UNIVERSITY OF CAMBRIDGE

Lecture 7: Computational Syntax

Suchir Salhan

Faculty of Modern and Medieval Languages and Linguistics

sas245@cam.ac.uk

Li18 Computational Linguistics

Introduction

We have explored various kinds of models:

- **Lecture 2: Finite-State Automata**
- **Lecture 3: Finite-State Transducers**
- Lecture 4: N-gram Language Models
- Lecture 5: Bag-of-words Naive Bayes
- Lecture 6: Hidden Markov Models

We have explored computational models of morphology (and phonology)

*Now, we turn to **syntax.***

Syntactic Theory and Computational Syntax

A theory of syntax should *characterise* which sentences are well-formed (grammatical) and which are not.

NB: descriptive v explanatory adequacy (Chomsky 1965, *Aspects of the Theory of Syntax*)

Syntactic Theory and Computational Syntax

A theory of syntax should ***characterise*** which sentences are well-formed (grammatical) and which are not:

- Note that well-formed is distinct from being meaningful

Computational Syntax explores a wide range of **formalisms** that represent syntactic processes and derivation.

Syntactic Theory and Computational Syntax

A theory of syntax should *characterise* which sentences are well-formed (grammatical) and which are not:

- Note that well-formed is distinct from being meaningful

Computational Syntax explores a wide range of **formalisms** that represent syntactic processes and derivation.

Computational linguists are interested in applying methods and formalisms (e.g., formal language theory) to model human syntactic competence.

Syntactic Theory and Computational Syntax

A theory of syntax should *characterise* which sentences are well-formed (grammatical) and which are not:

- Note that well-formed is distinct from being meaningful

Computational Syntax explores a wide range of **formalisms** that represent syntactic processes and derivation.

Computational linguists are interested in applying methods and formalisms (e.g., formal language theory) to model human syntactic competence.

Computational linguists also are interested in modelling human language processing ("performance"-related questions)

e.g., *processing relative clauses, prepositional attachment ambiguities*

Some Linguistic Preliminaries: Syntactic Categories and Phrase Structure

We may also want to capture **substitutability** at the phrasal level:

- POS categories indicate which words are substitutable, e.g. substituting **adjectives**:

I saw a red cat
I saw a former cat
I saw a billowy cat

- Phrasal categories indicate which phrases are substitutable, e.g. substituting a **noun phrase**:

Dogs sleep soundly
My next-door neighbours sleep soundly
Green ideas sleep soundly

Linguistic Preliminaries: Long-Distance Dependencies

The form of one word often depends on (agree with) another, even over arbitrarily long distances:

Sam sleeps soundly
Dogs sleep soundly

Sam, who is my cousin, sleeps soundly
Dogs often stay at my house and sleep soundly

Sam, the man with the red hair who is my cousin, sleeps soundly
Dogs in my neighbourhood often stay at my house and sleep soundly

Linguistic Preliminaries: PP Attachment Ambiguities

One common form of **syntactic ambiguity** is Prepositional Phrase Attachment Ambiguity.

Jo saw the man with one eye.
John saw the Mary in the garden with the goldfish.

Linguistic Preliminaries: Filler-Gap Dependencies

A **filler** (usually a wh-word, fronted phrase, or relative pronoun) is interpreted as filling a **gap** (an empty position) later in the sentence, even though the filler appears *at the front* of the clause. Syntacticians formalise constraints (e.g., **island constraints**) on the distribution of filler-gap dependencies.

Subject Gap:
Cheryl thought about who ∅ upset Sandra
Cheryl thought about who **some dog upset Sandra.*

Object Gap:
Joel discovered what Patricia might take ∅.
Joel discovered what Patricia might take **the vase.*

These include **island effects**. Filler-Gap Dependencies can also be long-distance dependencies with intervening constituents.

Computational Syntax, Parsing and Natural Language Processing

To parse a string *s* refers to one of two possible tasks:

- **Recognition:** It may just answer yes/no to answer *Does this sentence conform to the grammar?*
- **Derivation:** Produce a representation – or the most useful representation – of a string.

Computational Syntax, Parsing and Natural Language Processing

To parse a string *s* refers to one of two possible tasks:

- **Recognition:** It may just answer yes/no to answer *Does this sentence conform to the grammar?*
- **Derivation:** Produce a representation – or the most useful representation – of a string.

A parsing algorithm can be used to recognise whether a string is grammatical in a language and the steps to derive the structure of a string by recording the steps taken during recognition.

A parser might represent syntactically ambiguous sentences with distinct parses.

Note that most parsers do not (generally) include information about constituent (argument or verbal) movement.

Note, these syntax trees are MUCH more simplified than those that you will have seen in Li2 (or Li9)!

Computational Syntax, Parsing and Natural Language Processing

Parsing has been a historically important task in Natural Language Processing.

From a modern perspective (e.g., with LLMs), end-to-end systems render the task of parsing superfluous in many ways.

Computational Syntax, Parsing and Natural Language Processing

Parsing has been a historically important task in Natural Language Processing.

From a modern perspective (e.g., with LLMs), end-to-end systems render the task of parsing superfluous in many ways.

BUT …

parsing can be important to understand – or *probe* – structure in Language Models and the task is still important in scenarios where automatic annotation of linguistic structure can be useful.

Revisiting the Chomsky Hierarchy for Syntax

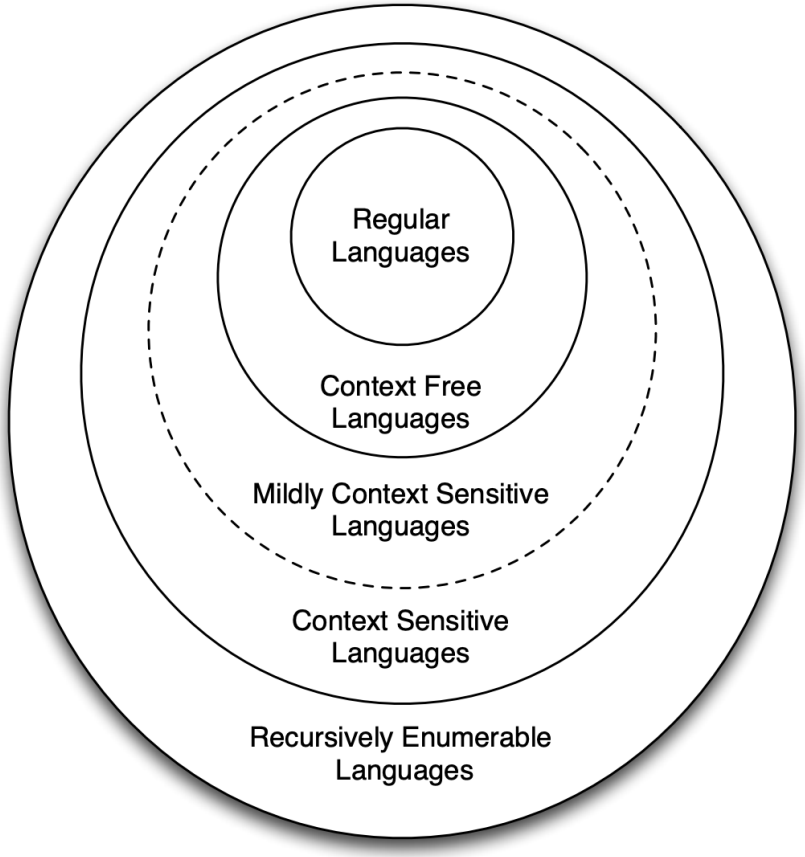


Illustration from Paula Buttery (2025) Notes on Formal Models of Languages: Grammars; https://www.cl.cam.ac.uk/teaching/2425/ForModLang/notes/Grammars_notes.pdf

The Chomsky Hierarchy: Syntax and Regular Grammars

Is syntax regular?

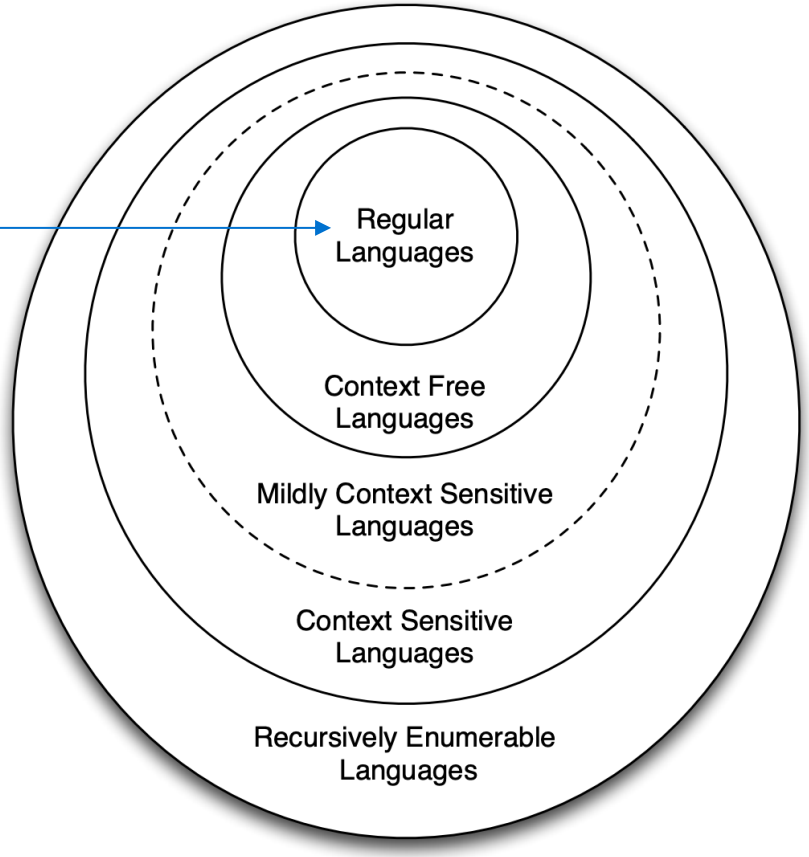
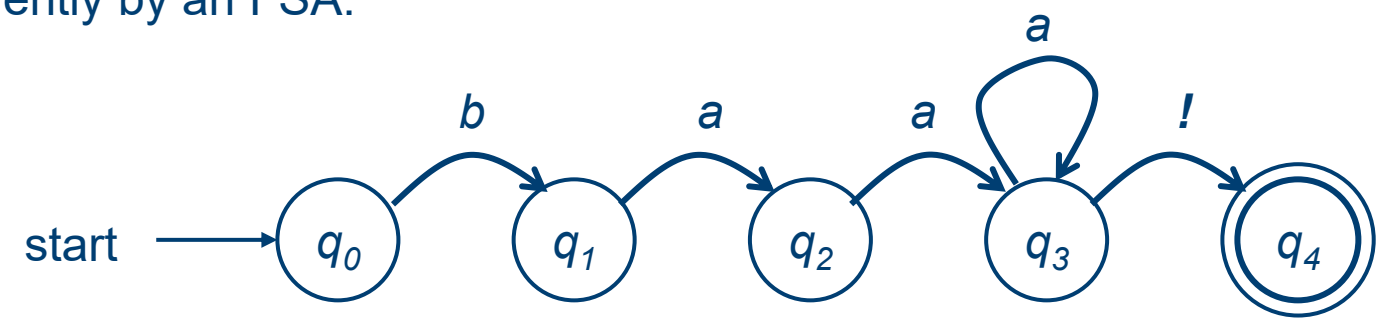


