CS 577: NATURAL LANGUAGE PROCESSING

Abulhair Saparov

Lecture 23: Syntax IV and Semantics

SYNTAX SO FAR

- We have discussed a handful of classes of grammars:
  - Regular grammars
    - Simple, easy to parse, but no recursion
  - Context-free grammars
    - Covers most (if not all) of natural language syntax
  - Combinatory categorial grammars (CCG)
- How do you parse non-context-free grammars?
  - How to parse CCG?
- What about more complex types of grammars?
- What about meaning (i.e., semantics)?

COMBINATORY CATEGORIAL GRAMMAR

- Combinatory categorial grammar (CCG; Steedman 1987) is a grammar formalism that can be used to describe non-context-free grammars.
- Each item in the vocabulary is assigned a syntactic type or category.
  - A simple vocabulary containing 4 items:

$$\text{the} : NP/N \qquad \text{dog} : N \qquad \text{John} : NP \qquad \text{bit} : (S\backslash NP)/NP$$

$$\frac{\text{the}}{NP/N} \qquad \frac{\text{dog}}{N} \qquad \frac{\text{bit}}{(S\backslash NP)/NP} \qquad \frac{\text{John}}{NP}$$

[wikipedia.org/wiki/Combinatory_categorial_grammar]

COMBINATORY CATEGORIAL GRAMMAR

- Combinatory categorial grammar (CCG; Steedman 1987) is a grammar formalism that can be used to describe non-context-free grammars.
- Each item in the vocabulary is assigned a syntactic type or category.
  - A simple vocabulary containing 4 items:

$$\text{the} : NP/N \qquad \text{dog} : N \qquad \text{John} : NP \qquad \text{bit} : (S\backslash NP)/NP$$

$$\frac{\dfrac{\text{the}}{NP/N} \qquad \dfrac{\text{dog}}{N}}{NP}> \qquad\qquad \dfrac{\text{bit}}{(S\backslash NP)/NP} \qquad \dfrac{\text{John}}{NP}$$

[wikipedia.org/wiki/Combinatory_categorial_grammar]

COMBINATORY CATEGORIAL GRAMMAR

- Combinatory categorial grammar (CCG; Steedman 1987) is a grammar formalism that can be used to describe non-context-free grammars.
- Each item in the vocabulary is assigned a syntactic type or category.
    - A simple vocabulary containing 4 items:

$$\text{the} : NP/N \qquad \text{dog} : N \qquad \text{John} : NP \qquad \text{bit} : (S\backslash NP)/NP$$

$$\dfrac{\dfrac{\text{the}}{NP/N} \quad \dfrac{\text{dog}}{N}}{NP} > \qquad \dfrac{\dfrac{\text{bit}}{(S\backslash NP)/NP} \quad \dfrac{\text{John}}{NP}}{S\backslash NP} >$$

[wikipedia.org/wiki/Combinatory_categorial_grammar]

COMBINATORY CATEGORIAL GRAMMAR

- Combinatory categorial grammar (CCG; Steedman 1987) is a grammar formalism that can be used to describe non-context-free grammars.
- Each item in the vocabulary is assigned a syntactic type or category.
  - A simple vocabulary containing 4 items:

$$\text{the} : NP/N \qquad \text{dog} : N \qquad \text{John} : NP \qquad \text{bit} : (S\backslash NP)/NP$$

$$\dfrac{\dfrac{\dfrac{\text{the}}{NP/N} \quad \dfrac{\text{dog}}{N}}{NP}> \qquad \dfrac{\dfrac{\text{bit}}{(S\backslash NP)/NP} \quad \dfrac{\text{John}}{NP}}{S\backslash NP}>}{S}<$$

[wikipedia.org/wiki/Combinatory_categorial_grammar]

COMBINATORY CATEGORIAL GRAMMAR

- Parsing ‘`I flew IcelandAir to Geneva and SwissAir to London`’.

$$
\begin{array}{cccccccc}
\textit{flew} & \textit{IcelandAir} & \textit{to} & \textit{Geneva} & \textit{and} & \textit{SwissAir} & \textit{to} & \textit{London} \\
\hline (VP/PP)/NP & NP & PP/NP & NP & CONJ & NP & PP/NP & NP \\
 & \overline{(VP/PP)\backslash((VP/PP)/NP)}^{<\mathbf{T}} & \multicolumn{2}{c}{\overline{PP}^{>}} & & \overline{(VP/PP)\backslash((VP/PP)/NP)}^{<\mathbf{T}} & \multicolumn{2}{c}{\overline{PP}^{>}} \\
 & & \multicolumn{2}{c}{\overline{VP\backslash(VP/PP)}^{<\mathbf{T}}} & & & \multicolumn{2}{c}{\overline{VP\backslash(VP/PP)}^{<\mathbf{T}}} \\
 & \multicolumn{3}{c}{\overline{VP\backslash((VP/PP)/NP)}^{<}} & & \multicolumn{3}{c}{\overline{VP\backslash((VP/PP)/NP)}^{<}} \\
 & \multicolumn{7}{c}{\overline{VP\backslash((VP/PP)/NP)}^{<\Phi>}} \\
\multicolumn{8}{c}{\overline{VP}^{<}}
\end{array}
$$

PARSING CCG

- How to parse CCG?
- Since CCG operations are unary or binary, we can extend CKY parsing.
  - CCG is well-suited for bottom-up parsing.
  - Vijay-Shanker and Weir (1993) describe this algorithm and show that it has running time $O(n^6)$.
- This can be made practically faster using beam search and better heuristics (e.g., A*).
  - But better heuristics do not change the worst-case running time.
  - And beam search sacrifices exactness/accuracy.

CONTEXT-SENSITIVE GRAMMARS

- A context-sensitive grammar is one where all production rules have the form

  $\alpha A \gamma$ -> $\alpha\beta\gamma$

where A is a nonterminal,

  α and γ are (possibly empty) sequences of terminals and nonterminals,

  and β is a non-empty sequence of terminals and nonterminals.

- This definition looks very similar to that of CFGs, with the addition of "context" requirements for each production rule.
  - In order to apply the rule for A, the additional "context" (i.e., α and γ) must match.

CONTEXT-SENSITIVE GRAMMARS

- Parsing context-sensitive grammars is PSPACE-complete.
  - PSPACE is the set of all problems that can be solved in polynomial space.
  - PSPACE is a superset of NP.
  - So PSPACE-complete is at least as difficult as NP-complete.
- Due to the computational cost of parsing, context-sensitive grammars are rarely used in practice.
- Although the question of whether the syntax of natural languages are context-free is debated by some linguists.
- The question of whether they are context-sensitive is not debated.

UNRESTRICTED GRAMMARS

- An unrestricted grammar is one where all production rules have the form

$$\alpha \rightarrow \beta$$

where α and β are sequences of terminals and nonterminals, and α is not empty.

