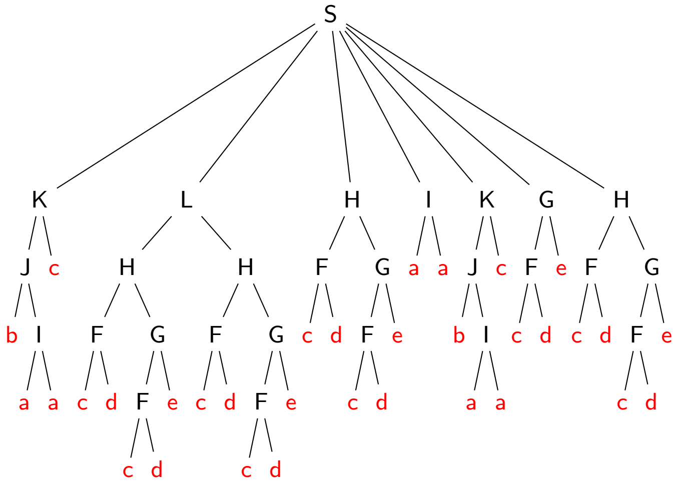
K H H H I K G H

$L \rightarrow H\ H$

K L H I K G H

$S \rightarrow K\ L\ H\ I\ K\ G\ H$

CFGs may be inferred using recursive **byte-pair encoding**


S
K L H I K G H
J c H H F G a a J c F e F G
b I F G F G c d F e b I c d c d F e
a a c d F e c d F e c d a a c d
c d c d

Byte-pair has **shortcomings** for grammar induction

Byte-pair encoding has benefits for encryption but shortcomings when it comes to grammar induction (especially of natural language):

- the algorithm is frequency driven and this might not lead to *appropriate* constituency
- in the circumstance that two pairs have the same frequency we make an arbitrary choice of which to reduce.
- the data is assumed to be non-noisy (all string sequences encountered are treated as valid)
- (for natural language) the algorithm learns from strings alone (a more *appropriate* grammar might be derived by including extra-linguistic information)

We might suggest improvements to the algorithm (such as allowing ternary branching) but in order to compare the algorithms we need a **learning paradigm** in which to study them.

Paradigms are defined over **grammatical systems**

Grammatical system:

- $\mathcal{H}$ a hypothesis space of language descriptions (e.g. all possible grammars)
- Ω a sample space (e.g. all possible strings)
- $\mathcal{L}$ a function that maps from a member of $\mathcal{H}$ to a subset of Ω

If we have $(\mathcal{H}_{cfg}, \Sigma^*, \mathcal{L})$ then for some $G \in \mathcal{H}_{cfg}$ we have:
$\mathcal{L}(G) = \{s_a, s_b, s_c ...\} \subseteq \Sigma*$

Learning function:

The learning function, F, maps from a subset of Ω to a member of $\mathcal{H}$

For $G \in \mathcal{H}_{cfg}$ then $F(\{s_d, s_e, s_f ...\}) = G$ for some $\{s_d, s_e, s_f ...\} \subseteq \Sigma*$

Note that the learning function is an algorithm (referred to as the **learner**) and that **learnability** is a property of a language class (when F surjective).

Learning paradigms specify the nature of **input**

Varieties of input given to the learner:

- **positive evidence**: the learner receives only valid examples from the sample space (i.e. if the underlying grammar is G then the learner receives samples, s_i, such that $s_i \in \mathcal{L}(G)$).
- **negative evidence**: the learner receives samples flagged as not being in the language.
- **exhaustive evidence**: the learner receives every relevant sample from the sample space.
- **non-string evidence**: the learner receives samples that are not strings.

Learning paradigms also specify...

- **assumed knowledge**: the things known to the learner before learning commences (for instance, the hypothesis space, $\mathcal{H}$ might be assumed knowledge).
- **nature of the algorithm**: are samples considered sequentially or as a batch? does the learner generate a hypothesis after every sample received in a sequence? does the learner generate a hypothesis after specific samples only?
- **required computation**: e.g. is the learner constrained to act in polynomial time.
- **learning success**: what are the criteria by which we measure success of the learner?

Gold's learning paradigms have been influential

Gold's best known paradigm modelled language learning as an infinite process in which a learner is presented with an infinite stream of strings of the target language:

- for a grammatical system $(\mathcal{G}, \Sigma^*, \mathcal{L})$
- select one of the languages L in the class defined by $\mathcal{L}$ (this is called the **target language**, $L = \mathcal{L}(G)$ where $G \in \mathcal{G}$)
- samples are presented to the learner one at a time $s_1, s_2, ...$ in an infinite sequence
- the learner receives only positive evidence (i.e. only s_i such that $s_i \in L$)
- after each sample the learner produces a hypothesis (i.e. learner produces G_n after having seen the data $s_1, ...s_n$
- the evidence is exhaustive, every $s \in L$ will be presented in the sequence.

Gold's learning paradigms have been influential

Gold defined **identification in the limit** as successful learning:

- There is some number N such that for all $i > N$, $G_i = G_N$ and $\mathcal{L}(G_N) = L$
- N is finite but there are no constraints placed on computation time of the learning function.

In this paradigm a **class** of languages is **learnable** if:

- Every language in the class can be identified in the limit *no matter what order* the samples appear in

Gold's learning paradigms have been influential

Well known results from Gold's paradigm include:

- The class of **suprafinite** languages are not learnable (a suprafinite class of languages is one that contains all finite languages and at least one infinite language)
- This means that e.g. the class of context-free languages are *not learnable* within Gold's paradigm.

We might care about this if we think that Gold's paradigm is a good model for natural language acquisition...(if we don't think this then it is just a fun result!).

Gold: **suprafinite** languages are not learnable

Short proof:

- Let L_∞ be an infinite language $L_\infty = \{s_1, s_2, ...\}$
- Now construct an infinite sequence of finite languages $L_1 = \{s_1\}$, $L_2 = \{s_1, s_2\}$, ...
- Consider a particular presentation order $s_1...s_1, s_2...s_2, s_3...$
- When learning L_1 we repeat s_1 until the learner predicts L_1
- When learning L_2 repeat s_1 until the learner predicts L_1 then repeat s_2 until it predicts L_2
- Continue like this for all L_i: either the learner fails to converge on one of these, or it ultimately fails to converge on L_∞ for finite N.
- We have found an ordering of the samples that makes the learner fail

Many people have investigated what IS learnable in this paradigm. We will look at one example, but to do so we introduce one more grammar.

Categorial grammars are **lexicalized grammars**

In a **classic categorial grammar** all symbols in the alphabet are associated with a finite number of **types**.

- Types are formed from primitive types using two operators, $\backslash$ and $/$.
- If P_r is the set of **primitive types** then the set of all types, T_p, satisfies:
  - $P_r \subset T_p$
  - if $A \in T_p$ and $B \in T_p$ then $A\backslash B \in T_p$
  - if $A \in T_p$ and $B \in T_p$ then $A/B \in T_p$
- Note that it is possible to arrange types in a hierarchy: a type A is a *subtype* of B if A occurs in B (that is, A is a subtype of B iff $A = B$; or ($B = B_1\backslash B_2$ or $B = B_1/B_2$) and A is a subtype of B_1 or B_2).

Categorial grammars are **lexicalized grammars**

- A relation, $\mathcal{R}$, maps symbols in the alphabet Σ to members of T_p.
- A grammar that associates at most one type to each symbol in Σ is called a **rigid grammar**
- A grammar that assigns at most k types to any symbol is a **k-valued grammar**.
- We can define a classic categorial grammar as $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:
  - Σ is the alphabet/set of terminals
  - P_r is the set of primitive types
  - S is a distinguished member of the primitive types $S \in P_r$ that will be the root of complete derivations
  - $\mathcal{R}$ is a relation $\Sigma \times T_p$ where T_p is the set of all types as generated from P_r as described above

Categorial grammars are **lexicalized grammars**

A string has a valid parse if the types assigned to its symbols can be combined to produce a derivation tree with root S.

Types may be combined using the two rules of function application:

- FORWARD APPLICATION is indicated by the symbol $>$:

$$\frac{A/B \qquad B}{A} >$$

- BACKWARD APPLICATION is indicated by the symbol $<$:

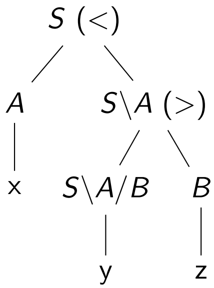
$$\frac{B \qquad A\backslash B}{A} <$$

Categorial grammars are **lexicalized grammars**

Derivation tree for the string *xyz* using the grammar $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:

$$\begin{aligned} Pr &= \{S, A, B\} \\ \Sigma &= \{x, y, z\} \\ S &= S \\ \mathcal{R} &= \{(x, A), (y, S\backslash A/B), (z, B)\} \end{aligned}$$

$$\dfrac{\dfrac{x}{A}\,\mathcal{R} \qquad \dfrac{\dfrac{y}{S\backslash A/B}\,\mathcal{R} \qquad \dfrac{z}{B}\,\mathcal{R}}{S\backslash A}\,>}{S}\,<$$

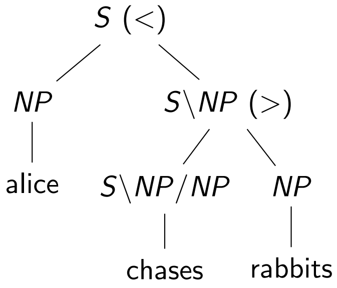


Categorial grammars are **lexicalized grammars**

Derivation tree for the string *Alice chases rabbits* using the grammar $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:

$$\begin{aligned}
Pr &= \{S, NP\}\\
\Sigma &= \{alice, chases, rabbits\}\\
S &= S\\
\mathcal{R} &= \{(alice, NP), (chases, S\backslash NP/NP),\\
&\quad (rabbits, NP)\}
\end{aligned}$$

$$\cfrac{\cfrac{alice}{NP}\,\mathcal{R} \qquad \cfrac{\cfrac{chases}{S\backslash NP/NP}\,\mathcal{R} \qquad \cfrac{rabbits}{NP}\,\mathcal{R}}{S\backslash NP}\,>}{S}\,<$$



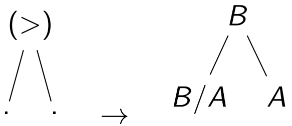
We can construct a **strongly equivalent** CFG

To create a context-free grammar $G_{cfg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$ with strong equivalence to $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ we can define G_{cfg} as:

$$
\begin{aligned}
\mathcal{N} &= P_r \cup range(\mathcal{R}) \\
\Sigma &= \Sigma \\
S &= S \\
P &= \{A \rightarrow B \;\; A\backslash B \mid A\backslash B \in range(\mathcal{R})\} \\
&\quad \cup \{A \rightarrow A/B \;\; B \mid A/B \in range(\mathcal{R})\} \\
&\quad \cup \{A \rightarrow a \mid \mathcal{R} : a \rightarrow A\}
\end{aligned}
$$

FYI: a categorial grammar learner within Gold's paradigm

- Buszkowski developed an algorithm for learning rigid grammars from **functor-argument structures**.
- The algorithm proceeds by inferring types from the available information
- Eg. for Forward Application:



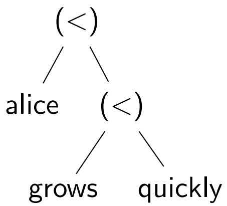
- Variables are unified across all encountered structures.
- Kanazawa constructed a proof to show that the algorithm could learn the class of rigid grammars from an **infinite stream** of functor-argument structures — as required to satisfy Gold's paradigm.

FYI: a categorial grammar learner within Gold's paradigm

Let G_i be the current hypothesis of the learner:

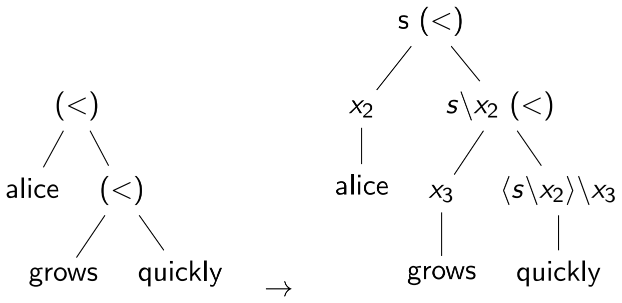
$$\begin{aligned} G_i : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s\backslash x_1 \end{aligned}$$

Let the next functor-argument structor encountered in the steam be:



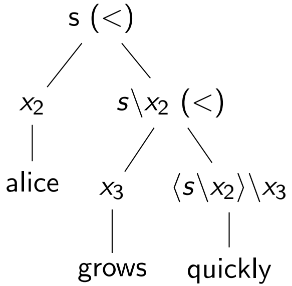
FYI: a categorial grammar learner within Gold's paradigm

Infer types to the new functor-argument structure:



FYI: a categorial grammar learner within Gold's paradigm

- Look up words at the leaf nodes of the new structure in G_i
- If the word exists in G_i, add types inferred at leaf nodes to the existing set of types for that word; else create new word entry.



$$\begin{aligned} G_i : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s\backslash x_1 \end{aligned}$$

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1, x_2 \\ \text{grows} &\rightarrow s\backslash x_1, x_3 \\ \text{quickly} &\rightarrow \langle s\backslash x_2\rangle\backslash x_3 \end{aligned}$$

FYI: a categorial grammar learner within Gold's paradigm

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1, x_2 \\ \text{grows} &\rightarrow s\backslash x_1, x_3 \\ \text{quickly} &\rightarrow \langle s\backslash x_2\rangle\backslash x_3 \end{aligned}$$

- Unify the set of types. If unification fails then fail.

$$\begin{aligned} x_2 &\mapsto x_1 \\ x_3 &\mapsto s\backslash x_1 \end{aligned}$$

- Output the lexicon.

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s\backslash x_1 \\ \text{quickly} &\rightarrow \langle s\backslash x_2\rangle\backslash\langle s\backslash x_1\rangle \end{aligned}$$

FYI: a categorial grammar learner within Gold's paradigm

Using this learner within Gold's paradigm over various sample spaces it is possible to prove:

- Rigid grammars are learnable from functor-argument structor and strings
- k-valued grammars (for a specific k) are learnable from functor-argument structor and strings

Note that the above mentioned grammars are subsets of the CFGs

Gold's paradigm **is not** much like human acquisition

- Gold's paradigm requires convergence in a finite number of steps (hypotheses of G) the amount of data it sees is unbounded.
- Gold's learner can use unbounded amounts of computation.

- A child only sees a limited amount of data, and has limited computational resources

- Success in this paradigm tells you absolutely nothing about the learner's state at any finite time.

- Children learn progressively

- The learner has to learn for every possible presentation of the samples (including presentations that have been chosen by an adversary with knowledge of the internal state of the learner).

- It is arguable that the distributions are in some way helpful: **parentese**

Gold's paradigm **is not** much like human acquisition

- Gold's learner is required to exactly identify the target language.

- We do not observe this in humans

  We observe agreement on *grammaticality* between adults and children approaching adult competence but we also observe differences in word choices and grammaticality judgments between adults speakers.

- Gold's learner requires a hypothesis to be selected after every step.

- In fact there is evidence that children only attend to selective evidence (Goldilocks effect)

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Languages transmit **information**

In previous lectures we have thought about language in terms of **computation**.

Today we are going to discuss language in terms of the **information** it conveys...

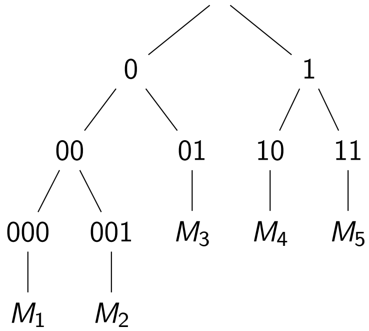
Entropy is a measure of information

- Information **sources** produce **information** as **events** or **messages**.
- Represented by a random variable X over a discrete set of symbols (or alphabet) $\mathcal{X}$.
- e.g. for a dice roll $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ for a source that produces characters of written English $\mathcal{X} = \{a...z, \}$
- **Entropy** (or **self-information**) may be thought of as:
  - the average amount of information produced by a source
  - the average amount of uncertainty of a random variable
  - the average amount of information we gain when receiving a message from a source
  - the average amount of information we lack before receiving the message
  - the average amount of uncertainty we have in a message we are about to receive

Entropy is a measure of information

- Entropy, H, is measured in **bits**.
- If X has M equally likely events: $H(X) = \log_2 M$
- Entropy gives us a **lower limit** on:
  - the number of bits we need to represent an event space.
  - the average number of bits you need per message code.

$$\begin{aligned} avg\_length &= \frac{(3*2)+(2*3)}{5} = 2.4 \\ &> H(5) = \log_2 5 = 2.32 \end{aligned}$$



Surprisal is also measured in bits

- Let $p(x)$ be the probability mass function of a random variable, X over a discrete set of symbols $\mathcal{X}$.
- The **surprisal** of x is $s(x) = \log_2 \left(\frac{1}{p(x)}\right) = -\log_2 p(x)$
- Surprisal is also measured in **bits**
- Surprisal gives us a measure of information that is inversely proportional to the probability of an event/message occurring
- i.e probable events convey a small amount of information and improbable events a large amount of information
- The average information (entropy) produced by X is the weighted sum of the surprisal (the average surprise): $H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x)$
- Note, that when all M items in $\mathcal{X}$ are equally likely (i.e. $p(x) = \frac{1}{M}$) then $H(X) = -\log_2 p(x) = \log_2 M$

The surprisal of the alphabet in *Alice in Wonderland*

x	$f(x)$	$p(x)$	$s(x)$
	26378	0.197	2.33
e	13568	0.101	3.30
t	10686	0.080	3.65
a	8787	0.066	3.93
o	8142	0.056	4.04
i	7508	0.055	4.16
…			
v	845	0.006	7.31
q	209	0.002	9.32
x	148	0.001	9.83
j	146	0.001	9.84
z	78	0.001	10.75

- If uniformly distributed: $H(X) = \log_2 27 = 4.75$
- As distributed in *Alice*: $H(X) = 4.05$
- Re. example 1:
- Average surprisal of a vowel = 4.16 bits (3.86 without u)
- Average surprisal of a consonant = 6.03 bits

Example 1

Last consonant removed:
Jus the he hea struc agains te roo o te hal: i fac se wa no rathe moe tha nie fee hig.
average missing information: 4.59 bits

Last vowel removed:
Jst thn hr hed strck aganst th rof f th hll: n fct sh ws nw rathr mor thn nin fet hgh.
average missing information: 3.85 bits

Original sentence:
Just then her head struck against the roof of the hall: in fact she was now rather more than nine feet high.

The surprisal of words in *Alice in Wonderland*

x	$f(x)$	$p(x)$	$s(x)$
the	1643	0.062	4.02
and	872	0.033	4.94
to	729	0.027	5.19
a	632	0.024	5.40
she	541	0.020	5.62
it	530	0.020	5.65
of	514	0.019	5.70
said	462	0.017	5.85
i	410	0.015	6.02
alice	386	0.014	6.11
...			
<any>	3	0.000	13.2
<any>	2	0.000	13.7
<any>	1	0.000	14.7

Example 2

She stretched herself up on tiptoe, and peeped over the edge of the mushroom, and her eyes immediately met those of a large blue caterpillar, that was sitting on the top with its arms folded, quietly smoking a long hookah, and taking not the smallest notice of her or of anything else.

Average information of *of* = 5.7 bits

Average information of low frequency compulsory content words = 14.7 bits (freq = 1), 13.7 bits (freq = 2), 13.2 bits (freq = 3)

Aside: Is written English a good code?

Highly efficient codes make use of regularities in the messages from the source using shorter codes for more probable messages.

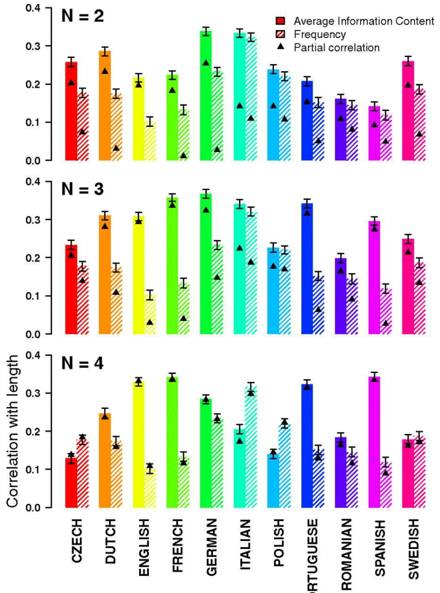
- From an encoding point of view, surprisal gives an indication of the number of bits we would want to assign a message symbol.
- It is efficient to give probable items (with low surprisal) a small bit code because we have to transmit them often.
- So, is English efficiently encoded?
- Can we predict the information provided by a word from its length?

Aside: Is written English a good code?

Piantadosi et al. investigated whether the surprisal of a word correlates with the word length.

- They calculated the average surprisal (average information) of a word w given its context c
- That is, $-\frac{1}{C}\sum_{i=1}^{C}\log_2 p(w|c_i)$
- Context is approximated by the n previous words.

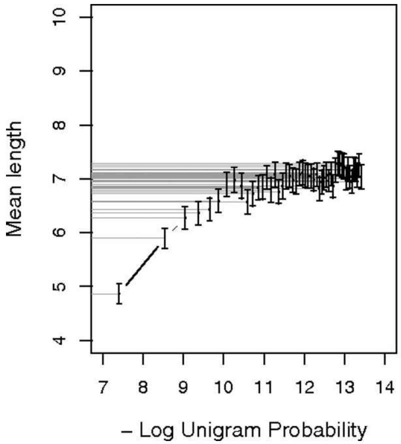
Aside: Is written English a good code?

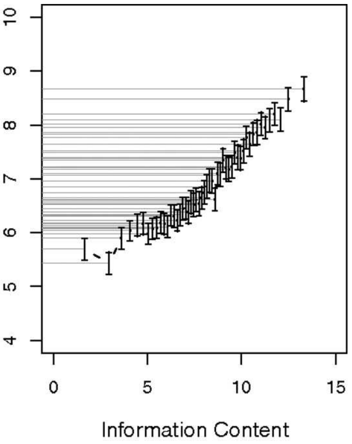


- Piantadosi et al. results for Google n-gram corpus.
- Spearman's rank on y-axis (0=no correlation, 1=monotonically related)
- Context approximated in terms of 2, 3 or 4-grams (i.e. 1, 2, or 3 previous words)
- Average information is a better predictor than frequency most of the time.

Aside: Is written English a good code?

Piantadosi et al: Relationship between frequency (negative log unigram probability) and length, and information content and length.





In language, events depend on context

Examples from *Alice in Wonderland*:

- Generated using $p(x)$ for $x \in \{a\text{-}z, \_\}$:

  dgnt_a_hi_tio__iui_shsnghihp_tceboi_c_ietl_ntwe_c_a_ad__ne_saa
  __hhpr___bre_c_ige_duvtnltueyi_tt__doe

- Generated using $p(x|y)$ for $x, y \in \{a\text{-}z, \_\}$:

  s_ilo_user_wa_le_anembe_t_anceasoke_ghed_mino_fftheak_ise_linld_met
  _thi_wallay_f_belle_y belde_se_ce

In language, events depend on context

Examples from *Alice in Wonderland*:

- Generated using $p(x)$ for $x \in \{\textit{words in Alice}\}$:

  didnt and and hatter out no read leading the time it two down to just this must goes getting poor understand all came them think that fancying them before this

- Generated using $p(x|y)$ for $x, y \in \{\textit{words in Alice}\}$:

  murder to sea i dont be on spreading out of little animals that they saw mine doesnt like being broken glass there was in which and giving it after that

In language, events depend on context

- **Joint entropy** is the amount of information needed on average to specify two discrete random variables:
  $H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_2 p(x,y)$
- **Conditional entropy** is the amount of extra information needed to communicate Y, given that X is already known:
  $H(Y|X) = \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_2 p(y|x)$
- **Chain rule** connects joint and conditional entropy:
  $H(X,Y) = H(X) + H(Y|X)$
  $H(X_1...X_n) = H(X_1) + H(X_2|X_1) + ... + H(X_n|X_1...X_{n-1})$

Example 3

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

Information in transitions of *Bandersnatch*:

- Surprisal of n given a = 2.45 bits
- Surprisal of d given n = 2.47 bits

Remember average surprisal of a character, $H(X)$, was 4.05 bits.
$H(X|Y)$ turns out to be about 2.8 bits.

What about Example 4?

Thank you, it's a very interesting dance to watch,' said Alice, feeling very glad that it was over at last.