Illustration from Paula Buttery (2025) Notes on Formal Models of Languages: Grammars; https://www.cl.cam.ac.uk/teaching/2425/ForModLang/notes/Grammars_notes.pdf

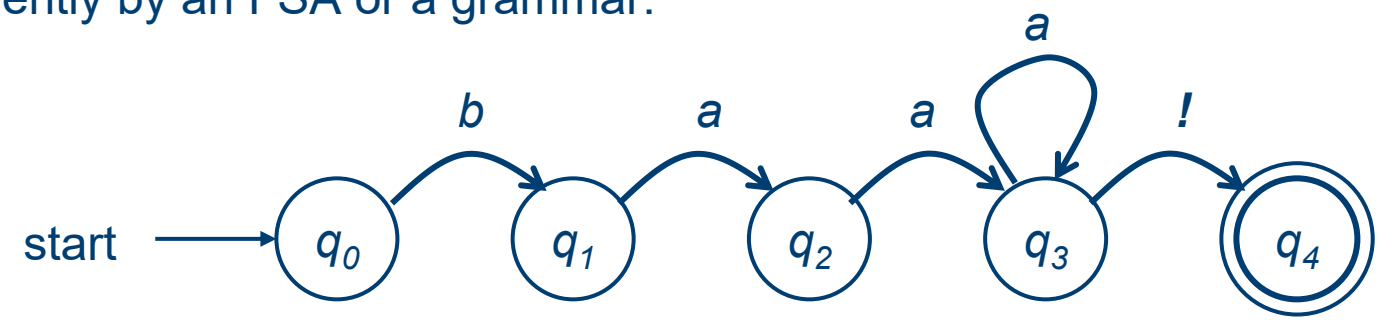
Representing an FSA as a grammar

Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA:



Representing an FSA as a grammar

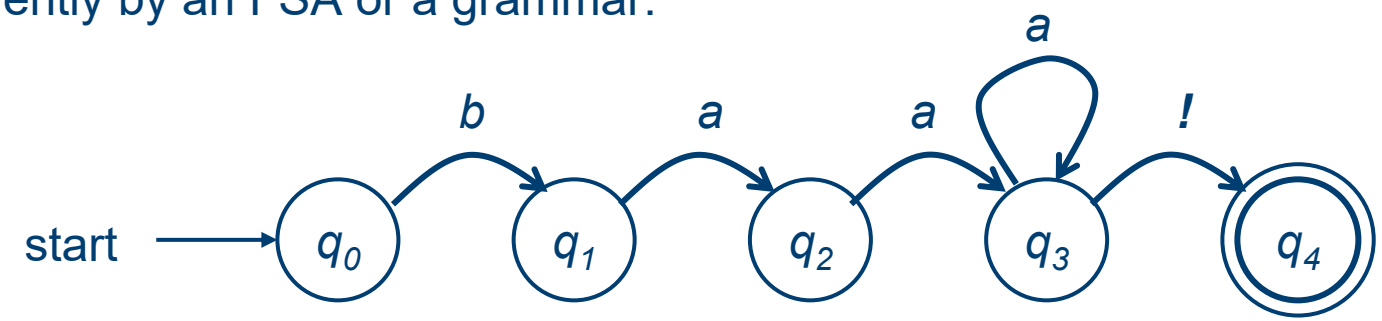
Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA or a grammar:



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

Representing an FSA as a grammar

Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA or a grammar:



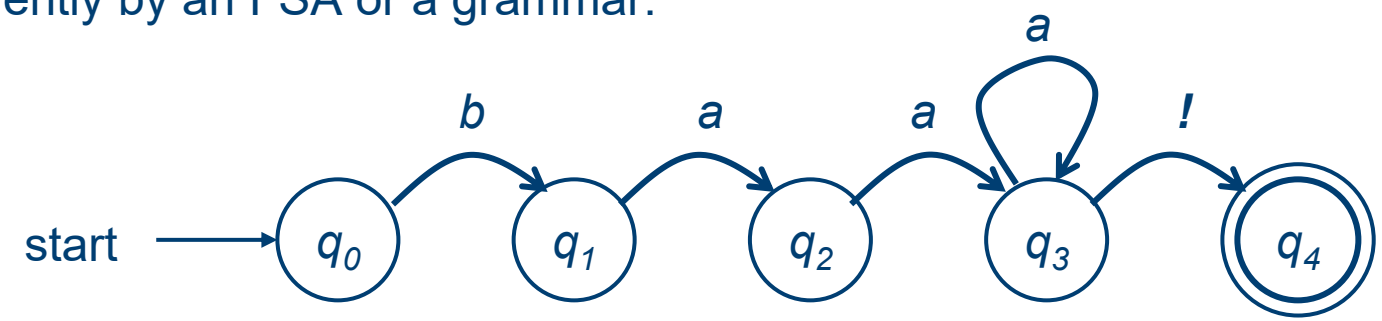
$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

Some terminology:

- production rules

Representing an FSA as a grammar

Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA or a grammar:



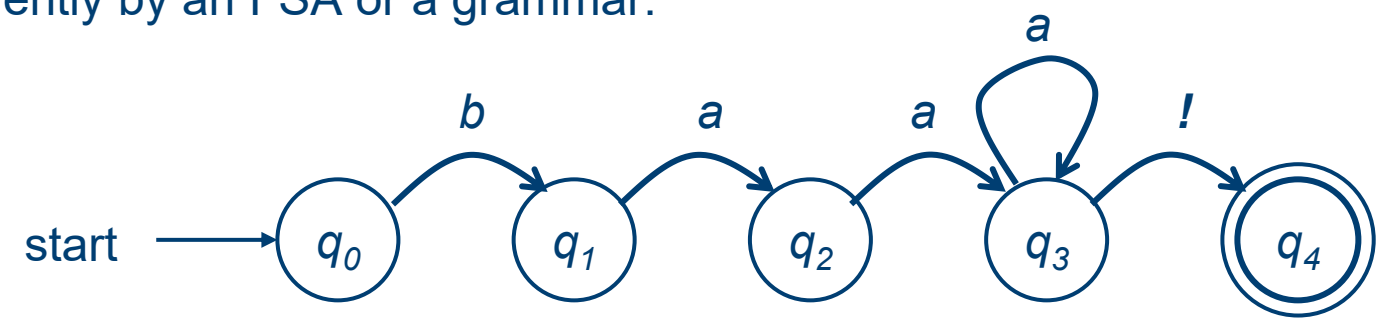
$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

Some terminology:

- production rules
- terminals

Representing an FSA as a grammar

Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA or a grammar:



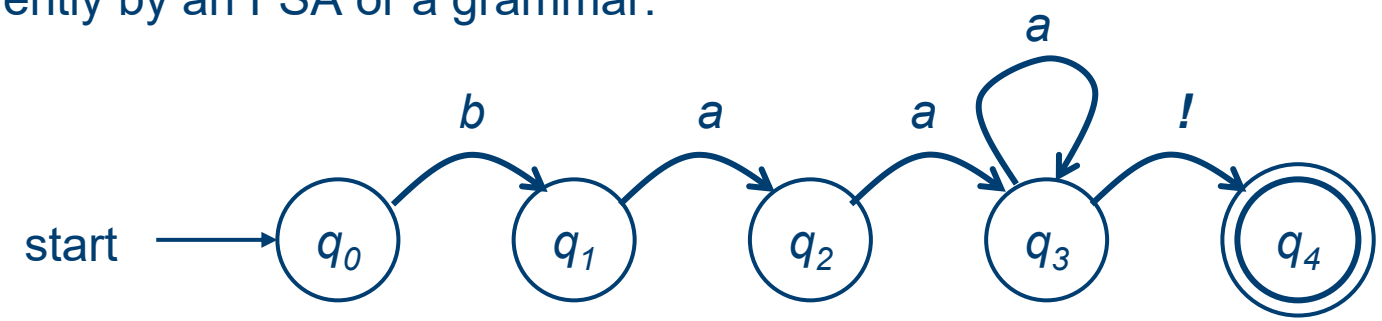
$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

Some terminology:

- production rules
- terminals
- non-terminals

Representing an FSA as a grammar

Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA or a grammar:



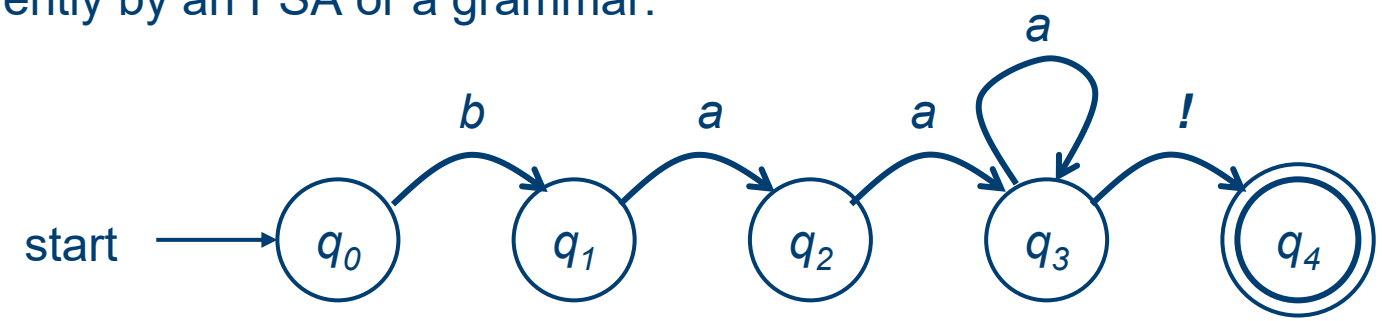
$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

Some terminology:

- production rules
- terminals
- non-terminals
- start symbol

Representing an FSA as a grammar

Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA or a grammar:



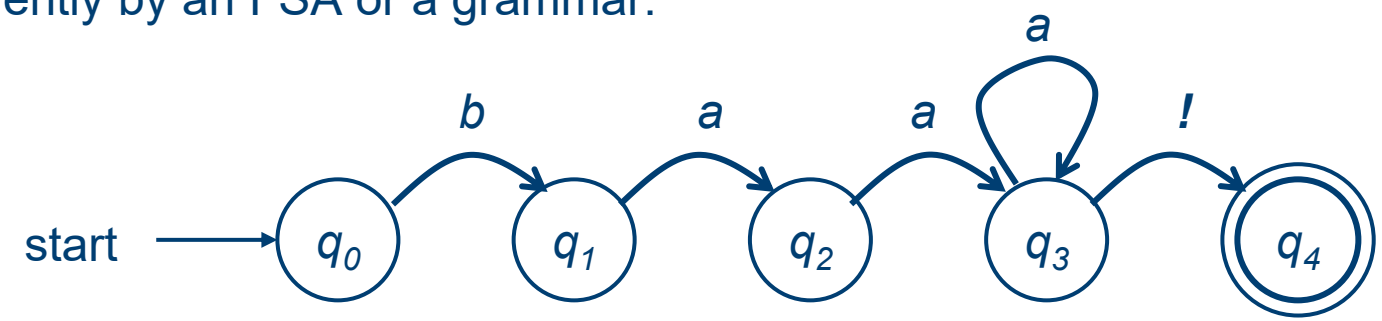
$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