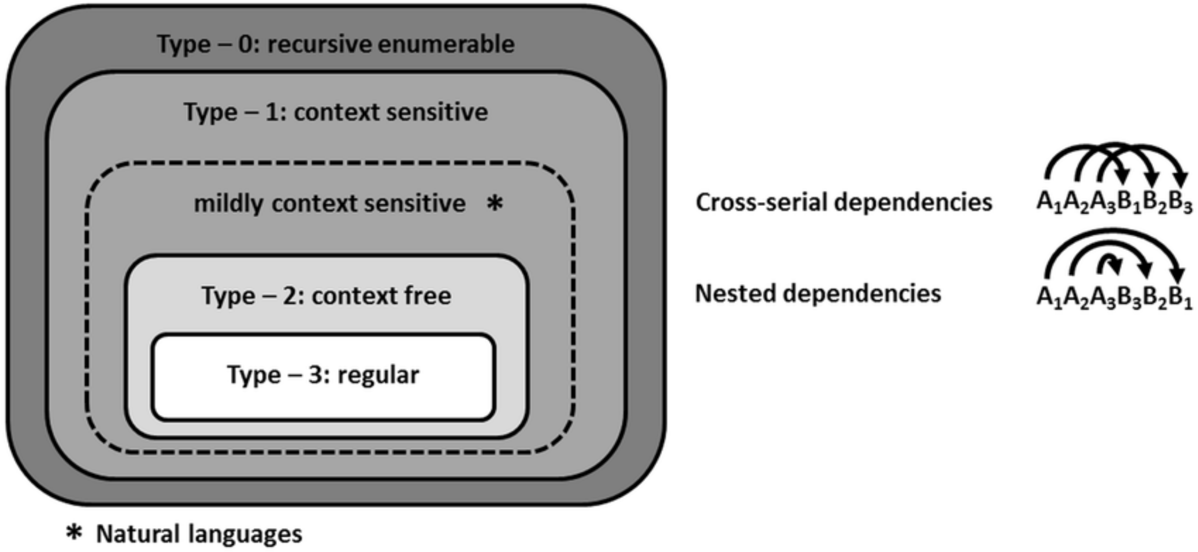
- Languages of unrestricted grammars are called recursively enumerable.
- This is the set of all grammars.
- Parsing unrestricted grammars is undecideable:
  - There is no algorithm such that, for any input string *s*, the algorithm outputs whether *s* is in the language of the grammar.
  - I.e., recognition in unrestricted grammars is undecideable

WHERE DOES CCG BELONG?

- CCG can describe non-context free languages.
- But there exists a polynomial time parsing algorithm (e.g., CKY).
- There are context-sensitive languages that cannot be described with CCG.
- CCG seems to belong to a category between context-free and context-sensitive grammars.
- There are other grammar formalisms that are shown to be equivalent to CCG.
  - I.e., they describe the same set of languages.
  - E.g., tree-adjoining grammar (TAG), linear indexed grammar, etc.
- This category is called mildly context-sensitive grammars.

CHOMSKY HIERARCHY

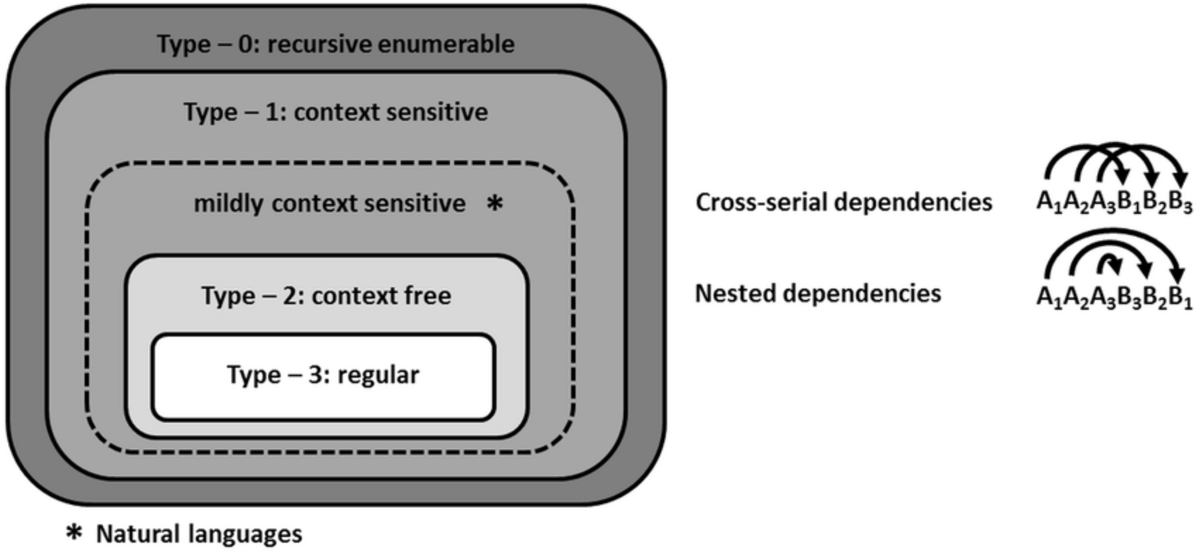
- Chomsky (1956) proposed to organize grammars according to their complexity.
- Consider the set of all languages:



[Öttl et al, 2015]

CHOMSKY HIERARCHY

- This is called the Chomsky hierarchy.
- Chomsky originally described 4 types:
  - Regular, context-free, context-sensitive, and recursively enumerable languages.



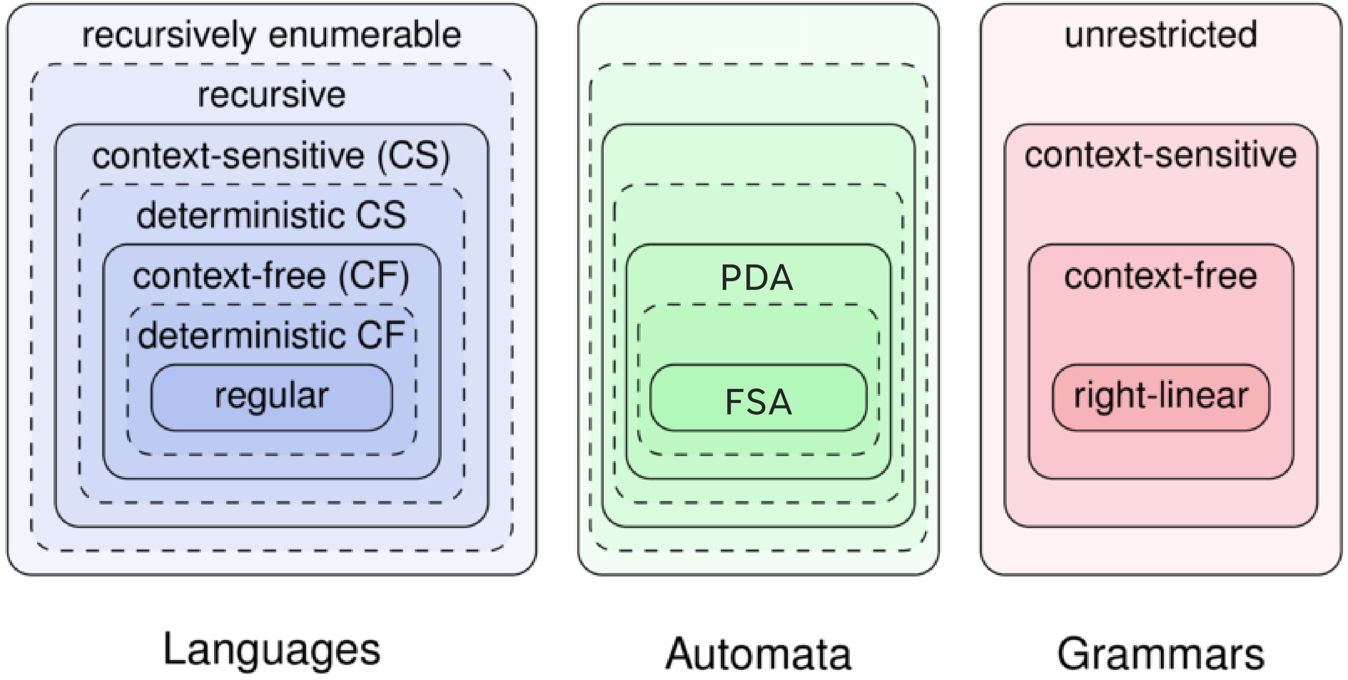
[Öttl et al, 2015]

CHOMSKY HIERARCHY

- This is called the Chomsky hierarchy.
- Chomsky originally described 4 types:
  - Regular, context-free, context-sensitive, and recursively enumerable languages.
  - Chomsky named them Type 3, Type 2, Type 1, and Type 0 languages, respectively.
  - But now there are many different categories of languages and grammars.
  - Some are not simple subsets of other categories.