To make predictions about when we insert *that* we need to think about **entropy rate**.

Entropy of a language is the **entropy rate**

- Language is a stochastic process generating a sequence of word tokens
- The entropy of the language is the entropy rate for the stochastic process:

$$H_{rate}(L) = \lim_{n\to\infty} \frac{1}{n} H(X_1 \ldots X_n)$$

- The entropy rate of language is the limit of the entropy rate of a sample of the language, as the sample gets longer and longer.

Hypothesis: **constant** rates of information are preferred

- The **capacity** of a communication **channel** is the number of bits on average that it can transmit
- Capacity defined by the noise in the channel—mutual information of the channel input and output (more next week)

- Assumption: language users want to maximize information transmission while minimizing comprehender difficulty.
- Hypothesis: language users prefer to distribute information uniformly throughout a message
- Entropy Rate Constancy Principle (Genzel & Charniak), Smooth Signal Redundancy Hypothesis (Aylett & Turk), Uniform Information Density (Jaeger)

Hypothesis: **constant** rates of information are preferred

Could apply the hypothesis at all levels of language use:

- In speech we can modulate the duration and energy of our vocalisations
- For vocabulary we can choose longer and shorter forms

  maths vs. *mathematics*, *don't* vs. *do not*
- At sentence level, we may make syntactic reductions:

  The rabbit (that was) chased by Alice.

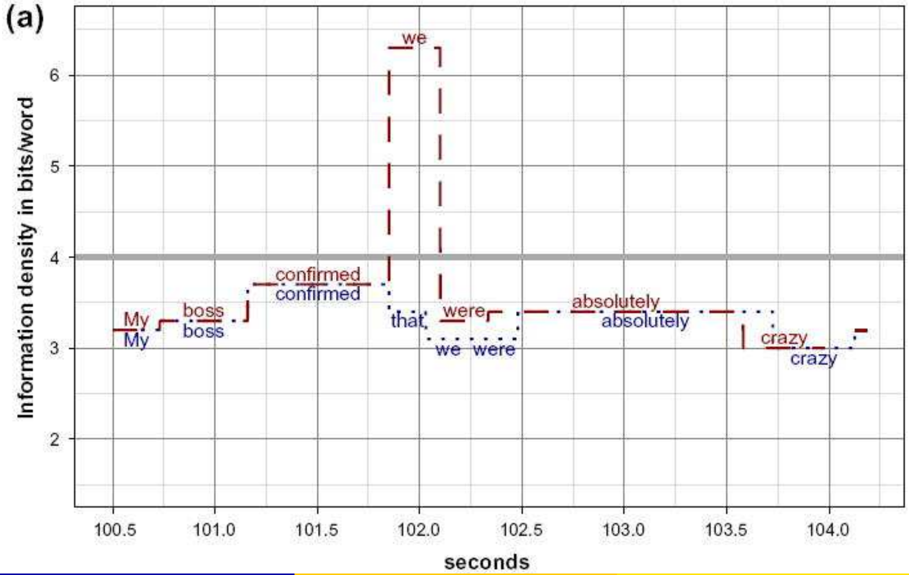
Hypothesis: **constant** rates of information are preferred

Uniform Information Density:

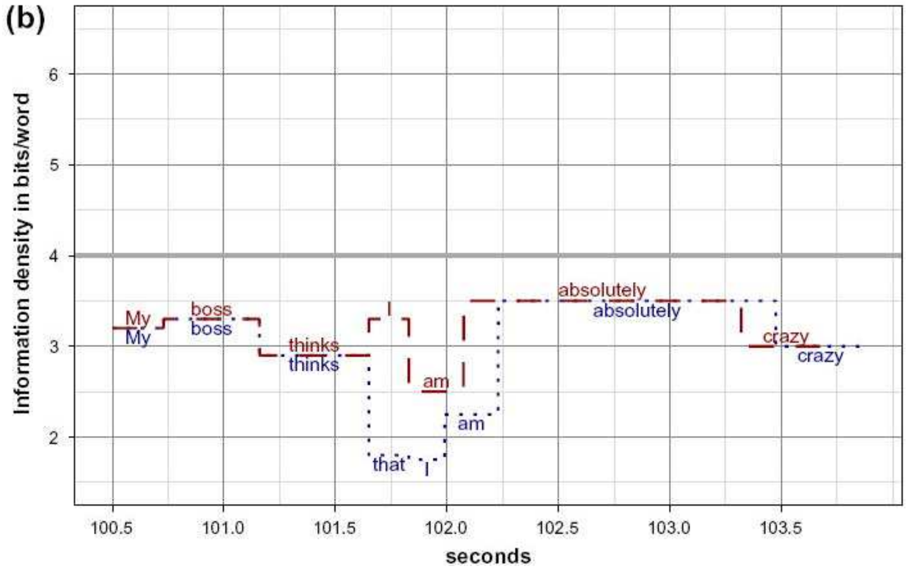
- Within the bounds defined by grammar, speakers prefer utterances that distribute information uniformly across the signal
- Where speakers have a choice between several variants to encode their message, they prefer the variant with more uniform information density

Evaluated on a large scale corpus study of complement clause structures in spontaneous speech (Switchboard Corpus of telephone dialogues)

Hypothesis: **constant** rates of information are preferred



Hypothesis: **constant** rates of information are preferred



- Notice that these information theoretic accounts are rarely explanatory (doesn't explicitly tell us what might be happening in the brain)
- An exception is Hale (2001) where we used surprisal to reason about parse trees and full parallelism
- Information theoretic accounts are unlikely to be the full story but they are predictive of certain phenomena

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

For communication, information has to be **transmitted**

Goal: To optimise, in terms of **throughput** and **accuracy**, the communication of messages in the presence of a **noisy channel**

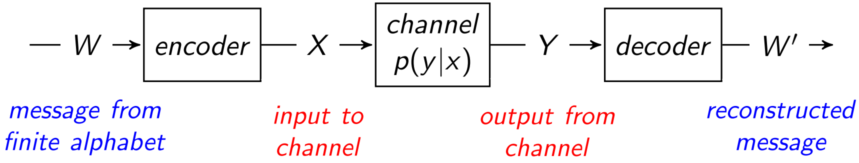
There is a trade off between:

- **compression**: making the most efficient code by removing all the redundancy
- **accuracy**: adding redundant information so that the input can still be recovered despite the presence of noise

Today we will:

- formalise the noisy channel more carefully
- look at some implications for natural language evolution
- see how the noisy channel model has inspired a framework for solving problems in Natural Language Processing.

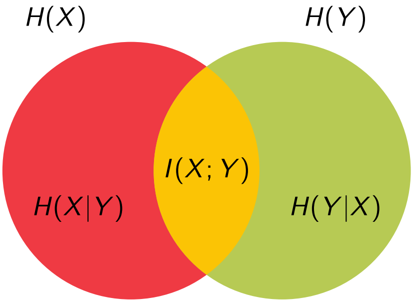
Transmission can be modelled using a **noisy channel**



- message should be efficiently encoded but with enough redundancy for the decoder to detect and correct errors
- the output depends probabilistically on the input
- the decoder finds the mostly likely original message given the output received

Mutual information: the information Y contains about X

- **Mutual Information** $I(X;Y)$ is a measure of the reduction in uncertainty of one random variable due to knowing about another
- Can also think of $I(X;Y)$ being the amount of information one random variable contains about another



$H(X)$ average information of input

$H(Y)$ average information in output

$H(X|Y)$ the uncertainty in (extra information needed for) X given Y is known

$I(X;Y)$ the mutual information; the information in Y that tells us about X

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \end{aligned}$$

Channel **capacity** is determined by **mutual information**

- The capacity of a channel is the maximum of the mutual information of X and Y over all input distributions of the input $p(X)$
$$C = \max_{p(X)} I(X;Y)$$
- C is the rate we can transmit information through a channel with an arbitrarily low probability of not being able to recover the input from the output
- As long as transmission rate is less than C we don't need to worry about errors (optimal rate is C)
- If transmission rate exceeds C then we need to slow down (e.g. by inserting a *that*—last lecture)
- In practical applications we reach the channel capacity by designing an encoding for the input that maximises mutual information.

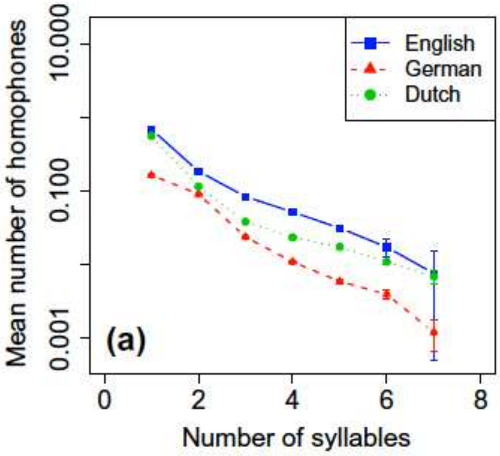
  What might this mean for the evolution of natural languages?

Piantadosi et al.—ambiguity has a communicative benefit

- If we are trying to maximise mutual information why has natural language evolved to be so ambiguous?
- Efficient communication systems will necessarily be globally ambiguous when context is informative about meaning.
- Notice that **ambiguity is not an issue in normal language** use and overloaded linguistic units are only ambiguous out of context:
  - Alice wanted **to** cry
  - Alice went **to** the garden
  - Alice saw **two** rabbits
  - Dinah saw some rabbits **too**.
- It is optimal to overload simple units for efficient transmission (we can assign the short efficient codes more than once and re-use them)

Piantadosi et al.—ambiguity has a communicative benefit

Some evidence to support the argument found in corpora: shorter words have more meanings



Implication: there must be enough information in the context to allow for the ambiguity in the simple units as well as any other noise in the channel.

Gibson et al.—a noisy channel can account for word order

Word order can provide context that is informative about meaning—this might account for observed word order in the world's languages

Most languages (out of 1,056 studied) exhibit one of two word orders:

- **subject-verb-object** (SVO) — 41% of languages

  the girl chases the rabbit (e.g. English)
- **subject-object-verb** (SOV) — 47% of languages

  the girl the rabbit chases (e.g. Japanese)

- For interest, 8% exhibit **verb-subject-object** (VSO) e.g. Welsh and Irish and 96% of languages have the subject before the object

Gibson et al.—noisy channel account of word order

- Experimental observations:
  - English speakers (SVO) were shown animations of simple events and asked to describe them using only gestures
  - For events in which a human acts on an inanimate object most participants use SOV despite being SVO speakers (e.g. *girl boy kicks*)
  - For events in which a human acts on another human most participants use SVO (e.g. *girl kicks boy*)
  - Preference in each case is around 70%
- Previous experiments show human preference for linguistic recapitulation of old information before introducing new information
- This might explain SOV gestures for SVO speakers—the verb is new information the people/objects are not.
- So why still use SVO for the animate-animate events? And why is English SVO?

Gibson et al.—noisy channel account of word order

Argument is that SVO ordering has a better chance of preserving information over a noisy channel.

- Consider the scenario of a girl kicking a boy
- Now let one of the nouns get lost in transmission.
- If the word order is SOV (*the girl the boy kicks*), the listener receives either:

  the girl kicks or *the boy kicks*
- If the word order is SVO (*the girl kicks the boy*) the listener receives either:

  the girl kicks or *kicks the boy*
- In the SVO case more information has made it through the noisy channel (preserved in the position relative to the verb)

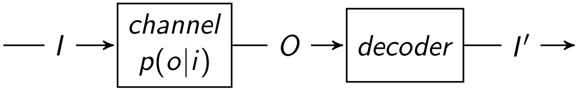
Gibson et al.—noisy channel account of word order

Further evidence for the argument is presented from the finding that there is a correlation between word order and case markings.

- Case marking means that words change depending on their syntactic function: e.g. *she* (subject), *her* (object)
- Case marking is rare in SVO languages (like English) and more common in SOV languages
- Suggestion is that when there are other information cues as to which noun is subject and which is object speakers can default to any natural preference for word order.

In Natural Language Processing, however, our starting point is **after** the evolutionary natural language encoding.

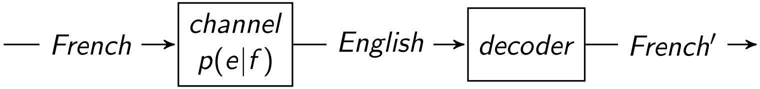
Noisy channel inspired an NLP problem-solving **framework**



- Many problems in NLP can be framed as trying to find the most likely input given an output: $I' = \underset{i}{\operatorname{argmax}}\, p(i|o)$
- $p(i|o)$ is often difficult to estimate directly and reliably, so use Bayes' theorem: $p(i|o) = \frac{p(o|i)p(i)}{p(o)}$
- Noting that $p(o)$ will have no effect on argmax function:
$$I' = \underset{i}{\operatorname{argmax}}\, p(i|o) = \underset{i}{\operatorname{argmax}}\, p(i)p(o|i)$$
- $p(i)$ is the probability of the input (a **language model**)
- $p(o|i)$ is the **channel probability** (the probability of getting an output from the channel given the input)

SMT is an intuitive (non-SOTA) example of noisy channel

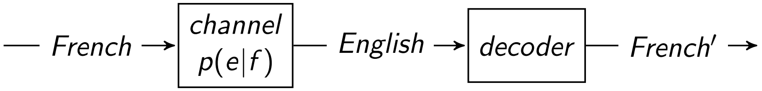
We want to translate a text from English to French:



- In statistical machine translation (SMT) noisy channel model assumes that original text is in French
- We pretend the original text has been through a noisy channel and come out as English (the word *hello* in the text is actually *bonjour* corrupted by the channel)
- To recover the French we need to decode the English:
  $f' = \underset{f}{\operatorname{argmax}}\, p(f|e) = \underset{f}{\operatorname{argmax}}\, p(f)p(e|f)$

SMT is an intuitive (now historic) example of noisy channel

Recover the French by decoding the English: $f' = \underset{f}{\operatorname{argmax}}\, p(f)p(e|f)$


$$\begin{tikzcd} \arrow[r, "\mathit{French}"] & \boxed{\begin{matrix}\mathit{channel}\\ p(e|f)\end{matrix}} \arrow[r, "\mathit{English}"] & \boxed{\mathit{decoder}} \arrow[r, "\mathit{French}'"] & {} \end{tikzcd}$$

- $p(f)$ is the **language model**.
- ensures **fluency** of the translation (usually a very large n-gram model)
- $p(e|f)$ is the **translation model**.
- ensures **fidelity** of the translation (derived from very large parallel corpora)

Noisy channel framework influenced many applications

Speech Recognition: recover word sequence by decoding the speech signal

$$\xrightarrow{\quad words \quad} \boxed{\begin{matrix} channel \\ p(s|w) \end{matrix}} \xrightarrow{\quad speech \quad} \boxed{decoder} \xrightarrow{\quad words' \quad}$$

$$words' = \underset{words}{\operatorname{argmax}}\, p(words)p(speech\_signal|words)$$

- $p(words)$ is the **language model** (n-gram model)
- $p(speech\_signal|words)$ is the **acoustic model**.

Noisy channel framework influenced many applications

Part-of-Speech Tagging:

$$\begin{tikzcd} {} \arrow[r, "tags"] & \boxed{\begin{array}{c} channel \\ p(w|t) \end{array}} \arrow[r, "words"] & \boxed{decoder} \arrow[r, "tags'"] & {} \end{tikzcd}$$

$$tags' = \underset{tags}{\text{argmax}}\, p(tags)p(words|tags)$$

Optical Character Recognition:

$$\begin{tikzcd} {} \arrow[r, "words"] & \boxed{\begin{array}{c} channel \\ p(e|w) \end{array}} \arrow[r, "errors"] & \boxed{decoder} \arrow[r, "words'"] & {} \end{tikzcd}$$

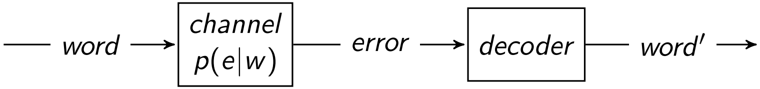
$$words' = \underset{words}{\text{argmax}}\, p(words)p(errors|words)$$

Spelling can be corrected using the noisy channel method

- There are two types of spelling error:
  - **non-word** errors: *Alcie* instead of *Alice*
  - **real-word** errors: *three* instead of *there*, *there* instead of *their*
- For illustration we will show how to use a noisy channel model to correct non-word errors
- Non-word correction is a significant problem:
  - 26%: of web queries *Wang et al.*
  - 13%: errors when asked to retype rather than backspace *Whitelaw et al.*
- **Detection** of non-word errors is easy (they fail to appear in a dictionary)
- The best candidate for **correction** will be the item that maximises the noisy channel equation.

Spelling can be corrected using the noisy channel method

Spelling correction:



$$word' = \underset{word}{\operatorname{argmax}}\, p(word)p(error|word)$$

- $p(word)$ can be obtained from a corpus
- $p(error|word)$ can be modelled using minimum text edit distance or minimum pronunciation distance (the probability of the edit)

Spelling can be corrected using the noisy channel method

- Damerau-Levenshtein is edit distance model that counts: *insertions*. *deletions*, *substitutions*, *transpositions*

error	candidate correction	corrected letters	error letters	type
acress	actress	t		deletion
acress	cress	-	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	-	s	insertion
acress	acres	-	s	insertion

- 80% of errors are within edit distance 1
- Almost all errors within edit distance 2

Spelling can be corrected using the noisy channel method

- $p(error|word)$ may be calculated from confusion tables created from error annotated training data
- e.g. substitution(x,w) confusion matrix

	a	b	c	d	e	...
a	0	0	7	1	342	...
b	0	0	9	9	2	...
c	6	5	0	16	0	...
d	1	10	13	0	12	...
e	338	0	3	11	0	...

- if misspelled word is $x = x_1, x_2 ... x_m$
- and corrected word is $w = w_1, w_2 ... w_n$
- If proposed edit at x_i is a substitution $p(x|w) = \frac{substitution(x_i, w_i)}{count(w_i)}$
- similar equations for a *deletion*, *insertion* and *transposition*

Spelling can be corrected using the noisy channel method

- For typo *acress* chosen word is

$$= \underset{word}{\operatorname{argmax}}\, p(word|error) = \underset{word}{\operatorname{argmax}}\, p(word)p(error|word)$$

candidate	corr	err	$x\|w$	$p(x\|w)$	p(w)	$p(x\|w)p(w)10^9$
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	.00078
caress	ca	ac	ac|ca	.00000164	.00000170	.0028
access	c	r	r|c	.000000209	.0000916	.019
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Noisy channel could be used to model comprehension

- For many natural language applications, noisy channel models have been surpassed by data hungry sequence-to-sequence neural models (more in *NLP* course next year)
- Natural language communication is an area where the model might still yield research insights
- Classic assumption in sentence processing: input to the parser is an error-free sequence of words (e.g. Hale and Yngve)
- This assumption is problematic because we are communicating across a noisy channel
- The ultimate interpretation of a sentence should depend on the proximity of plausible alternatives under the noise model
- This could be modelled in terms of insertions and deletions (just like spelling correction)...

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

You shall know a word by the **company** it keeps—Firth

Consider the following sentences about the *rabbit* in *Alice in Wonderland*:

- *Suddenly a white rabbit with pink eyes ran close by her.*
- *She was walking by the white rabbit who was peeping anxiously into her face.*
- *The rabbit actually took a watch out of its waistcoat pocket and looked at it.*
- *'Oh hush', the rabbit whispered, in a frightened tone.*
- *The white rabbit read out at the top of his shrill little voice the name Alice.*

We learn a lot about the rabbit from the words in the local context.

You shall know a word by the **company** it keeps—Firth

- So far, we have been discussing grammars with discrete alphabets and algorithms that have discrete symbols as input.
- Many Natural Language Processing tasks require some notion of similarity between the symbols.

  e.g. *The queen looked angry. Her majesty enjoyed beheading.*

  To understand the implication of these sentences we need to know that *the queen* and *her majesty* are similar ways of expressing the same thing.
- Instead of symbols we can represent a word by a collection of key words from its context (as a proxy to its meaning)

  e.g instead of rabbit we could use

  rabbit = {white, pink, eyes, voice, read, watch, waistcoat, ...}

You shall know a word by the **company** it keeps—Firth

- But which key words do we include in the collection?
- We could look at a $\pm n$-word context **window** around the **target** word.
- We could select (and weight) keywords based on their frequency in the window:

  rabbit = {the 56, white 22, a 17, was 11, in 10, it 9, said 8, and 8, to 7...}

- This would become a little more informative if we removed the function words:

  rabbit ={white 22, said 8, alice 7, king 4, hole 4, hush 3, say 3, anxiously 2...}

  queen ={said 21, king 6, shouted 5, croquet 4, alice 4, play 4, hearts 4, head 3... }

  cat ={said 19, alice 5, cheshire 5, sitting 3, think 3, queen 2, vanished 2, grin 2...}

- This is all just illustrative, we can of course, do this for all words (not just the characters)— called distributional semantics.

We can replace symbols with **vector representations**

- Two words can be expected to be semantically similar if they have similar word co-occurrence behaviour in texts.

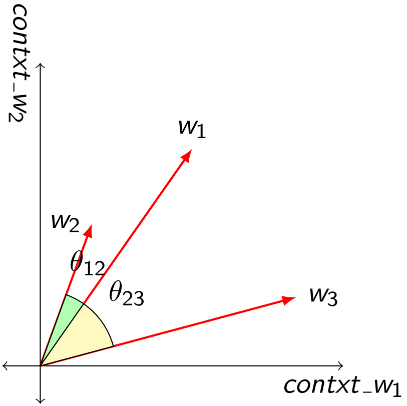
  e.g. in large amounts of general text we would expect *queen* and *monarch* to have similar word co-occurrences.
- Simple collections of context words don't help us easily calculate any notion of similarity.
- A trend in modern Natural Language Processing technology is to replace symbolic representation with a **vector representation**
- Every word is encoded into some vector that represents a point in a multi-dimensional word space.

	alice	croquet	grin	hurried	king	say	shouted	vanished
rabbit	7	0	0	2	4	3	0	1
queen	4	4	0	1	6	1	5	0
cat	5	1	2	0	0	0	0	2

We can replace symbols with **vector representations**

- Note that there is an issue with polysemy (words that have more than one meaning):
- E.g. we have obtained the following vector for cat:
  cat = [5, 1, 2, 0, 0, 0, 0 2]
- But cat referred to two entities in our story:
  I wish I could show you our cat Dinah
  I didn't know that Cheshire cats always grinned in fact I didn't know that cats could grin

- The vector provides the coordinates of point/vector in the multi-dimensional word space.
- Assumption: **proximity in word space** correlates with **similarity** in meaning
- Similarity can now be measured using distance measures such as Jaccard, Cosine, Euclidean...



- e.g. cosine similarity $\text{cosine}(\mathbf{v_1}, \mathbf{v_2}) = \frac{\mathbf{v_1} \cdot \mathbf{v_2}}{\|\mathbf{v_1}\| \|\mathbf{v_2}\|}$
- Equivalent to dot product of normalised vectors (not affected by magnitude)
- cosine is 0 between orthogonal vectors
- cosine is 1 if $v_1 = \alpha v_2$, where $\alpha > 0$

Automatically derived vectors will be very **large** and **sparse**

- In certain circumstances we might select dimensions **expertly**
- For general purpose vectors we want to simply count in a large collection of texts, the number of times each word appears inside a window of a particular size around the target word.
- This leads to very **large sparse** vectors (remember Zipf's law)
- There are an estimated 13 *million* tokens for the English language—we can reduce this a bit by removing (or discounting) function words, grouping morphological variants (e.g, *grin*, *grins*, *grinning*)
- Is there some k-**dimensional space** (such that $k << 13 million$) that is sufficient to encode the word meanings of natural language?
- Dimensions might hypothetically encode tense (past vs. present vs. future), count (singular vs. plural), and gender (masculine vs. feminine)...

