Doing Formal Language Theory

Huteng Dai (huteng.dai@rutgers.edu)

Link to the Python code: https://tinyurl.com/2p99j4u7

Roadmap

- Jäger & Rogers (2012)
    1. Chomsky Hierarchy and cognitive complexity
    2. Application in natural language syntax
    3. Application in phonology: subregular hierarchy
       (return to first-order logic in last meeting)
    4. Artificial Grammar Learning (AGL)
- Team-based games incoming—get ready!

We will learn

- A fresh perspective from (theoretical) computer science
- …even questions in coding interviews.
- I will only talk about discrete models here.

Assumptions

- I will only talk about discrete models here.

Chomsky Hierarchy: overview

Background

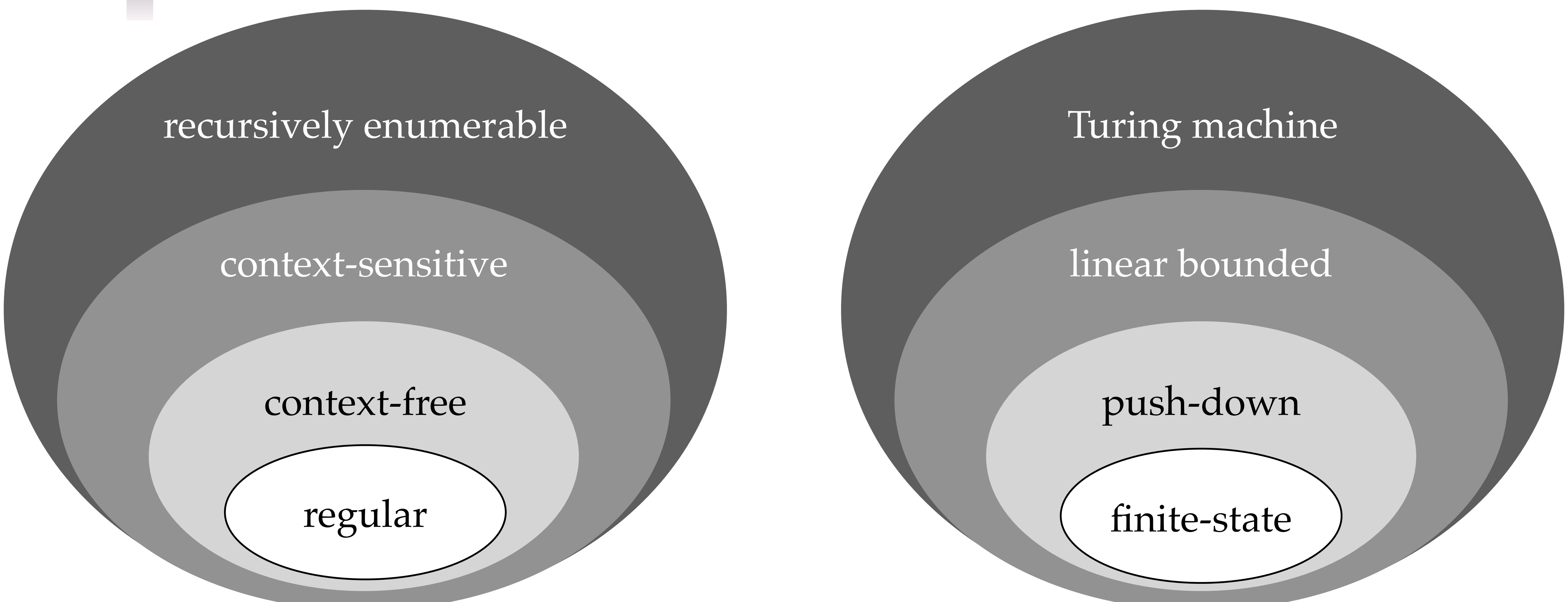
1. Turing (1950): “Can machines think?”
   - Turing proved that any machine can simulate the behavior of any other machine, given enough memory and time.
   - Also earliest state machine and the idea of learning machine.
2. Chomsky (1956) “Three Models for The Description of Language”
   - Military-sponsored project to teach computers to understand English commands, but "...basically the military didn’t care what you were doing."
   - Chomsky asked: How can we provide a finite grammar that generates all the sentences of English and only these?

Chomsky Hierarchy

more working memory
more complex
more logical power
less restricted

grammars (left) and automata (right)

Chomsky 1956, 1959; Chomsky & Schützenberger 1963



Regular language

rewrite rules (left) and automata (right)