CHOMSKY HIERARCHY

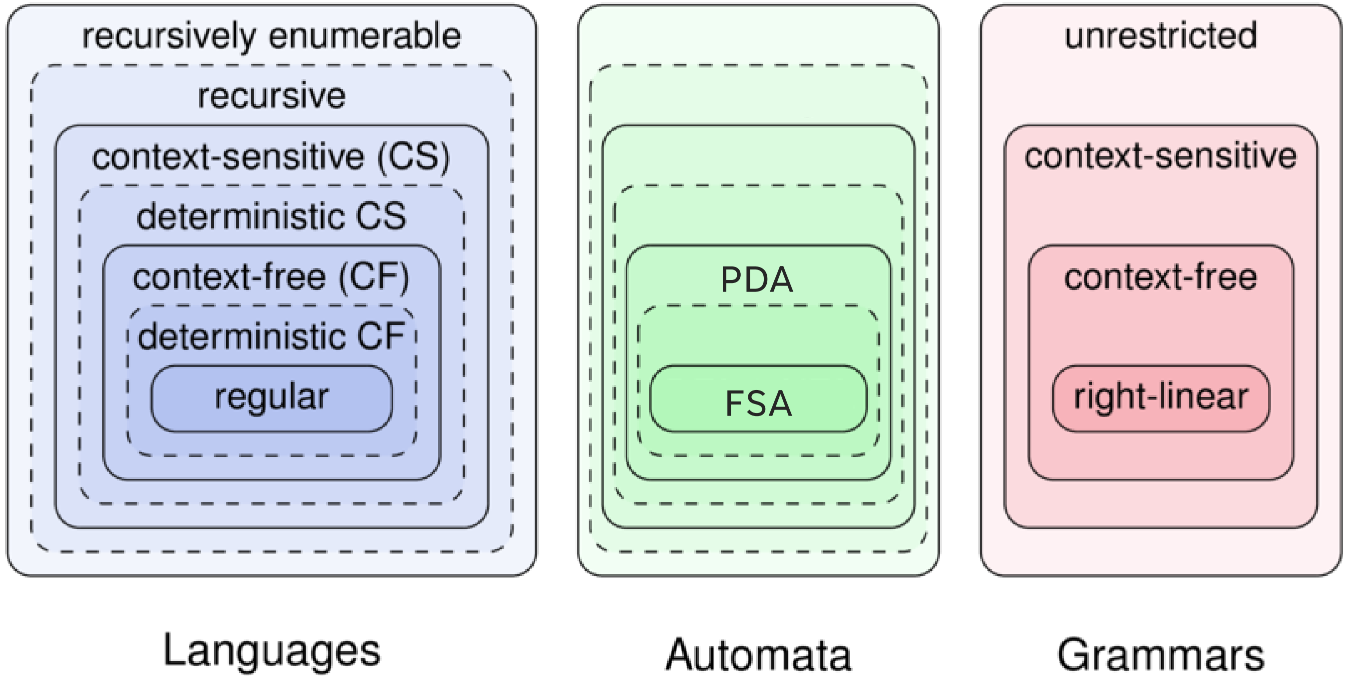
- Each category in the Chomsky hierarchy can also be characterized by the computational expressiveness required to parse/recognize languages in that category.



[Endrullis, joerg.endrullis.de/automata/21_chomsky_hierarchy.pdf]

CHOMSKY HIERARCHY

- There is a 1-to-1 correspondence between regular languages, regular grammars, and finite state automata (FSA).



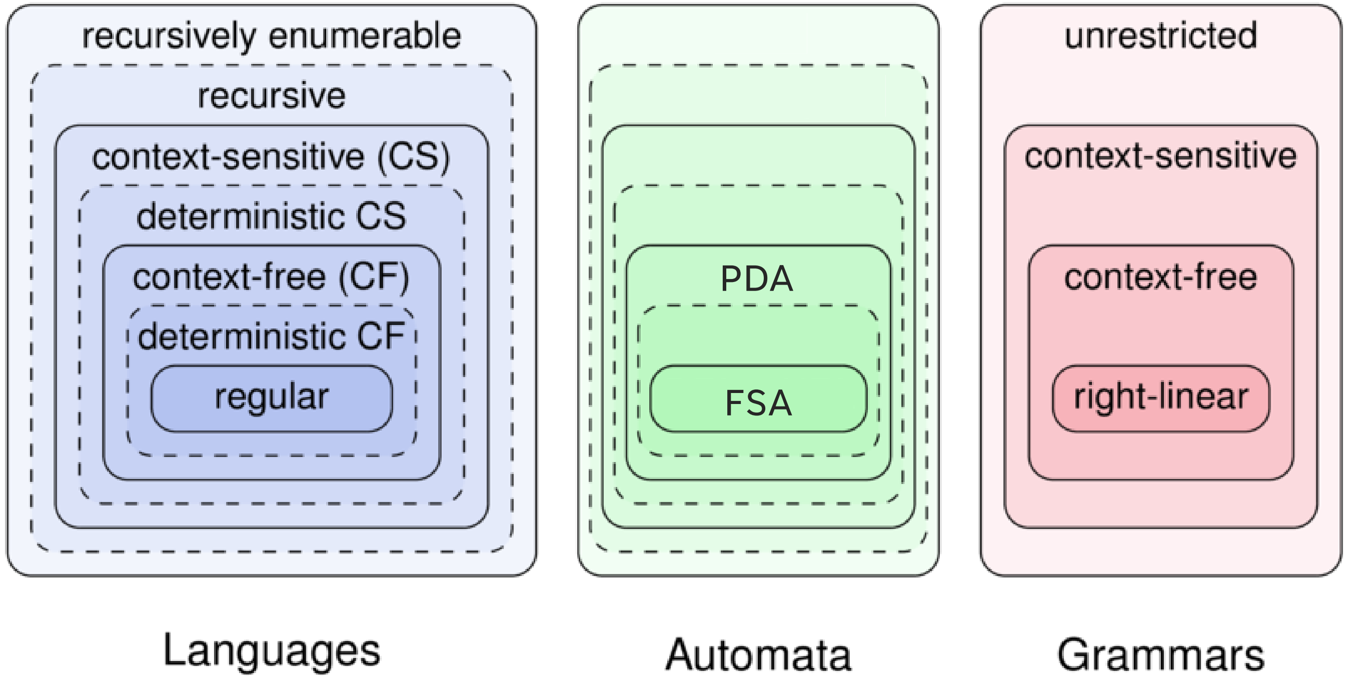
[Endrullis, joerg.endrullis.de/automata/21_chomsky_hierarchy.pdf]

CHOMSKY HIERARCHY

- There is a 1-to-1 correspondence between regular languages, regular grammars, and finite state automata (FSA).
  - For each regular language, there is a regular grammar that describes it.
  - For each regular language, there is an FSA that recognizes strings in that languages.
    - I.e., returns `SUCCESS` if an input string s belongs to the language, or `FAIL` otherwise.
  - For each FSA, the set of input strings for which the output is `SUCCESS` is a regular language.
  - Etc.