It is possible to **reduce** the **dimensions** of the vector

To find reduced dimensionality vectors (usually called **word embeddings**)

- Loop over a massive dataset and accumulate word co-occurrence counts in some form of a large sparse matrix X (dimensions n x n where n is vocabulary size)
- Perform **Singular Value Decomposition** on X to get a USV^T decomposition of X.

$$\begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & X & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} = \begin{bmatrix} u_{11} & \vdots & \vdots & \vdots \\ \vdots & u_2 & \dots & u_n \\ u_{1n} & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} s_1 & 0 & 0 & \dots \\ 0 & s_2 & 0 & \dots \\ 0 & 0 & \ddots & \dots \\ \vdots & \vdots & \vdots & s_n \end{bmatrix} \begin{bmatrix} v_{1n} & \dots & v_{1n} \\ \dots & v_2 & \dots \\ \dots & \vdots & \dots \\ \dots & v_n & \dots \end{bmatrix}$$

It is possible to **reduce** the **dimensions** of the vector

- Note S matrix has diagonal entries only.
- Cut diagonal matrix at index k based on desired dimensionality (can be decided by desired percentage variance): $(\sum_{i=1}^{k} s_i)/(\sum_{i=1}^{n} s_i)$

$$\begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & X' & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} = \begin{bmatrix} u_{11} & \vdots & \vdots \\ \vdots & \dots & u_k \\ u_{1n} & \vdots & \vdots \end{bmatrix} \begin{bmatrix} s_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & s_k \end{bmatrix} \begin{bmatrix} v_{1n} & \dots & v_{1n} \\ \dots & \vdots & \dots \\ \dots & v_k & \dots \end{bmatrix}$$

- Use rows of U for the word embeddings.
- This gives us a k-dimensional representation of every word in the vocabulary.

It is possible to **reduce** the **dimensions** of the vector

Things to note:

- Need all the counts before we do the SVD reduction.
- The matrix is extremely **sparse** (most words do not co-occur)
- The matrix is very **large** ($\approx 10^6 x 10^6$)
- SVD is **quadratic**

Points of methodological variation:

- Due to Zipf distribution of words there is **large variance** in co-occurrence frequencies (need to do something about this e.g. discount/remove stop words)
- Refined approaches might **weight** the co-occurrence counts based on distance between the words

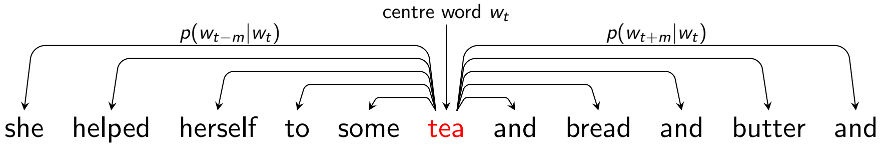
Predict models can be more efficient than **count** models

- **word2vec** is a **predict model**, in contrast to the distributional models already mentioned which are **count models**.
- Instead of computing and storing a large matrix from a very large dataset, use a model that learns **iteratively**, eventually encoding the probability of a word given its context.

- The parameters of the model are the word embeddings.
- The model is trained on a certain objective.
- At every iteration we run our model, evaluate the errors, and then adjust the model parameters that caused the error.

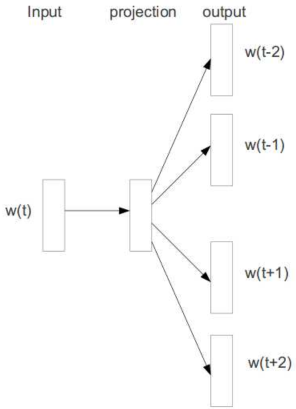
Predict models can be more efficient than **count** models

- There are two main word2vec architectures:
- **Continuous Bag of Words** CBOW: given some context word embeddings, predict the target word embedding.
- **Skip-gram**: given a target word embedding, predict the context word embeddings (below).



- **skip-gram** model predicts relationship between a centre word w_t and its context words: $p(context|w_t) = \ldots$
- Predict context word embeddings based on the target word embedding.
- A loss function is used to score the prediction (usually **cross-entropy** loss function).

  (Cross-entropy measures the information difference between the expected word embeddings and the predicted ones.)
- Adjust the word embeddings to minimise the loss function.
- Repeat over many positions of t in a very big language corpus.



Distributional models have improved NLP applications

In general, distributional models have had a positive impact on NLP and provided improvement over symbolic systems:

- There has been a change in state-of-the-art for some applications: (e.g. Google Translate)
- Multi-modal experiments have become more straightforward (by combining vector representations)
- But these models are statistical (need very large amounts of data and have to find a way to handle unseen words)
- There has been a lot of hype and not much work on the problems the distributional models can't solve.

Predict models versus **count** models

+ Predict models can be more efficient than count models because we can learn **iteratively** and don't have to hold statistics on the whole dataset.
– Need to **initialise** the word embeddings (several possible methods).
± The size of the embeddings is a chosen parameter of the system (usually a few hundred).
+ Predict models are learning structure **without hand-crafting** of features.
– Dimensionality of the embeddings are assumed to capture meaningful generalisations, but the dimensions are **not directly interpretable**.

Predict models versus **count** models

- After training, predict models are found to be equivalent to a count model with dimensionality reduction.
- Tuning hyper-parameters is a matter of much (often brute-force) experimentation.
- Predict models perform better than count models with dimensionality on some tasks (but perhaps due to tuning hyper-parameters).
- For some tasks count vectors without dimensionality reduction are the most effective.

Word embeddings can **correlate** with human **intuitions**

Researchers test their word embeddings against datasets of **human similarity judgements**:

- For a test set of words, participants rate word pairs for **relatedness** (e.g. Miller & Charles, Rubenstein & Goodenough)
- A rank of relatedness can be drawn up between items in the test set.
- A **rank correlation** between embeddings and human judgements can be calculated.
- Good embeddings have a correlation of 0.8 or better with the human judgements.

Reasoning may be possible based on word embeddings

- *Mikolov et al.* analogy puzzles:

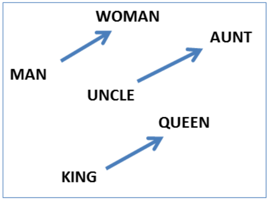
  Can we use word embeddings to solve puzzles such as:

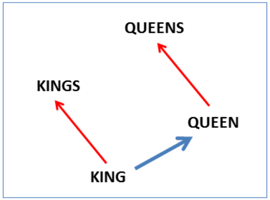
  man is to woman as king is to .... queen

- Can we do vector-oriented reasoning based on the offsets between words?

Reasoning may be possible based on word embeddings

- Derive the vector between the pair of words *man* and *woman* and then add it to *king*.
- The nearest word to the region of vector space that results will be the answer to the analogy.

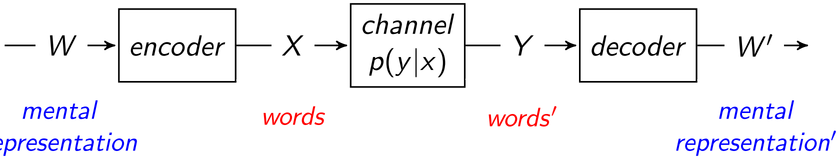




- *Mikolov* found that word2vec embeddings are good at capturing syntactic and semantic regularities in language, and that each relationship is characterised by a relation-specific vector offset.
- Note that the space is very sparse and that there are word pairs for which this does not work...

Relationship between embeddings and **brain activity**?

- Humans have the capacity to translate thoughts into words, and to infer others' thoughts from their words.
- There must be some mental representations of meaning that are mapped to language, but we have no direct access to these representations.



- Do word embeddings provide a model that successfully captures some aspects of our mental representation of meaning?

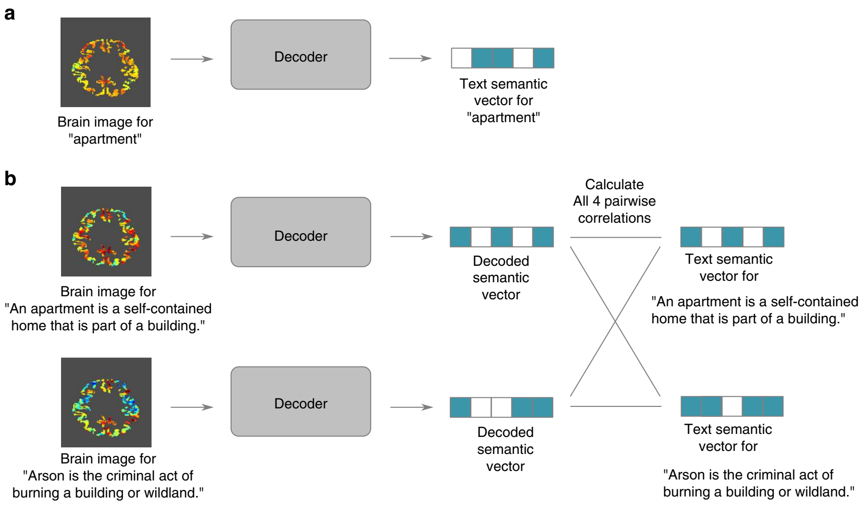
Relationship between embeddings and **brain activity**?

- Natural language appears to be a discrete symbolic system.
- The brain encodes information through continuous signals of activation.
- Language symbols are transmitted via continuous signals of sound/vision.

- *Pereira et al.* trained a system using brain imaging data and word embeddings.
- Demonstrated the ability to generalise to new meanings from limited imaging data.

`https://www.nature.com/articles/s41467-018-03068-4`

Relationship between embeddings and **brain activity**?


a
Brain image for "apartment"
Decoder
Text semantic vector for "apartment"
b
Brain image for "An apartment is a self-contained home that is part of a building."
Decoder
Decoded semantic vector
Calculate All 4 pairwise correlations
Text semantic vector for
"An apartment is a self-contained home that is part of a building."
Brain image for "Arson is the criminal act of burning a building or wildland."
Decoder
Decoded semantic vector
Text semantic vector for
"Arson is the criminal act of burning a building or wildland."

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 handout
- 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, Σ, 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 Σ is an ordered n-tuple of elements of Σ.

Σ^* denotes the set of all strings over Σ of finite length.

- If $\Sigma = \{a, b\}$ then ϵ, ba, bab, aab are examples of strings over Σ.
- 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 Σ any subset of Σ^* is a **formal language** over alphabet Σ.

[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 Σ^* 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 Σ^*:

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

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

DERIVATIONS

Given a set of axioms and rules for inductively defining a subset, $\mathcal{L}$, of Σ^*, 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:

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

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

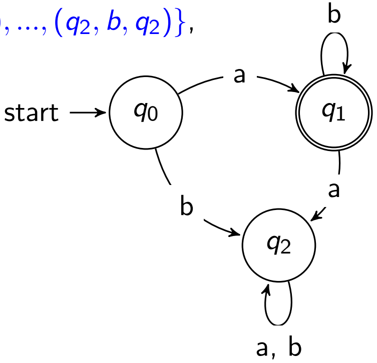
Recall that a language is regular if it is equal to the set of strings accepted by some deterministic finite-state automaton (DFA).
A DFA is defined as $M = (\mathcal{Q}, \Sigma, \Delta, s, \mathcal{F})$ where:

- $\mathcal{Q} = \{q_0, q_1, q_2...\}$ is a finite set of states.
- Σ is the alphabet: a finite set of transition symbols.
- $\Delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a function $\mathcal{Q} \times \Sigma \to \mathcal{Q}$ which we write as δ. 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

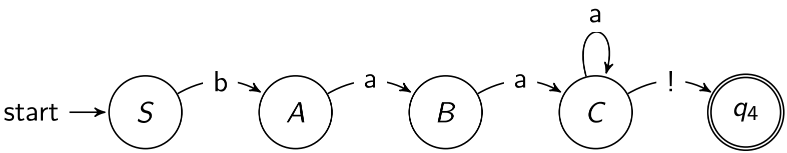
Reminder: **regular languages** are accepted by **DFAs**

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

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



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


$$\begin{tikzcd}
\text{start} \arrow[r] & S \arrow[r, bend left, "b"] & A \arrow[r, bend left, "a"] & B \arrow[r, bend left, "a"] & C \arrow[loop above, "a"] \arrow[r, bend left, "!"] & \boxed{q_4}
\end{tikzcd}$$

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

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

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:

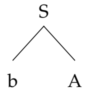
- $\mathcal{N} = \{\mathcal{Q}\}$ the non-terminals are the states of M
- $\Sigma = \Sigma$ the terminals are the set of transition symbols of M
- $S = s$ the starting symbol is the starting state of M
- $\mathcal{P} = q_i \rightarrow aq_j$ when $\delta(q_i, a) = q_j \in \Delta$
  or $q_i \rightarrow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

Strings are **derived** from production rules

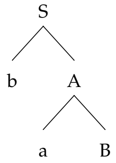
In order to derive a string from a grammar

- 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.

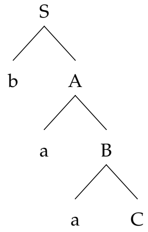
The rewrite rules derive the members of a language from their internal structure (or **phrase structure**)



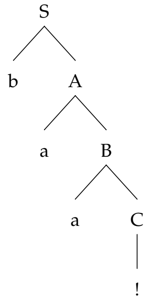
$S \rightarrow bA$



$A \rightarrow aB$



$B \rightarrow aC$



$C \rightarrow !$

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:

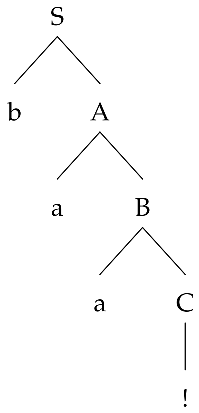
$$\begin{aligned} X &\rightarrow aY \\ X &\rightarrow a \end{aligned}$$

or alternatively, they can all be expressed as:

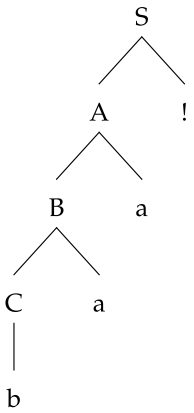
$$\begin{aligned} X &\rightarrow Ya \\ X &\rightarrow a \end{aligned}$$

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



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



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

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

A phrase structure grammar over an alphabet Σ 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 Σ: Terminal symbols (often lowercase letters) are elements of Σ 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 $\alpha \rightarrow \beta \in \mathcal{P}$

This is written $w \underset{G}{\Rightarrow} v$

A string in the language $\mathcal{L}(G)$ is a member of Σ^* 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 Σ^* 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?

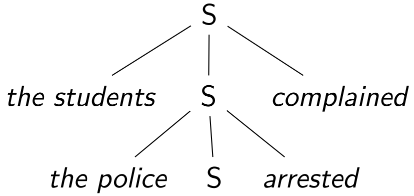
- We'd like fast algorithms for natural language processing applications.
- Potentially tells us something about human processing and acquisition (more in later lectures).

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

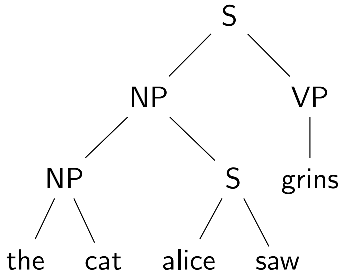
But for finite *n* we can still model English using a DFA—we can design the states to capture finite levels of embedding.

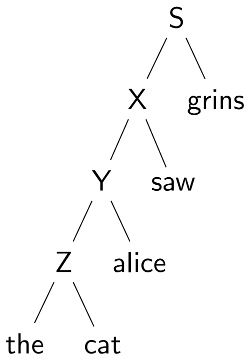
So are there any other reasons not to just use a regular grammar?

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



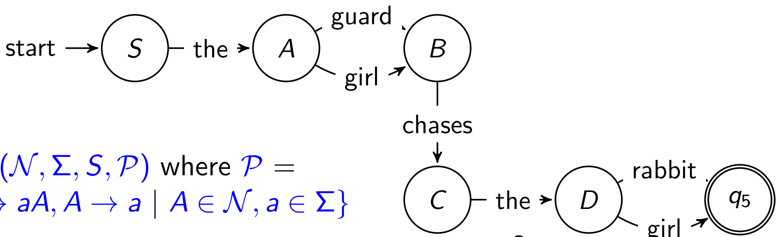


Formal Models of Language

Paula Buttery

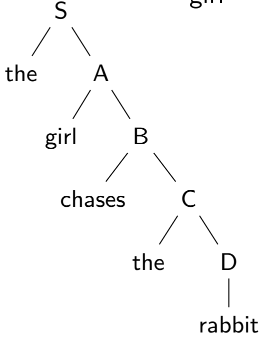
Dept of Computer Science & Technology, University of Cambridge

Regular grammars give us **linear** trees

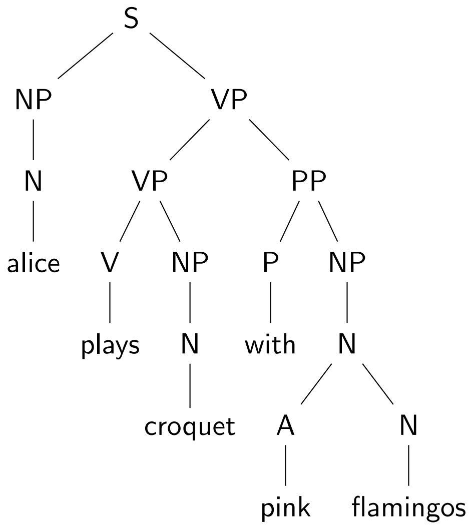


$G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where $\mathcal{P} = \{A \rightarrow aA, A \rightarrow a \mid A \in \mathcal{N}, a \in \Sigma\}$

- $\mathcal{N} = \{S, A, B, C, D, q_5\}$
- $\Sigma = \{the, girl, guard, \ldots\}$
- $\mathcal{S} = S$
- $\mathcal{P} = \{S \rightarrow the \;\; A,$
  $A \rightarrow guard \;\; B \mid girl \;\; B,$
  $B \rightarrow chases \;\; C,$
  $C \rightarrow the \;\; D,$
  $D \rightarrow girl \mid rabbit\}$



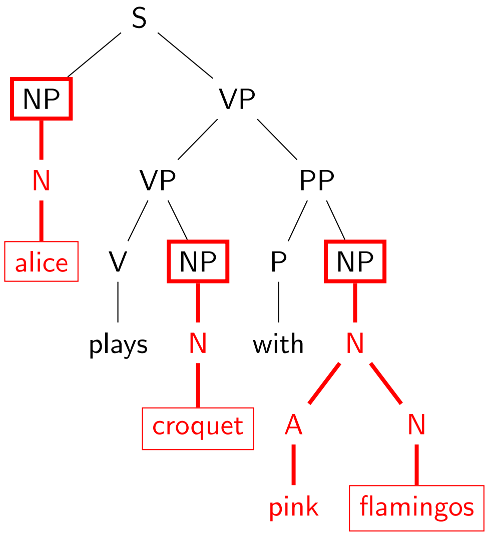
Context-free grammars capture **phrase structure**



$G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where
$\mathcal{P} = \{A \to \alpha \mid A \in \mathcal{N}, \alpha \in (\mathcal{N} \cup \Sigma)^*\}$

A brief excursion into linguistic terminology...

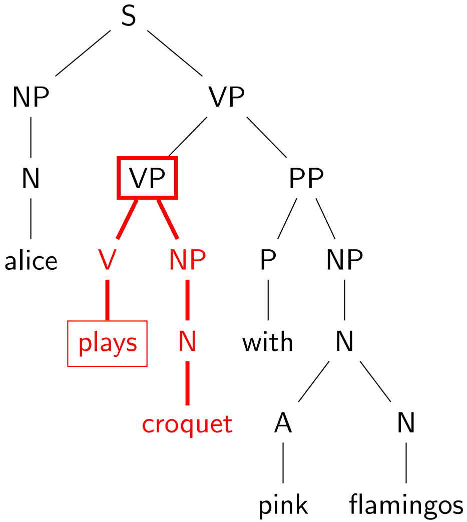
Context-free grammars capture **phrase structure**



When modelling natural language, linguists label the non-terminal symbols with names that encode the most *influential* word in the phrase. They call this influential word the **head**.

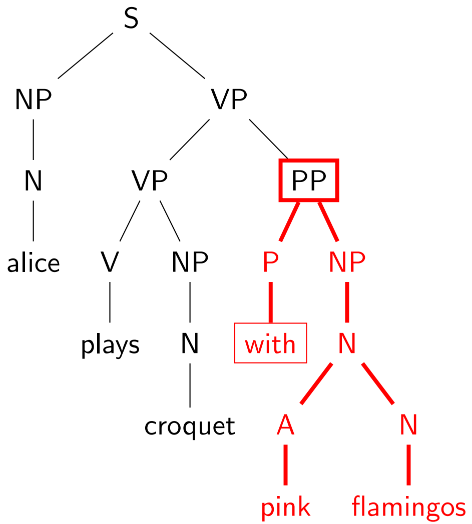
- noun phrases, *NP*, have a head noun

Context-free grammars capture **phrase structure**



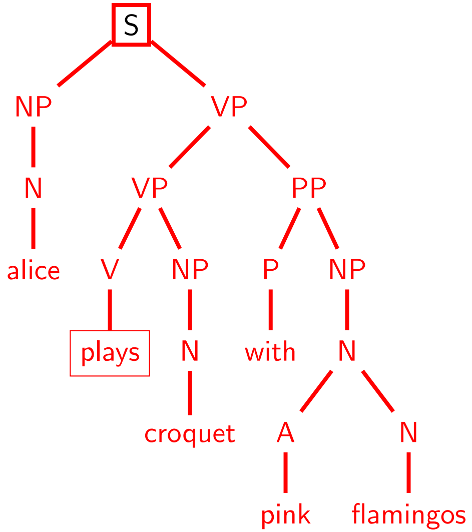
- verb phrases, *VP*, have a head verb

Context-free grammars capture **phrase structure**



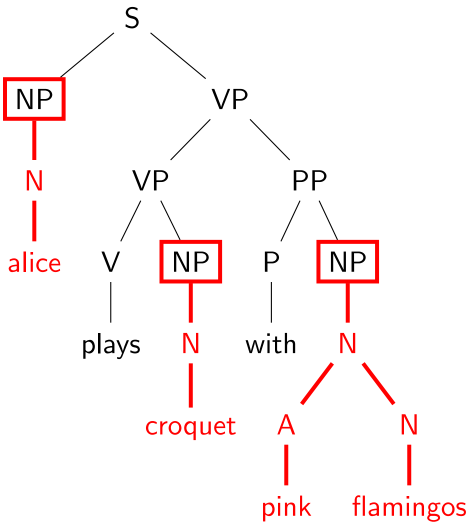
- prepositional phrases, *PP*, have a head preposition

Context-free grammars capture **phrase structure**



- the head of the whole string, S, is always the main verb

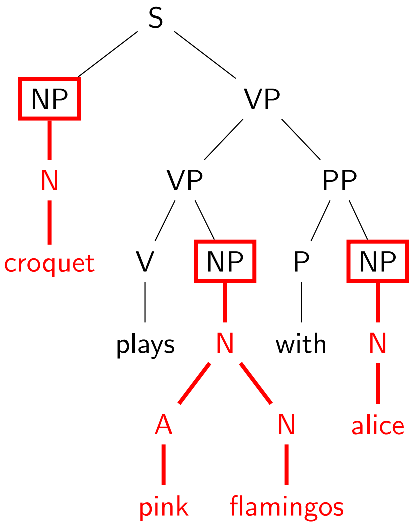
Context-free grammars capture **phrase structure**



Trees below nodes of the same type are interchangeable to yield another string in the language:

- $NP \rightarrow N$
- $N \rightarrow A\ N$
- $N \rightarrow alice|croquet|...$

Context-free grammars capture **phrase structure**



Trees below nodes of the same type are interchangeable to yield another string in the language:

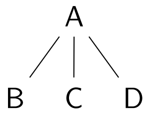
- $NP \rightarrow N$
- $N \rightarrow A\ N$
- $N \rightarrow alice | croquet | ...$

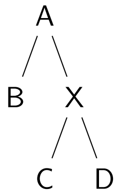
CFGs are often written in **Chomsky Normal Form**

Chomsky normal form: every production rule has the form, $A \rightarrow BC$, or, $A \rightarrow a$ where $A, B, C \in \mathcal{N}$, and, $a \in \Sigma$.

Conversion to Chomsky Normal Form

For every CFG there is a weakly equivalent CNF alternative.
$A \rightarrow BCD$ may be rewritten as the two rules, $A \rightarrow BX$, and, $X \rightarrow CD$.





CNF is a requirement for some parsing algorithms.