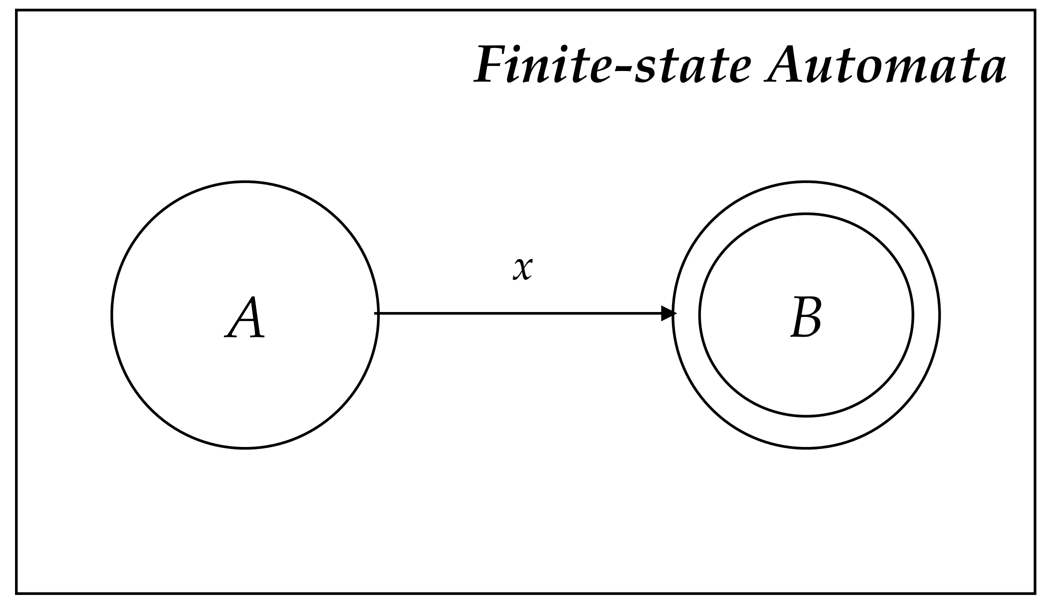
“replace A with xB“

Format: $A \rightarrow xB$ *or* $A \rightarrow x$

x: terminal

A/B: nonterminal

can’t have $A \rightarrow xBy$

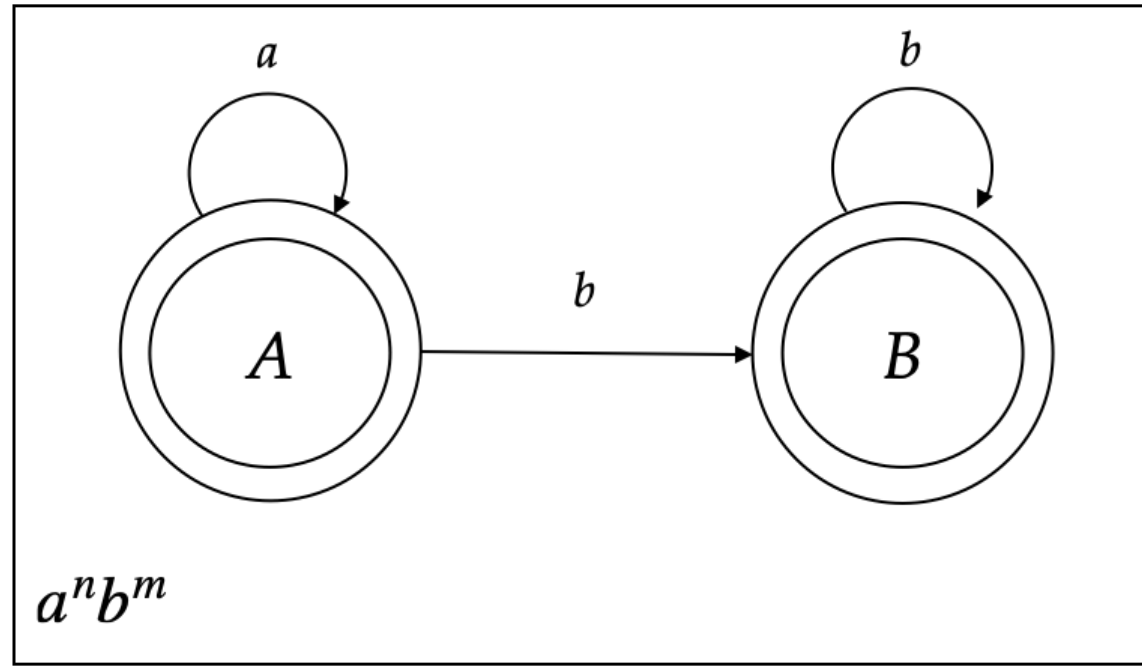


regular

finite-state

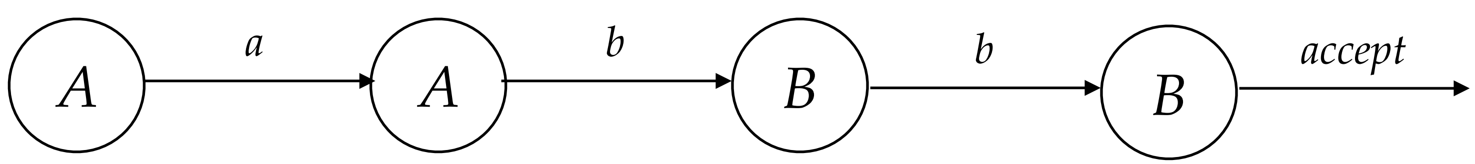
1st line: starting symbol

$$S \rightarrow aA$$
$$A \rightarrow bB$$
$$A \rightarrow aA$$
$$B \rightarrow bB$$
$$B \rightarrow \varepsilon$$

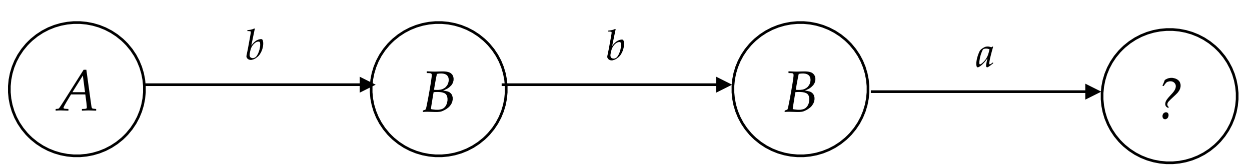


Finite-state Automata

Input: *abb*



Input: *bba*



Link to the Python code: https://tinyurl.com/2p99j4u7

Pikachu language

pi	*kapichu
pika	*kachu
pikachu	*chupi
pikapika	*chu
pikapikachuuuuu	*pichu

FSA?