Some terminology:

- production rules
- terminals
- non-terminals
- start symbol
- parents

Representing an FSA as a grammar

Recall the sheep language defined by the regular expression `/baa+!/` and equivalently by an FSA or a grammar:



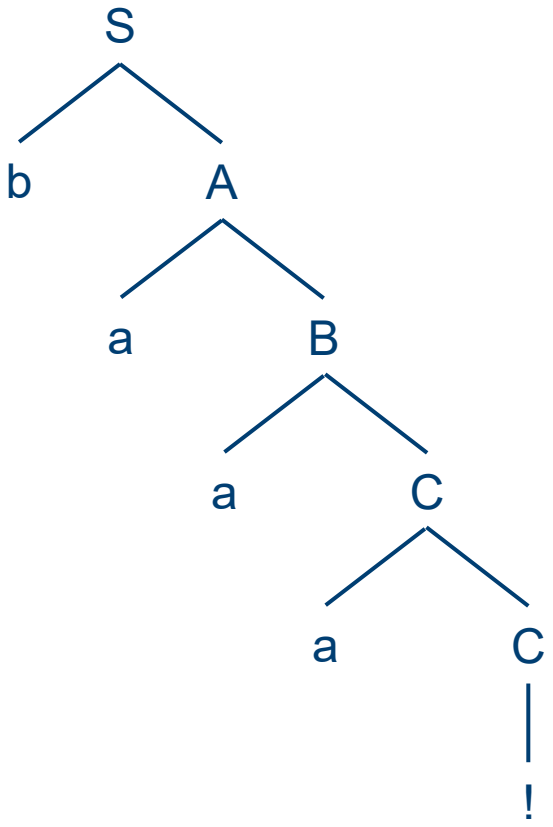
$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

Some terminology:

- production rules
- terminals
- non-terminals
- start symbol
- parents
- children

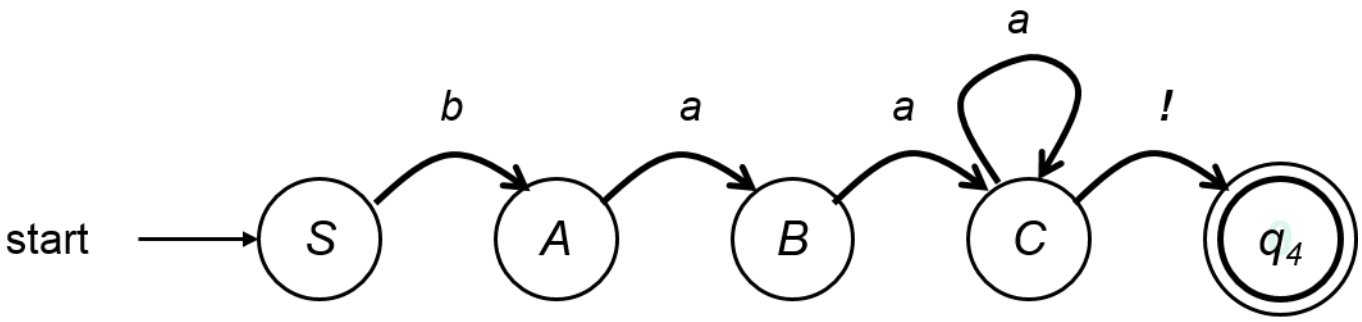
Representing an FSA as a grammar

Rewrite rules naturally give rise to tree structures:



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

Representing an FSA as a grammar



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

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

Formal Grammars

- A formal grammar consists of:
  - A set of terminal symbols
  - A set of non-terminal symbols
  - A start symbol (one of the non-terminals)
  - A set of rewrite rules

Regular Grammars

- In a regular grammar:
  - Each rule has one parent and at most one non-terminal child
  - A non-terminal child is always the last child (or always the first child)

- Regular grammars are equivalent to FSAs and regular expressions:
  - Terminals → alphabet
  - Non-terminals → states (plus an extra final state)
  - Start symbol → start state
  - Rules with a non-terminal child → transitions to a non-final state
  - Rules with no non-terminal child → transitions to the final state

Using a regular grammar to model natural language

Do we need anything more powerful than a regular grammar in order to model natural language?

Centre Embedding: i.e. infinitely recursive structures described by the rule $A \rightarrow aAb$, which generates language examples of the form $a^n b^n$.

Using a regular grammar to model natural language

Do we need anything more powerful than a regular grammar in order to model natural language?

Centre Embedding: i.e. infinitely recursive structures described by the rule A→aAb, which generates language examples of the form $a^n b^n$.

1. The students the police arrested complained.
2. The luggage that the passengers checked arrived.
3. The luggage that the passengers that the storm delayed checked arrived.

Using a regular grammar to model natural language

Do we need anything more powerful than a regular grammar in order to model natural language?

Centre Embedding: i.e. infinitely recursive structures described by the rule $A \rightarrow aAb$, which generates language examples of the form $a^n b^n$.

1. The students the police arrested complained.

2. The luggage that the passengers checked arrived.

3. The luggage that the passengers that the storm delayed checked arrived.

But for finite *n* we can still model the language using an FSA: we can design the states to capture finite levels of embedding.

So are there any other reasons not to just use an FSA?

Using a regular grammar to model natural language

Do we need anything more powerful than a regular grammar in order to model natural language?

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

Using a regular grammar to model natural language

Do we need anything more powerful than a regular grammar in order to model natural language?

Useful internal structures: structures derived by Regular Grammars are generally not very useful for higher level NLP applications. We need informative structure so that we can, for example, build up good semantic representations.

Long-range dependencies in FSAs

- Some kinds of long-range dependencies can be captured by an FSA:
  - `/a.*a|b.*b/`
    - matches: `acccccca` and `bccccccb`
    - but not: `accccccb` or `bccccccа`

Long-range dependencies in FSAs

- Some kinds of long-range dependencies can be captured by an FSA:
  - `/a.*a|b.*b/`
    - matches: `acccccca` and `bccccccb`
    - but not: `accccccb` or `bccccccа`

- Are there dependencies that are *impossible* for an FSA?
- Are there dependencies that are possible but *difficult* for an FSA?

FSAs can model limited kinds of long-distance dependencies ***as long as the amount of long-distance information is bounded****.*

Regular vs. Context-Free Grammars

- In a regular grammar:
  - Each rule has one parent and at most one non-terminal child
  - A non-terminal child is always the last child (or always the first child)

- In a context-free grammar:
  - Each rule has one parent

Regular vs. Context-Free Grammars

- In a regular grammar:
  - Each rule has one parent and at most one non-terminal child
  - A non-terminal child is always the last child (or always the first child)

- In a context-free grammar:
  - Each rule has one parent

- Example of a language that is context-free but not regular:
  - $\{ a^n b^n \} = \{ ab, aabb, aaabbb, aaaabbbb, \ldots \}$

Regular vs. Context-Free Grammars

- In a regular grammar:
  - Each rule has one parent and at most one non-terminal child
  - A non-terminal child is always the last child (or always the first child)

- In a context-free grammar:
  - Each rule has one parent

- Example of a language that is context-free but not regular:
  - $\{ a^n b^n \} = \{ ab, aabb, aaabbb, aaaabbbb, \ldots \}$
  - Generated by two rules: $S \rightarrow aSb$
    $S \rightarrow ab$

Shallow Parsing: FSAs and Partial Grammars

Shallow Parsing using FSAs can be useful information for certain tasks like Named Entity Recognition, e.g. Chase Manhattan, J.P. Morgan, Barack Obama, etc.

Consider:

NP → nnsb NP

NP → np1 NP

NP → np1

You could match a titled name like, *Prof. Stephen Hawking*.

Shallow Parsing: FSAs and Partial Grammars

Shallow Parsing using FSAs can be useful information for certain tasks like Named Entity Recognition, e.g. Chase Manhattan, J.P. Morgan, Barack Obama, etc.

Consider:

NP → nnsb NP

NP → np1 NP

NP → np1

For more "complex" NPs, e.g., with relative clauses (RC) or possessives, this is insufficient, but since the RC is not part of the named entity, it does not need to be captured in the partial grammar

T*he man who likes the dog which bites postmen*.

The Chomsky Hierarchy beyond Regular Grammars

- More complex rewrite rules can describe more complex languages
- Chomsky initially identified four levels:
  - Recursively enumerable (no restrictions)
  - Context-sensitive ($\alpha A \beta \rightarrow \alpha \gamma \beta$, where α, β, γ are strings of symbols, and where γ is not the empty string)
  - Context-free ($A \rightarrow \gamma$, where γ is a string of symbols)
  - Regular ($A \rightarrow xB$ or $A \rightarrow x$, where x is a string of terminal symbols)

The Extended Chomsky Hierarchy

- Many more levels are now known (in fact infinitely many!), such as:
  - Recursively enumerable (no restrictions)
  - Context-sensitive ($\alpha A \beta \rightarrow \alpha \gamma \beta$)
  - Mildly context-sensitive (e.g. Combinatory Categorial Grammar)
  - Context-free ($A \rightarrow \gamma$)
  - Regular ($A \rightarrow xB$ or $A \rightarrow x$)
  - Strictly local (e.g. n-gram models; each n is a separate level)

The Chomsky Hierarchy: Syntax and Context-Free Grammars

Is syntax context-free?