CHOMSKY HIERARCHY

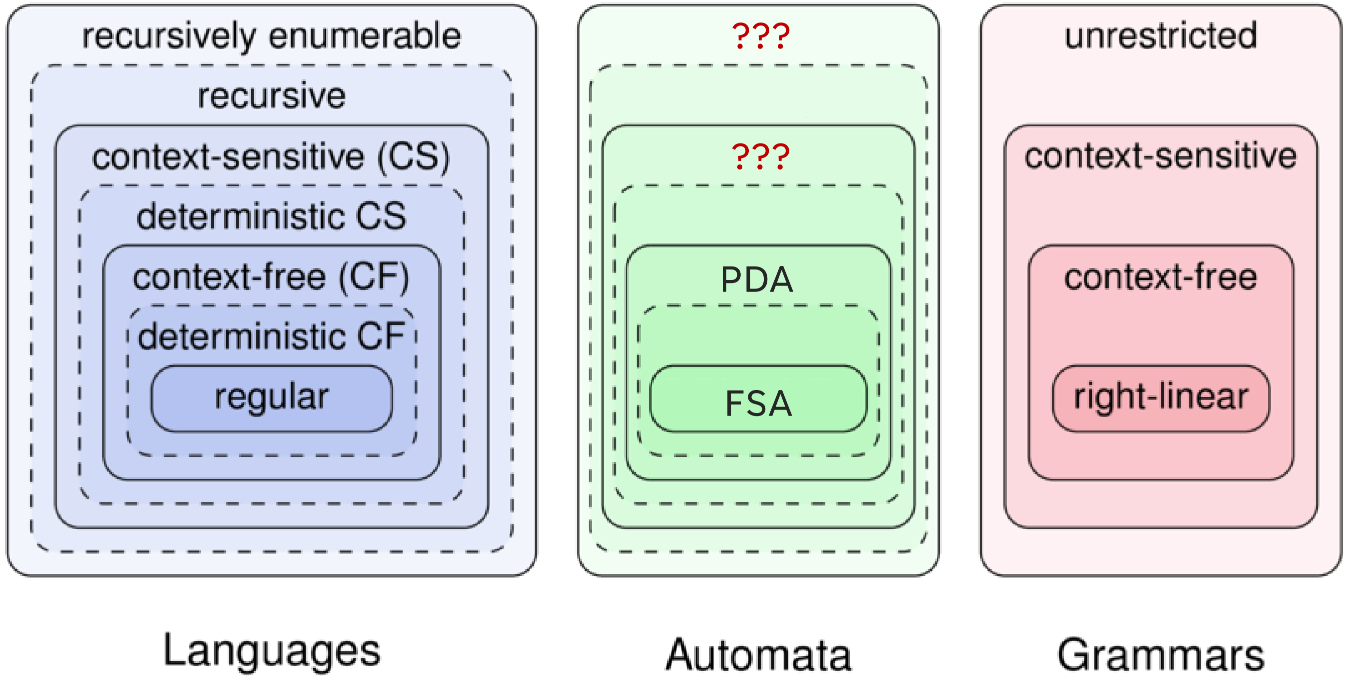
- There is a similar 1-to-1 correspondence between context-free languages, CFGs, and pushdown automata (PDA).



[Endrullis, joerg.endrullis.de/automata/21_chomsky_hierarchy.pdf]

CHOMSKY HIERARCHY

- But what about other language/grammar classes?
  - E.g., unrestricted grammars?



[Endrullis, joerg.endrullis.de/automata/21_chomsky_hierarchy.pdf]

TURING MACHINES

- For any recursively enumerable language `L`, there exists a Turing machine that outputs `SUCCESS` for any input string s if and only if s is an element of `L`.
- For any Turing machine, if `L` is the set of input strings for which the output is `SUCCESS`, `L` is recursively enumerable.
- Okay, so what are Turing machines?
  - We can define Turing machines as an extension of FSMs and PDAs.
  - Recall that a PDA is defined as an FSM with the addition of a stack.
    - State transitions can push/pop elements to/from the top of the stack.
    - State transitions can also depend on the value at the top of the stack.

TURING MACHINES

- A Turing machine is an FSM with the addition of an infinite tape.
  - First described by Turing (1936).
- Each position on the tape stores a symbol.
  - A read/write head is located over the tape at one position.
- The state transitions in the FSM can write a new symbol on the tape at the position under the head.
  - They can also move the head one position to the left or right.
  - The state transitions can depend on the current symbol under the head.

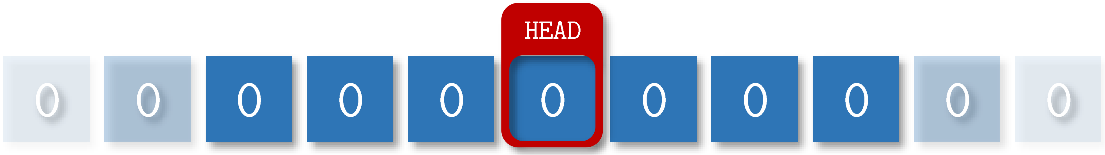
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: A



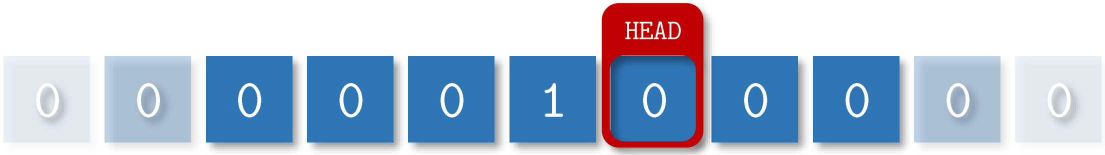
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: B



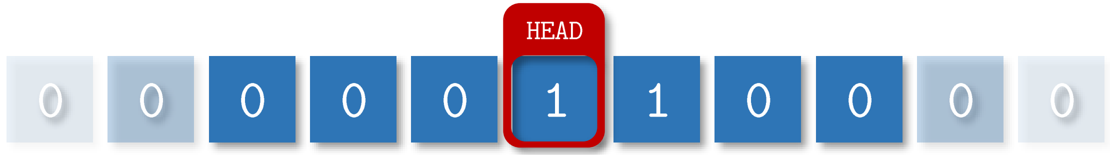
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: A



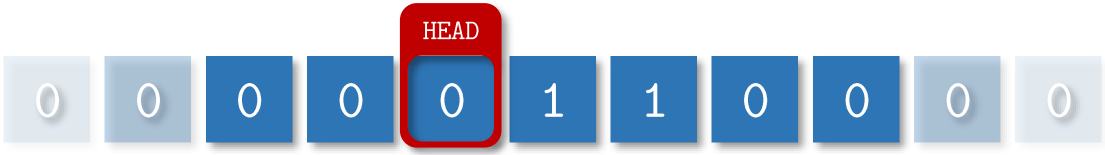
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: C



EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: B

EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: A

EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: B

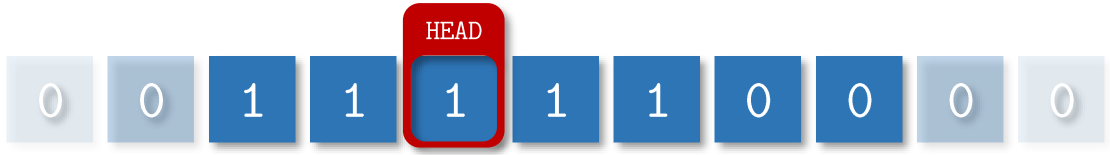
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: B



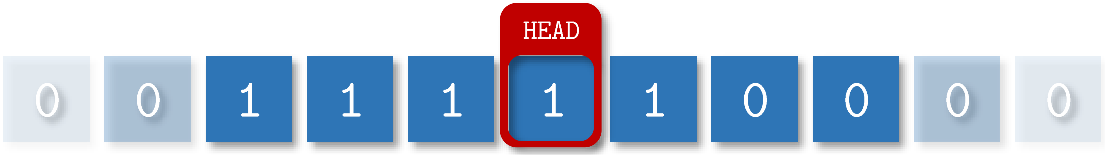
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: B



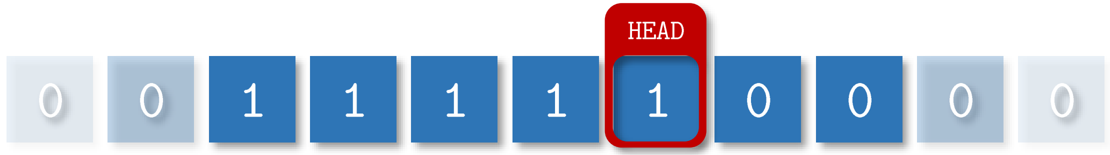
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: B



EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: B

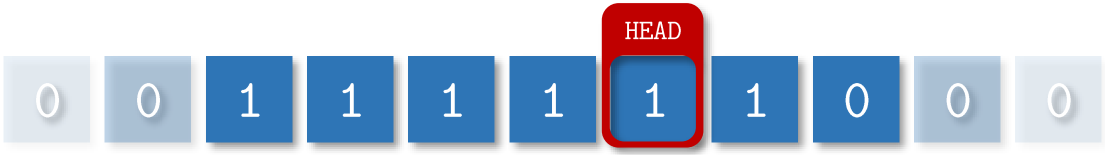
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: A



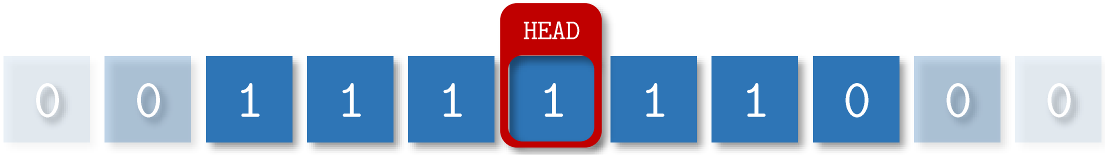
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: C



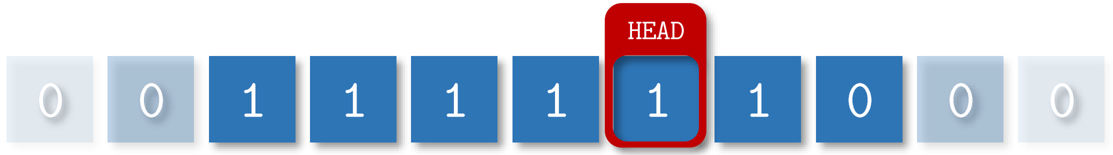
EXAMPLE TURING MACHINE

Initially all tape cells are marked with 0.

State table for 3-state, 2-symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Current state: HALT



- This example is a 3-state busy beaver Turing machine.

  (the Turing machine with 3 states that writes the most 1's to the tape and halts)

TURING MACHINES

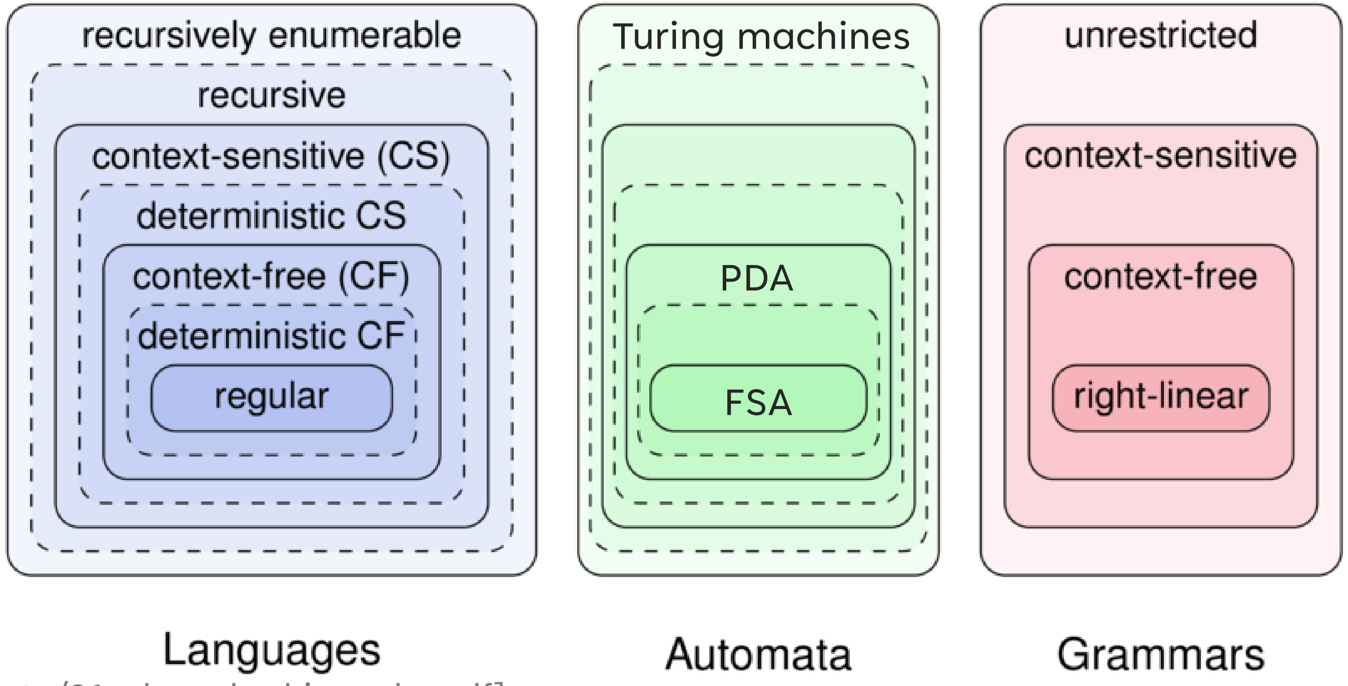
- Why are Turing machines important?
- Church-Turing thesis:
  - Any computable function can be written as a Turing machine.
  - Every Turing machine expresses a computable function.
- Any algorithm can be expressed as a Turing machine, and vice versa.
- Halting problem:
  - There is no algorithm that can compute for any Turing machine and any input, whether the Turing machine will halt.

TURING MACHINES

- How are Turing machines related to recursively enumerable languages or unrestricted grammars?
- For any recursively enumerable language `L`, we can write a Turing machine that will output `SUCCESS` if and only if the input string s is an element of `L`.
- However, for any given input s, it is impossible to determine whether the Turing machine will halt, or output `SUCCESS` eventually.
- This the problem of recognition is not only computationally expensive,
  - It is undecidable.

CHOMSKY HIERARCHY

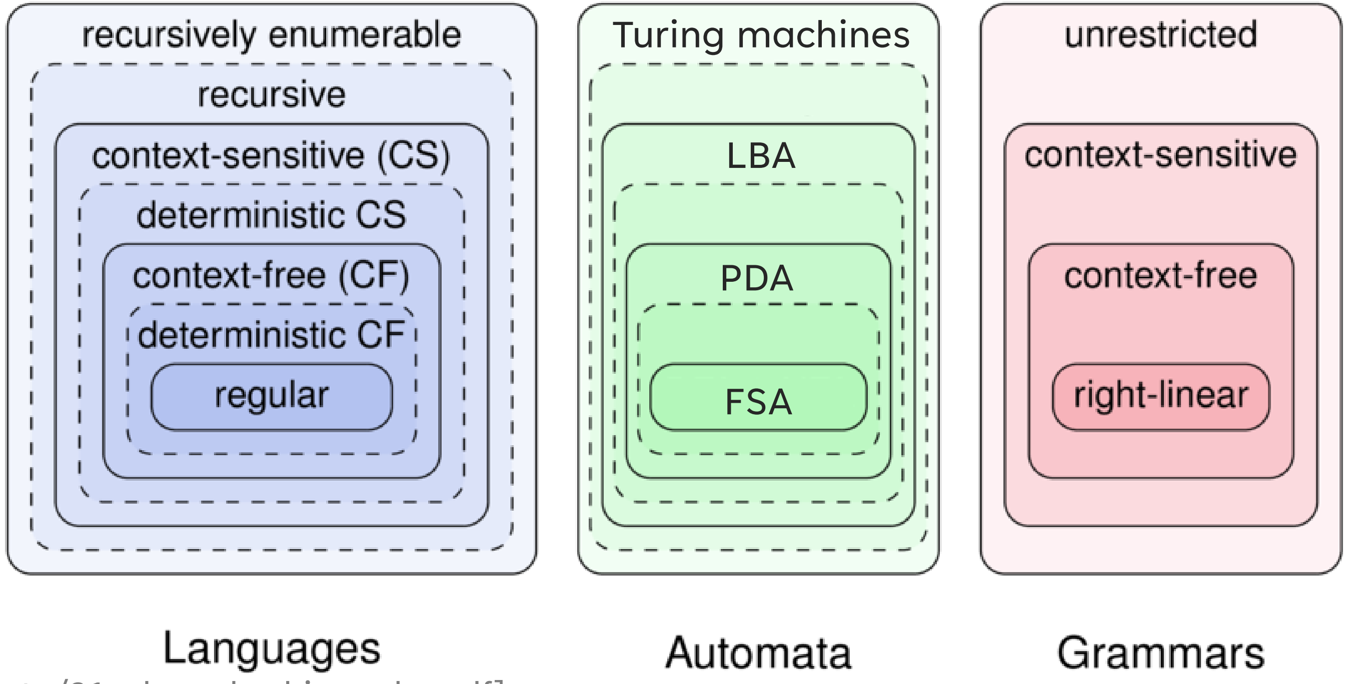
- What about context-sensitive languages?
  - The equivalent automata is a linear-bounded automata.