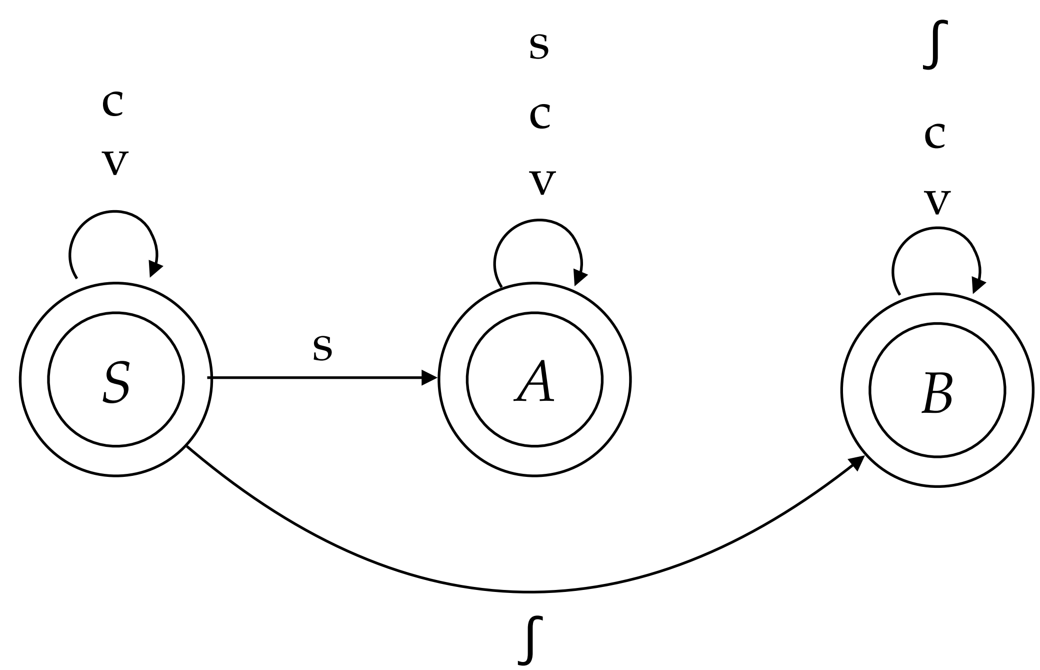
Navajo sibilant harmony

$S \rightarrow cS$
$S \rightarrow vS$
$S \rightarrow sA$
$S \rightarrow ʃB$

$A \rightarrow cA$
$A \rightarrow vA$
$A \rightarrow sA$

$B \rightarrow cB$
$B \rightarrow vB$
$B \rightarrow ʃB$

$S \rightarrow \varepsilon$
$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$



v: any vowel
c: any consonant other than {s, ʃ}

from Heinz (2010)

Stress patterns

A. Primary stress falls on the initial syllable and there is no secondary stress.
B. Primary stress falls on the final syllable. and secondary stress falls on other odd syllables, counting from the right.

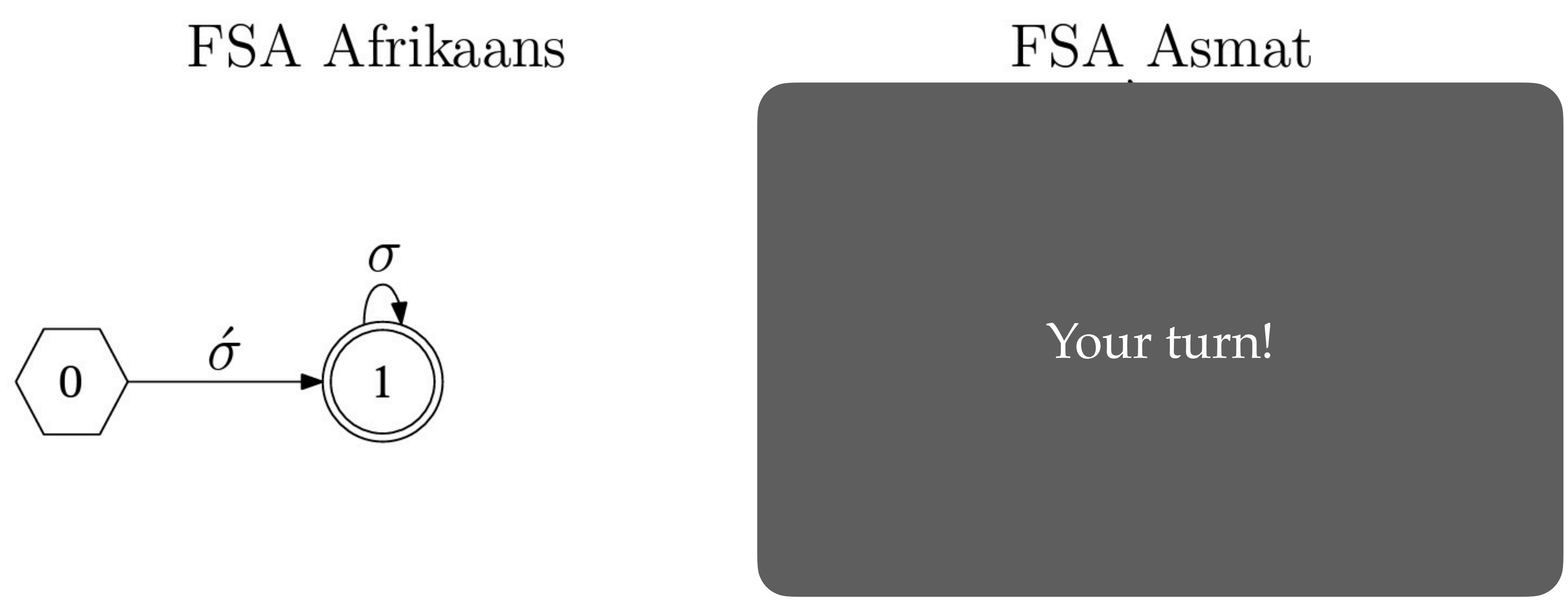


Figure 1: The stress patterns of Afrikaans and Asmat

Some syntactic rules are regular

start symbol: A

A	$\rightarrow$	*someone* B
B	$\rightarrow$	*really* B
B	$\rightarrow$	*ran* C
B	$\rightarrow$	*ran*
C	$\rightarrow$	*and* A
C	$\rightarrow$	*and* B
C	$\rightarrow$	*really* D
D	$\rightarrow$	*really* D
D	$\rightarrow$	*quickly* C
D	$\rightarrow$	*quickly*

(a) Rewrite-rule representation

Your turn!

(b) Graphical representation

Figure 2: A simple finite-state grammar, represented graphically and as rewrite rules

from Hunter (2020)

FSAs are awesome

1. Well-defined: Myhill-Nerode Theorem;
   - You can intersect several FSAs to get another FSA;
   - You can encode an entire corpus in an FSA;
2. You don’t need a separate memory storage;
3. They are **directed graphic models**: you can convert them to Hidden Markov Models and Bayesian networks with *some* twists.