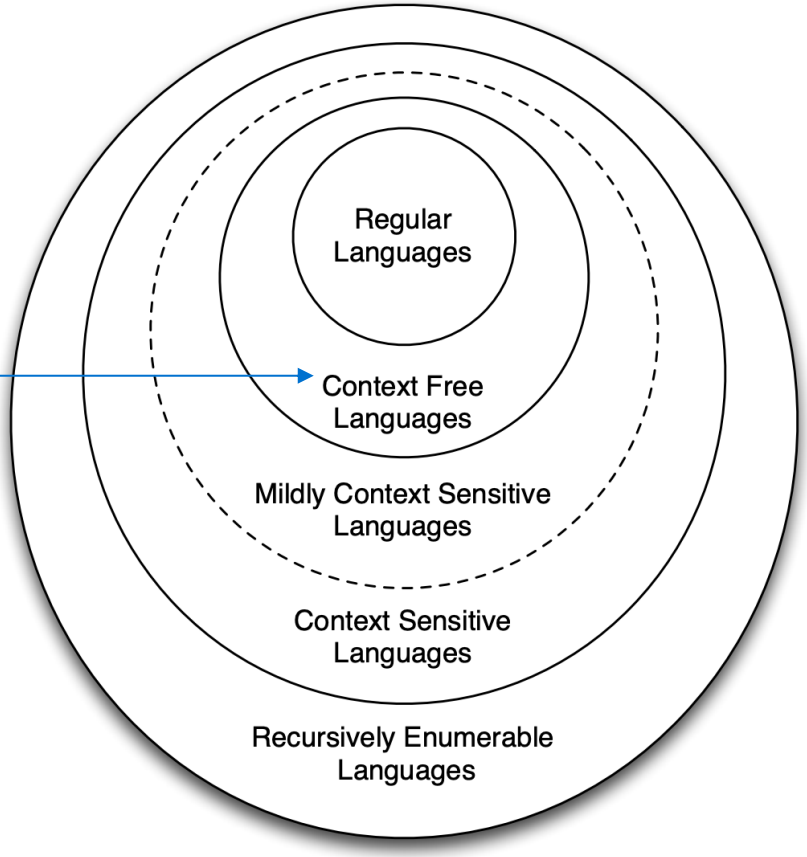


Illustration from Paula Buttery (2025) Notes on Formal Models of Languages: Grammars; https://www.cl.cam.ac.uk/teaching/2425/ForModLang/notes/Grammars_notes.pdf

A toy CFG

S →	NP VP	(sentences)
NP →	Pron	(noun phrases)
NP →	Det N	
NP →	N	
NP →	NP PP	
VP →	V	(verb phrases)
VP →	V NP	
VP →	VP PP	
PP →	P NP	(prepositional phrases)
Det →	{a, the, this, these}	(determiners)
N →	{river, rivers, fish, December}	(nouns)
Pron →	{they, them, she, her, he, him}	(pronouns)
V →	{eat, eats, fish, fishes}	(verbs)
P →	{in, on}	(prepositions)

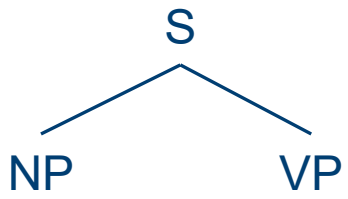
A toy CFG

S →	NP VP
NP →	Pron
NP →	Det N
NP →	N
NP →	NP PP
VP →	V
VP →	V NP
VP →	VP PP
PP →	P NP
Det →	{a, the, this, these}
N →	{river, rivers, fish, December}
Pron →	{they, them, she, her, he, him}
V →	{eat, eats, fish, fishes}
P →	{in, on}

S

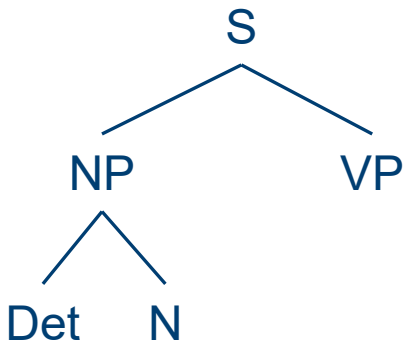
A toy CFG

S →	NP VP
NP →	Pron
NP →	Det N
NP →	N
NP →	NP PP
VP →	V
VP →	V NP
VP →	VP PP
PP →	P NP
Det →	{a, the, this, these}
N →	{river, rivers, fish, December}
Pron →	{they, them, she, her, he, him}
V →	{eat, eats, fish, fishes}
P →	{in, on}



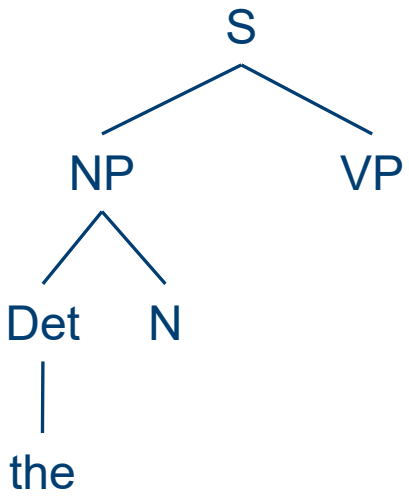
A toy CFG

S →	NP VP
NP →	Pron
NP →	Det N
NP →	N
NP →	NP PP
VP →	V
VP →	V NP
VP →	VP PP
PP →	P NP
Det →	{a, the, this, these}
N →	{river, rivers, fish, December}
Pron →	{they, them, she, her, he, him}
V →	{eat, eats, fish, fishes}
P →	{in, on}



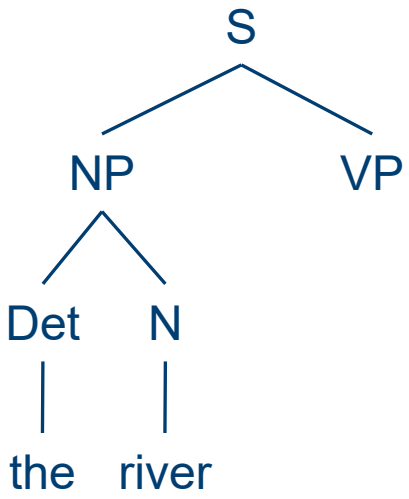
A toy CFG

S	→	NP VP
NP	→	Pron
NP	→	Det N
NP	→	N
NP	→	NP PP
VP	→	V
VP	→	V NP
VP	→	VP PP
PP	→	P NP
Det	→	{a, the, this, these}
N	→	{river, rivers, fish, December}
Pron	→	{they, them, she, her, he, him}
V	→	{eat, eats, fish, fishes}
P	→	{in, on}



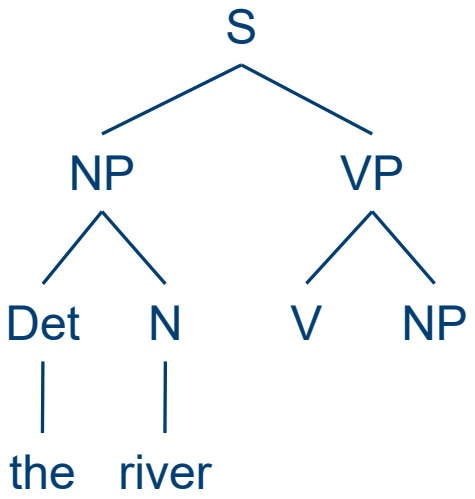
A toy CFG

S →	NP VP
NP →	Pron
NP →	Det N
NP →	N
NP →	NP PP
VP →	V
VP →	V NP
VP →	VP PP
PP →	P NP
Det →	{a, the, this, these}
N →	{river, rivers, fish, December}
Pron →	{they, them, she, her, he, him}
V →	{eat, eats, fish, fishes}
P →	{in, on}



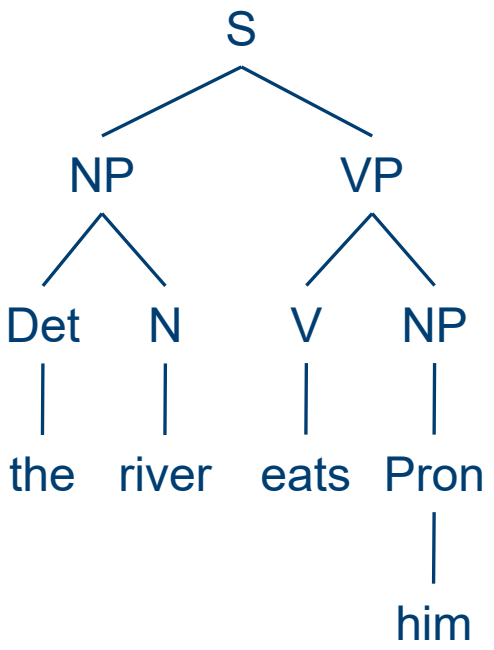
A toy CFG

S →	NP VP
NP →	Pron
NP →	Det N
NP →	N
NP →	NP PP
VP →	V
VP →	V NP
VP →	VP PP
PP →	P NP
Det →	{a, the, this, these}
N →	{river, rivers, fish, December}
Pron →	{they, them, she, her, he, him}
V →	{eat, eats, fish, fishes}
P →	{in, on}



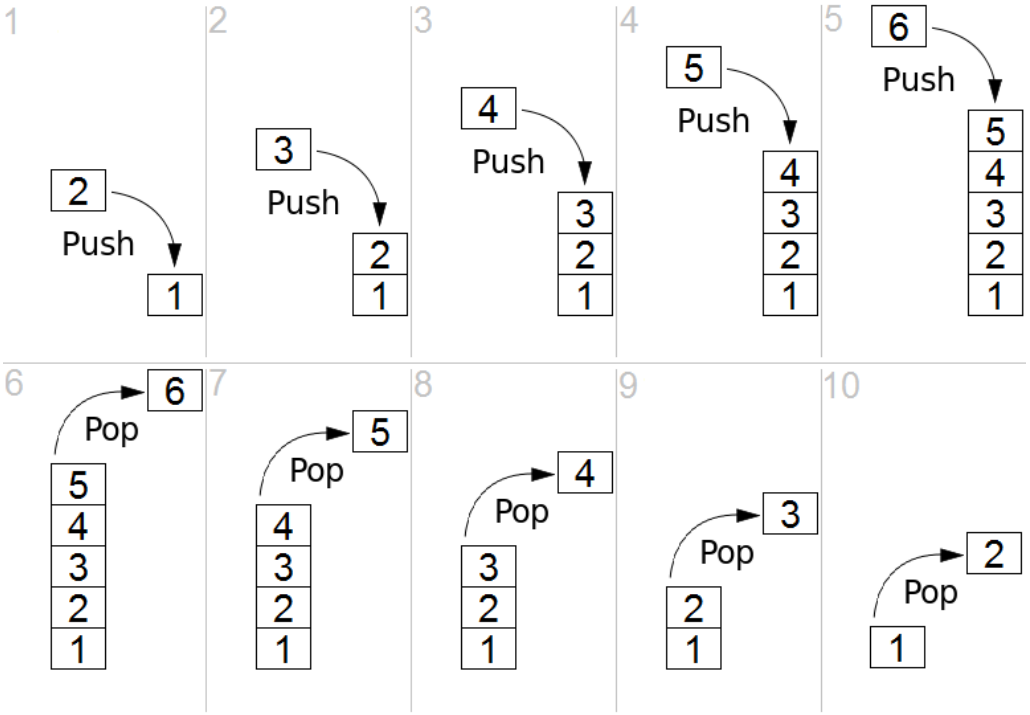
A toy CFG

S →	NP VP
NP →	Pron
NP →	Det N
NP →	N
NP →	NP PP
VP →	V
VP →	V NP
VP →	VP PP
PP →	P NP
Det →	{a, the, this, these}
N →	{river, rivers, fish, December}
Pron →	{they, them, she, her, he, him}
V →	{eat, eats, fish, fishes}
P →	{in, on}