Context-free languages are accepted by **push down automata**

A PDA is defined as $M = (\mathcal{Q}, \Sigma, \Gamma, \Delta, s, \perp, \mathcal{F})$ where:

- $\mathcal{Q} = \{q_0, q_1, q_2 ...\}$ is a finite set of states.
- Σ is the input alphabet.
- Γ is the stack alphabet.
- $\Delta \subseteq (\mathcal{Q} \times (\Sigma \cup \epsilon) \times \Gamma) \times (\mathcal{Q} \times \Gamma^*)$ is a relation $(\mathcal{Q} \times (\Sigma \cup \epsilon) \times \Gamma) \to (\mathcal{Q} \times \Gamma^*)$ which we write as δ. Given $q \in \mathcal{Q}$, $i \in \Sigma$ and $A \in \Gamma$ then $\delta(q, i, A)$ returns (q', α), that is, a new state $q' \in \mathcal{Q}$ and replaces A at the top of the stack with $\alpha \in \Gamma^*$
- s is the starting state
- $\perp$ is the initial stack symbol
- $\mathcal{F}$ is the set of all end states

Moving from one state to the next we may **push** or **pop**

- in state q_x on encountering transition symbol a transition to state q_y popping A from the top of the stack and pushing B onto the stack

$$\begin{tikzcd} q_x \arrow[r, bend left, "a : A/B"] & q_y \end{tikzcd}$$

BEFORE	AFTER
A	B
z_0	z_0

- in state q_x transition to state q_y pushing A onto the stack

$$\begin{tikzcd} q_x \arrow[r, bend left, "\epsilon : \epsilon/A"] & q_y \end{tikzcd}$$

BEFORE	AFTER
z_0	A
	z_0

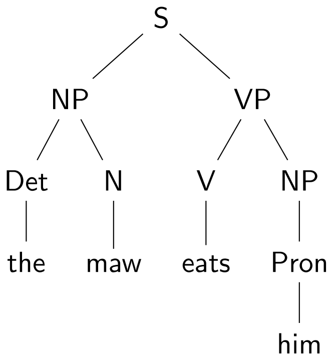
- in state q_x transition to state q_y popping A from the stack

$$\begin{tikzcd} q_x \arrow[r, bend left, "\epsilon : A/\epsilon"] & q_y \end{tikzcd}$$

BEFORE	AFTER
A	z_0
z_0	

A toy context-free grammar

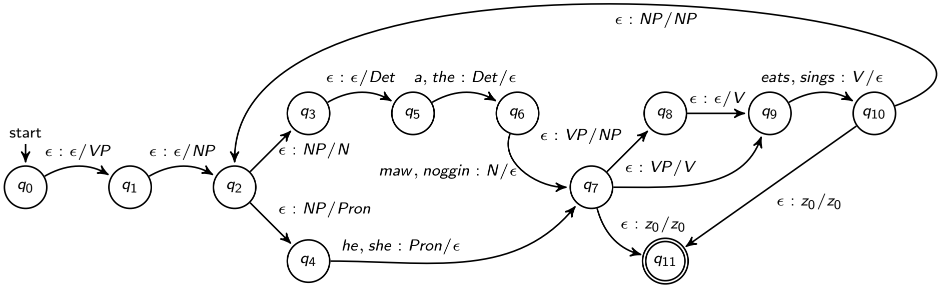
$$
\begin{array}{rcl}
\text{S} & \rightarrow & \text{NP VP} \\
\text{NP} & \rightarrow & \text{Pron} \\
\text{NP} & \rightarrow & \text{Det N} \\
\text{VP} & \rightarrow & \text{V} \\
\text{VP} & \rightarrow & \text{V NP} \\
\text{Det} & \rightarrow & \{\text{a, the}\} \\
\text{N} & \rightarrow & \{\text{maw, noggin, ...}\} \\
\text{Pron} & \rightarrow & \{\text{he, she, him, her}\} \\
\text{V} & \rightarrow & \{\text{eats, sings}\}
\end{array}
$$



Recognising a string with a push down automaton

S	$\rightarrow$	NP VP
NP	$\rightarrow$	Pron
NP	$\rightarrow$	Det N
VP	$\rightarrow$	V
VP	$\rightarrow$	V NP

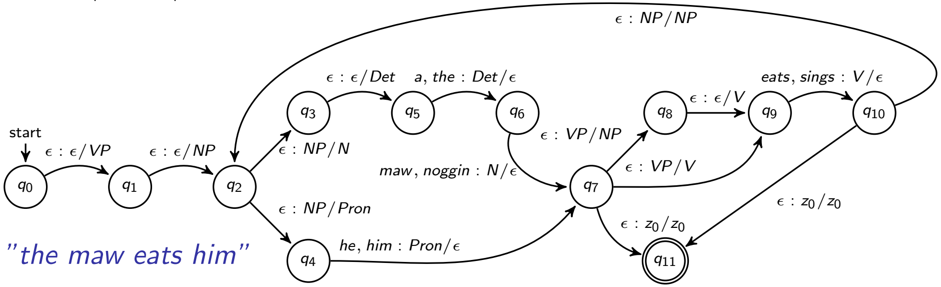
Det	$\rightarrow$	{a,the}
N	$\rightarrow$	{maw, noggin, ...}
Pron	$\rightarrow$	{he, him, her}
V	$\rightarrow$	{eats, sings}



Is *'the maw eats him'* a string in the language?

the	q_0	z_0			
the	q_0-q_1	VP	z_0		
the	q_1-q_2	NP	VP	z_0	
the	q_2-q_3	N	VP	z_0	
the	q_3-q_5	Det	N	VP	z_0
maw	q_5-q_6	N	VP	z_0	
eats	q_6-q_7	VP	z_0		
eats	q_7-q_8	NP	z_0		
eats	q_8-q_9	V	NP	z_0	
him	q_9-q_{10}	NP	z_0		
him	q_{10}-q_2	NP	z_0		
him	q_2-q_4	Pron	z_0		
him	q_4-q_7	z_0			
ϵ	q_7-q_{11}	z_0			

S	$\rightarrow$	NP VP
NP	$\rightarrow$	Pron
NP	$\rightarrow$	Det N
VP	$\rightarrow$	V
VP	$\rightarrow$	V NP
Det	$\rightarrow$	{a,the}
N	$\rightarrow$	{maw, noggin, ...}
Pron	$\rightarrow$	{he, him, her}
V	$\rightarrow$	{eats, sings}



"the maw eats him"

Can context-free grammars model natural language?

Cross Serial Dependencies

A small number of languages exhibit strings of the form

$$\begin{tikzcd}
noun_1 \arrow[rrrr, bend right] & noun_2 \arrow[rrrr, bend right] & \cdots & noun_n \arrow[rrrr, bend right] & verb_1 & verb_2 & \cdots & verb_n
\end{tikzcd}$$

Zurich dialect of Swiss German

mer d'chind em Hans es huus haend wele laa hälfe aastriiche.

we the children Hans the house have wanted to let help paint.

we have wanted to let the children help Hans paint the house

Such expressions, i.e. of the form $/a^n b^m c^n d^m/$, may not be derivable by a context-free grammar.

mer d'chind^n em Hans^m es huus haend wele laa^n hälfe^m aastriiche.

$\rightarrow /wa^n b^m x c^n d^m y/$

Use the **pumping lemma** to prove **not** context-free

The pumping lemma for context-free languages (CFLs) is used to show that a language is not context-free. The pumping lemma property for CFLs is:

All $w \in \mathcal{L}$ with $|w| \geq k$ can be expressed as a concatenation of five strings, $w = u_1 y u_2 z u_3$, where u_1, y, u_2, z and u_3 satisfy:

- $|yz| \geq 1$ (i.e. we cannot have $y = \epsilon$ and $z = \epsilon$)
- $|yu_2z| \leq k$
- for all $n \geq 0$, $u_1 y^n u_2 z^n u_3 \in \mathcal{L}$
  (i.e. $u_1 u_2 u_3 \in \mathcal{L}$, $u_1 y u_2 z u_3 \in \mathcal{L}$, $u_1 yy u_2 zz u_3 \in \mathcal{L}$ etc.)

To prove that Swiss German is not context-free, similar proof as for **centre embeddings** (last lecture). Except that you need to remember that:
$\mathcal{L}_{reg1} \cap \mathcal{L}_{cfg1} = \mathcal{L}_{cfg2}$

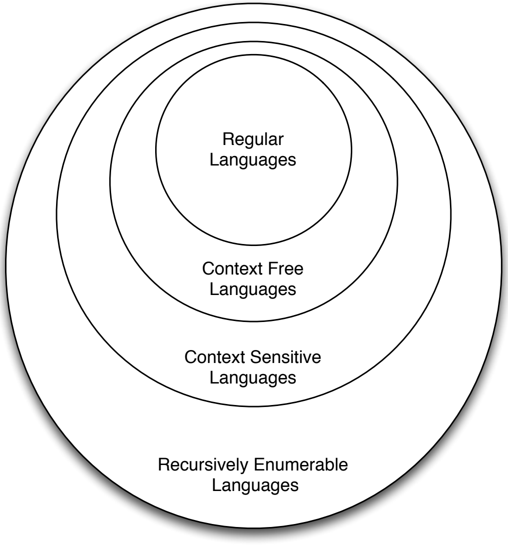
Are **CSGs** required to model natural languages?

Remember the **complexity** of a language class was defined in terms of the **recognition problem**.

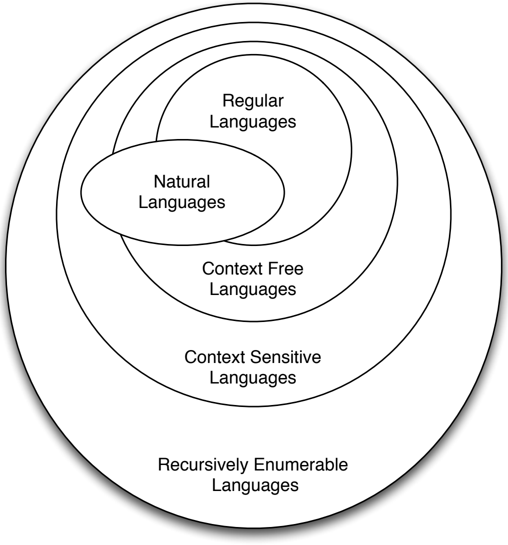
Type	Language Class	Complexity	machine
3	regular	$O(n)$	DFA
2	context-free	$O(n^c)$	PDA
1	context-sensitive	$O(c^n)$	LBA
0	recursively enumerable	*undecidable*	Turing

- Modelling natural languages using context-sensitive grammars is very expensive. In practice we don't have to because only very limited constructions are not captured by context-free grammars.
- However, it is still fun to place a limit on the complexity of natural languages — we are not limited to discussing language classes only in terms of the Chomsky hierarchy.

We are not limited to the **Chomsky hierarchy**



We are not limited to the **Chomsky hierarchy**



The mildly context-sensitive grammars

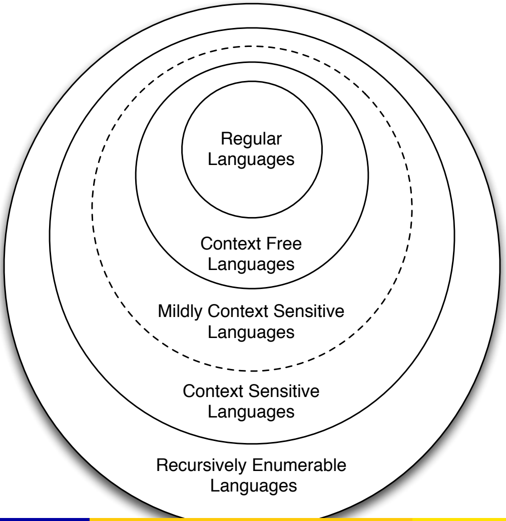
Joshi defined a class of languages that is more expressive than context-free languages, less expressive than context-sensitive languages and also sits neatly in the Chomsky hierarchy.

MILDLY CONTEXT-SENSITIVE languages

An abstract language class has the following properties:

- it includes all the context-free languages;
- members of the languages in the class may be recognised in polynomial time;
- the languages in the class account for all the constructions in natural language that context-free languages fail to account for (such as cross-serial dependencies).

Mildly CSGs are a **subset** of CSGs that account for natural language


Regular Languages
Context Free Languages
Mildly Context Sensitive Languages
Context Sensitive Languages
Recursively Enumerable Languages

In **Tree Adjoining Grammars** trees are rewritten as trees.

In phrase structure grammar symbols were rewritten with other symbols

In **Tree Adjoining Grammars** trees are rewritten as other trees.

The grammar consists of sets of two types of elementary tree:

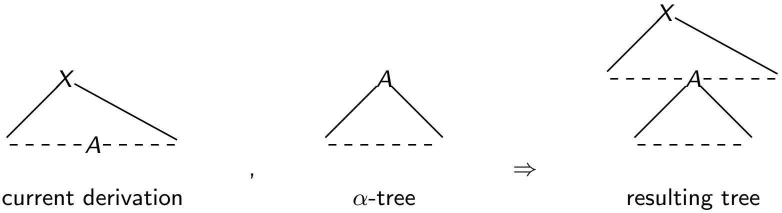
- **initial trees** or α trees
- **auxiliary trees** or β trees

A derivation is the result of recursive composition of elementary trees via one of two operations:

- **substitution**
- **adjunction**.

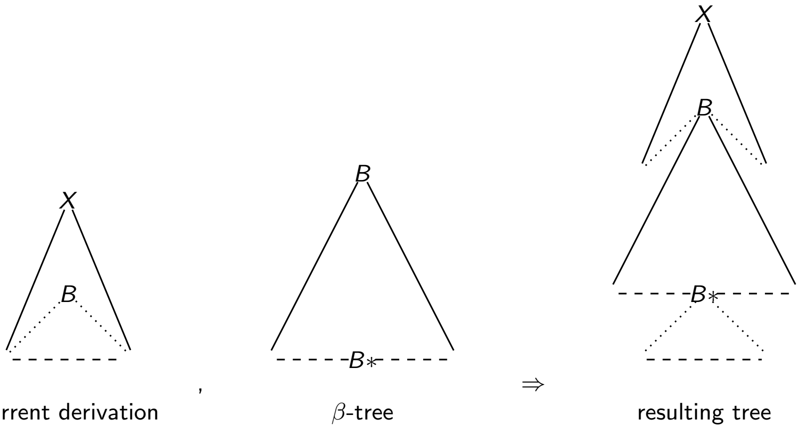
Tree adjoining grammars: the **substitution** operation

- SUBSTITUTION: a substitution may occur when a non-terminal leaf (that is, some $A \in \mathcal{N}$) of the current derivation tree is replaced by an α-tree that has A at its root.



Tree adjoining grammars: the **adjunction** operation

- ADJUNCTION:an adjunction may occur when an internal non-terminal node of the current derivation (some $B \in \mathcal{N}$) tree is replaced by a β tree that has a B at its root and *foot*.



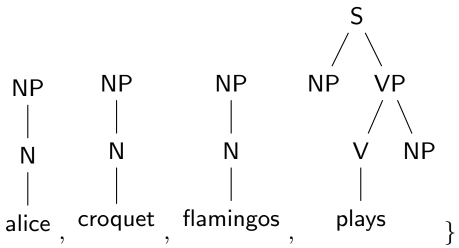
Tree adjoining grammars: definition

- $\mathcal{N}$ is the set of non-terminals
- Σ is the set of terminals
- S is a distinguished non-terminal $S \in \mathcal{N}$ that will be the root of complete derivations
- $\mathcal{I}$ is a set of **initial trees** (also known as α trees). Internal nodes of an α tree are drawn from $\mathcal{N}$ and the leaf nodes from $\Sigma \cup \mathcal{N} \cup \epsilon$.
- $\mathcal{A}$ is a set of **auxiliary trees** (also know as β trees). Internal nodes of an β-tree are drawn from $\mathcal{N}$ and the leaf nodes from $\Sigma \cup \mathcal{N} \cup \epsilon$. One leaf of a β-tree is distinguished as the **foot** and will be the same non-terminal as at its root (the foot is often indicated with an asterisk).

Tree adjoining grammars: natural language example

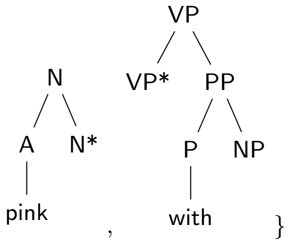
$G_{tag} = (\mathcal{N}, \Sigma, S, \mathcal{I}, \mathcal{A})$ where:

$\mathcal{I} = \{$



$\}$

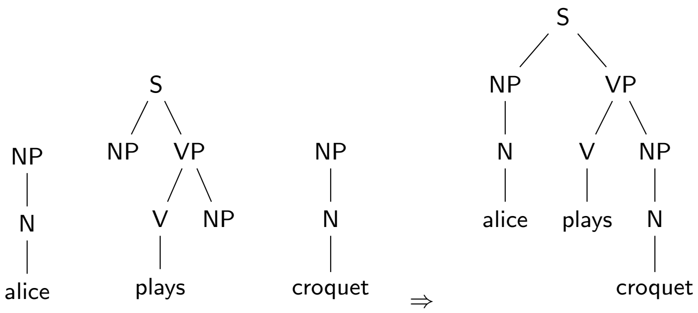
$\mathcal{A} = \{$



$\}$

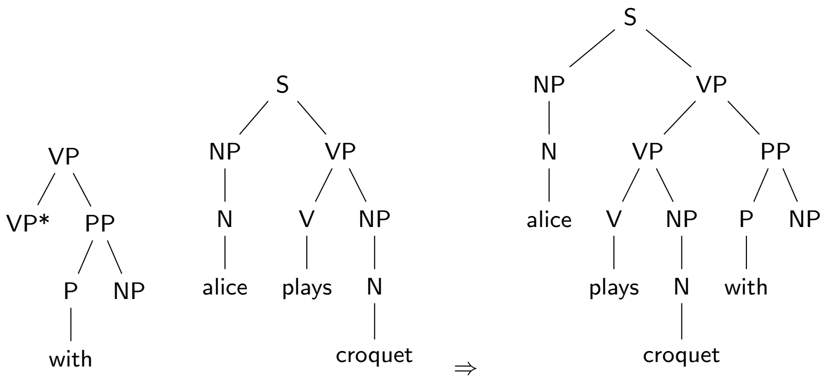
Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



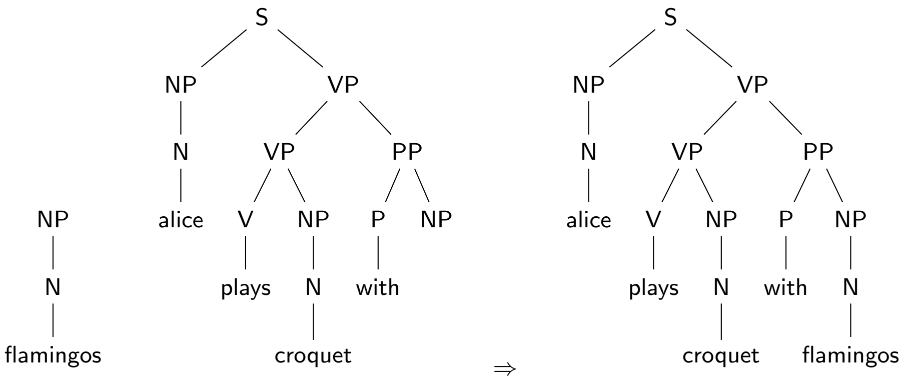
Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



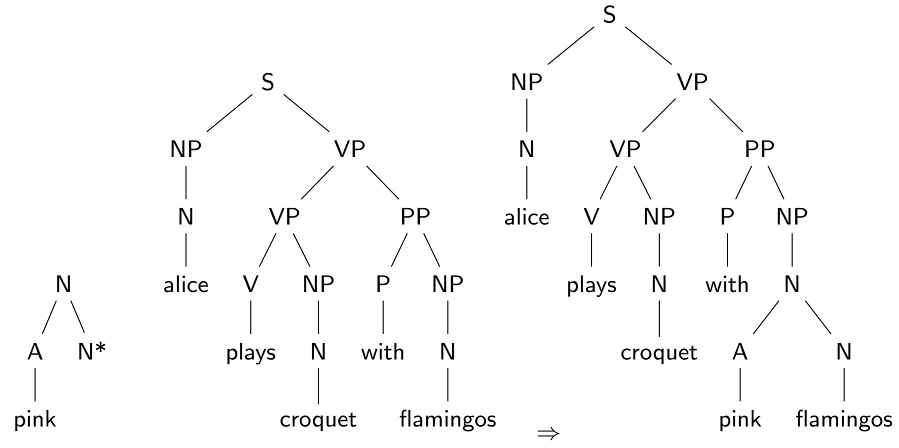
Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



Tree adjoining grammars: natural language example

Deriving: *Alice plays croquet with pink flamingos*



Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Shift-reduce parsers are useful for **deterministic** languages

LR(k) Shift-reduce parsers are most useful for recognising the strings of **deterministic** languages (languages where no string has more than one analysis) which have been described by an unambiguous grammar.

Quick reminder:

- The parsing algorithm has two actions: SHIFT and REDUCE
- Initially the input string is held in the buffer and the stack is empty.
- Symbols are **shifted** from the buffer to the stack
- When the top items of the stack match the RHS of a rule in the grammar then they are **reduced**, that is, they are replaced with the LHS of that rule.
- k refers to the look-ahead.

Reminder: shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\,C, \\
&\quad C \rightarrow c\,D, \\
&\quad D \rightarrow d\}
\end{aligned}$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE
AbC		REDUCE
AB		REDUCE
S		

Reminder: properties of **Deterministic** CFLs

Deterministic context-free languages:

- are a proper subset of the context-free languages
- are accepted by deterministic push-down automata
- can be modelled by an unambiguous grammar
- can be parsed in linear time
- parser can be automatically generated from the grammar

CFGs used to model natural language are **not** deterministic

- Natural languages (with all their inherent ambiguity) are not well suited to shift-reduce parsers which operate deterministically recognising a single derivation without backtracking
- However, natural language parsing can be achieved deterministically by selecting parsing actions using a machine learning classifier (more on this next time).
- All CFGs (including those exhibiting ambiguity) can be recognised in polynomial time using chart parsing algorithms.

Ambiguous grammars derive a **parse forest**

Number of binary trees is proportional to the Catalan number

$$\texttt{Num of trees for sentence length n} = \prod_{k=2}^{n-1} \frac{(n-1)+k}{k}$$

sentence length	number of trees	sentence length	number of trees
3	2	8	429
4	5	9	1430
5	14	10	4862
6	42	11	16796
7	132	12	58786

We need parsing algorithms that can efficiently store the parse forest and not derive shared parts of tree more than once—chart parsers

The **Earley parser** is a **chart parsing** algorithm

The **Earley parser** is a dynamic programming algorithm that records partial derivations in a CHART (a table).

- Uses a top-down approach to explore the whole search space, recovering multiple derivations where they exist.
- The progress of the algorithm is encoded in something called a **dotted rule** or **progress rule**:

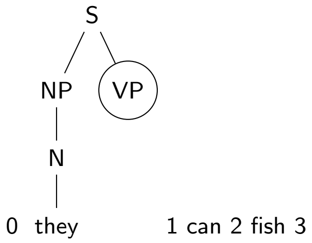
  $A \rightarrow {\bullet}\alpha\beta \mid \alpha{\bullet}\beta \mid \alpha\beta{\bullet}$ where $A \rightarrow \alpha\beta \in \mathcal{P}$.
- Rules of the form $A \rightarrow {\bullet}\alpha\beta$ have all symbols still to be explored;
- Rules of the form $A \rightarrow \alpha\beta{\bullet}$ have been completely *used up* deriving a portion of the string.

Partial derivations are recorded in a **chart**

- By convention, each row of the chart is referred to as an **edge**.
- An edge in the chart records a dotted rule, and its **span**.
- The span refers to the portion of the input string which is consistent with the partial tree.
- If we wish to discover the structure of a parse, an edge must also record the derivation **history** of the immediately previous partial tree(s) that made the current partial tree possible.

Partial derivations are recorded in a **chart**

For an illustration, consider the partial tree below which has been derived when attempting to parse the sentence *they can fish*:



ID	RULE	[start, end]	HIST
⋮			
e_i	$S \rightarrow NP_\bullet VP$	$[0, 1]$	h_k

Partial derivations are recorded in a **chart**

For input string $u = a_1 ... a_n$ and grammar $G_{cfg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$:

An edge $A \rightarrow \alpha_\bullet \beta\ [i, j]$ is added if

$S \underset{G^*}{\Longrightarrow} a_1 ... a_i A \gamma$ where γ are symbols in u yet to be parsed

and $\alpha \underset{G^*}{\Longrightarrow} a_{i+1} ... a_j$

- The chart is **initialised** with the edge $S \rightarrow {}_\bullet \alpha\beta\ [0, 0]$;
- The input string $u = a_1 ... a_n$ is **recognised** when we add the edge $S \rightarrow \alpha\beta_\bullet\ [0, n]$.