Myhill-Nerode Theorem

Given a language L and x, y are string over $\sum^*$, if for every string $z \in \sum^*$, xz, $yz \in L$ or xz, $yz \notin L$ then x and y are said to be indistinguishable over language L. Formally, we denote that x and y are indistinguishable over L by the following notation : $x \equiv_L y$.

A language is regular if and only if $\equiv_L$ partitions $\sum^*$ into finitely many equivalence classes. If $\equiv_L$ partitions $\sum^*$ into n equivalence classes, then a minimal DFA recognizing L has exactly n states.

- In practice, it's used to minimize DFAs by merging states that are equivalent, thus simplifying the automaton without changing the language it recognizes.
- Theoretically, it provides a way to prove that certain languages are not regular by demonstrating that there are infinitely many equivalence classes (meaning there's no way to construct a finite automaton to recognize the language).

Regular *vs.* non-regular

Table 2. Regular and non-regular languages.

regular languages	non-regular languages
$a^n b^m$	$a^n b^n$
the set of strings x such that the number of 'a's in x is a multiple of 4	the set of strings x such that the number of 'a's and the number of 'b's in x are equal
the set of natural numbers that leave a remainder of 3 when divided by 5	

Link to the Python code: https://tinyurl.com/2p99j4u7

from Jäger & Rogers (2012)

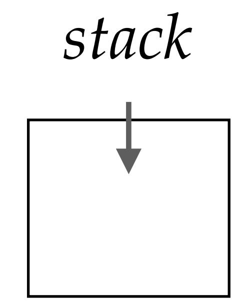
Homework

Describe Japanese syllable structure in **the most restrictive logic and automata** you learned. Most syllables in Japanese conform to (C)V(N), where C is a consonant, V is a vowel, and N is a nasal consonant that can appear at the end of syllables.

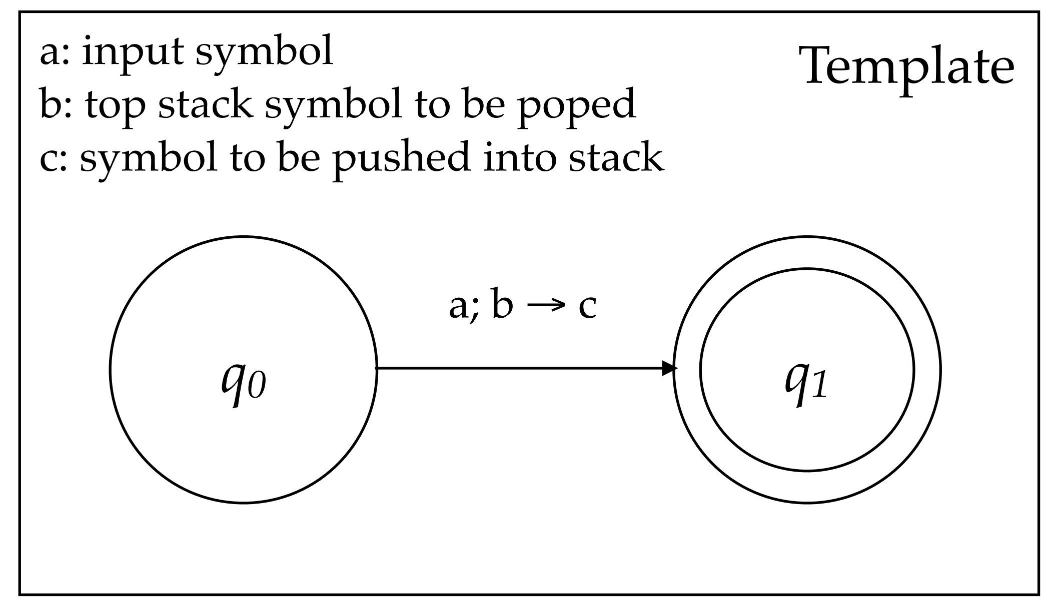
Japanese	**English**	**Syllable**
sakura さくら	Cherry blossom	CV.CV.CV
tomodachi ともだち	Friend	CV.CV.CV.CV
shinbun しんぶん	Newspaper	CVN.CVN
nihongo にほんご	Japanese Language	CV.CVN.CV
tsukue つくえ	Desk	CV.CV.V

Optional: write a code to recognize Japanese syllable structure!