Modelling a CFG with a Push Down Automaton

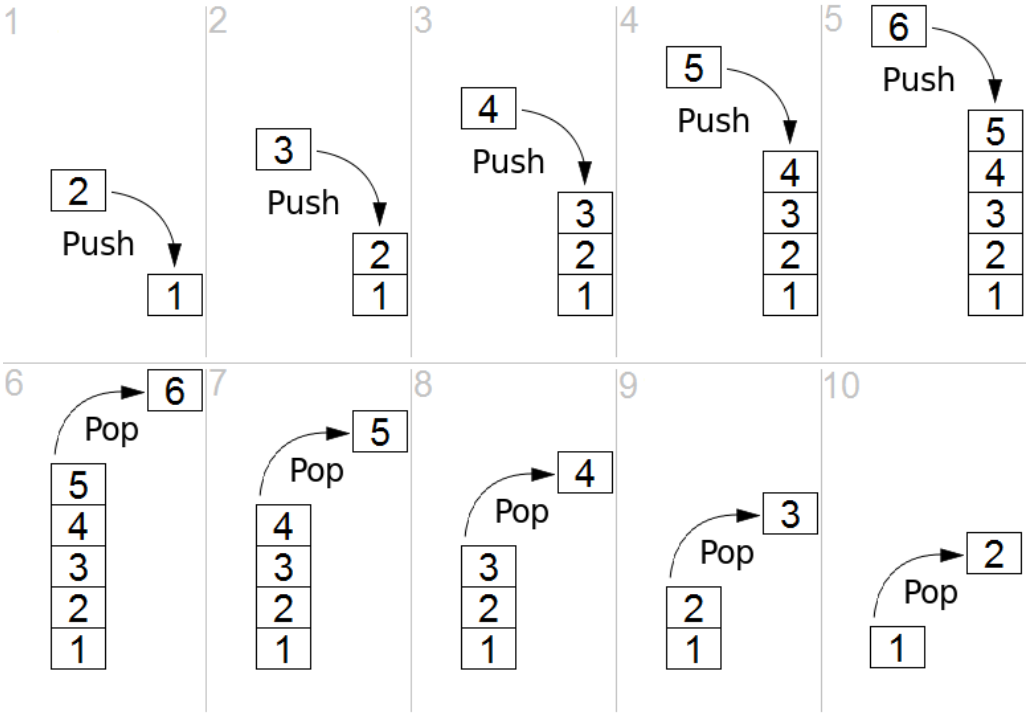
Due to centre embedding, not all CFGs can be represented as FSA: we need a **pushdown automaton** – an FSA with additional memory called a **stack**.



* Image courtesy of Maxtremus on Wikimedia

Modelling a CFG with a Push Down Automaton

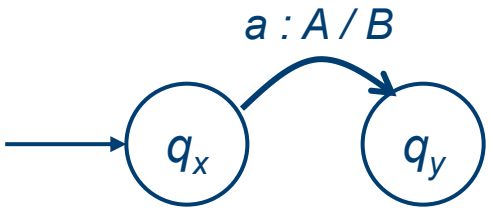
Pushdown Automata are Finite-State Automata with an associated stack (with **Last-In First-Out Memory**).



* Image courtesy of Maxtremus on Wikimedia

Modelling a CFG with a Push Down Automaton

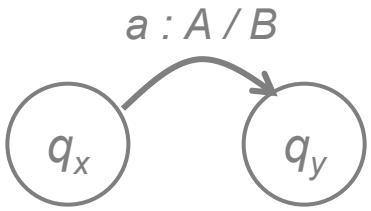
Read symbol a, pop A from the top of the stack, push B onto the stack



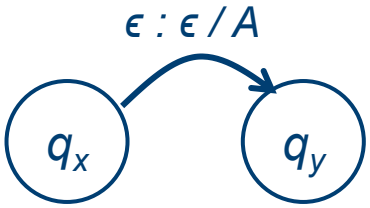
BEFORE	AFTER
A	B
C	C
D	D

Modelling a CFG with a Push Down Automaton

Push A onto the stack



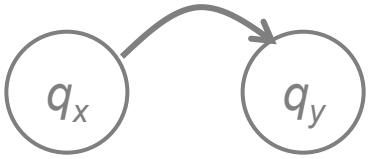
BEFORE	AFTER
A	B
C	C
D	D



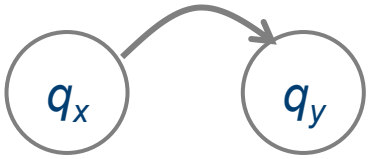
BEFORE	AFTER
	A
C	C
D	D

Modelling a CFG with a Push Down Automaton

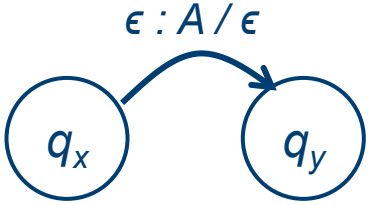
Pop A from the stack



BEFORE	AFTER
A	B
C	C
D	D



BEFORE	AFTER
	A
C	C
D	D



BEFORE	AFTER
A	
C	C
D	D

Formal Definition of a Finite-State Automaton

Recall how to define a (possibly non-deterministic) finite-state automaton:

- Q, a finite set of states
- $q_0 \in Q$, a start state
- $F \subseteq Q$, a set of accepting states
- Σ, a finite input alphabet
- $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$, a transition relation

Formal Definition of a Pushdown Automaton

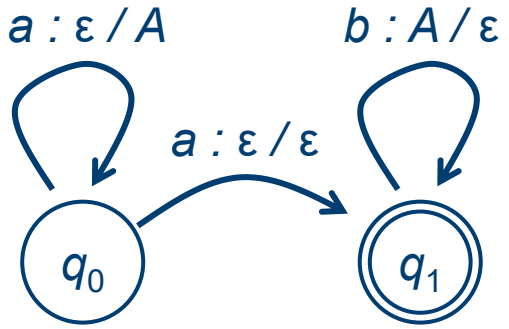
We need to augment the FSA definition with a stack:

- Q, a finite set of states
- $q_0 \in Q$, a start state
- $F \subseteq Q$, a set of accepting states
- Σ, a finite input alphabet
- Γ, a finite stack alphabet
- $z_0 \in \Gamma$, an initial stack symbol
- $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q \times (\Gamma \cup \{\varepsilon\})$, a transition relation

Once the stack is empty, the automaton must halt. A string is accepted if the automaton halts at the end of the string in an accepting state with an empty stack.

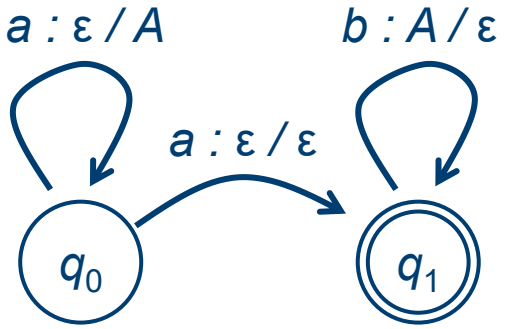
A Pushdown Automaton for a non-regular language

PDA with start state q_0 and initial stack symbol A:



A Pushdown Automaton for a non-regular language

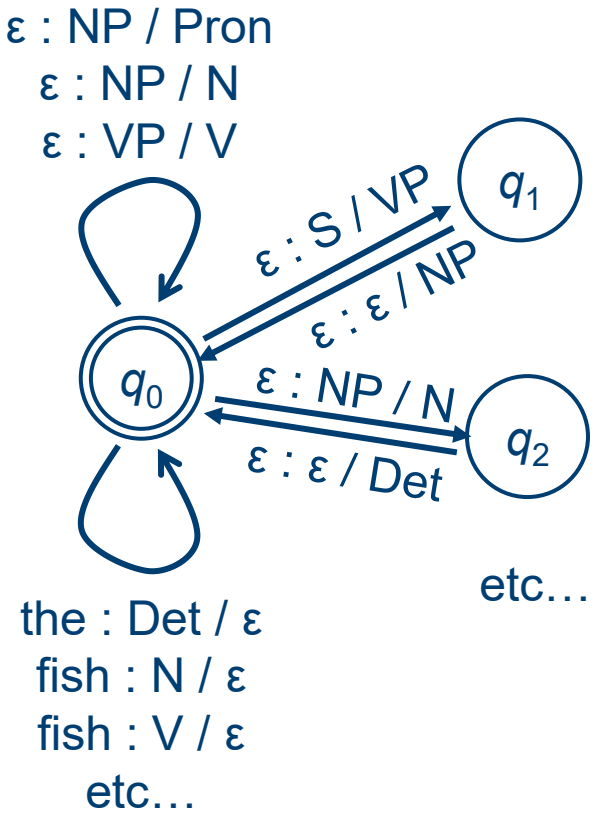
PDA with start state q_0 and initial stack symbol A:



- Reaching q_1 means we have seen a^n in the input, and have a stack of size n
- Emptying the stack means we have seen $a^n b^n$ in the input

Our toy CFG, as a PDA

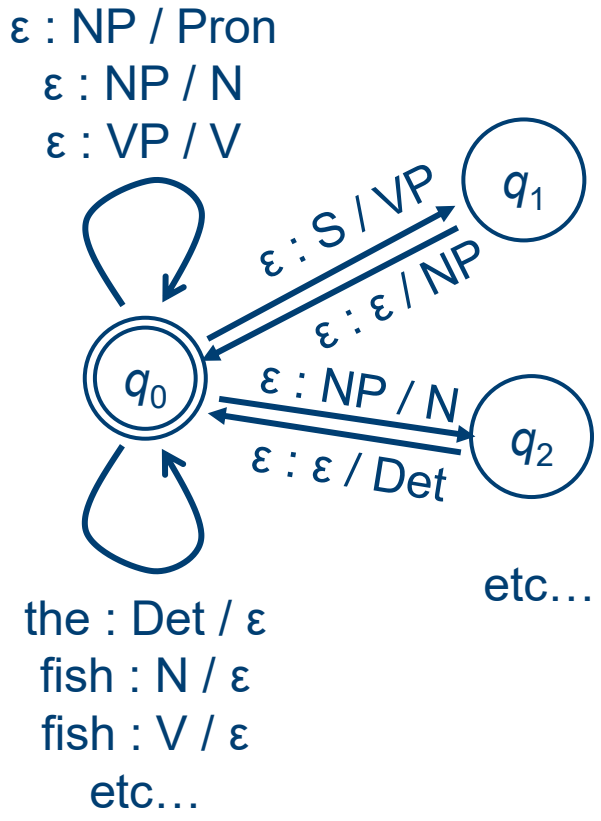
S →	NP VP
NP →	Pron
NP →	Det N
NP →	N
NP →	NP PP
VP →	V
VP →	V NP
VP →	VP PP
PP →	P NP
Det →	{a, the, this, these}
N →	{river, rivers, fish, December}
Pron →	{they, them, she, her, he, him}
V →	{eat, eats, fish, fishes}
P →	{in, on}