Today's toy grammar

We will parse the sentence *they can fish* using $G_{cfg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where:

$$\begin{array}{rcl}
\mathcal{N} & = & \{S, NP, VP, PP, N, V, P\} \\
\Sigma & = & \{can, fish, in, rivers, they \ldots\} \\
S & = & S \\
\mathcal{P} & = & \{S \rightarrow NP\ VP \\
 & & NP \rightarrow N\ PP \mid N \\
 & & PP \rightarrow P\ NP \\
 & & VP \rightarrow VP\ PP \mid V\ VP \mid V\ NP \mid V \\
 & & N \rightarrow can \mid fish \mid rivers \mid \ldots \\
 & & P \rightarrow in \mid \ldots \\
 & & V \rightarrow can \mid fish \mid \ldots\ \}
\end{array}$$

0 they 1 can 2 fish 3

Initialise the chart

The chart is initialised with $S \rightarrow {}_\bullet \alpha\beta\ [0,0]$.

ID	RULE	[start, end]	HIST
e_0	$S \rightarrow {}_\bullet\ NP\ VP$	$[0,0]$	

In rule induction notation:

$$\frac{}{S \rightarrow {}_\bullet \alpha\beta\ [0,0]}\ \text{(induction step)}$$

Three steps of the Earley parser: **predict** step

This step adds new edges to the chart and can be thought of as expanding tree nodes in the top-down derivation.

ID	RULE	[start, end]	HIST
e_0	$S \rightarrow {\bullet}\ NP\ VP$	$[0,0]$	
e_1	$NP \rightarrow {\bullet}\ N$	$[0,0]$	
e_2	$NP \rightarrow {\bullet}\ N\ PP$	$[0,0]$	

In rule induction notation:

$$\frac{A \rightarrow \alpha_{\bullet} B\beta\ [i,j]}{B \rightarrow {\bullet}\gamma\ [j,j]}\ \text{(predict step) where } B \rightarrow \gamma \in \mathcal{P}$$

Three steps of the Earley parser: **scan** step

This step allows us to check if we have a node that is consistent with the input sentence. If the input sentence is $u = a_1 ... a_n$ we can add a new edge if $A \rightarrow {\bullet} a\ [i, j-1]$ and $a = aj$.

ID	RULE	[start, end]	HIST
e_0	$S \rightarrow \bullet\ NP\ VP$	$[0,0]$	
e_1	$NP \rightarrow \bullet\ N$	$[0,0]$	
e_2	$NP \rightarrow \bullet\ N\ PP$	$[0,0]$	
e_3	$N \rightarrow they\ \bullet$	$[0,1]$	

In rule induction notation:

$$\frac{A \rightarrow {\bullet} a\ [i, j-1]}{A \rightarrow a {\bullet}\ [i, j]} \text{ (scan step) when } a = a_j$$

Three steps of the Earley parser: **scan** step

- For natural language sentence parsing tasks, Σ can be the finite set of words in the language (a very large set).
- when carrying out the predict step from a rule like $NP \rightarrow \bullet N$ we would end up adding a new edge for every *noun* in the language.
- To save us from creating all these edges we can privilege a set of the non-terminals and perform a forward look-up of the next a_j to see whether it will be consistent.
- In our example this set would be $\mathcal{N}_{PofS} = \{N, V, P\}$, that is, all the non-terminal symbols that represent the **parts-of-speech** of the language (such as *nouns*, *verbs*, *adjectives*...).
- During the scanning step, we find edges containing non-terminals in $\mathcal{N}_{PofS}$ with a dot on their LHS and check if the upcoming word is consistent with the part-of-speech. Iff it is consistent then we add an edge to the chart.

Three steps of the Earley parser: **complete** step

This step propagates fully explored tree nodes in the chart.

ID	RULE	[start, end]	HIST
e_0	$S \to \bullet\ NP\ VP$	$[0, 0]$	
e_1	$NP \to \bullet\ N$	$[0, 0]$	
e_2	$NP \to \bullet\ N\ PP$	$[0, 0]$	
e_3	$N \to they\ \bullet$	$[0, 1]$	
e_4	$NP \to N\ \bullet$	$[0, 1]$	e_3
e_5	$NP \to N\ \bullet\ PP$	$[0, 1]$	e_3
e_6	$S \to NP\ \bullet\ VP$	$[0, 1]$	e_4

In rule induction notation:

$$\frac{A \to \alpha_\bullet B\beta\ [i,k] \qquad B \to \gamma_\bullet\ [k,j]}{A \to \alpha B_\bullet \beta\ [i,j]}\ \text{(complete step)}$$

ID	RULE	[start, end]	HIST	word n
e_0	$S \rightarrow \bullet\ NP\ VP$	$[0,0]$		**word 0**
e_1	$NP \rightarrow \bullet\ N$	$[0,0]$		**word 1**
e_2	$NP \rightarrow \bullet\ N\ PP$	$[0,0]$		
e_3	$N \rightarrow they \bullet$	$[0,1]$		
e_4	$NP \rightarrow N \bullet$	$[0,1]$	(e_3)	
e_5	$NP \rightarrow N \bullet\ PP$	$[0,1]$	(e_3)	
e_6	$S \rightarrow NP \bullet\ VP$	$[0,1]$	(e_4)	
e_7	$PP \rightarrow \bullet\ P\ NP$	$[1,1]$		**word 2**
e_8	$VP \rightarrow \bullet\ V$	$[1,1]$		
e_9	$VP \rightarrow \bullet\ V\ NP$	$[1,1]$		
e_{10}	$VP \rightarrow \bullet\ V\ VP$	$[1,1]$		
e_{11}	$VP \rightarrow \bullet\ VP\ PP$	$[1,1]$		
e_{12}	$V \rightarrow can \bullet$	$[1,2]$		
e_{13}	$VP \rightarrow V \bullet$	$[1,2]$	(e_{12})	
e_{14}	$VP \rightarrow V \bullet\ NP$	$[1,2]$	(e_{12})	
e_{15}	$VP \rightarrow V \bullet\ VP$	$[1,2]$	(e_{12})	
e_{16}	$S \rightarrow NP\ VP \bullet$	$[0,2]$	(e_4, e_{13})	
e_{17}	$VP \rightarrow VP \bullet\ PP$	$[1,2]$	(e_{13})	
e_{18}	$NP \rightarrow \bullet\ N$	$[2,2]$		**word 3**
e_{19}	$NP \rightarrow \bullet\ N\ PP$	$[2,2]$		
e_{20}	$VP \rightarrow \bullet\ V$	$[2,2]$		
e_{21}	$VP \rightarrow \bullet\ V\ NP$	$[2,2]$		

The run time of the Earley parser is **polynominal**

- The **complete** step dominates run time $O(n^2)$
- Running time of the Earley parser is $O(n^3)$
- Run time is reduced in various scenarios, e.g. when the grammar is unambiguous or left-recursive .[1]

So what makes a sentence **complex** for a human to process?

[1] See `https://homepages.cwi.nl/~jve/lm2005/earley.pdf` for a full discussion

The term **complexity** can be used to describe human processing difficulty

The term **complexity** is also used to describe the perceived human processing difficulty of a sentence: work in this area is generally referred to as **computational psycholinguistics**.

Complexity within this domain can refer to:

- the **time and space requirements** of the algorithm that your brain is posited to require while processing a sentence.
- the **information theoretic content** of the sentence itself in isolation from the human processor (more in later lectures on this)

The term **complexity** can be used to describe human processing difficulty

Traditional work in this area has looked mainly at parsing algorithms to discover whether they exhibit properties that correlate with measurable predictors of complexity in human linguistic behaviour.

Two general assumptions are made in this work:

1) Sentences will take **longer to process** if they are more complicated for the human parser.
    - Processing time is usually measured as the time it takes to read a sentence.
    - This can be done with eye-tracking machines which also identify whether the subject reread any parts of a sentence.
    - Researchers also use neuro-imaging techniques (MEG, fMRI)

The term **complexity** can be used to describe human processing difficulty

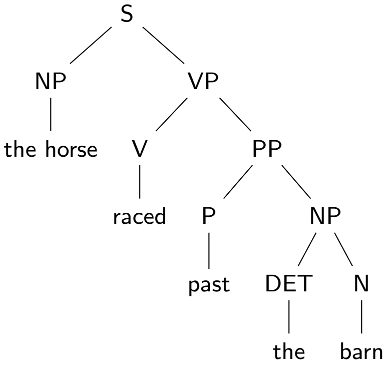
2) Sentences **will not occur frequently in the spoken language** if they are complicated to produce or comprehend.
   - Frequencies are calculated by counting constructions of interest in spoken language corpora.

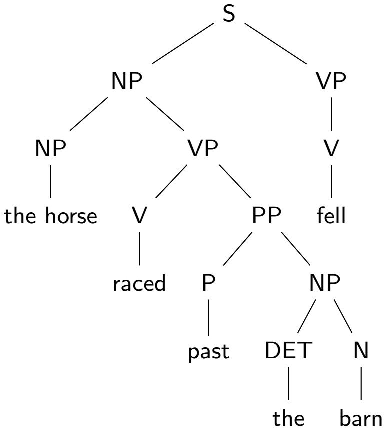
The assumption then is that one (or both) of the two measurements of perceived complexity above will correlate with time and space requirements of the parsing algorithm.

What makes a sentence expensive to process?

Example: long distance syntactic dependencies (e.g. garden-paths)

- The horse raced past the barn
- The horse raced past the barn fell—comparatively slow reading time





Hale — Earley parser as a model of sentence processing

Using predictability as a measure of difficulty

- The cognitive effort associated with a word in a sentence can be measured by the word's **surprisal** (negative log conditional probability): $\log \frac{1}{P(w_i|w_{1 \cdots i-1})}$ (more on this in later lectures)
- The suggestion is that probabilistic context-free grammars (PCFGs) can be used to model human language processing.

  $G_{pcfg} = (\Sigma, \mathcal{N}, S, \mathcal{P}, q)$ where q is a mapping from rules in $\mathcal{P}$ to a probability and $\sum\limits_{A \rightarrow \alpha \in \mathcal{P}} q(A \rightarrow \alpha) = 1$
- A probabilistic Earley parser is used as a model of online eager sentence processing.

Hale — Earley parser as a model of sentence processing

- The probabilistic Earley parser computes all parses of its input.
- As a psycholinguistic theory it is one of **total parallelism** (as opposed to a reanalysis theory)
- Calculate **prefix probabilities** i.e. probabilities of partially derived trees.
- Hypothesis is that the cognitive effort expended to parse a given prefix is **proportional to the total probability** of all the structural analyses which are not compatible with the prefix.
- Generates predictions about word-by-word reading times by comparing the total effort expended before some word to the total effort after.
- The explanation for garden-pathing is then **the reduction in the probability** of the new tree set compared with the previous tree set.
- The model accounts successfully for reading times.

Hale — Earley parser as a model of sentence processing

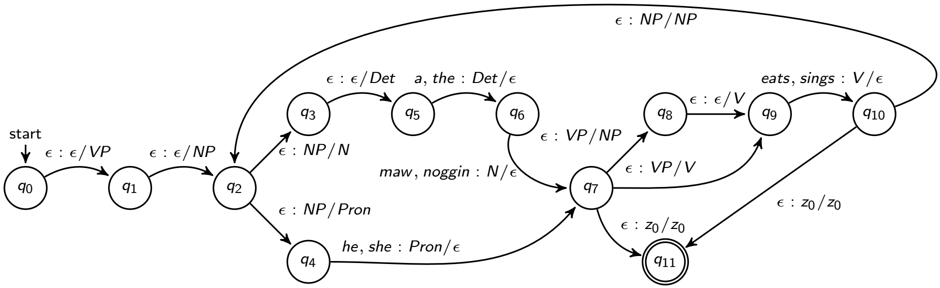
Toy grammar with probabilities

S	$\rightarrow$	NP VP	1
NP	$\rightarrow$	N PP	0.2
NP	$\rightarrow$	N	0.8
PP	$\rightarrow$	P NP	1
VP	$\rightarrow$	VP PP	0.1
VP	$\rightarrow$	V VP	0.2
VP	$\rightarrow$	V NP	0.4
VP	$\rightarrow$	V	0.3
N	$\rightarrow$	{it, fish, rivers, December, they}	0.2
P	$\rightarrow$	{in}	1
V	$\rightarrow$	{can, fish}	0.5

Hale — Earley parser as a model of sentence processing

$edge_n$	DOTTED RULE	[S, W]	HIST	**Prob**	**MaxProb**
e_0	S → ● NP VP	[0,0]			P(S → NP VP)=1
e_1	NP → ● N	[0,0]			P(e_0)P(NP → N)=1*0.8=0.8
e_2	NP → ● N PP	[0,0]			P(e_0)P(NP → N PP)=1*0.2=0.2
e_3	N → they ●	[0,1]		P(N → they)=0.2	
e_4	NP → N ●	[0,1]	(e_3)	P(e_3)P(NP → N) =0.2*0.8 =0.16	
e_5	NP → N ● PP	[0,1]	(e_3)		
e_6	S → NP ● VP	[0,1]	(e_4)		
e_7	PP → ● P NP	[1,1]			P(N → they)P(e_2)P(PP → P NP) =0.2*1*0.2*1=0.04
e_8	VP → ● V	[1,1]			P(N → they)P(e_1)P(VP → V) =0.2*1*0.8*0.3=0.048
e_9	VP → ● V NP	[1,1]			P(N → they)P(e_1)P(VP → V NP) =0.2*1*0.8*0.4=0.064
e_{10}	VP → ● V VP	[1,1]			P(N → they)P(e_1)P(VP → V VP) =0.2*1*0.8*0.2=0.032
e_{11}	VP → ● VP PP	[1,1]			P(N → they)P(e_1)P(VP → VP PP) =0.2*1*0.8*0.1=0.0016
e_{12}	V → can ●	[1,2]		P(V → can)=0.5	
e_{13}	VP → V ●	[1,2]	(e_{12})	P(e_{12})P(VP → V) =0.5*0.3 =0.15	
e_{14}	VP → V ● NP	[1,2]	(e_{12})		
e_{15}	VP → V ● VP	[1,2]	(e_{12})		
e_{16}	S → NP VP ●	[0,2]	(e_4,e_{13})	P(e_4)P(e_{13})P(S → NP VP) =0.2*0.8*0.5*0.3*1 =0.024	
e_{17}	VP → VP ● PP	[1,2]	(e_{13})		

Yngve—PDA as a model of sentence processing



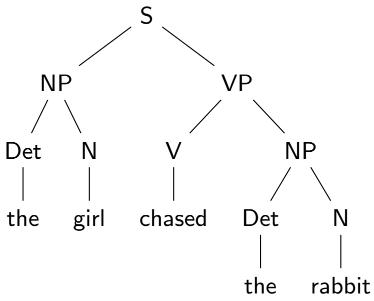
- **Hypothesis**: the size of the stack correlates with working memory load.
- **Prediction**: sentences which require many items to be placed on the stack will be difficult to process and also less frequent in the language.
- **Prediction**: when multiple parses are possible we should prefer the one with the minimised stack.

Yngve—PDA as a model of sentence processing

- Yngve formulated the problem as interaction between:
  - a **register** (which holds the current node) and
  - the **stack** (which contains all the nodes left to explore)
- Sentences are constructed top-down and left-to-right.
- Under these circumstances the size of the stack is hypothesised to correlate with working memory load.

Hypothesis: stack correlates with **working memory load**

S→NP VP
NP→Det N
VP→V NP
Det→the
N→girl
N→rabbit
V→chased



Register	Stack
S	
NP	VP
Det	N VP
the	N VP
N	VP
girl	VP
VP	
V	NP
chased	NP
NP	
Det	N
the	N
N	
rabbit	

Hypothesis: stack correlates with **working memory load**

Yngve's model makes **predictions** about centre embedding:

- Consider:

  This is the malt that the rat that the cat that the dog worried killed ate.

  STACK: N VP VP VP

- as opposed to:

  This is the malt that was eaten by the rat that was killed by the cat that was worried by the dog.

- Yngve evaluated his predictions by looking at frequencies of constructions in corpus data.

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Recap:

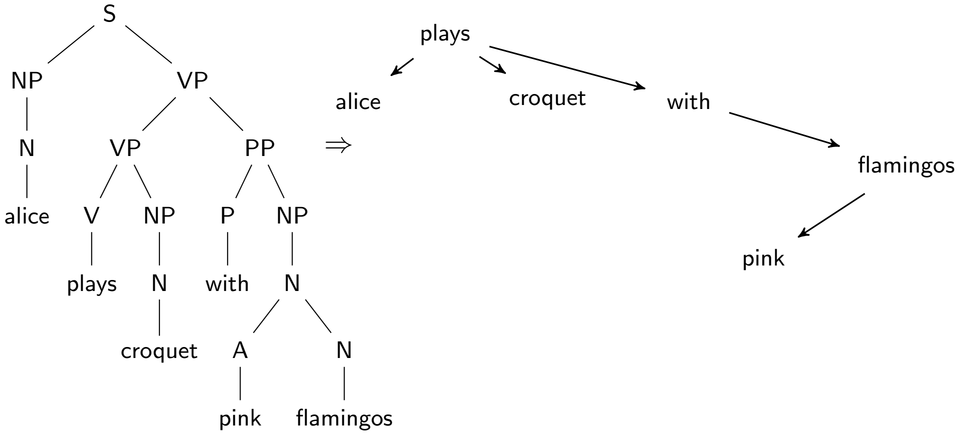
- We said LR shift-reduce parser wasn't a good fit for natural language because it proceeds deterministically and natural language is too ambiguous.
- We used the Earley parser to explore the whole tree-space, recording partial derivations in a chart.

However,

- We can use a modified version of the shift-reduce parser in order to parse natural language.
- First we're going to learn about **dependency grammars**.

A **dependency tree** is a directed graph

A **dependency tree** is a directed graph representation of a string—each edge represents a grammatical relationship between the symbols.



A **dependency grammar** derives **dependency trees**

Formally $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$ where:

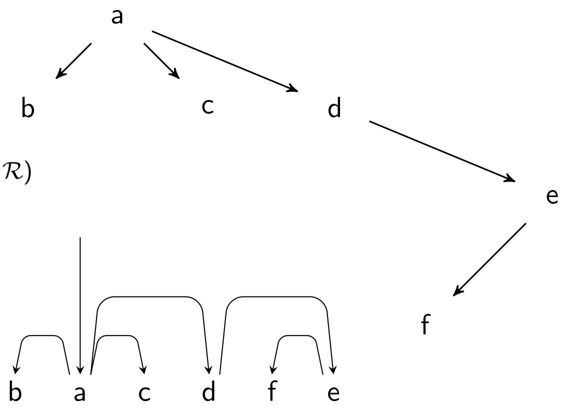
- Σ is the finite set of alphabet symbols
- $\mathcal{D}$ is the set of symbols to indicate whether the dependent symbol (the one on the RHS of the rule) will be located on the left or right of the current item within the string $\mathcal{D} = \{\mathcal{L}, \mathcal{R}\}$
- s is the root symbol for the dependency tree (we will use $s \in \Sigma$ but sometimes a special extra symbol is used)
- $\perp$ is a symbol to indicate a halt in the generation process
- $\mathcal{P}$ is a set of rules for generating dependencies: $\mathcal{P} = \{(\alpha \to \beta,\ d) \mid \alpha \in (\Sigma\ \cup\ s),\ \beta \in (\Sigma \cup \perp),\ d \in \mathcal{D}\}$

In dependency grammars we refer to the term on the LHS of a rule as the **head** and the RHS as the **dependent** (as opposed to *parents* and *children* in phrase structure grammars).

Dependency trees have **several representations**

Two diagrammatic representations of a dependency tree for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$ where:

$$\begin{aligned}
\Sigma &= \{a...f\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{a} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L}) \\
&\quad (b \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R}) \\
&\quad (c \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R}) \\
&\quad (f \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R})\}
\end{aligned}$$



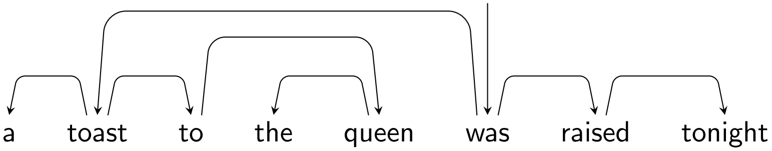
The same rules would have been used to generate the string *badfec*.

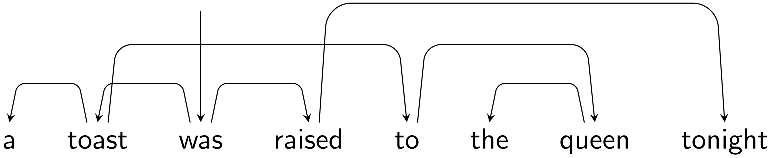
Useful when there is flexibility in the symbol order of grammatical strings.

Valid trees may be **projective** or **non-projective**

Valid derivation is one that is **rooted** in *s* and is **weakly connected**.

- Derivation trees may be **projective** or **non-projective**.
- Non-projective trees can be needed for long distance dependencies.





- The difference has implications for parsing complexity.

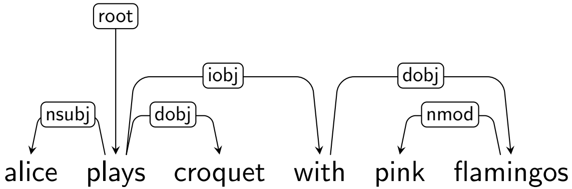
Labels can be added to the dependency edges

A label can be added to each generated dependency:

$$\mathcal{P} = \{(\alpha \rightarrow \beta : r, d) \mid \alpha \in (\Sigma \cup s),\ \beta \in (\Sigma \cup \bot),\ d \in \mathcal{D},\ r \in \mathcal{B}\}$$

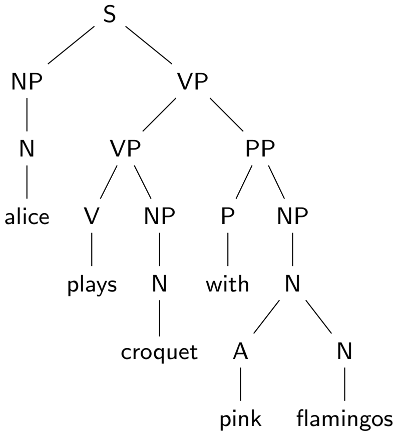
where $\mathcal{B}$ is the set of dependency labels.

When used for natural language parsing, dependency grammars will often label each dependency with the *grammatical function* (or the **grammatical relation**) between the words.

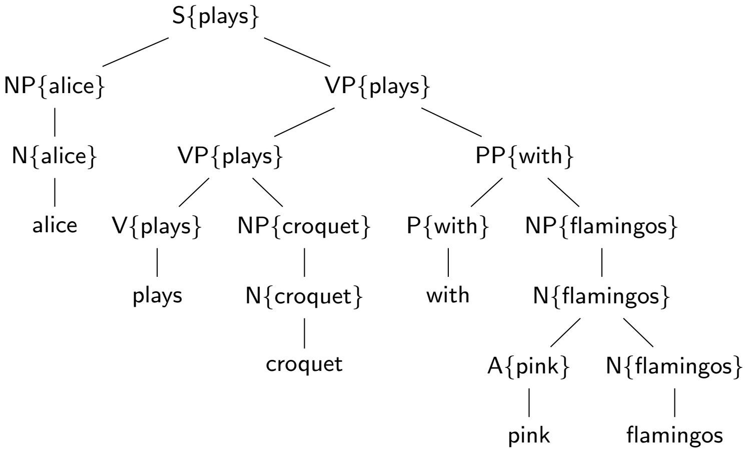


Dependency grammars can be **weakly equivalent** to CFGs

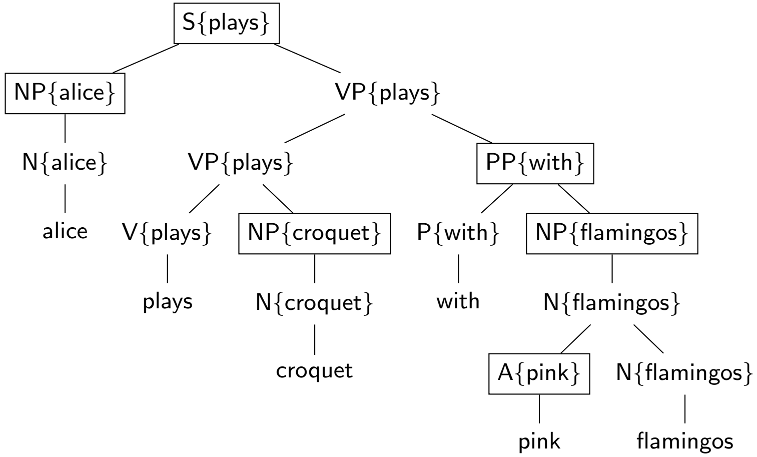
Projective dependency grammars can be shown to be **weakly equivalent** to context-free grammars.