Context-free language

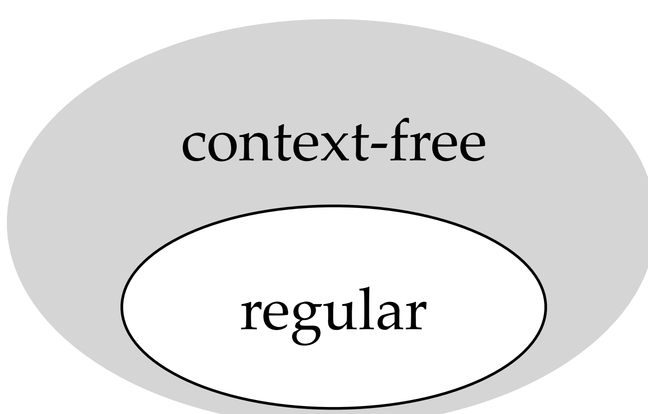


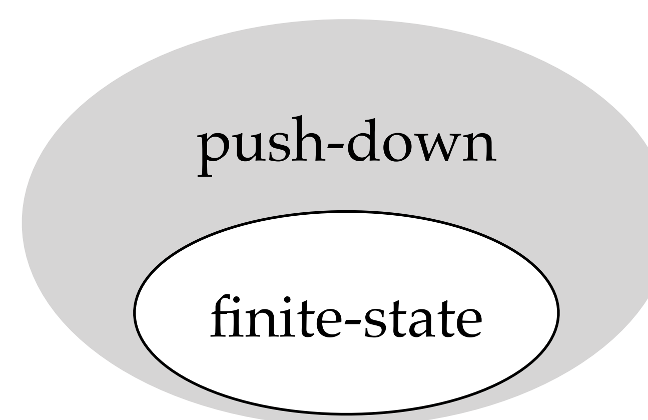
First in last out



Format: $A \rightarrow \omega$

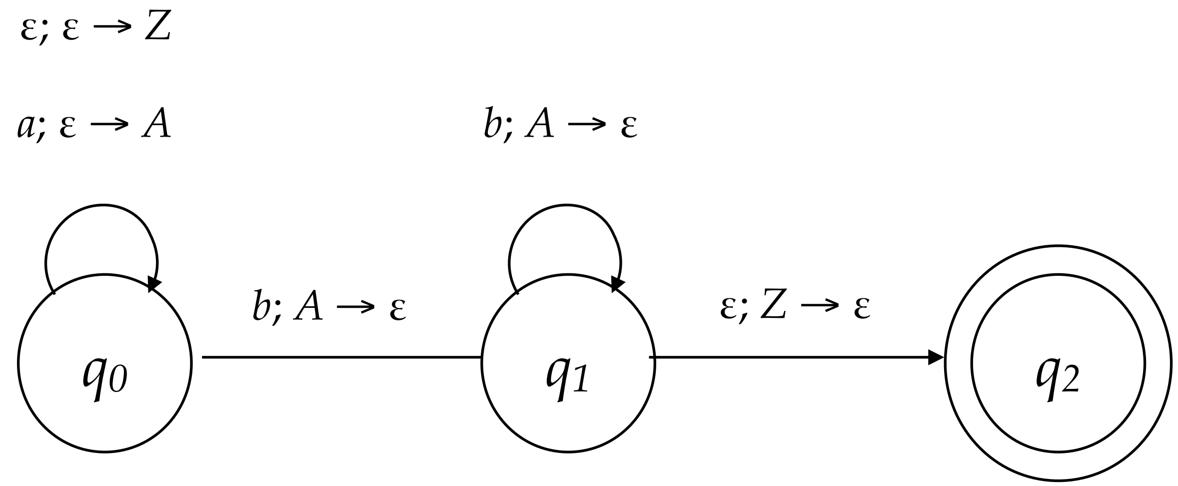
ω: non-empty string of either terminal or non-terminal





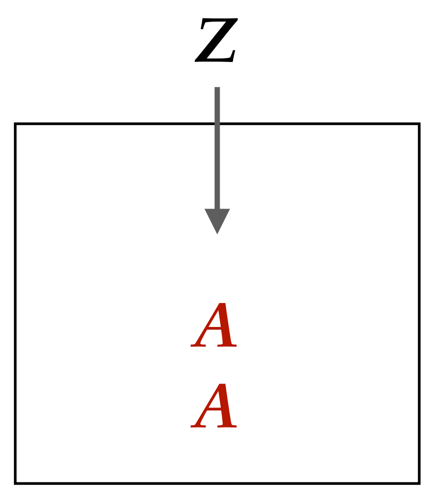
$a^n b^n, n \geq 1$

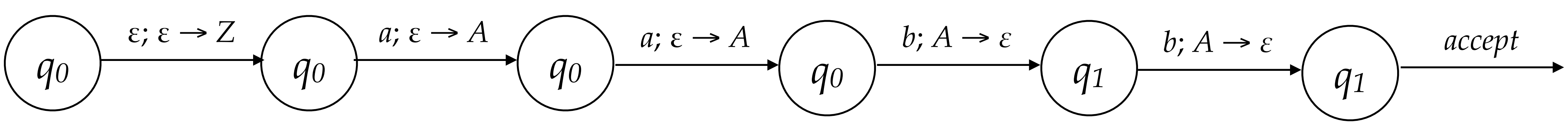
$$S \rightarrow aAb$$
$$A \rightarrow aAb$$
$$A \rightarrow \varepsilon$$



Input: *aabb*

Push-down Automata





English nested embeddings

$A \rightarrow$ *neither* A *nor*

$A \rightarrow \varepsilon$

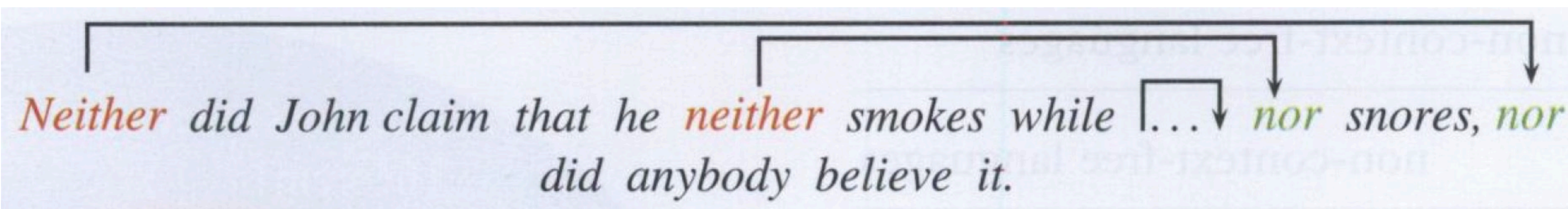


Table 1. Context-free and non-context-free languages

context-free languages	non-context-free languages
mirror language (i.e. the set of strings xy over a given Σ such that y is the mirror image of x)	*copy language* (i.e. the set of strings xx over a given Σ such that x is an arbitrary string of symbols from Σ)
palindrome language (i.e. the set of strings x that are identical to their mirror image)	
$a^n b^n$	$a^n b^n c^n$
$a^n b^m c^m d^n$	$a^n b^m c^n d^m$
well-formed programs of Python (or any other high-level programming language)	
Dyck language (the set of well-nested parentheses)	
well-formed arithmetic expression	

Homework 2: Dyck

Describe Dyck language, the set of well nested parentheses in pushdown automata.