Our toy CFG, as a PDA

Input: the river eats him

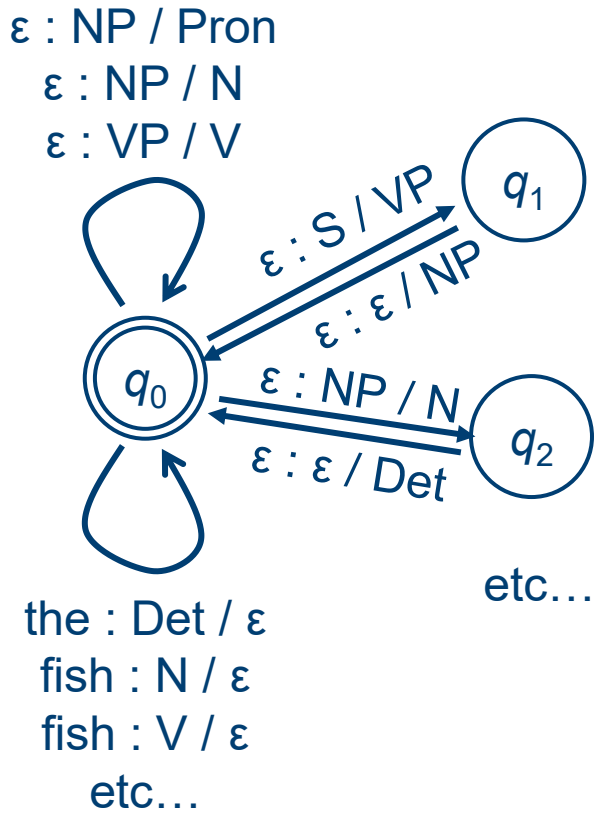
Stack: S



Our toy CFG, as a PDA

Input: the river eats him

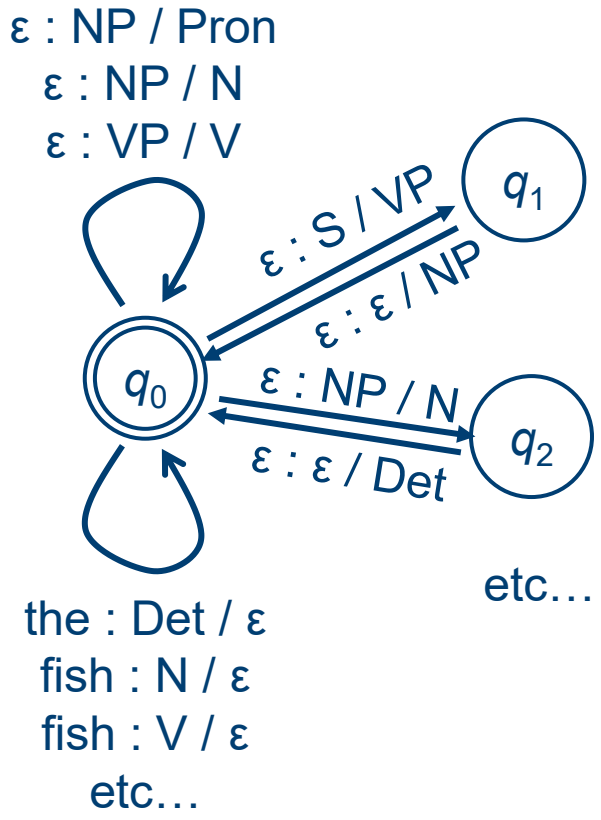
Stack: NP
VP



Our toy CFG, as a PDA

Input: the river eats him

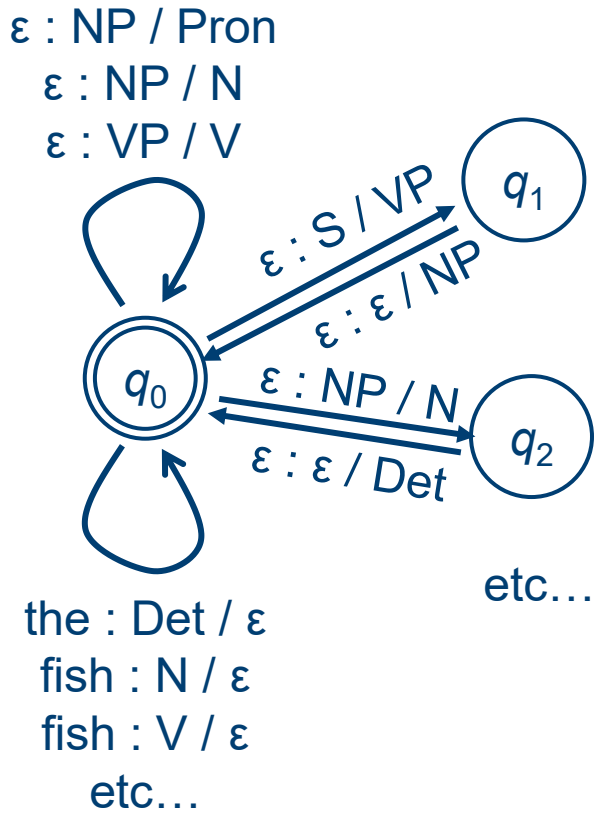
Stack: Det
N
VP



Our toy CFG, as a PDA

Input: the river eats him

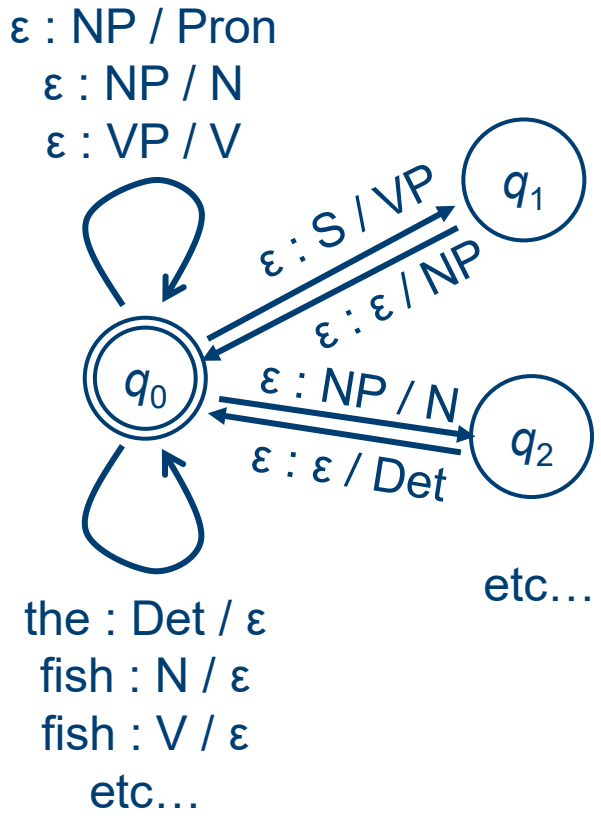
Stack: N
VP



Our toy CFG, as a PDA

Input: the river eats him

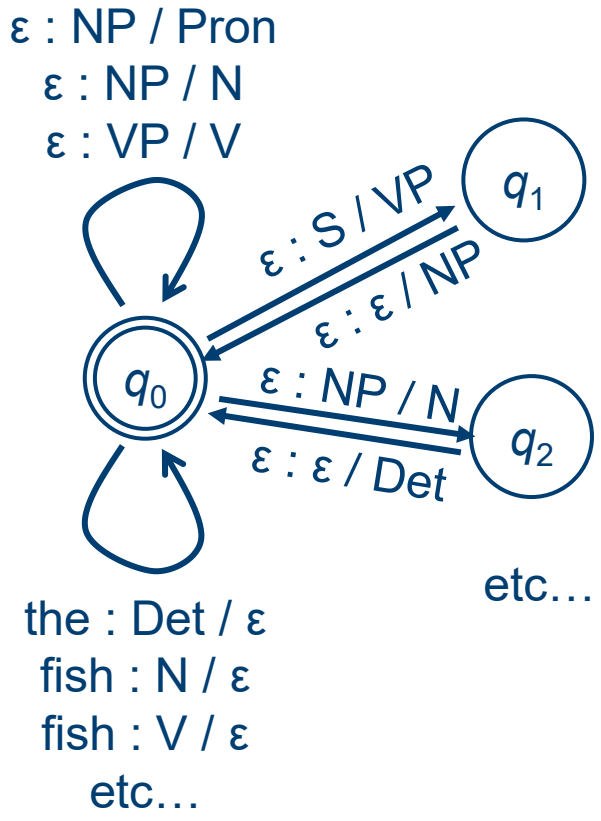
Stack: VP



Our toy CFG, as a PDA

Input: the river eats him

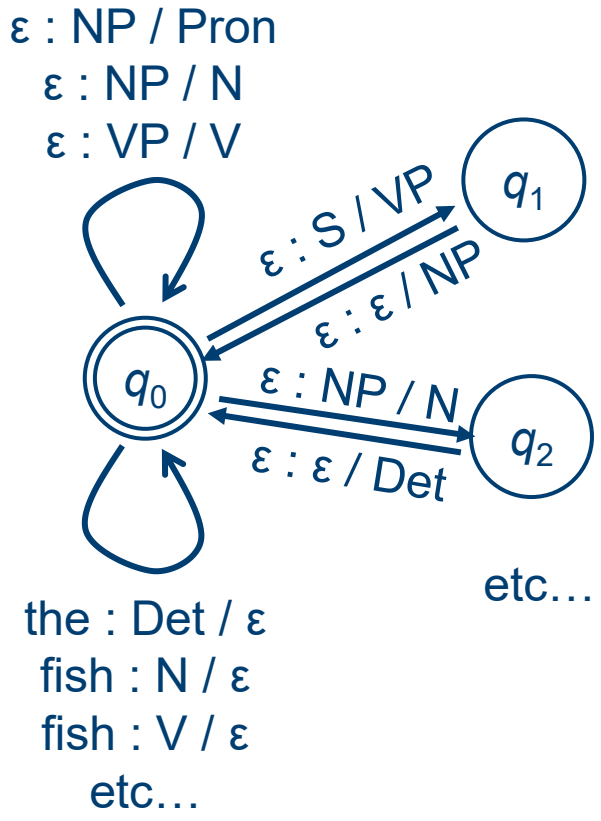
Stack: V
NP



Our toy CFG, as a PDA

Input: the river eats him

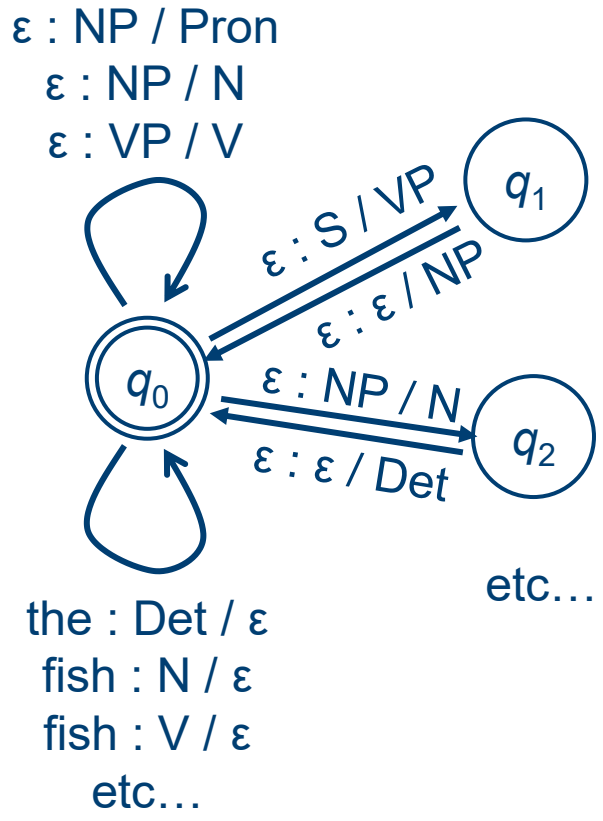
Stack: NP



Our toy CFG, as a PDA

Input: the river eats him

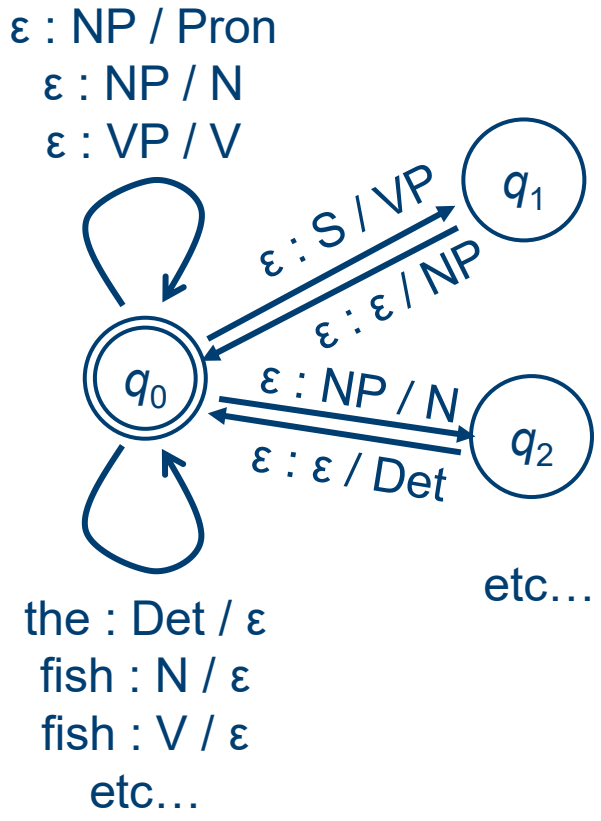
Stack: Pron



Our toy CFG, as a PDA

Input: the river eats him

Stack:



Natural Language: Context-Free to Mild-Context Sensitivity

If we are to use formal languages as a computational model of syntax, we should consider where they will appear in the Chomsky Hierarchy.

Natural language believed to be mildly context-sensitive (Joshi 1985).

- (But there are also mildly context-sensitive languages and even regular languages that have structure unlike any natural language.)
- The formal languages in this class account for constructions like **cross-serial dependencies** which context-free languages and PDAs still fail to account for.

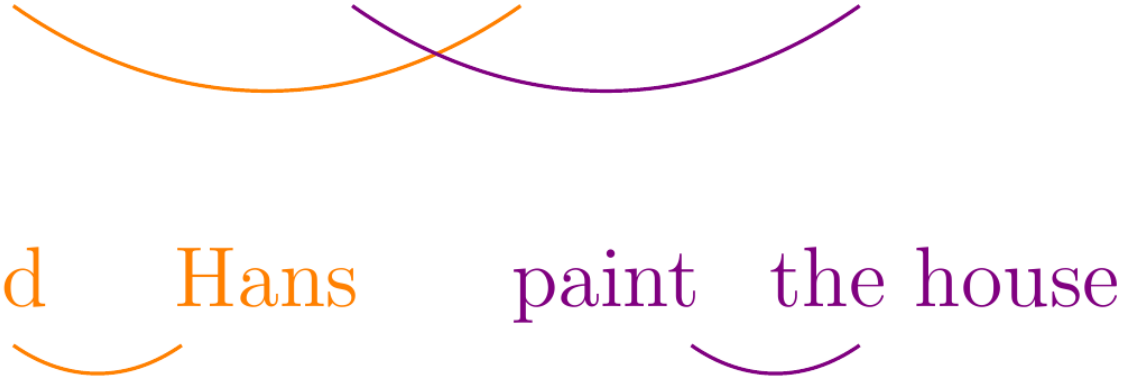
Cross-Serial Dependencies: An Example from Swiss German

Swiss-German:

...mer em Hans es huss hälfed aastriiche

English:

...we helped Hans paint the house



Example from **Shieber (1985)** *Evidence against the Context-Freeness of Natural Language*
https://link.springer.com/article/10.1007/BF00630917

Natural Language Syntax and the Chomsky Hierarchy

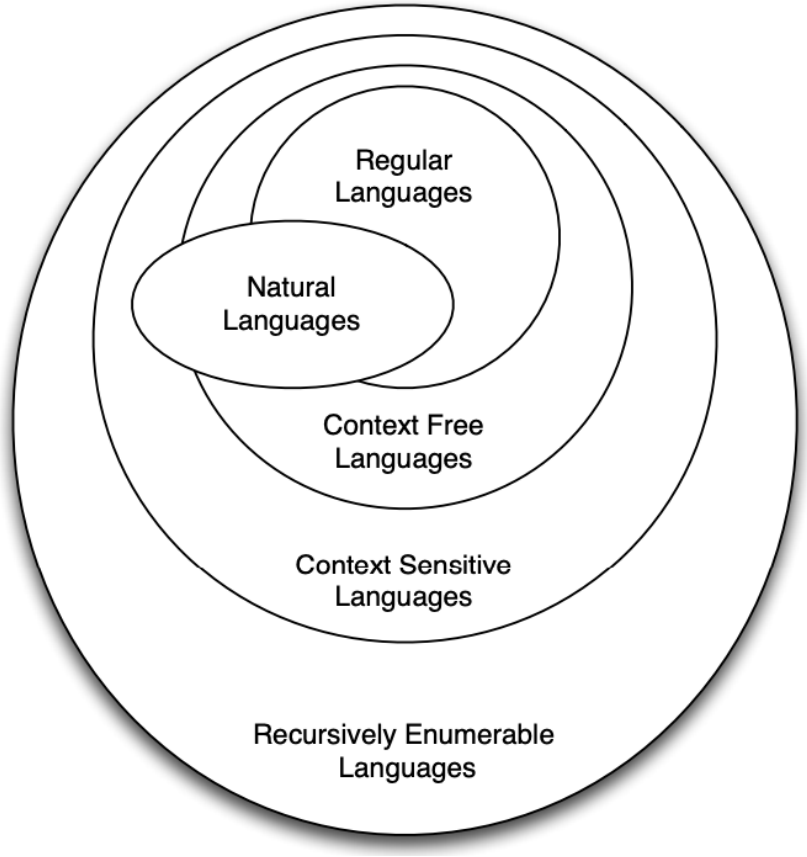


Illustration from Paula Buttery (2025) Notes on Formal Models of Languages: Grammars; https://www.cl.cam.ac.uk/teaching/2425/ForModLang/notes/Grammars_notes.pdf

Computational Syntax and Grammar Formalisms

Historically, computational linguists have modelled natural language using a much wider range of grammatical formalisms and frameworks beyond Phrase Structure Grammars :

1. **Head-Driven Phrase Structure Grammars** and **Feature-Based Grammars**
2. **Combinatory Categorial Grammars (CCG)**
3. **Tree Adjoining Grammars**, the grammar that Joshi defined to handle mild-context sensitive properties of Natural Language.

Computational Syntax and Grammar Formalisms

Historically, computational linguists have modelled natural language using a much wider range of grammatical formalisms and frameworks beyond Phrase Structure Grammars :

1. **Head-Driven Phrase Structure Grammars** and **Feature-Based Grammars**
2. **Combinatory Categorial Grammars (CCG)**
3. **Tree Adjoining Grammars**, the grammar that Joshi defined to handle mild-context sensitive properties of Natural Language.