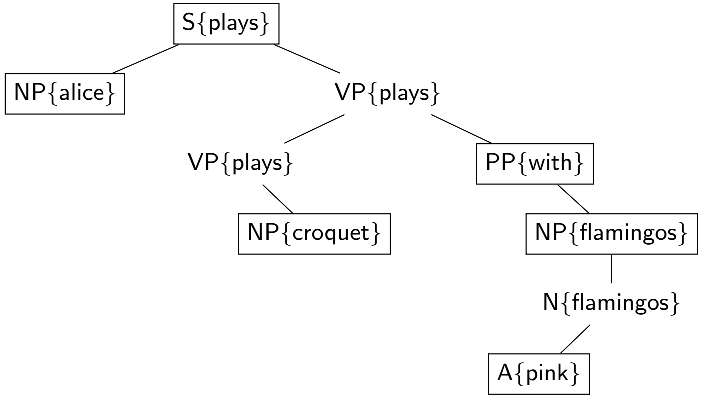
Dependency grammars can be **weakly equivalent** to CFGs



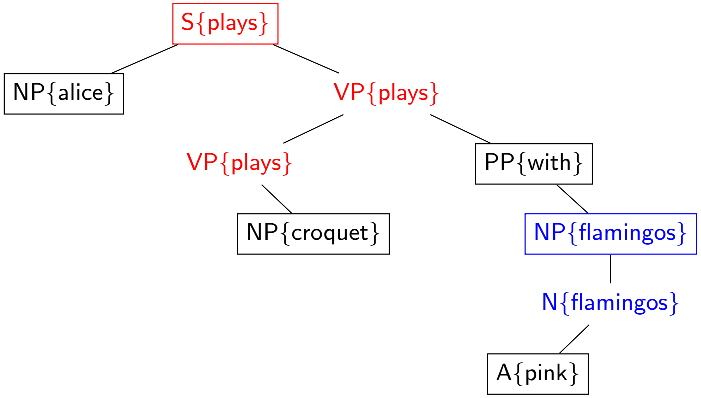
Dependency grammars can be **weakly equivalent** to CFGs



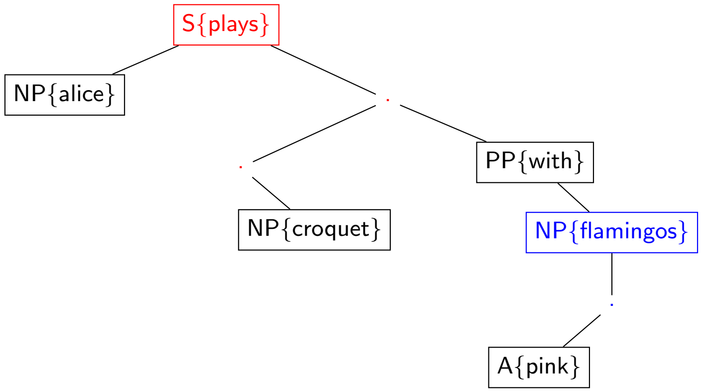
Dependency grammars can be **weakly equivalent** to CFGs



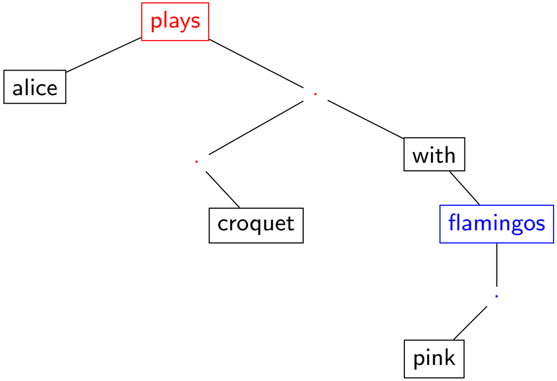
Dependency grammars can be **weakly equivalent** to CFGs



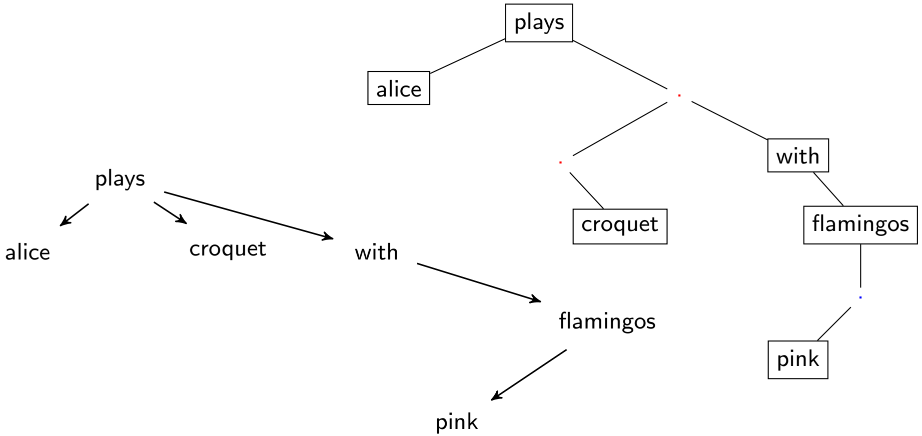
Dependency grammars can be **weakly equivalent** to CFGs



Dependency grammars can be **weakly equivalent** to CFGs



Dependency grammars can be **weakly equivalent** to CFGs



Projective dependency grammars can be shown to be **weakly equivalent** to context-free grammars.

Dependency parsers use a **modified** shift-reduce parser

- A common method for dependency parsing of natural language involves a modification of the LR shift-reduce parser
- The **shift** operator continues to move items of the input string from the buffer to the stack
- The **reduce** operator is replaced with the operations **left-arc** and **right-arc** which *reduce* the top two stack symbols leaving the *head* on the stack

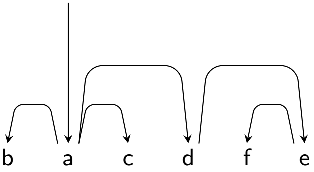
Consider $\mathcal{L}(G_{dep}) \subseteq \Sigma^*$, during parsing the stack may hold γab where $\gamma \in \Sigma*$ and $a, b \in \Sigma$, and b is at the top of the stack:

- LEFT-ARC reduces the stack to γb and records use of rule $b \rightarrow a$
- RIGHT-ARC reduces the stack to γa and records the use of rule $a \rightarrow b$

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{a} \\
\mathcal{P} &= \{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \to e, \mathcal{R}) \\
&\quad (e \to f, \mathcal{L})\}
\end{aligned}$$



STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \to b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \to c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \to f$
ade		RIGHT-ARC	$d \to e$
ad		RIGHT-ARC	$a \to d$
a		TERMINATE	$root \to a$

Note that, for a deterministic parse here, a lookahead is needed

Data driven dependency parsing is **grammarless**

- For natural language there would be considerable effort in manually defining $\mathcal{P}$—this would involve determining the dependencies between all possible words in the language.
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can be achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items on the stack and in the buffer as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

We can use a **beam** search to record the parse forest

- The classifier can return a **probability** of an action.
- To avoid the problem of early incorrect resolution of an ambiguous parse, multiple competing parses can be recorded and a **beam search** used to keep track of the best alternative parses.
- Google's *Parsey McParseface* is an English language dependency parser that uses word-embeddings as features and a neural network to score parse actions. A beam search is used to compare competing parses.

Dependency parsers can be useful for parsing **speech**

The most obvious difference between spoken and written language is the mode of transmission:

- **Prosody** refers to the patterns of stress and intonation in a language.
- **Stress** refers to the relative emphasis or prominence given to a certain part of a word (e.g. CON-tent (the stuff included in something) vs. con-TENT (happy))
- **Intonation** refers to the way speakers' pitch rises and falls in line with words and phrases, to signal a question, for example.
- Co-speech gestures involve parts of the body which move in coordination with what a speaker is saying, to emphasise, disambiguate or otherwise.

We can use some of these extra features to help the parse-action-classifier when parsing spoken language.

Prosody has been used to resolve parsing ambiguity

- Briscoe suggested using a shift-reduce parser that favours shift over reduce wherever both are possible.
- In the absence of extra-linguistic information the parser delays resolution of the grammatical dependency.
- Extra features enable an override of the shift preference at the point where the ambiguity arises, including:
  - prosodic information (intonational phrase boundary)

  The model accounts for frequencies of certain syntactic constructions as attested in corpora.

Spoken language **lacks** string delimitation

- A fundamental issue that affects syntactic parsing of spoken language is the **lack of the sentence unit** (i.e string delimitation)—indicated in writing by a full-stop and capital letter.
- **Speech units** may be identified by pauses, intonation (e.g. rising for a question, falling for a full-stop), change of speaker.
- Speech units are not much like written sentences due to **speaker overlap**, **co-constructions**, **ellipsis**, **hesitation**, **repetitions** and **false starts**.
- Speech units often contain words and grammatical constructions that would not appear in the written form of the language.

Spoken language **lacks** string delimitation

Excerpt from the Spoken section of the British National Corpus

set your sights realistically haven't you and there's a lot of people unemployed and what are you going to do when you eventually leave college if you get there you're not gonna step straight into television mm right then let's see now what we're doing where's that recipe book for that chocolate and banana cake chocolate and banana cake which book was it oh right oh some of these chocolate cakes are absolutely mm mm mm right what's the topping what's that icing sugar cocoa powder and vanilla essence oh luckily I've got all those I think yes

Spoken language **lacks** string delimitation

Excerpt from the Spoken section of the British National Corpus

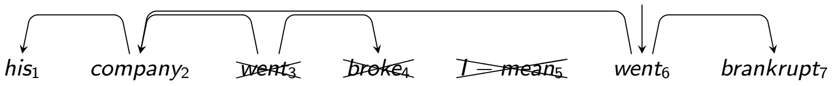
Set your sights realistically haven't you? And there's a lot of people unemployed. And what are you going to do when you eventually leave college? If you get there. You're not gonna step straight into television. Mm right then, let's see now what we're doing... Where's that recipe book for that chocolate and banana cake? Chocolate and banana cake which book was it? Oh right. Oh, some of these chocolate cakes are absolutely mm mm mm. Right, what's the topping? what's that? Icing sugar, cocoa powder and vanilla essence. Oh luckily I've got all those I think, yes!

Dependency parsers can be useful for parsing **speech**

- Spoken language can look noisy and somewhat *grammarless* but the disfluencies are predictable
- Honnibal & Johnson's Redshift parser introduces an **edit** action, to remove disfluent items from spoken language:
    - EDIT: on detection of disfluency, remove connected words and their dependencies.
- Parser uses extra classifier features to detect disfluency.

Example of dependency parser using an **edit** action

STACK	BUFFER	ACTION	RECORD
	$his_1 \dots bankrupt_7$	SHIFT	
his_1	$company_2 \dots bankrupt_7$	SHIFT	
$his_1\ company_2$	$went_3 \dots bankrupt_7$	LEFT-ARC	$company_2 \rightarrow his_1$
$company_2$	$went_3 \dots bankrupt_7$	SHIFT	
$company_2\ went_3$	$broke_4 \dots bankrupt_7$	LEFT-ARC	~~$went_3 \rightarrow company_2$~~
$went_3$	$broke_4 \dots bankrupt_7$	SHIFT	
$went_3\ broke_4$	$I - mean_5 \dots bankrupt_7$	RIGHT-ARC	~~$went_3 \rightarrow broke_4$~~
$went_3$	$I - mean_5 \dots bankrupt_7$	SHIFT	
$went_3\ I - mean_5$	$went_6\ bankrupt_7$	EDIT	
$company_2$	$went_6\ bankrupt_7$	SHIFT	
$company_2\ went_6$	$bankrupt_7$	LEFT-ARC	$went_6 \rightarrow company_2$
$went_6$	$bankrupt_7$	SHIFT	
$went_3\ bankrupt_7$		RIGHT-ARC	$went_6 \rightarrow bankrupt_7$
$went_3$		TERMINATE	$root \rightarrow went_6$



Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

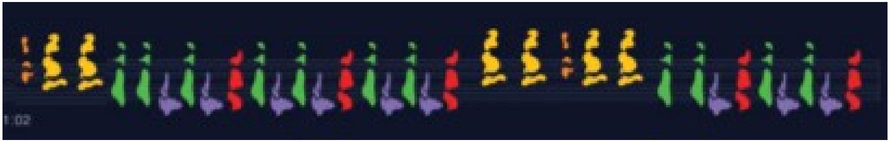
- Last time we looked at ways to parse without ever building a grammar
- But what if we want to know what a grammar is for a set of strings?

- Today we will look at grammar induction.

  ...we'll start with an example

CFGs may be inferred using recursive **byte-pair encoding**

The following is a *speech unit* of whale song:



b a a c c d c d e c d c d e c d c d e a a b a a c c d e c d c d e

We are going to infer some rules for this string using the following algorithm:

- count the frequency of all adjacent pairs in the string
- reduce the most frequent pair to a non-terminal
- repeat until there are no pairs left with a frequency > 1

This is used for compression—once we have removed all the repeated strings we have less to transmit or store (we have to keep the grammar to decompress)

CFGs may be inferred using recursive **byte-pair encoding**

b a a c c d c d e c d c d e c d c d e a a b a a c c d e c d c d e

$F \rightarrow c\ d$

b a a c F F e F F e F F e a a b a a c F e F F e

$G \rightarrow F\ e$

b a a c F G F G F G a a b a a c G F G

$H \rightarrow F\ G$

b a a c H H H a a b a a c G H

$I \rightarrow a\ a$

b I c H H H I b I c G H

$J \rightarrow b\ I$

J c H H H I J c G H

$K \rightarrow J\ c$

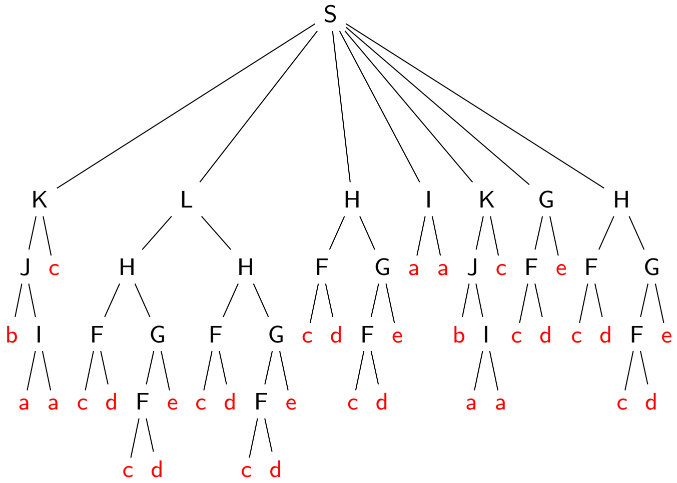
K H H H I K G H

$L \rightarrow H\ H$

K L H I K G H

$S \rightarrow K\ L\ H\ I\ K\ G\ H$

CFGs may be inferred using recursive **byte-pair encoding**


S
K L H I K G H
J c H H F G a a J c F e F G
b I F G F G c d F e b I c d c d F e
a a c d F e c d F e c d a a c d
c d c d

Byte-pair has **shortcomings** for grammar induction

Byte-pair encoding has benefits for encryption but shortcomings when it comes to grammar induction (especially of natural language):

- the algorithm is frequency driven and this might not lead to *appropriate* constituency
- in the circumstance that two pairs have the same frequency we make an arbitrary choice of which to reduce.
- the data is assumed to be non-noisy (all string sequences encountered are treated as valid)
- (for natural language) the algorithm learns from strings alone (a more *appropriate* grammar might be derived by including extra-linguistic information)

We might suggest improvements to the algorithm (such as allowing ternary branching) but in order to compare the algorithms we need a **learning paradigm** in which to study them.

Paradigms are defined over **grammatical systems**

Grammatical system:

- $\mathcal{H}$ a hypothesis space of language descriptions (e.g. all possible grammars)
- Ω a sample space (e.g. all possible strings)
- $\mathcal{L}$ a function that maps from a member of $\mathcal{H}$ to a subset of Ω

If we have $(\mathcal{H}_{cfg}, \Sigma^*, \mathcal{L})$ then for some $G \in \mathcal{H}_{cfg}$ we have:
$\mathcal{L}(G) = \{s_a, s_b, s_c ...\} \subseteq \Sigma*$

Learning function:

The learning function, F, maps from a subset of Ω to a member of $\mathcal{H}$

For $G \in \mathcal{H}_{cfg}$ then $F(\{s_d, s_e, s_f ...\}) = G$ for some $\{s_d, s_e, s_f ...\} \subseteq \Sigma*$

Note that the learning function is an algorithm (referred to as the **learner**) and that **learnability** is a property of a language class (when F surjective).

Learning paradigms specify the nature of **input**

Varieties of input given to the learner:

- **positive evidence**: the learner receives only valid examples from the sample space (i.e. if the underlying grammar is G then the learner receives samples, s_i, such that $s_i \in \mathcal{L}(G)$).
- **negative evidence**: the learner receives samples flagged as not being in the language.
- **exhaustive evidence**: the learner receives every relevant sample from the sample space.
- **non-string evidence**: the learner receives samples that are not strings.

Learning paradigms also specify...

- **assumed knowledge**: the things known to the learner before learning commences (for instance, the hypothesis space, $\mathcal{H}$ might be assumed knowledge).
- **nature of the algorithm**: are samples considered sequentially or as a batch? does the learner generate a hypothesis after every sample received in a sequence? does the learner generate a hypothesis after specific samples only?
- **required computation**: e.g. is the learner constrained to act in polynomial time.
- **learning success**: what are the criteria by which we measure success of the learner?

Gold's learning paradigms have been influential

Gold's best known paradigm modelled language learning as an infinite process in which a learner is presented with an infinite stream of strings of the target language:

- for a grammatical system $(\mathcal{G}, \Sigma^*, \mathcal{L})$
- select one of the languages L in the class defined by $\mathcal{L}$ (this is called the **target language**, $L = \mathcal{L}(G)$ where $G \in \mathcal{G}$)
- samples are presented to the learner one at a time $s_1, s_2, \ldots$ in an infinite sequence
- the learner receives only positive evidence (i.e. only s_i such that $s_i \in L$)
- after each sample the learner produces a hypothesis (i.e. learner produces G_n after having seen the data $s_1, \ldots s_n$)
- the evidence is exhaustive, every $s \in L$ will be presented in the sequence.

Gold's learning paradigms have been influential

Gold defined **identification in the limit** as successful learning:

- There is some number N such that for all $i > N$, $G_i = G_N$ and $\mathcal{L}(G_N) = L$
- N is finite but there are no constraints placed on computation time of the learning function.

In this paradigm a **class** of languages is **learnable** if:

- Every language in the class can be identified in the limit *no matter what order* the samples appear in

Gold's learning paradigms have been influential

Well known results from Gold's paradigm include:

- The class of **suprafinite** languages are not learnable (a suprafinite class of languages is one that contains all finite languages and at least one infinite language)
- This means that e.g. the class of context-free languages are *not learnable* within Gold's paradigm.

We might care about this if we think that Gold's paradigm is a good model for natural language acquisition...(if we don't think this then it is just a fun result!).

Gold: **suprafinite** languages are not learnable

Short proof:

- Let L_∞ be an infinite language $L_\infty = \{s_1, s_2, ...\}$
- Now construct an infinite sequence of finite languages $L_1 = \{s_1\}$, $L_2 = \{s_1, s_2\}$, ...
- Consider a particular presentation order $s_1...s_1, s_2...s_2, s_3...$
- When learning L_1 we repeat s_1 until the learner predicts L_1
- When learning L_2 repeat s_1 until the learner predicts L_1 then repeat s_2 until it predicts L_2
- Continue like this for all L_i: either the learner fails to converge on one of these, or it ultimately fails to converge on L_∞ for finite N.
- We have found an ordering of the samples that makes the learner fail

Many people have investigated what IS learnable in this paradigm. We will look at one example, but to do so we introduce one more grammar.

Categorial grammars are **lexicalized grammars**

In a **classic categorial grammar** all symbols in the alphabet are associated with a finite number of **types**.

- Types are formed from primitive types using two operators, $\backslash$ and $/$.
- If P_r is the set of **primitive types** then the set of all types, T_p, satisfies:
  - $P_r \subset T_p$
  - if $A \in T_p$ and $B \in T_p$ then $A \backslash B \in T_p$
  - if $A \in T_p$ and $B \in T_p$ then $A/B \in T_p$
- Note that it is possible to arrange types in a hierarchy: a type A is a *subtype* of B if A occurs in B (that is, A is a subtype of B iff $A = B$; or ($B = B_1 \backslash B_2$ or $B = B_1/B_2$) and A is a subtype of B_1 or B_2).

Categorial grammars are **lexicalized grammars**

- A relation, $\mathcal{R}$, maps symbols in the alphabet Σ to members of T_p.
- A grammar that associates at most one type to each symbol in Σ is called a **rigid grammar**
- A grammar that assigns at most k types to any symbol is a **k-valued grammar**.
- We can define a classic categorial grammar as $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:
  - Σ is the alphabet/set of terminals
  - P_r is the set of primitive types
  - S is a distinguished member of the primitive types $S \in P_r$ that will be the root of complete derivations
  - $\mathcal{R}$ is a relation $\Sigma \times T_p$ where T_p is the set of all types as generated from P_r as described above

Categorial grammars are **lexicalized grammars**

A string has a valid parse if the types assigned to its symbols can be combined to produce a derivation tree with root S.

Types may be combined using the two rules of function application:

- FORWARD APPLICATION is indicated by the symbol $>$:

$$\frac{A/B \qquad B}{A}>$$

- BACKWARD APPLICATION is indicated by the symbol $<$:

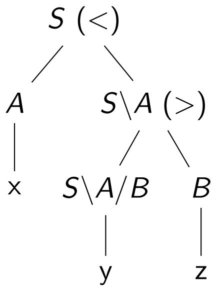
$$\frac{B \qquad A\backslash B}{A}<$$

Categorial grammars are **lexicalized grammars**

Derivation tree for the string *xyz* using the grammar $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:

$$\begin{array}{rcl} Pr & = & \{S, A, B\} \\ \Sigma & = & \{x, y, z\} \\ S & = & S \\ \mathcal{R} & = & \{(x, A), (y, S\backslash A/B), (z, B)\} \end{array}$$

$$\dfrac{\dfrac{x}{A}\,\mathcal{R} \qquad \dfrac{\dfrac{y}{S\backslash A/B}\,\mathcal{R} \qquad \dfrac{z}{B}\,\mathcal{R}}{S\backslash A}\,>}{S}\,<$$

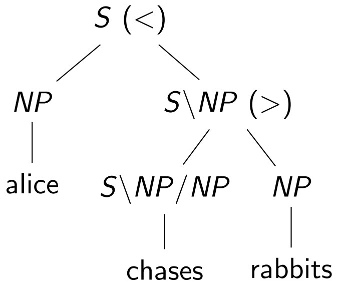


Categorial grammars are **lexicalized grammars**

Derivation tree for the string *Alice chases rabbits* using the grammar $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:

$$\begin{aligned} Pr &= \{S, NP\} \\ \Sigma &= \{alice, chases, rabbits\} \\ S &= S \\ \mathcal{R} &= \{(alice, NP), (chases, S\backslash NP/NP), \\ &\quad (rabbits, NP)\} \end{aligned}$$

$$\dfrac{\dfrac{alice}{NP}\,\mathcal{R} \qquad \dfrac{\dfrac{chases}{S\backslash NP/NP}\,\mathcal{R} \qquad \dfrac{rabbits}{NP}\,\mathcal{R}}{S\backslash NP}\,>}{S}\,<$$



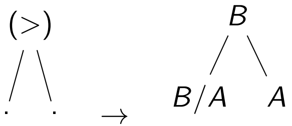
We can construct a **strongly equivalent** CFG

To create a context-free grammar $G_{cfg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$ with strong equivalence to $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ we can define G_{cfg} as:

$$\begin{array}{rcl} \mathcal{N} & = & P_r \cup range(\mathcal{R}) \\ \Sigma & = & \Sigma \\ S & = & S \\ P & = & \{A \rightarrow B \ \ A\backslash B \mid A\backslash B \in T_p\} \\ & & \cup \ \{A \rightarrow A/B \ \ B \mid A/B \in T_p\} \\ & & \cup \ \{A \rightarrow a \mid \mathcal{R} : a \rightarrow A\} \end{array}$$

FYI: a categorial grammar learner within Gold's paradigm

- Buszkowski developed an algorithm for learning rigid grammars from **functor-argument structures**.
- The algorithm proceeds by inferring types from the available information
- Eg. for Forward Application:



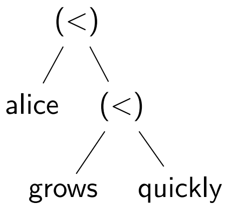
- Variables are unified across all encountered structures.
- Kanazawa constructed a proof to show that the algorithm could learn the class of rigid grammars from an **infinite stream** of functor-argument structures — as required to satisfy Gold's paradigm.