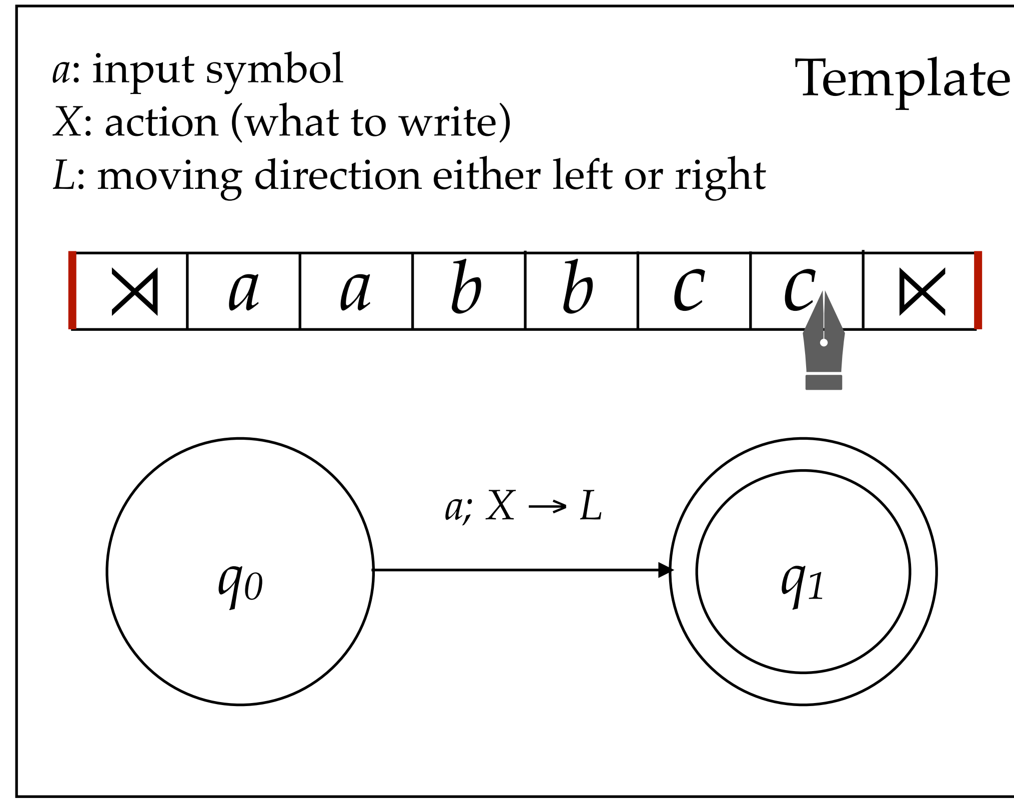
Optional: write a code to recognize Dyck language!

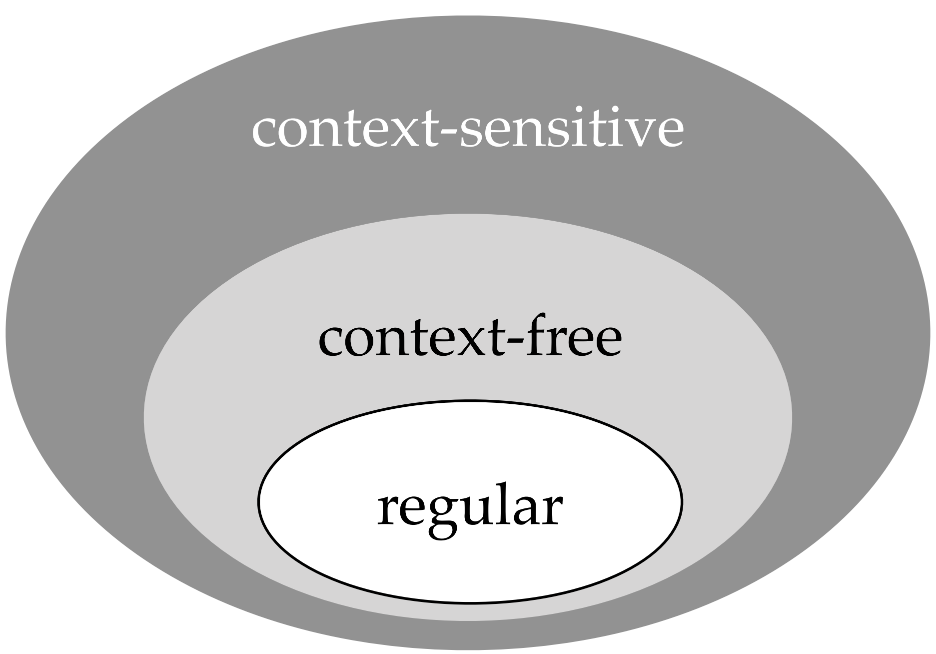
Context-sensitive language

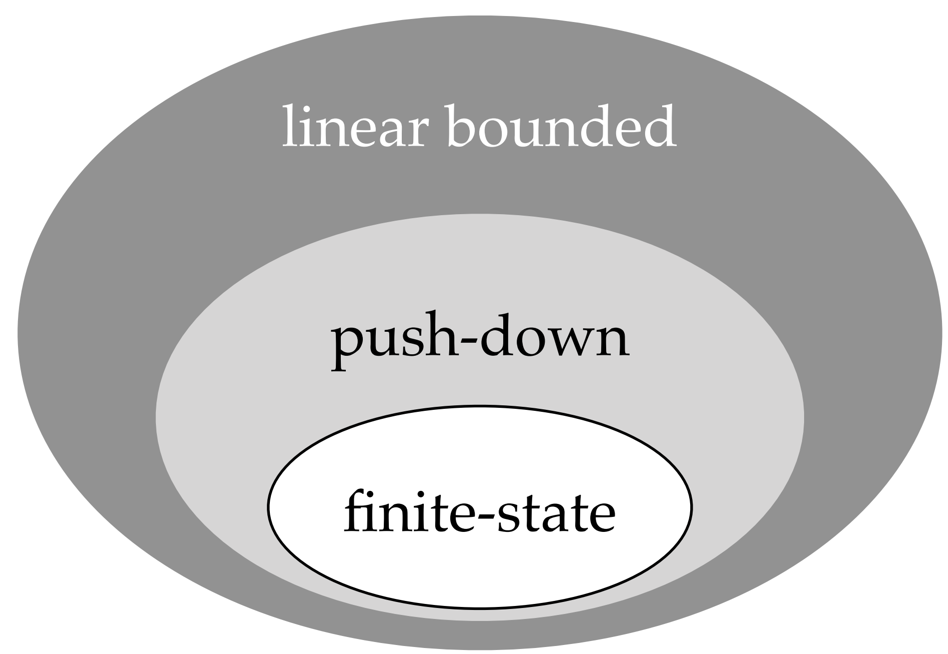
Format: $\phi A \psi \rightarrow \phi \omega \psi$