[Endrullis, joerg.endrullis.de/automata/21_chomsky_hierarchy.pdf]

CHOMSKY HIERARCHY

- What about context-sensitive languages?
  - The equivalent automata is a linear-bounded automata.
  - A linear-bounded automaton is a Turing machine with a finite tape.



[Endrullis, joerg.endrullis.de/automata/21_chomsky_hierarchy.pdf]

ML MODELS AND THE CHOMSKY HIERARCHY

- Where do ML/NLP models fit in the Chomsky hierarchy?
- In an earlier lecture, we saw that there exists regular languages that transformers are unable to learn.
  - But they are able to learn some non-regular languages.
- Delétang and Ruoss (2023) trained various neural architectures on different kinds of languages,
  - They tested whether each model could generalize out-of-distribution on each language.

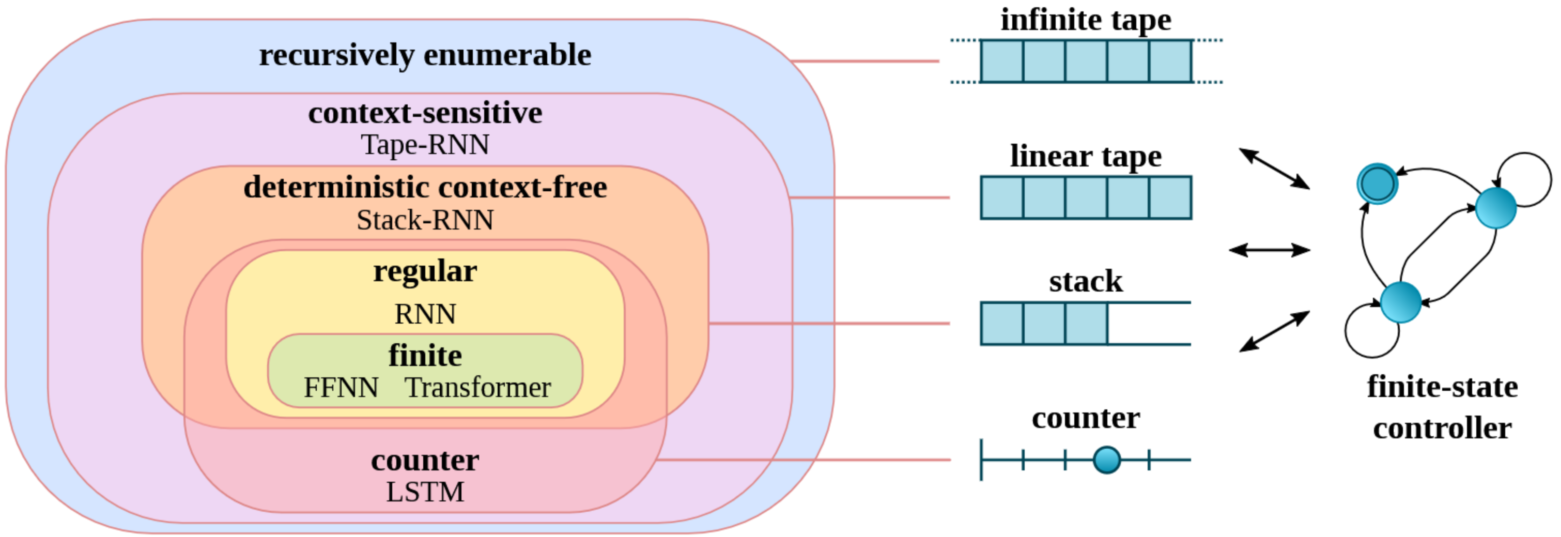
ML MODELS AND THE CHOMSKY HIERARCHY

Level	Task	RNN	Stack-RNN	Tape-RNN	Transformer	LSTM
R	Even Pairs	**100.0**	**100.0**	**100.0**	**96.4**	**100.0**
	Modular Arithmetic (Simple)	**100.0**	**100.0**	**100.0**	24.2	**100.0**
	Parity Check†	**100.0**	**100.0**	**100.0**	52.0	**100.0**
	Cycle Navigation†	**100.0**	**100.0**	**100.0**	61.9	**100.0**
DCF	Stack Manipulation	56.0	**100.0**	**100.0**	57.5	59.1
	Reverse String	62.0	**100.0**	**100.0**	62.3	60.9
	Modular Arithmetic	41.3	**96.1**	**95.4**	32.5	59.2
	Solve Equation°	51.0	56.2	64.4	25.7	67.8
CS	Duplicate String	50.3	52.8	**100.0**	52.8	57.6
	Missing Duplicate	52.3	55.2	**100.0**	56.4	54.3
	Odds First	51.0	51.9	**100.0**	52.8	55.6
	Binary Addition	50.3	52.7	**100.0**	54.3	55.5
	Binary Multiplication×	50.0	52.7	58.5	52.2	53.1
	Compute Sqrt	54.3	56.5	57.8	52.4	57.5
	Bucket Sort†★	27.9	78.1	70.7	**91.9**	**99.3**

[Delétang and Ruoss, 2023]

ML MODELS AND THE CHOMSKY HIERARCHY

- Their results led them to assign various ML models to different parts of the Chomsky hierarchy.
  - This assignment is not perfect, as they only tested a handful of languages within each level.
  - E.g., transformers performed well on one context-sensitive task.



[Delétang and Ruoss, 2023]

DOES CHAIN-OF-THOUGHT HELP?

- Merrill and Sabharwal (2024) showed that for any Turing machine,
  There exists a parameterization of a transformer that can simulate it.
- They proved this fact by construction:
  - They constructed a transformer that simulates one iteration of a Turing machine per forward pass.
  - Each output token represents a "diff" of the Turing machine tape.
  - The transformer can combine the diffs to reconstruct the symbol value under the read/write head.
- Implication: With CoT, transformers can express any algorithm.
- Not addressed: Can transformers learn any algorithm?

SEMANTICS

HOW TO MODEL MEANING?

- How do we model the meaning of natural language utterances?
- Language can convey lots of different kinds of meaning:
  - Truth-functional: ‘`Cats are mammals`’, ‘`Fish are not mammals.`’
  - Modality:
    - ‘`You must follow the law.`’
    - ‘`Compute the average of the list of numbers.`’
  - Emphasis/topicalization:
    - ‘`It wasn’t them who robbed the bank yesterday.`’
    - ‘`The bank wasn’t what they robbed yesterday.`’
    - ‘`It wasn’t yesterday that they robbed the bank.`’
  - This is not a comprehensive list.