FYI: a categorial grammar learner within Gold's paradigm

Let G_i be the current hypothesis of the learner:

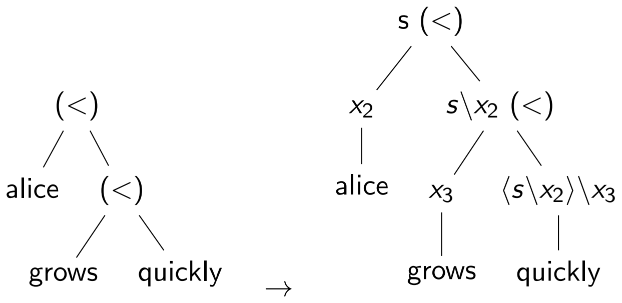
$$\begin{aligned} G_i : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s\backslash x_1 \end{aligned}$$

Let the next functor-argument structor encountered in the steam be:



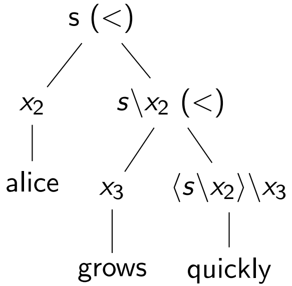
FYI: a categorial grammar learner within Gold's paradigm

Infer types to the new functor-argument structure:



FYI: a categorial grammar learner within Gold's paradigm

- Look up words at the leaf nodes of the new structure in G_i
- If the word exists in G_i, add types inferred at leaf nodes to the existing set of types for that word; else create new word entry.



$$\begin{aligned} G_i : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s\backslash x_1 \end{aligned}$$

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1, x_2 \\ \text{grows} &\rightarrow s\backslash x_1, x_3 \\ \text{quickly} &\rightarrow \langle s\backslash x_2\rangle\backslash x_3 \end{aligned}$$

FYI: a categorial grammar learner within Gold's paradigm

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1, x_2 \\ \text{grows} &\rightarrow s\backslash x_1, x_3 \\ \text{quickly} &\rightarrow \langle s\backslash x_2\rangle\backslash x_3 \end{aligned}$$

- Unify the set of types. If unification fails then fail.

$$\begin{aligned} x_2 &\mapsto x_1 \\ x_3 &\mapsto s\backslash x_1 \end{aligned}$$

- Output the lexicon.

$$\begin{aligned} G_{i+1} : \text{alice} &\rightarrow x_1 \\ \text{grows} &\rightarrow s\backslash x_1 \\ \text{quickly} &\rightarrow \langle s\backslash x_1\rangle\backslash\langle s\backslash x_1\rangle \end{aligned}$$

FYI: a categorial grammar learner within Gold's paradigm

Using this learner within Gold's paradigm over various sample spaces it is possible to prove:

- Rigid grammars are learnable from functor-argument structor and strings
- k-valued grammars (for a specific k) are learnable from functor-argument structor and strings

Note that the above mentioned grammars are subsets of the CFGs

Gold's paradigm **is not** much like human acquisition

- Gold's paradigm requires convergence in a finite number of steps (hypotheses of G) the amount of data it sees is unbounded.
- Gold's learner can use unbounded amounts of computation.

- A child only sees a limited amount of data, and has limited computational resources

- Success in this paradigm tells you absolutely nothing about the learner's state at any finite time.

- Children learn progressively

- The learner has to learn for every possible presentation of the samples (including presentations that have been chosen by an adversary with knowledge of the internal state of the learner).

- It is arguable that the distributions are in some way helpful: **parentese**

Gold's paradigm **is not** much like human acquisition

- Gold's learner is required to exactly identify the target language.

- We do not observe this in humans

  We observe agreement on *grammaticality* between adults and children approaching adult competence but we also observe differences in word choices and grammaticality judgments between adults speakers.

- Gold's learner requires a hypothesis to be selected after every step.

- In fact there is evidence that children only attend to selective evidence (Goldilocks effect)

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Languages transmit **information**

In previous lectures we have thought about language in terms of **computation**.

Today we are going to discuss language in terms of the **information** it conveys...

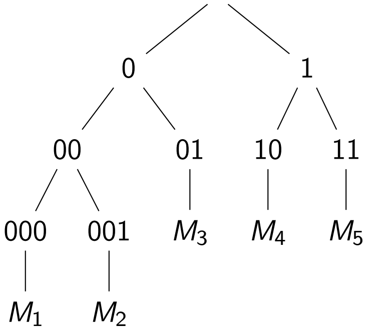
Entropy is a measure of information

- Information **sources** produce **information** as **events** or **messages**.
- Represented by a random variable X over a discrete set of symbols (or alphabet) $\mathcal{X}$.
- e.g. for a dice roll $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ for a source that produces characters of written English $\mathcal{X} = \{a...z, \}$
- **Entropy** (or **self-information**) may be thought of as:
  - the average amount of information produced by a source
  - the average amount of uncertainty of a random variable
  - the average amount of information we gain when receiving a message from a source
  - the average amount of information we lack before receiving the message
  - the average amount of uncertainty we have in a message we are about to receive

Entropy is a measure of information

- Entropy, H, is measured in **bits**.
- If X has M equally likely events: $H(X) = \log_2 M$
- Entropy gives us a **lower limit** on:
  - the number of bits we need to represent an event space.
  - the average number of bits you need per message code.

$$\begin{aligned} avg\_length &= \frac{(3*2)+(2*3)}{5} = 2.4 \\ &> H(5) = \log_2 5 = 2.32 \end{aligned}$$



Surprisal is also measured in bits

- Let $p(x)$ be the probability mass function of a random variable, X over a discrete set of symbols $\mathcal{X}$.
- The **surprisal** of x is $s(x) = \log_2 \left(\frac{1}{p(x)}\right) = -\log_2 p(x)$
- Surprisal is also measured in **bits**
- Surprisal gives us a measure of information that is inversely proportional to the probability of an event/message occurring
- i.e probable events convey a small amount of information and improbable events a large amount of information
- The average information (entropy) produced by X is the weighted sum of the surprisal (the average surprise): $H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x)$
- Note, that when all M items in $\mathcal{X}$ are equally likely (i.e. $p(x) = \frac{1}{M}$) then $H(X) = -\log_2 p(x) = \log_2 M$

The surprisal of the alphabet in *Alice in Wonderland*

x	$f(x)$	$p(x)$	$s(x)$
	26378	0.197	2.33
e	13568	0.101	3.30
t	10686	0.080	3.65
a	8787	0.066	3.93
o	8142	0.056	4.04
i	7508	0.055	4.16
…			
v	845	0.006	7.31
q	209	0.002	9.32
x	148	0.001	9.83
j	146	0.001	9.84
z	78	0.001	10.75

- If uniformly distributed: $H(X) = \log_2 27 = 4.75$
- As distributed in *Alice*: $H(X) = 4.05$
- Re. example 1:
- Average surprisal of a vowel = 4.16 bits (3.86 without u)
- Average surprisal of a consonant = 6.03 bits

Example 1

Last consonant removed:
Jus the he hea struc agains te roo o te hal: i fac se wa no rathe moe tha nie fee hig.
average missing information: 4.59 bits

Last vowel removed:
Jst thn hr hed strck aganst th rof f th hll: n fct sh ws nw rathr mor thn nin fet hgh.
average missing information: 3.85 bits

Original sentence:
Just then her head struck against the roof of the hall: in fact she was now rather more than nine feet high.

The surprisal of words in *Alice in Wonderland*

x	$f(x)$	$p(x)$	$s(x)$
the	1643	0.062	4.02
and	872	0.033	4.94
to	729	0.027	5.19
a	632	0.024	5.40
she	541	0.020	5.62
it	530	0.020	5.65
of	514	0.019	5.70
said	462	0.017	5.85
i	410	0.015	6.02
alice	386	0.014	6.11
...			
<any>	3	0.000	13.2
<any>	2	0.000	13.7
<any>	1	0.000	14.7

Example 2

She stretched herself up on tiptoe, and peeped over the edge of the mushroom, and her eyes immediately met those of a large blue caterpillar, that was sitting on the top with its arms folded, quietly smoking a long hookah, and taking not the smallest notice of her or of anything else.

Average information of *of* = 5.7 bits

Average information of low frequency compulsory content words = 14.7 bits (freq = 1), 13.7 bits (freq = 2), 13.2 bits (freq = 3)

Aside: Is written English a good code?

Highly efficient codes make use of regularities in the messages from the source using shorter codes for more probable messages.

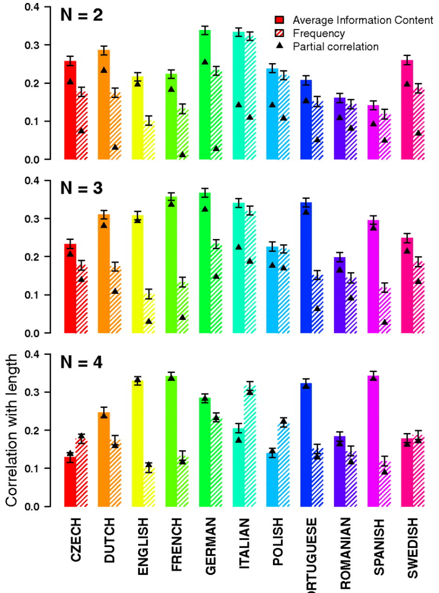
- From an encoding point of view, surprisal gives an indication of the number of bits we would want to assign a message symbol.
- It is efficient to give probable items (with low surprisal) a small bit code because we have to transmit them often.
- So, is English efficiently encoded?
- Can we predict the information provided by a word from its length?

Aside: Is written English a good code?

Piantadosi et al. investigated whether the surprisal of a word correlates with the word length.

- They calculated the average surprisal (average information) of a word w given its context c
- That is, $-\frac{1}{C}\sum_{i=1}^{C}\log_2 p(w|c_i)$
- Context is approximated by the n previous words.

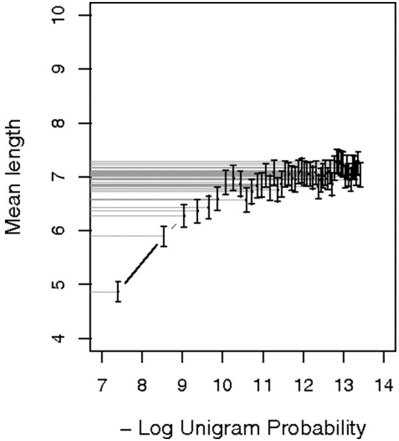
Aside: Is written English a good code?

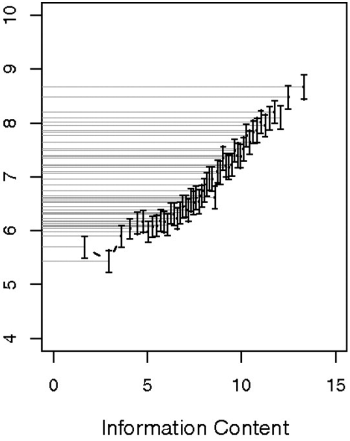


- Piantadosi et al. results for Google n-gram corpus.
- Spearman's rank on y-axis (0=no correlation, 1=monotonically related)
- Context approximated in terms of 2, 3 or 4-grams (i.e. 1, 2, or 3 previous words)
- Average information is a better predictor than frequency most of the time.

Aside: Is written English a good code?

Piantadosi et al: Relationship between frequency (negative log unigram probability) and length, and information content and length.





In language, events depend on context

Examples from *Alice in Wonderland*:

- Generated using $p(x)$ for $x \in \{a\text{-}z, \_\}$:

  dgnt_a_hi_tio__iui_shsnghihp_tceboi_c_ietl_ntwe_c_a_ad__ne_saa
  __hhpr___bre_c_ige_duvtnltueyi_tt__doe

- Generated using $p(x|y)$ for $x, y \in \{a\text{-}z, \_\}$:

  s_ilo_user_wa_le_anembe_t_anceasoke_ghed_mino_fftheak_ise_linld_met
  _thi_wallay_f_belle_y belde_se_ce

In language, events depend on context

Examples from *Alice in Wonderland*:

- Generated using $p(x)$ for $x \in \{\textit{words in Alice}\}$:

  didnt and and hatter out no read leading the time it two down to just this must goes getting poor understand all came them think that fancying them before this

- Generated using $p(x|y)$ for $x, y \in \{\textit{words in Alice}\}$:

  murder to sea i dont be on spreading out of little animals that they saw mine doesnt like being broken glass there was in which and giving it after that

In language, events depend on context

- **Joint entropy** is the amount of information needed on average to specify two discrete random variables:
  $H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_2 p(x,y)$

- **Conditional entropy** is the amount of extra information needed to communicate Y, given that X is already known:
  $H(Y|X) = \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_2 p(y|x)$

- **Chain rule** connects joint and conditional entropy:
  $H(X,Y) = H(X) + H(Y|X)$
  $H(X_1 ... X_n) = H(X_1) + H(X_2|X_1) + ... + H(X_n|X_1 ... X_{n-1})$

Example 3

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

Information in transitions of *Bandersnatch*:

- Surprisal of n given a = 2.45 bits
- Surprisal of d given n = 2.47 bits

Remember average surprisal of a character, $H(X)$, was 4.05 bits.
$H(X|Y)$ turns out to be about 2.8 bits.

What about Example 4?

Thank you, it's a very interesting dance to watch,' said Alice, feeling very glad that it was over at last.

To make predictions about when we insert *that* we need to think about **entropy rate**.

Entropy of a language is the **entropy rate**

- Language is a stochastic process generating a sequence of word tokens
- The entropy of the language is the entropy rate for the stochastic process:

$$H_{rate}(L) = \lim_{n \to \infty} \frac{1}{n} H(X_1 \ldots X_n)$$

- The entropy rate of language is the limit of the entropy rate of a sample of the language, as the sample gets longer and longer.

Hypothesis: **constant** rates of information are preferred

- The **capacity** of a communication **channel** is the number of bits on average that it can transmit
- Capacity defined by the noise in the channel—mutual information of the channel input and output (more next week)

- Assumption: language users want to maximize information transmission while minimizing comprehender difficulty.
- Hypothesis: language users prefer to distribute information uniformly throughout a message
- Entropy Rate Constancy Principle (Genzel & Charniak), Smooth Signal Redundancy Hypothesis (Aylett & Turk), Uniform Information Density (Jaeger)

Hypothesis: **constant** rates of information are preferred

Could apply the hypothesis at all levels of language use:

- In speech we can modulate the duration and energy of our vocalisations
- For vocabulary we can choose longer and shorter forms
  maths vs. *mathematics*, *don't* vs. *do not*
- At sentence level, we may make syntactic reductions:
  The rabbit (that was) chased by Alice.

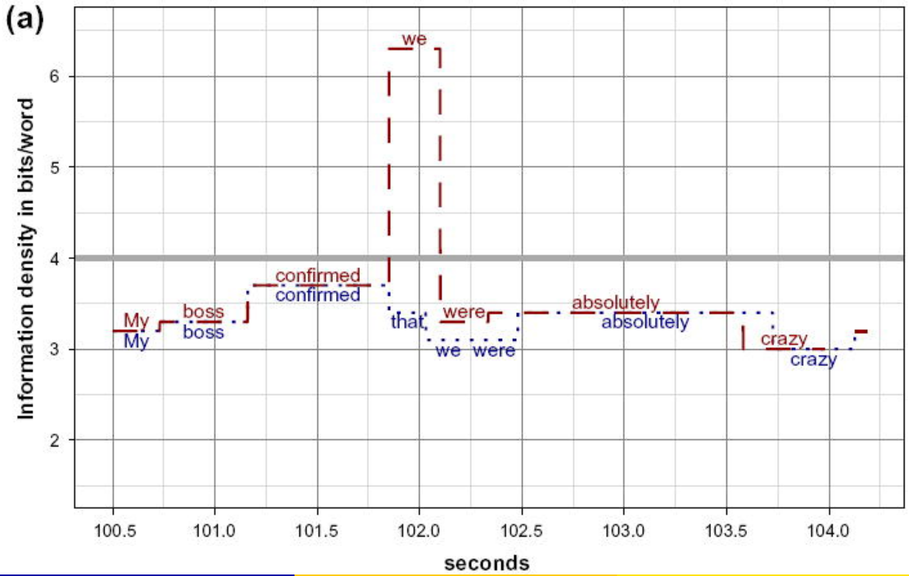
Hypothesis: **constant** rates of information are preferred

Uniform Information Density:

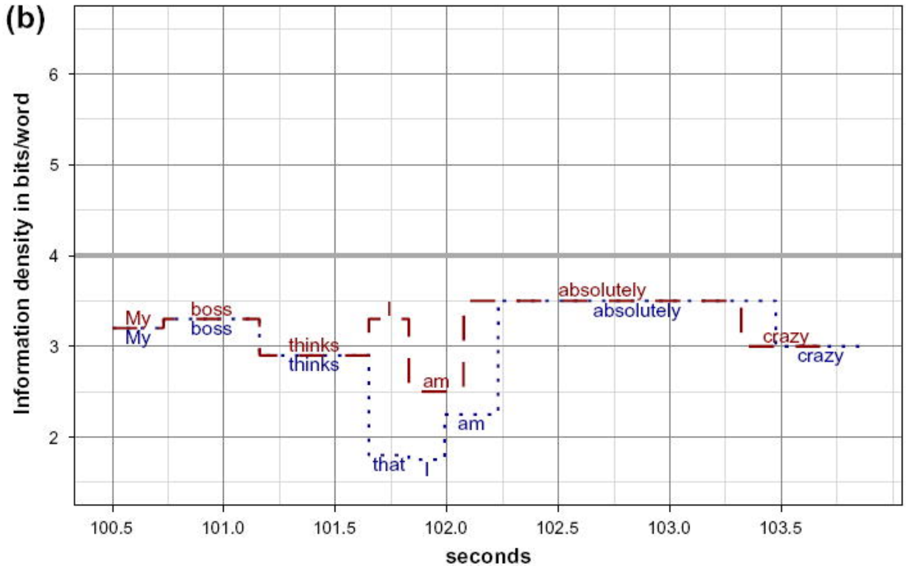
- Within the bounds defined by grammar, speakers prefer utterances that distribute information uniformly across the signal
- Where speakers have a choice between several variants to encode their message, they prefer the variant with more uniform information density

Evaluated on a large scale corpus study of complement clause structures in spontaneous speech (Switchboard Corpus of telephone dialogues)

Hypothesis: **constant** rates of information are preferred



Hypothesis: **constant** rates of information are preferred



- Notice that these information theoretic accounts are rarely explanatory (doesn't explicitly tell us what might be happening in the brain)
- An exception is Hale (2001) where we used surprisal to reason about parse trees and full parallelism
- Information theoretic accounts are unlikely to be the full story but they are predictive of certain phenomena

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

For communication, information has to be **transmitted**

Goal: To optimise, in terms of **throughput** and **accuracy**, the communication of messages in the presence of a **noisy channel**

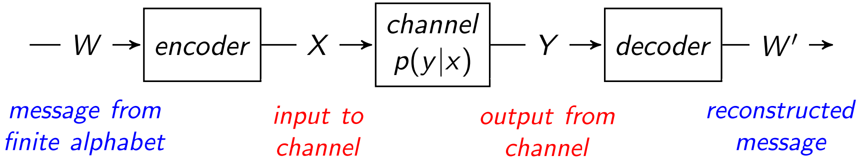
There is a trade off between:

- **compression**: making the most efficient code by removing all the redundancy
- **accuracy**: adding redundant information so that the input can still be recovered despite the presence of noise

Today we will:

- formalise the noisy channel more carefully
- look at some implications for natural language evolution
- see how the noisy channel model has inspired a framework for solving problems in Natural Language Processing.

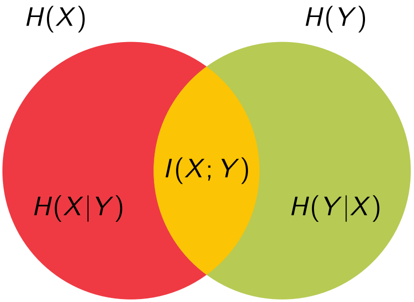
Transmission can be modelled using a **noisy channel**



- message should be efficiently encoded but with enough redundancy for the decoder to detect and correct errors
- the output depends probabilistically on the input
- the decoder finds the mostly likely original message given the output received

Mutual information: the information Y contains about X

- **Mutual Information** $I(X;Y)$ is a measure of the reduction in uncertainty of one random variable due to knowing about another
- Can also think of $I(X;Y)$ being the amount of information one random variable contains about another



$H(X)$ average information of input

$H(Y)$ average information in output

$H(X|Y)$ the uncertainty in (extra information needed for) X given Y is known

$I(X;Y)$ the mutual information; the information in Y that tells us about X

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \end{aligned}$$

Channel **capacity** is determined by **mutual information**

- The capacity of a channel is the maximum of the mutual information of X and Y over all input distributions of the input $p(X)$

$$C = \max_{p(X)} I(X;Y)$$

- C is the rate we can transmit information through a channel with an arbitrarily low probability of not being able to recover the input from the output
- As long as transmission rate is less than C we don't need to worry about errors (optimal rate is C)
- If transmission rate exceeds C then we need to slow down (e.g. by inserting a *that*—last lecture)
- In practical applications we reach the channel capacity by designing an encoding for the input that maximises mutual information.

  What might this mean for the evolution of natural languages?

Piantadosi et al.—ambiguity has a communicative benefit

- If we are trying to maximise mutual information why has natural language evolved to be so ambiguous?

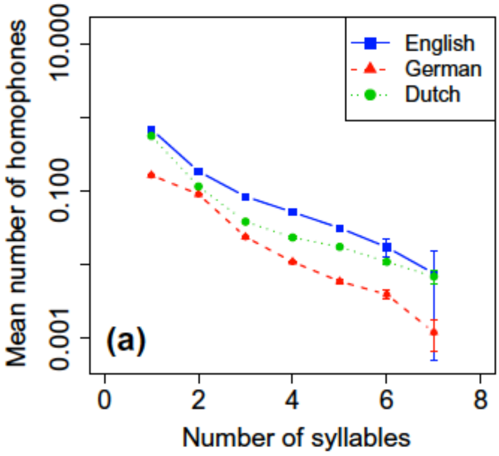
- Efficient communication systems will necessarily be globally ambiguous when context is informative about meaning.

- Notice that **ambiguity is not an issue in normal language** use and overloaded linguistic units are only ambiguous out of context:
  - Alice wanted **to** cry
  - Alice went **to** the garden
  - Alice saw **two** rabbits
  - Dinah saw some rabbits **too**.

- It is optimal to overload simple units for efficient transmission (we can assign the short efficient codes more than once and re-use them)

Piantadosi et al.—ambiguity has a communicative benefit

Some evidence to support the argument found in corpora: shorter words have more meanings



Implication: there must be enough information in the context to allow for the ambiguity in the simple units as well as any other noise in the channel.

Gibson et al.—a noisy channel can account for word order

Word order can provide context that is informative about meaning—this might account for observed word order in the world's languages

Most languages (out of 1,056 studied) exhibit one of two word orders:

- **subject-verb-object** (SVO) — 41% of languages
  the girl chases the rabbit (e.g. English)
- **subject-object-verb** (SOV) — 47% of languages
  the girl the rabbit chases (e.g. Japanese)

- For interest, 8% exhibit **verb-subject-object** (VSO) e.g. Welsh and Irish and 96% of languages have the subject before the object

Gibson et al.—noisy channel account of word order

- Experimental observations:
  - English speakers (SVO) were shown animations of simple events and asked to describe them using only gestures
  - For events in which a human acts on an inanimate object most participants use SOV despite being SVO speakers (e.g. *girl ball kicks*)
  - For events in which a human acts on another human most participants use SVO (e.g. *girl kicks boy*)
  - Preference in each case is around 70%
- Previous experiments show human preference for linguistic recapitulation of old information before introducing new information
- This might explain SOV gestures for SVO speakers—the verb is new information the people/objects are not.
- So why still use SVO for the animate-animate events? And why is English SVO?