These follow different assumptions about:

1. Syntactic Derivation (e.g., **bottom-up derivations**; potentially similar to more "recent" assumptions in Minimalist Syntax; *see also Minimalist Grammars*)
2. Lexicalisation
3. Learnability (in particular CCG supports practically computable models of L1 acquisition)

Grammar Engineering

Engineering formal grammars offers an explicit link between linguistic theory and NLP.

Explicit linguistic formalisation encoding principles and operations.

Broad-Coverage Grammars

One approach to computational syntax is to manually write a grammar

- **Takes years to reach broad coverage, when developed from scratch**
- Re-using analyses for other languages can accelerate grammar development for a new language, e.g. Grammar Matrix and CoreGram projects, which use Head-driven Phrase Structure Grammar (HPSG)
- Developing a grammar requires explicitly formalising syntactic theory
- Maintaining a large grammar can use principles from software engineering
- Selecting the best parse for a sentence typically requires information that is not in the grammar itself

Treebanks

An alternative or complementary approach is to annotate a corpus with syntactic structure. This is called a *treebank*.

- Annotated data is useful for machine learning
- Training a parser can accelerate further annotation, where annotators edit the output of the parser
- Annotating without a grammar allows flexibility, but it also creates scope for inconsistencies and disagreements between annotators.
- Maintaining consistency typically requires extensive annotation guidelines
- Treebanks allow empirical investigation of syntactic phenomena

The Penn Treebank

One of the best-known treebanks is the Penn Treebank:

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	infinitive to	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+, %, &*	WRB	wh-adverb	*how, where*

Illustration from SLP3 Ch17

The Penn Treebank

One of the best-known treebanks is the Penn Treebank:

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	infinitive to	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	+, %, &	WRB	wh-adverb	*how, where*

(17.1) There/PRON/EX are/VERB/VBP 70/NUM/CD children/NOUN/NNS there/ADV/RB ./PUNC/.

(17.2) Preliminary/ADJ/JJ findings/NOUN/NNS were/AUX/VBD reported/VERB/VBN in/ADP/IN today/NOUN/NN 's/PART/POS London/PROPN/NNP Journal/PROPN/NNP of/ADP/IN Medicine/PROPN/NNP

Illustration from SLP3 Ch17

The Penn Treebank

One of the best-known treebanks is the Penn Treebank:

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

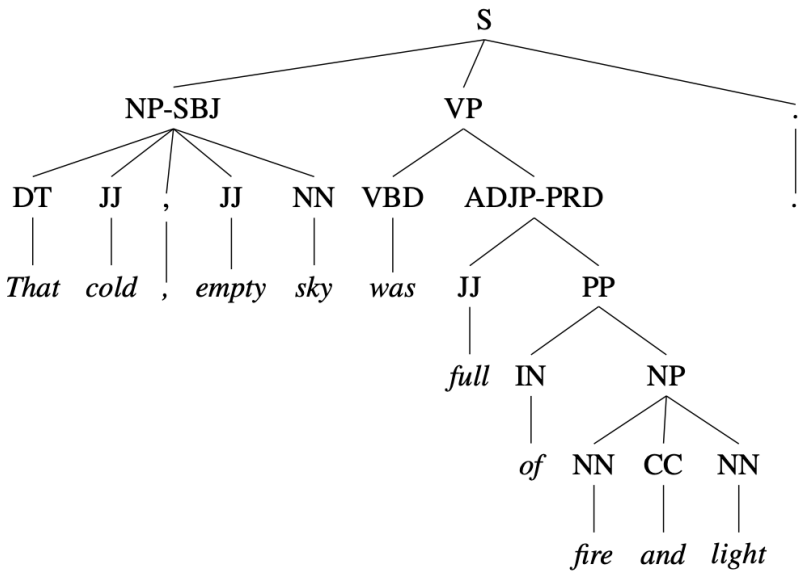


Illustration from SLP3 Ch18

The Penn Treebank

One of the best-known treebanks is the Penn Treebank

- This has also been re-annotated or semi-automatically converted to several syntactic formalisms, including HPSG, CCG, Minimalist Grammar…

Language Models and Computational Syntax

Language Models do not have any explicit encoding of syntactic structure during pretraining.