FORMAL SEMANTICS

- Formal semantics is the study of formal representations of meaning.
- Ideally, a good model of meaning can capture all semantic information.
- For simplicity, we will focus on truth-functional meaning.
  - We can interpret the meaning of each sentence as a function.
  - The input to the function is a possible world (or "model").
  - The output is `TRUE` or `FALSE`.
- E.g., '`All insects have six legs`'
  - If the input possible world is one in which every instance of an insect has six legs, the output value is `TRUE`.
  - If the input possible world has an insect that does not have six legs, the output value is `FALSE`.

HOW TO MODEL MEANING?

- Truth-functional meaning representations are useful for tasks that require complex reasoning.
- For example:
  - Question answering
    - Especially multi-hop QA
  - Mathematical reasoning
  - Code generation
- Logic provides a natural way to represent meaning in these applications.

PROPOSITIONAL LOGIC

- Propositional logic or propositional calculus is a logic that describes propositions and relations between them.
  - Propositions are represented as symbols or variables.
  - Each proposition can either be `TRUE` or `FALSE`.
- E.g., the logical form of '`Sally is a cat`' is `sally_is_cat = TRUE`.
  - '`Sally is a cat and Bob likes dogs`'
    - `sally_is_cat & bob_likes_dogs`
    - `sally_is_cat ∧ bob_likes_dogs`
  - '`Sally is a cat or Bob likes dog`'
    - `sally_is_cat | bob_likes_dogs`
    - `sally_is_cat ∨ bob_likes_dogs`

PROPOSITIONAL LOGIC

- Propositional logic has the following operators/relations:
  - Conjunction ("and"):
    - `A & B` is true if and only if both `A` and `B` are true
  - Disjunction ("or"):
    - `A | B` is true if either `A` or `B` are true
  - Negation ("not"):
    - `~A` or `!A` or `-A` or `¬A` is true if `A` is false
  - Implication ("if-then", "implies"):
    - `A -> B` or `A => B` or `A ⊃ B` is true if `B` is true whenever `A` is true

PROPOSITIONAL LOGIC

- One common problem in reasoning is deduction.
  - Given some premises, prove whether a conclusion is `TRUE` or `FALSE`.
  - E.g., Given `sally_is_cat` and `bob_likes_dogs`,
  - Prove: `(sally_is_cat & bob_likes_dogs) | charlie_has_wings`
- This is equivalent to proving that in any possible world where the premises are true, the conclusion is necessarily true.
  - In propositional logic, we can search over all possible truth value assignments to the propositions.
    - Exponential running time
  - Inference in propositional logic can be shown to be NP-complete.

BETTER MODEL OF REASONING?

- Searching over all possible truth assignments is not a good model of human reasoning.
  - And it may not be a good model of reasoning more generally if we want to design systems that can solve complex reasoning problems.
- Can we model reasoning as a step-by-step process?
- We can define inference rules that tell us how to deduce new true facts.

$$\frac{A \; \mathit{true} \quad B \; \mathit{true}}{A \wedge B \; \mathit{true}} \wedge I$$

- In this rule, given the premises `A` is true and `B` is true,
  - We can conclude that `A & B` is true.

PROOFS

- In addition to conjunction introduction, we can define a conjunction elimination rule.

$$\frac{A \wedge B \ true}{A \ true} \wedge E$$

- We can similarly define inference rules for other logical connectives:

$$\frac{A \ true}{A \vee B \ true} \vee I$$

$$\frac{A \vee B \ true \quad \begin{array}{c} \overline{A \ true}^{\,u} \\ \vdots \\ C \ true \end{array} \quad \begin{array}{c} \overline{B \ true}^{\,w} \\ \vdots \\ C \ true \end{array}}{C \ true} \vee E^{u,w}$$

This rule is also called proof-by-cases

PROOFS

- We can similarly define inference rules for other logical connectives:

$$\frac{A \supset B \ \mathit{true} \quad A \ \mathit{true}}{B \ \mathit{true}} \supset E$$

This rule is also called "modus ponens"

$$\frac{\begin{array}{c}\overline{A \ \mathit{true}}^{\,u} \\ \vdots \\ B \ \mathit{true}\end{array}}{A \supset B \ \mathit{true}} \supset I^u$$

[Pfenning, Automated Theorem Proving Course Notes, 2004]

PROOFS

- We can similarly define inference rules for other logical connectives:

$$\dfrac{\dfrac{}{\neg A}u \\ \vdots \\ \bot}{A}\bot_C^u$$

This rule is also called proof by contradiction

$$\dfrac{\neg A\ \mathit{true} \qquad A\ \mathit{true}}{C\ \mathit{true}}\neg\mathrm{E}$$

[Pfenning, Automated Theorem Proving Course Notes, 2004]

PROOFS

- We can use proofs to solve the earlier reasoning example:
  - E.g., Given `sally_is_cat` and `bob_likes_dogs`,
  - Prove: `(sally_is_cat & bob_likes_dogs) | charlie_has_wings`

$$\dfrac{\dfrac{\dfrac{}{\texttt{sally\_is\_cat}}\,\texttt{Ax} \qquad \dfrac{}{\texttt{bob\_likes\_dogs}}\,\texttt{Ax}}{\texttt{sally\_is\_cat \& bob\_likes\_dogs}}\,\texttt{\&I}}{\texttt{(sally\_is\_cat \& bob\_likes\_dogs) | charlie\_has\_wings}}\,\texttt{|I}$$

- The step-by-step proof also provides a better model of chain-of-thought-style reasoning in LMs.
- The above is an example of a depth-2 proof.

PROPOSITIONAL LOGIC LIMITATIONS

- Propositional logic does not cover the meaning of all natural language.
- E.g., '`All states have capitals`' or '`There is a city on the river.`'
- In propositional logic, we can only express statements about specific objects.
  - In order to represent the above sentences, we need a way to express statements over all objects.
- Propositional logic can be extended by adding variables and quantifiers.
  - '`All states have capitals`'
    - `∀x(state(x) -> ∃y(capital(y) & has(x,y)))`
  - '`There is a city on the river`'
    - `∃x(city(x) & on(x,the_river))`

FIRST-ORDER LOGIC

- First-order logic (FOL) is the logic containing propositional logic with universal and existential quantifiers.
- To enable reasoning with first-order logic, we can define additional inference rules for the quantifiers:

$$\frac{[a/x]A \ \mathit{true}}{\forall x.\ A \ \mathit{true}}\ \forall\mathrm{I}^a \qquad\qquad \frac{\forall x.\ A \ \mathit{true}}{[t/x]A \ \mathit{true}}\ \forall\mathrm{E}$$

- $[a/x]A$ denotes the result of substituting x with a in A.

[Pfenning, Automated Theorem Proving Course Notes, 2004]

FIRST-ORDER LOGIC

- First-order logic (FOL) is the logic containing propositional logic with universal and existential quantifiers.
- To enable reasoning with first-order logic, we can define additional inference rules for the quantifiers:

$$\frac{[t/x]A\ \mathit{true}}{\exists x.\ A\ \mathit{true}}\ \exists\mathrm{I}
\qquad\qquad
\frac{\exists x.\ A\ \mathit{true} \qquad \begin{array}{c}\overline{[a/x]A\ \mathit{true}}^{\,u} \\ \vdots \\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \exists\mathrm{E}^{a,u}$$

- $[a/x]A$ denotes the result of substituting x with a in A.

[Pfenning, Automated Theorem Proving Course Notes, 2004]

FIRST-ORDER LOGIC

- In the worst-case, reasoning in first-order logic is significantly more computationally expensive than in propositional logic.
- In fact, the problem of determining whether a given logical form is true in first-order logic is undecidable.
- However, FOL is also highly expressive.
- Mathematics can largely be written in FOL.
  - See Zermelo-Fraenkel set theory.
  - The most common "foundation of mathematics."