ϕ, ω, ψ: non-empty string of either terminal or non-terminal

$|\phi| \leq |\psi|$

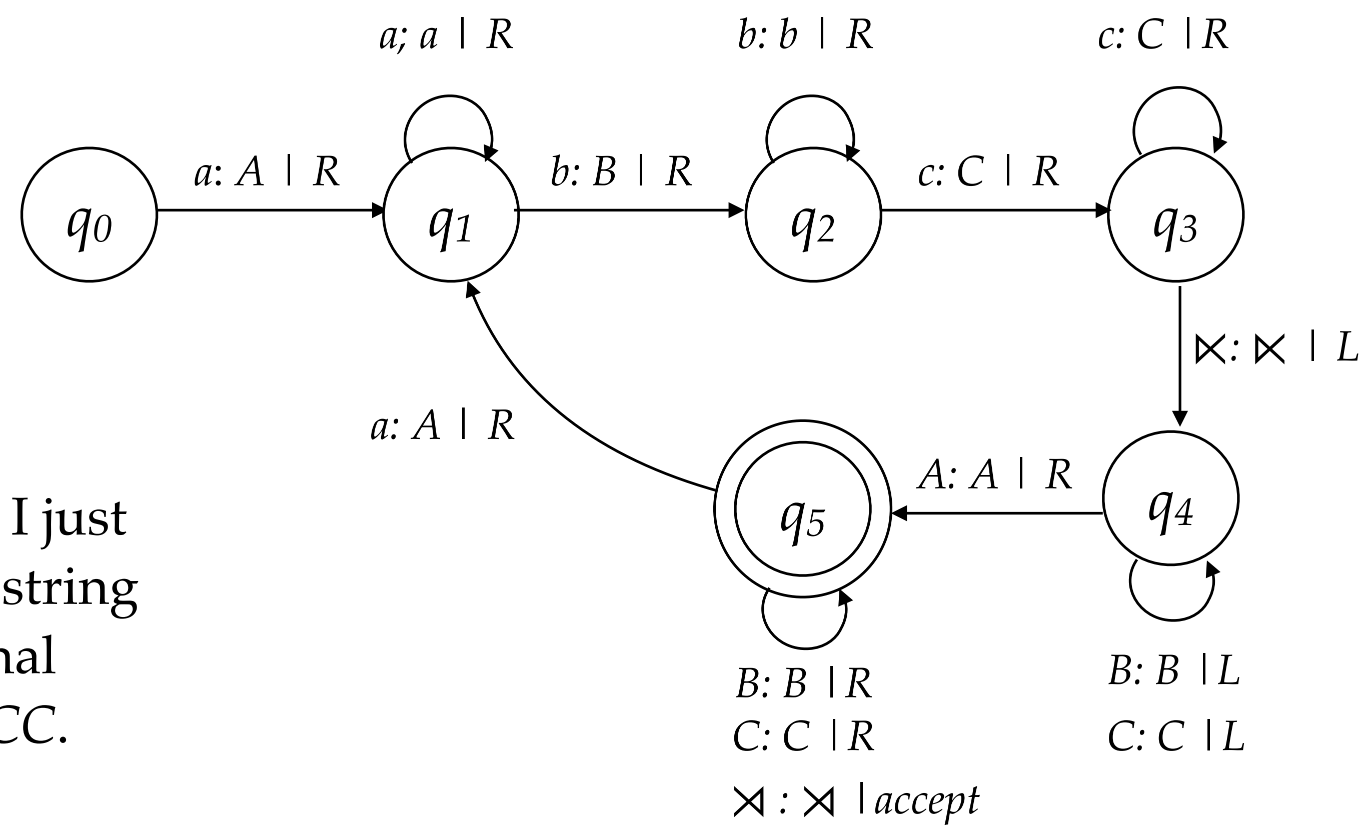






$a^n b^n c^n, n \geq 1$

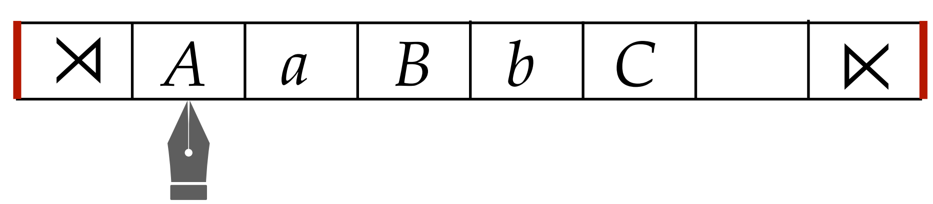
S → *abc* | *aAbc*
Ab → *bA*
Ac → *Bbcc*
bB → *Bb*
aB → *aa* | *aaA*

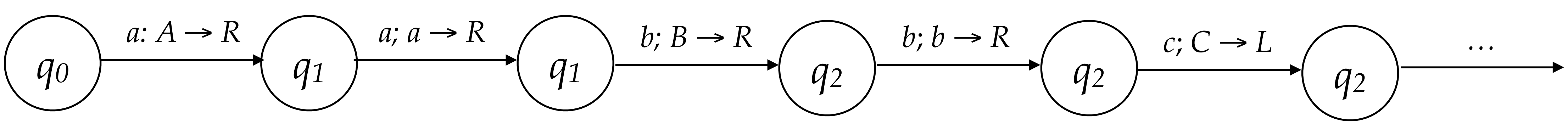


The head can go any direction, here I just show one alternative automata; the string is accepted once there is only terminal symbols on the tape, such as *AABBCC*.