Gibson et al.—noisy channel account of word order

Argument is that SVO ordering has a better chance of preserving information over a noisy channel.

- Consider the scenario of a girl kicking a boy
- Now let one of the nouns get lost in transmission.
- If the word order is SOV (*the girl the boy kicks*), the listener receives either:
  the girl kicks or *the boy kicks*
- If the word order is SVO (*the girl kicks the boy*) the listener receives either:
  the girl kicks or *kicks the boy*
- In the SVO case more information has made it through the noisy channel (preserved in the position relative to the verb)

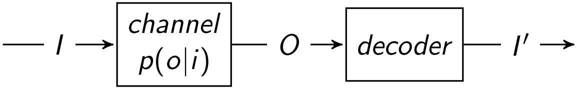
Gibson et al.—noisy channel account of word order

Further evidence for the argument is presented from the finding that there is a correlation between word order and case markings.

- Case marking means that words change depending on their syntactic function: e.g. *she* (subject), *her* (object)
- Case marking is rare in SVO languages (like English) and more common in SOV languages
- Suggestion is that when there are other information cues as to which noun is subject and which is object speakers can default to any natural preference for word order.

In Natural Language Processing, however, our starting point is **after** the evolutionary natural language encoding.

Noisy channel inspired an NLP problem-solving **framework**


$$\begin{tikzcd} {} \arrow[r, "I"] & \boxed{\begin{array}{c}\textit{channel}\\ p(o|i)\end{array}} \arrow[r, "O"] & \boxed{\textit{decoder}} \arrow[r, "I'"] & {} \end{tikzcd}$$

- Many problems in NLP can be framed as trying to find the most likely input given an output: $I' = \underset{i}{\operatorname{argmax}}\, p(i|o)$
- $p(i|o)$ is often difficult to estimate directly and reliably, so use Bayes' theorem: $p(i|o) = \frac{p(o|i)p(i)}{p(o)}$
- Noting that $p(o)$ will have no effect on argmax function:
$$I' = \underset{i}{\operatorname{argmax}}\, p(i|o) = \underset{i}{\operatorname{argmax}}\, p(i)p(o|i)$$
- $p(i)$ is the probability of the input (a **language model**)
- $p(o|i)$ is the **channel probability** (the probability of getting an output from the channel given the input)

SMT is an intuitive (non-SOTA) example of noisy channel

We want to translate a text from English to French:

$$\begin{tikzcd} {} \arrow[r, "French"] & \boxed{\begin{array}{c}\textit{channel}\\ p(e|f)\end{array}} \arrow[r, "English"] & \boxed{\textit{decoder}} \arrow[r, "French'"] & {} \end{tikzcd}$$

- In statistical machine translation (SMT) noisy channel model assumes that original text is in French
- We pretend the original text has been through a noisy channel and come out as English (the word *hello* in the text is actually *bonjour* corrupted by the channel)
- To recover the French we need to decode the English:

  $$f' = \underset{f}{\operatorname{argmax}}\, p(f|e) = \underset{f}{\operatorname{argmax}}\, p(f)p(e|f)$$

SMT is an intuitive (non-SOTA) example of noisy channel

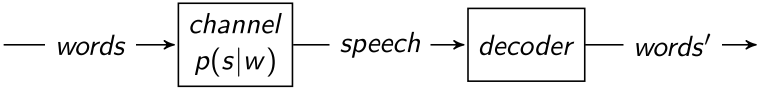
Recover the French by decoding the English: $f' = \underset{f}{\operatorname{argmax}}\, p(f)p(e|f)$

$$\begin{tikzcd} {} \arrow[r, dash, "\textit{French}"] & \boxed{\begin{matrix}\textit{channel}\\ p(e|f)\end{matrix}} \arrow[r, "\textit{English}"] & \boxed{\textit{decoder}} \arrow[r, "\textit{French}'"] & {} \end{tikzcd}$$

- $p(f)$ is the **language model**.
  - ensures **fluency** of the translation (e.g. a very large n-gram model)
- $p(e|f)$ is the **translation model**.
  - ensures **fidelity** of the translation (derived from very large parallel corpora)

Noisy channel framework influenced many applications

Speech Recognition: recover word sequence by decoding the speech signal



$$words' = \underset{words}{\operatorname{argmax}}\, p(words)p(speech\_signal|words)$$

- $p(words)$ is the **language model** (e.g. n-gram model)
- $p(speech\_signal|words)$ is the **acoustic model**.

Noisy channel framework influenced many applications

Part-of-Speech Tagging:

$$\begin{tikzcd} {} \arrow[r, "tags"] & \boxed{\begin{array}{c} \textit{channel} \\ p(w|t) \end{array}} \arrow[r, "words"] & \boxed{\textit{decoder}} \arrow[r, "tags'"] & {} \end{tikzcd}$$

$$tags' = \underset{tags}{\operatorname{argmax}}\, p(tags)p(words|tags)$$

Optical Character Recognition:

$$\begin{tikzcd} {} \arrow[r, "words"] & \boxed{\begin{array}{c} \textit{channel} \\ p(e|w) \end{array}} \arrow[r, "errors"] & \boxed{\textit{decoder}} \arrow[r, "words'"] & {} \end{tikzcd}$$

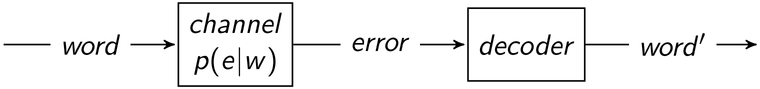
$$words' = \underset{words}{\operatorname{argmax}}\, p(words)p(errors|words)$$

Spelling can be corrected using the noisy channel method

- There are two types of spelling error:
  - **non-word** errors: *Alcie* instead of *Alice*
  - **real-word** errors: *three* instead of *there*, *there* instead of *their*
- For illustration we will show how to use a noisy channel model to correct non-word errors
- Non-word correction is a significant problem:
  - 26%: of web queries *Wang et al.*
  - 13%: errors when asked to retype rather than backspace *Whitelaw et al.*
- **Detection** of non-word errors is easy (they fail to appear in a dictionary)
- The best candidate for **correction** will be the item that maximises the noisy channel equation.

Spelling can be corrected using the noisy channel method

Spelling correction:



$$word' = \underset{word}{\operatorname{argmax}}\, p(word)p(error|word)$$

- $p(word)$ can be obtained from a corpus
- $p(error|word)$ can be modelled using minimum text edit distance or minimum pronunciation distance (the probability of the edit)

Spelling can be corrected using the noisy channel method

- Damerau-Levenshtein is edit distance model that counts: *insertions*. *deletions*, *substitutions*, *transpositions*

error	candidate correction	corrected letters	error letters	type
acress	actress	t		deletion
acress	cress	-	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	-	s	insertion
acress	acres	-	s	insertion

- 80% of errors are within edit distance 1
- Almost all errors within edit distance 2

Spelling can be corrected using the noisy channel method

- $p(error|word)$ may be calculated from confusion tables created from error annotated training data
- e.g. substitution(x,w) confusion matrix

	a	b	c	d	e	...
a	0	0	7	1	342	...
b	0	0	9	9	2	...
c	6	5	0	16	0	...
d	1	10	13	0	12	...
e	338	0	3	11	0	...

- if misspelled word is $x = x_1, x_2 ... x_m$
- and corrected word is $w = w_1, w_2 ... w_n$
- If proposed edit at x_i is a substitution $p(x|w) = \frac{substitution(x_i, w_i)}{count(w_i)}$
- similar equations for a *deletion*, *insertion* and *transposition*

Spelling can be corrected using the noisy channel method

- For typo *acress* chosen word is

$$= \underset{word}{\operatorname{argmax}}\, p(word|error) = \underset{word}{\operatorname{argmax}}\, p(word)p(error|word)$$

candidate	corr	err	$x\|w$	$p(x\|w)$	p(w)	$p(x\|w)p(w)10^9$
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	.00078
caress	ca	ac	ac|ca	.00000164	.00000170	.0028
access	c	r	r|c	.000000209	.0000916	.019
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Noisy channel could be used to model comprehension

- For many natural language applications, noisy channel models have been surpassed by data hungry sequence-to-sequence neural models (more in *NLP* course next year)
- Natural language communication is an area where the model might still yield research insights
- Classic assumption in sentence processing: input to the parser is an error-free sequence of words (e.g. Hale and Yngve)
- This assumption is problematic because we are communicating across a noisy channel
- The ultimate interpretation of a sentence should depend on the proximity of plausible alternatives under the noise model
- This could be modelled in terms of insertions and deletions (just like spelling correction)...

Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

- What is examinable?
- What will the exam be like?
- Emails?

You shall know a word by the **company** it keeps—Firth

Consider the following sentences about the *rabbit* in *Alice in Wonderland*:

- *Suddenly a white rabbit with pink eyes ran close by her.*
- *She was walking by the white rabbit who was peeping anxiously into her face.*
- *The rabbit actually took a watch out of its waistcoat pocket and looked at it.*
- *'Oh hush', the rabbit whispered, in a frightened tone.*
- *The white rabbit read out at the top of his shrill little voice the name Alice.*

We learn a lot about the rabbit from the words in the local context.

You shall know a word by the **company** it keeps—Firth

- So far, we have been discussing grammars with discrete alphabets and algorithms that have discrete symbols as input.
- Many Natural Language Processing tasks require some notion of similarity between the symbols.

  e.g. *The queen looked angry. Her majesty enjoyed beheading.*

  To understand the implication of these sentences we need to know that *the queen* and *her majesty* are similar ways of expressing the same thing.
- Instead of symbols we can represent a word by a collection of key words from its context (as a proxy to its meaning)

  e.g instead of rabbit we could use

  rabbit = {white, pink, eyes, voice, read, watch, waistcoat, ...}

You shall know a word by the **company** it keeps—Firth

- But which key words do we include in the collection?
- We could look at a $\pm n$-word context **window** around the **target** word.
- We could select (and weight) keywords based on their frequency in the window:

  rabbit = {the 56, white 22, a 17, was 11, in 10, it 9, said 8, and 8, to 7...}

- This would become a little more informative if we removed the function words:

  rabbit ={white 22, said 8, alice 7, king 4, hole 4, hush 3, say 3, anxiously 2...}

  queen ={said 21, king 6, shouted 5, croquet 4, alice 4, play 4, hearts 4, head 3... }

  cat ={said 19, alice 5, cheshire 5, sitting 3, think 3, queen 2, vanished 2, grin 2...}

- This is all just illustrative, we can of course, do this for all words (not just the characters)—called distributional semantics.

We can replace symbols with **vector representations**

- Two words can be expected to be semantically similar if they have similar word co-occurrence behaviour in texts.

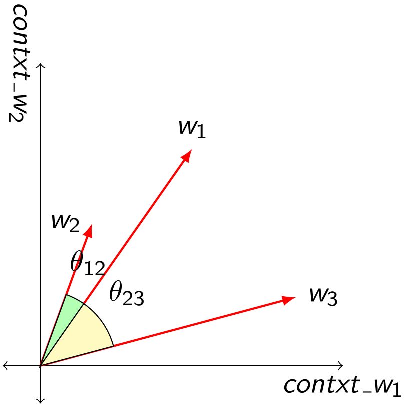
  e.g. in large amounts of general text we would expect *queen* and *monarch* to have similar word co-occurrences.
- Simple collections of context words don't help us easily calculate any notion of similarity.
- A trend in modern Natural Language Processing technology is to replace symbolic representation with a **vector representation**
- Every word is encoded into some vector that represents a point in a multi-dimensional word space.

	alice	croquet	grin	hurried	king	say	shouted	vanished
rabbit	7	0	0	2	4	3	0	1
queen	4	4	0	1	6	1	5	0
cat	5	1	2	0	0	0	0	2

We can replace symbols with **vector representations**

- Note that there is an issue with polysemy (words that have more than one meaning):
- E.g. we have obtained the following vector for cat:
  cat = [5, 1, 2, 0, 0, 0, 0 2]
- But cat referred to two entities in our story:
  I wish I could show you our cat Dinah
  I didn't know that Cheshire cats always grinned in fact I didn't know that cats could grin

- The vector provides the coordinates of point/vector in the multi-dimensional word space.
- Assumption: **proximity in word space** correlates with **similarity** in meaning
- Similarity can now be measured using distance measures such as Jaccard, Cosine, Euclidean...



- e.g. cosine similarity
  $\text{cosine}(\mathbf{v_1}, \mathbf{v_2}) = \frac{\mathbf{v_1} \cdot \mathbf{v_2}}{\|\mathbf{v_1}\| \|\mathbf{v_2}\|}$
- Equivalent to dot product of normalised vectors (not affected by magnitude)
- cosine is 0 between orthogonal vectors
- cosine is 1 if $v_1 = \alpha v_2$, where $\alpha > 0$

Automatically derived vectors will be very **large** and **sparse**

- In certain circumstances we might select dimensions **expertly**
- For general purpose vectors we want to simply count in a large collection of texts, the number of times each word appears inside a window of a particular size around the target word.
- This leads to very **large sparse** vectors (remember Zipf's law)
- There are an estimated 13 *million* tokens for the English language—we can reduce this a bit by removing (or discounting) function words, grouping morphological variants (e.g, *grin*, *grins*, *grinning*)
- Is there some k-**dimensional space** (such that $k << 13 \textit{million}$) that is sufficient to encode the word meanings of natural language?
- Dimensions might hypothetically encode tense (past vs. present vs. future), count (singular vs. plural), and gender (masculine vs. feminine)...

It is possible to **reduce** the **dimensions** of the vector

To find reduced dimensionality vectors (usually called **word embeddings**)

- Loop over a massive dataset and accumulate word co-occurrence counts in some form of a large sparse matrix X (dimensions n x n where n is vocabulary size)
- Perform **Singular Value Decomposition** on X to get a USV^T decomposition of X.

$$\begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & X & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} = \begin{bmatrix} u_{11} & \vdots & \vdots & \vdots \\ \vdots & u_2 & \dots & u_n \\ u_{1n} & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} s_1 & 0 & 0 & \dots \\ 0 & s_2 & 0 & \dots \\ 0 & 0 & \ddots & \dots \\ \vdots & \vdots & \vdots & s_n \end{bmatrix} \begin{bmatrix} v_{1n} & \dots & v_{1n} \\ \dots & v_2 & \dots \\ \dots & \vdots & \dots \\ \dots & v_n & \dots \end{bmatrix}$$

It is possible to **reduce** the **dimensions** of the vector

- Note S matrix has diagonal entries only.
- Cut diagonal matrix at index k based on desired dimensionality (can be decided by desired percentage variance): $(\sum_{i=1}^{k} s_i)/(\sum_{i=1}^{n} s_i)$

$$\begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & X' & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} = \begin{bmatrix} u_{11} & \vdots & \vdots \\ \vdots & \dots & u_k \\ u_{1n} & \vdots & \vdots \end{bmatrix} \begin{bmatrix} s_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & s_k \end{bmatrix} \begin{bmatrix} v_{1n} & \dots & v_{1n} \\ \dots & \vdots & \dots \\ \dots & v_k & \dots \end{bmatrix}$$

- Use rows of U for the word embeddings.
- This gives us a k-dimensional representation of every word in the vocabulary.

It is possible to **reduce** the **dimensions** of the vector

Things to note:

- Need all the counts before we do the SVD reduction.
- The matrix is extremely **sparse** (most words do not co-occur)
- The matrix is very **large** ($\approx 10^6 x 10^6$)
- SVD is **quadratic**

Points of methodological variation:

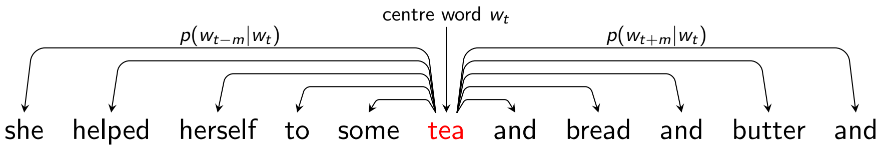
- Due to Zipf distribution of words there is **large variance** in co-occurrence frequencies (need to do something about this e.g. discount/remove stop words)
- Refined approaches might **weight** the co-occurrence counts based on distance between the words

Predict models can be more efficient than **count** models

- **word2vec** is a **predict model**, in contrast to the distributional models already mentioned which are **count models**.
- Instead of computing and storing a large matrix from a very large dataset, use a model that learns **iteratively**, eventually encoding the probability of a word given its context.
- The parameters of the model are the word embeddings.
- The model is trained on a certain objective.
- At every iteration we run our model, evaluate the errors, and then adjust the model parameters that caused the error.

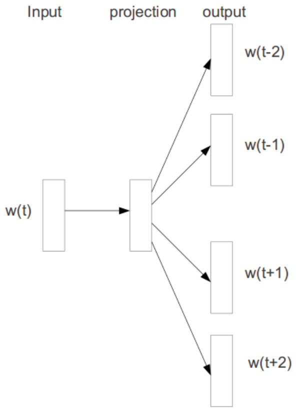
Predict models can be more efficient than **count** models

- Two simple word2vec architectures:
- **Continuous Bag of Words** CBOW: given some context word embeddings, predict the target word embedding.
- **Skip-gram**: given a target word embedding, predict the context word embeddings (below).



- **skip-gram** model predicts relationship between a centre word w_t and its context words: $p(context|w_t) = \ldots$
- Predict context word embeddings based on the target word embedding.
- A loss function is used to score the prediction (usually **cross-entropy** loss function).

  (Cross-entropy measures the information difference between the expected word embeddings and the predicted ones.)
- Adjust the word embeddings to minimise the loss function.
- Repeat over many positions of t in a very big language corpus.



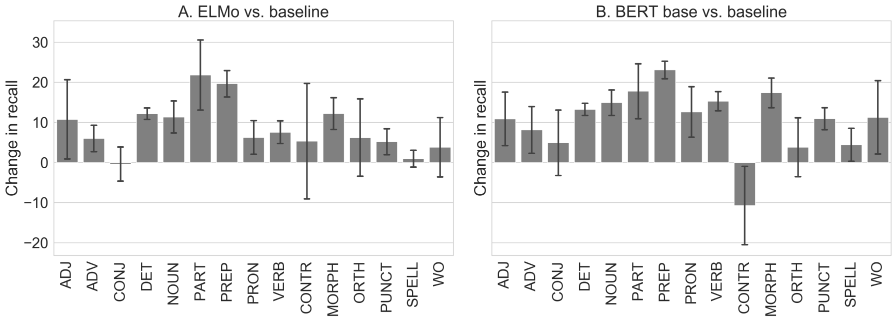
Distributional models have improved NLP applications

In general, distributional models have had a positive impact on NLP and provided improvement over symbolic systems:

- There has been a change in state-of-the-art for some applications: (e.g. Google Translate)
- Multi-modal experiments have become more straightforward (by combining vector representations)
- But these models are statistical (need very large amounts of data and have to find a way to handle unseen words)
- There has been a lot of hype and not much work on the problems the distributional models can't solve.

Methods for predict word models are fast moving research

- There are many different methods for training word embeddings.
- A method can be considered better than a previous method if it gives us an improvement for a task.
- e.g. using contextual embeddings for grammatical error detection



- *Part III project by Sam Bell 2019*

Predict models versus **count** models

- + Predict models can be more efficient than count models because we can learn **iteratively** and don't have to hold statistics on the whole dataset.
- − Need to **initialise** the word embeddings (several possible methods).
- ± The size of the embeddings is a chosen parameter of the system (usually a few hundred).
- + Predict models are learning structure **without hand-crafting** of features.
- − Dimensionality of the embeddings are assumed to capture meaningful generalisations, but the dimensions are **not directly interpretable**.

Predict models versus **count** models

- After training, predict models are found to be equivalent to a count model with dimensionality reduction.
- Tuning hyper-parameters is a matter of much (often brute-force) experimentation.
- Predict models perform better than count models with dimensionality on some tasks (but perhaps due to tuning hyper-parameters).
- For some tasks count vectors without dimensionality reduction are the most effective.

Word embeddings can **correlate** with human **intuitions**

Researchers test their word embeddings against datasets of **human similarity judgements**:

- For a test set of words, participants rate word pairs for **relatedness** (e.g. Miller & Charles, Rubenstein & Goodenough)
- A rank of relatedness can be drawn up between items in the test set.
- A **rank correlation** between embeddings and human judgements can be calculated.
- Good embeddings have a correlation of 0.8 or better with the human judgements.

Reasoning may be possible based on word embeddings

- *Mikolov et al.* analogy puzzles:

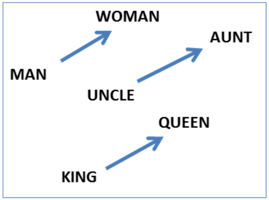
  Can we use word embeddings to solve puzzles such as:

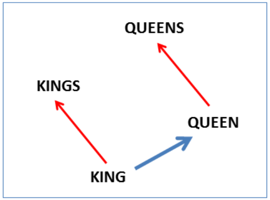
  man is to woman as king is to .... queen

- Can we do vector-oriented reasoning based on the offsets between words?

Reasoning may be possible based on word embeddings

- Derive the vector between the pair of words *man* and *woman* and then add it to *king*.
- The nearest word to the region of vector space that results will be the answer to the analogy.

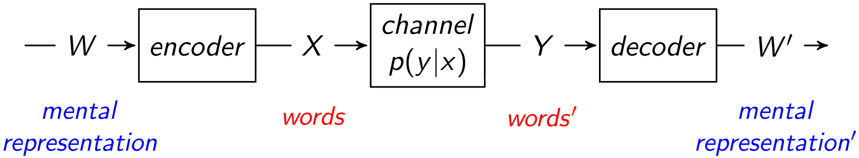




- *Mikolov* found that word2vec embeddings are good at capturing syntactic and semantic regularities in language, and that each relationship is characterised by a relation-specific vector offset.
- Note that the space is very sparse and that there are word pairs for which this does not work...

Relationship between embeddings and **brain activity**?

- Humans have the capacity to translate thoughts into words, and to infer others' thoughts from their words.
- There must be some mental representations of meaning that are mapped to language, but we have no direct access to these representations.



- Do word embeddings provide a model that successfully captures some aspects of our mental representation of meaning?

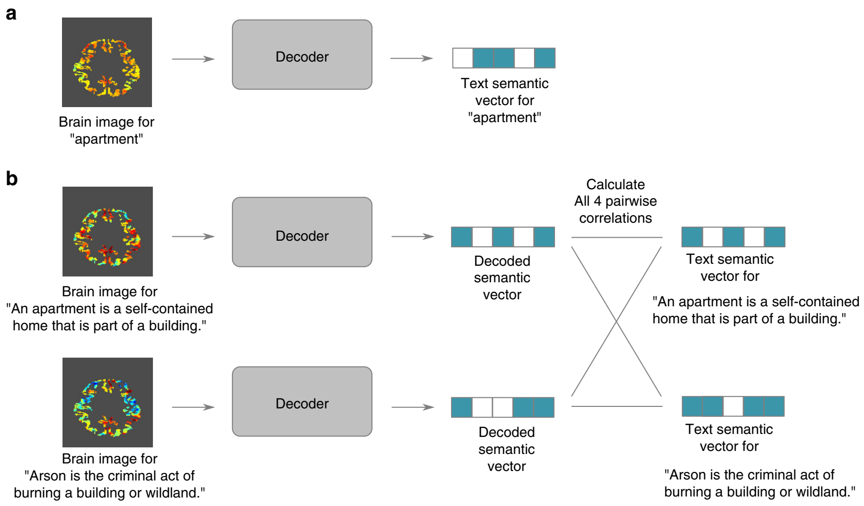
Relationship between embeddings and **brain activity**?

- Natural language appears to be a discrete symbolic system.
- The brain encodes information through continuous signals of activation.
- Language symbols are transmitted via continuous signals of sound/vision.

- *Pereira et al.* trained a system using brain imaging data and word embeddings.
- Demonstrated the ability to generalise to new meanings from limited imaging data.

`https://www.nature.com/articles/s41467-018-03068-4`

Relationship between embeddings and **brain activity**?


a
Brain image for "apartment"
Decoder
Text semantic vector for "apartment"
b
Brain image for "An apartment is a self-contained home that is part of a building."
Decoder
Decoded semantic vector
Calculate All 4 pairwise correlations
Text semantic vector for "An apartment is a self-contained home that is part of a building."
Brain image for "Arson is the criminal act of burning a building or wildland."
Decoder
Decoded semantic vector
Text semantic vector for "Arson is the criminal act of burning a building or wildland."