As a reminder from **Lecture 4 (n-grams):**

The task of language modelling is to assign probabilities to a sequence of tokens. The goal is to model the likelihood of a sequence in a language.

$$P(w) = P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1})$$

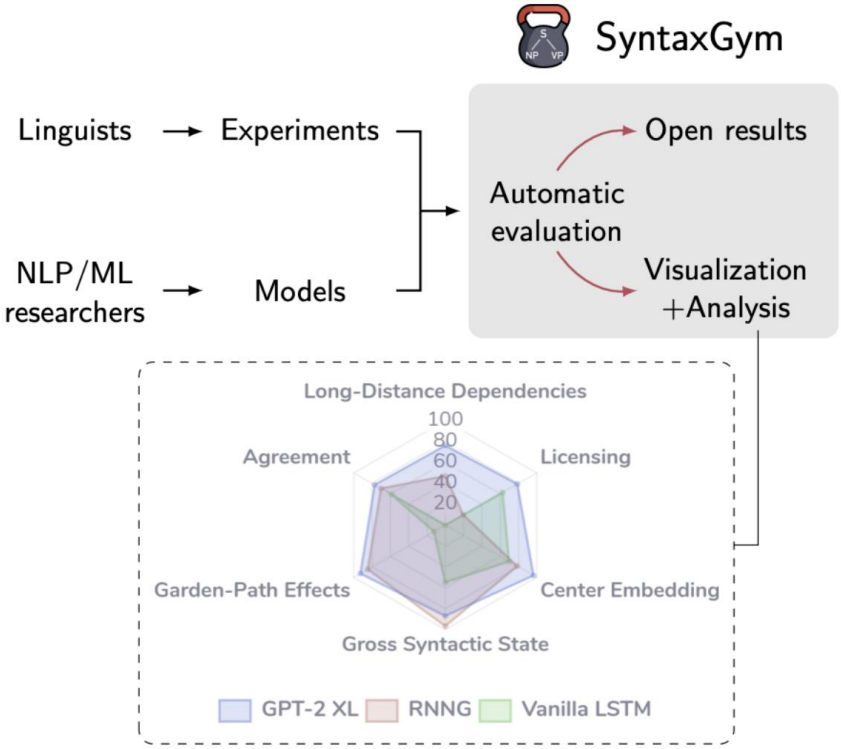
But, we can evaluate the **syntactic capabilities** of Language Models, learnt unsupervised from string probabilities.

Language Models and Computational Syntax

Minimal Pairs Evaluation: The linguistic capabilities of language models can be intrinsically evaluated using minimal pairs of datasets consisting of pairs of contrasting grammatical and ungrammatical sentences.

- The Benchmark of Linguistic Minimal Pairs for English (Warstadt et al 2020)
- MultiBLiMP: A Massively Multilingual Benchmark of Linguistic Minimal Pairs (Jumelet et al, 2025), based on **Universal Dependencies** (*more on this next week*)**.**
- Lots of language-specific BLiMPs for Turkish, Japanese, Chinese, etc. [see references]
- But mixed correlations with log-probabilities of minimal pairs and acceptability judgements and datasets might have poor separation between grammatical and ungrammatical strings.

Language Models and Computational Syntax

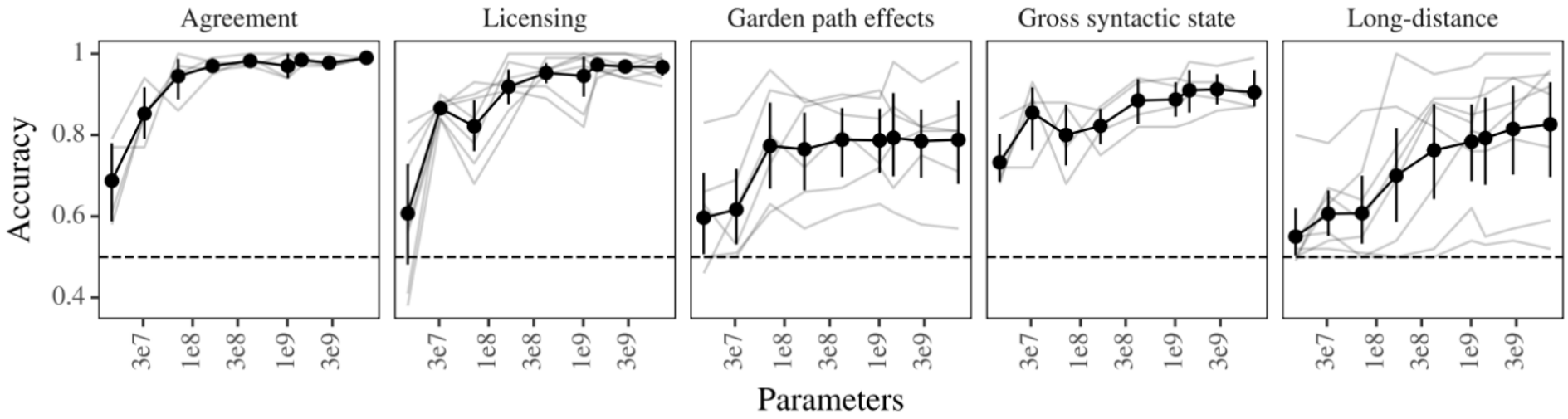


Targeted Syntactic Evaluation and Syntactic Generalisation of Language Models:

- Phenomena: Agreement and *Long-Distance Agreement*, Licensing, Garden Path Effects and Gross Syntactic State (CausalGym and SyntaxGym)

Language Models and Computational Syntax

Interpretability: Localising “circuits” in Language Models to understand linguistic mechanisms in model families (example from CausalGYM)



Accuracy of the Pythia Language Model Family on the CausalGYM tasks; grouped by type and scale. Dashed line is chance accuracy (50%).

Next Steps: Algorithms for Syntactic Parsing

Essentially there are two approaches to parsing

1. Searching for a parse within the automata is a search problem over all the possible paths.
2. Searching for a parse within the grammar is a search problem through all the possible trees defined by the grammar.

Since the automata produce languages equivalent to their grammar, we can choose to pose the problem as is most convenient for us.

Next Steps: Algorithms for Syntactic Parsing

A major consideration of all these approaches is how to keep track of:

- Which of the many derivations in search space have been explored
- Which are successful
- Which are yet to be explored

A second consideration is the order in which you apply the rules when there is a choice of rules to pick from.

Next week we will look at parsing algorithms and probabilistic CFGs which will involve all these issues. We will also consider how parsing algorithms can model aspects of human language processing.

Summary of Key Points

- Goals of Computational Syntax and Introduction to the task of Syntactic Parsing (*more on parsing algorithms next week)*
- Phrase Structure Grammars and The Chomsky Hierarchy – ***pushdown automata and context-free languages*** *and* ***mild-context sensitivity***
- Introduction to other **Grammar Frameworks (TAGs, CCG and HPSG)** in Computational Syntax (*see suggested readings)*
- Grammar Engineering and Treebanking
- Computational Syntax and Language Models *(see suggested readings)*

Thanks for listening!

With special thanks to Guy Emerson!

Slides partially based on material prepared by Nigel Collier. Additional content partially draws on by material prepared by Suchir Salhan, Paula Buttery & Fermin Moscoso del Prado Martin for the L95 MPhil Course in the Department of Computer Science & Technology.

Further Reading

- J&M (2nd edition), Chapters 12 and 16
- **Treebanking:** Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). “Building a Large Annotated Corpus of English: The Penn Treebank” https://aclanthology.org/J93-2004
- **Grammar Engineering**
  - Guy Emerson and Emily M. Bender (2021) "Computational linguistics and grammar engineering", particularly sections 1, 2, and 5 https://zenodo.org/record/5599868/files/259-M%C3%BCllerEtAl-2021-25.pdf?download=1
  - Olga Zamaraeva, Lorena S. Allegue, and Carlos Gómez-Rodríguez. 2024. Spanish Resource Grammar Version 2023. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 15093–15104, Torino, Italia. ELRA and ICCL. ***A recent grammar for Spanish (useful exposition to Grammar Engineering)***

Further Reading: Syntactic Frameworks

- **CCG:**
  - A General Introduction to CCG: Steedman (2022) https://homepages.inf.ed.ac.uk/steedman/papers/ccg/moravcsik2.pdf
  - Mark Steedman and Jason Baldridge (2011) "Combinatory Categorial Grammar", particularly sections 1 and 2 https://www.research.ed.ac.uk/files/24976508/SteedmanBaldridgeNTSyntax.pdf
- **Minimalist Grammars (MGs):**
  - Stabler, Edward P., ' Computational Perspectives on Minimalism', in Cedric Boeckx (ed.), *The Oxford Handbook of Linguistic Minimalism* (2011; online edn, Oxford Academic, 18 Sept. 2012), https://doi.org/10.1093/oxfordhb/9780199549368.013.0027, accessed 21 Nov. 2025. (*More advanced reading)*

Further Reading: Syntax and Language Models

- **Learnability:**
  - Ethan Gotlieb Wilcox, Richard Futrell, Roger Levy; Using Computational Models to Test Syntactic Learnability. Linguistic Inquiry 2024; 55 (4): 805–848. Available online here: https://muse.jhu.edu/article/938707/figure/fig04
- **Syntax and Language Models:**
  - Warstadt, A., Parrish, A., Liu, H., Mohananey, A., Peng, W., Wang, S. F., & Bowman, S. R. (2020). BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, *8*, 377-392.
  - MultiBLiMP 1.0 Paper (TACL): https://arxiv.org/pdf/2504.02768 Datasets on HuggingFace: https://huggingface.co/datasets/jumelet/multiblimp
  - SyntaxGym (ACL Systems Demo 2020): https://aclanthology.org/2020.acl-demos.10.pdf
  - CausalGym (ACL 2024): https://aclanthology.org/2024.acl-long.785.pdf

Exercises (supervision 3, part 1/2)

- "Computational linguists should place their efforts on annotating treebanks rather than writing grammars." Discuss. (Short essay, ~1 side)
- Give examples of under- and over-generation in the toy grammar. Modify the grammar to mitigate at least one problem, and explain your changes.

Pre-Lecture Exercise for Lecture 8

Given our toy CFG and a sentence, how might you systematically work out whether the grammar licenses that sentence?