CAN LLMS REASON USING FOL?

- Han et al (2024) create a dataset of FOL reasoning problems, written in both natural language and logic.

A FOLIO example based on the Wild Turkey Wikipedia page: https://en.wikipedia.org/wiki/Wild_turkey

NL premises	NL Conclusions -> Labels
1. There are six types of wild turkeys: Eastern wild turkey, Osceola wild turkey, Gould's wild turkey, Merriam's wild turkey, Rio Grande wild turkey, and the Ocellated wild turkey. 2. Tom is not an Eastern wild turkey. 3. Tom is not an Osceola wild turkey. 4. Tom is also not a Gould's wild turkey. 5. Tom is neither a Merriam's wild turkey, nor a Rio Grande wild turkey. 6. Tom is a wild turkey.	A. Tom is an Ocellated wild turkey. -> True B. Tom is an Eastern wild turkey. -> False C. Joey is a wild turkey. -> Unknown
FOL Premises	**FOL conclusions -> Labels**
1. $\forall x(\text{WildTurkey}(x) \rightarrow (\text{EasternWildTurkey}(x) \lor \text{OsceolaWildTurkey}(x) \lor \text{GouldsWildTurkey}(x) \lor \text{MerriamsWildTurkey}(x) \lor \text{RiograndeWildTurkey}(x) \lor \text{OcellatedWildTurkey}(x)))$ 2. $\neg\text{EasternWildTurkey}(tom)$ 3. $\neg\text{OsceolaWildTurkey}(tom))$ 4. $\neg\text{GouldsWildTurkey}(tom)$ 5. $\neg\text{MerriamsWildTurkey}(tom) \land \neg\text{RiograndeWildTurkey}(tom)$ 6. $\text{WildTurkey}(tom)$	A. $\text{OcellatedWildTurkey}(tom)$ -> True B. $\text{EasternWildTurkey}(tom)$ -> False C. $\text{WildTurkey}(joey)$ -> Unknown

[Han et al, 2004]

CAN LLMS REASON USING FOL?

- They use Wikipedia as the source of their examples.

A FOLIO example based on the Wild Turkey Wikipedia page: https://en.wikipedia.org/wiki/Wild_turkey	
NL premises 1. There are six types of wild turkeys: Eastern wild turkey, Osceola wild turkey, Gould's wild turkey, Merriam's wild turkey, Rio Grande wild turkey, and the Ocellated wild turkey. 2. Tom is not an Eastern wild turkey. 3. Tom is not an Osceola wild turkey. 4. Tom is also not a Gould's wild turkey. 5. Tom is neither a Merriam's wild turkey, nor a Rio Grande wild turkey. 6. Tom is a wild turkey.	**NL Conclusions -> Labels** A. Tom is an Ocellated wild turkey. -> True B. Tom is an Eastern wild turkey. -> False C. Joey is a wild turkey. -> Unknown
FOL Premises 1. $\forall x(\text{WildTurkey}(x) \rightarrow (\text{EasternWildTurkey}(x) \vee \text{OsceolaWildTurkey}(x) \vee \text{GouldsWildTurkey}(x) \vee \text{MerriamsWildTurkey}(x) \vee \text{RiograndeWildTurkey}(x) \vee \text{OcellatedWildTurkey}(x)))$ 2. $\neg\text{EasternWildTurkey}(tom)$ 3. $\neg\text{OsceolaWildTurkey}(tom))$ 4. $\neg\text{GouldsWildTurkey}(tom)$ 5. $\neg\text{MerriamsWildTurkey}(tom) \wedge \neg\text{RiograndeWildTurkey}(tom)$ 6. $\text{WildTurkey}(tom)$	**FOL conclusions -> Labels** A. $\text{OcellatedWildTurkey}(tom)$ -> True B. $\text{EasternWildTurkey}(tom)$ -> False C. $\text{WildTurkey}(joey)$ -> Unknown

[Han et al, 2004]

CAN LLMS REASON USING FOL?

- They find LLMs perform decently.
  - But data contamination may confound their results.

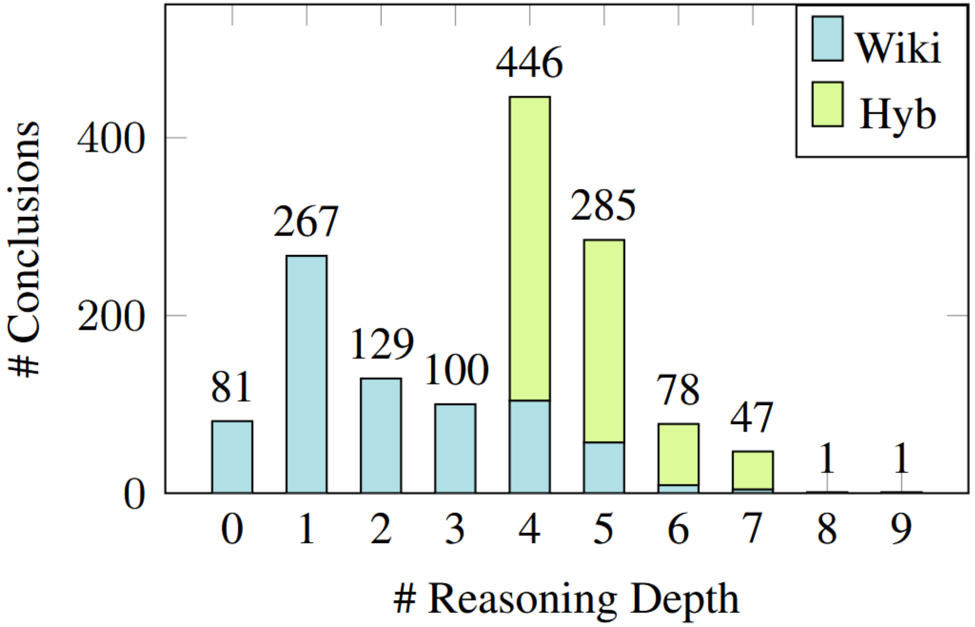
Model	Size	Acc (%)
majority baseline	-	38.5%
random probability	-	33.3 %
Fully supervised fine-tune		
BERT-base	110M	56.8
BERT-large	340M	59.0
RoBERTa-base	110M	56.8
RoBERTa-large	340M	62.1
Flan-T5-Large	783M	**65.9**
0-shot NL Prompt		
GPT-3.5-Turbo	-	53.1
GPT-4	-	61.3

8-shot NL Prompt		
LLama-13B	13B	33.6
LLama-70B	70B	44.0
LLama-70B - CoT	70B	47.8
LLama-70B - ToT	70B	48.4
text-davinci-002	-	49.5
GPT-3.5-Turbo	-	58.3
GPT-4	-	64.2
GPT-4 - CoT (2022b)	-	68.9
GPT-4 - CoT with SC (2023)	-	69.5
GPT-4 ToT (2023)	-	70.0
LR-specific Methods		
Logic-LM (2023)	-	78.1
LINC (2023)	-	73.1
DetermLR (2023)	-	77.5

[Han et al, 2004]

CAN LLMS REASON USING FOL?

- They find LLMs perform decently.
  - But data contamination may confound their results.
  - The complexity of their examples is rather limited.



[Han et al, 2004]

HOW DO WE CONVERT NL INTO LF?

- We have described two logics to represent the meaning of natural language sentences:
  - Propositional logic
  - First-order logic
- How do you actually convert a sentence into logical form?
- Is FOL a good representation for meaning in natural language?
  - ‘`Mark drives`’ can be parsed as `drive(mark)`,
  - But how would you parse ‘`Mark drives quickly`’?
- We will discuss these questions further next time.

QUESTIONS?