Input: *aabbcc*

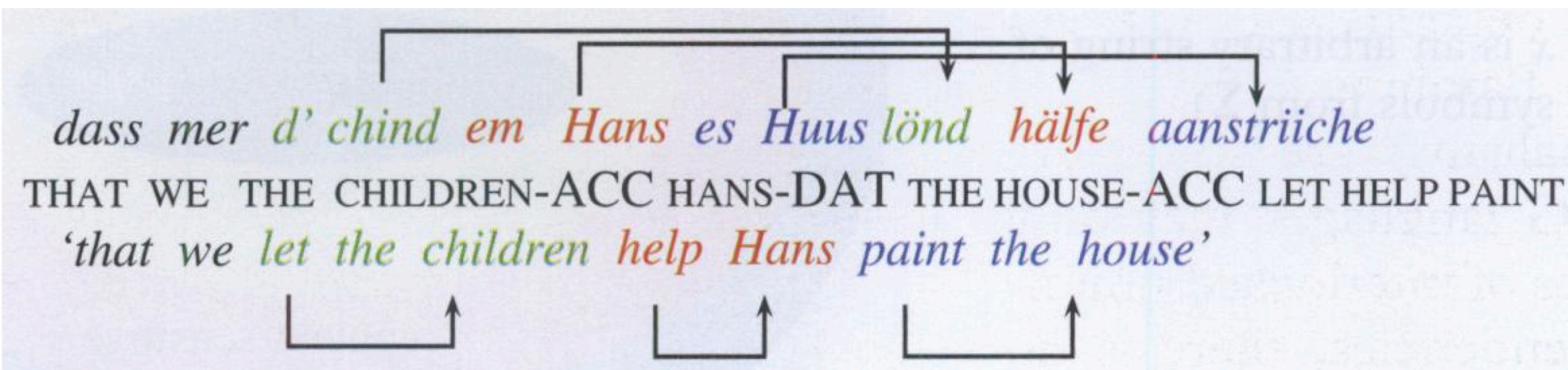
Linear bounded Automata





Swiss German cross-serial dependencies

$$a^n b^n c^n, n \geq 1$$

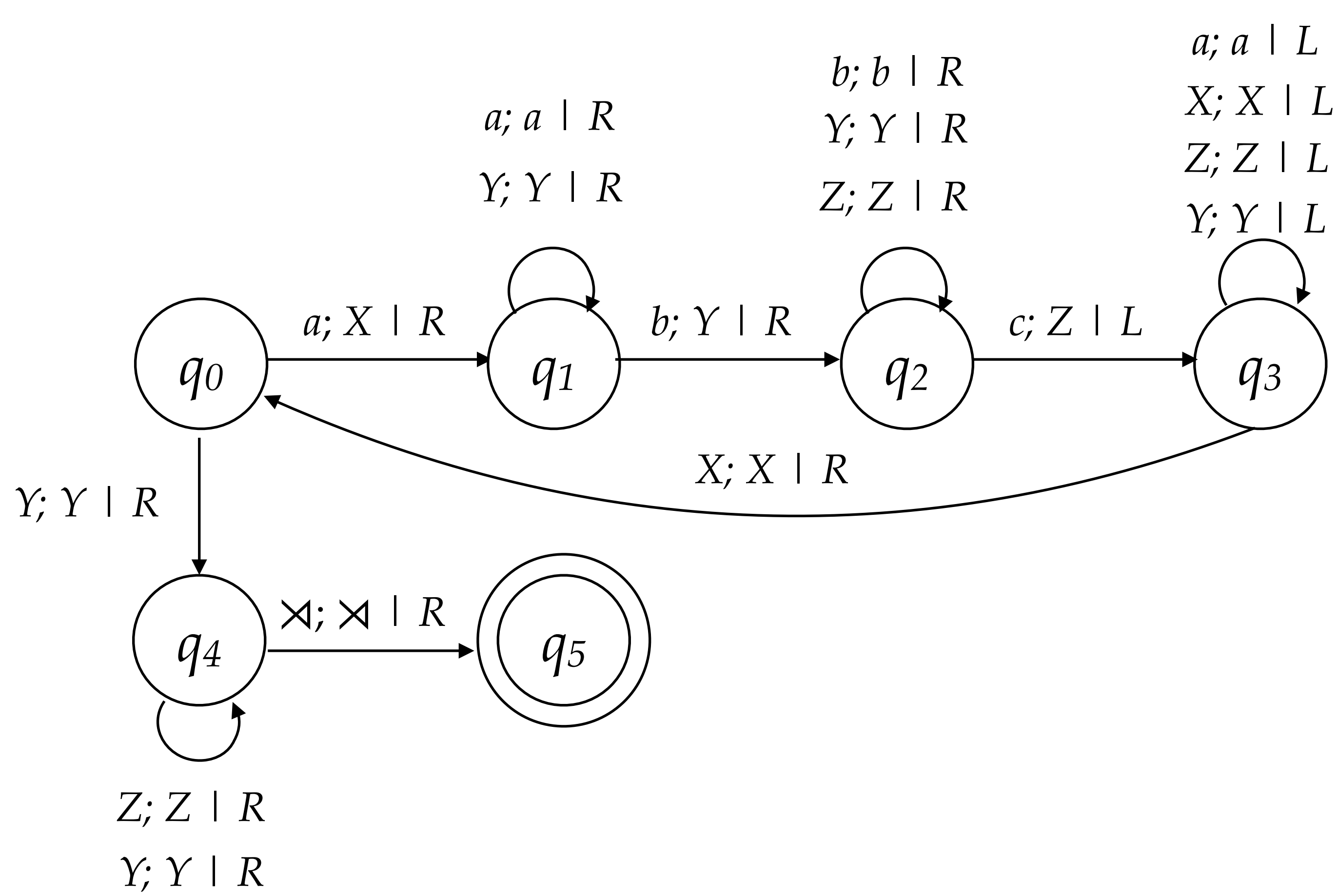


Three models

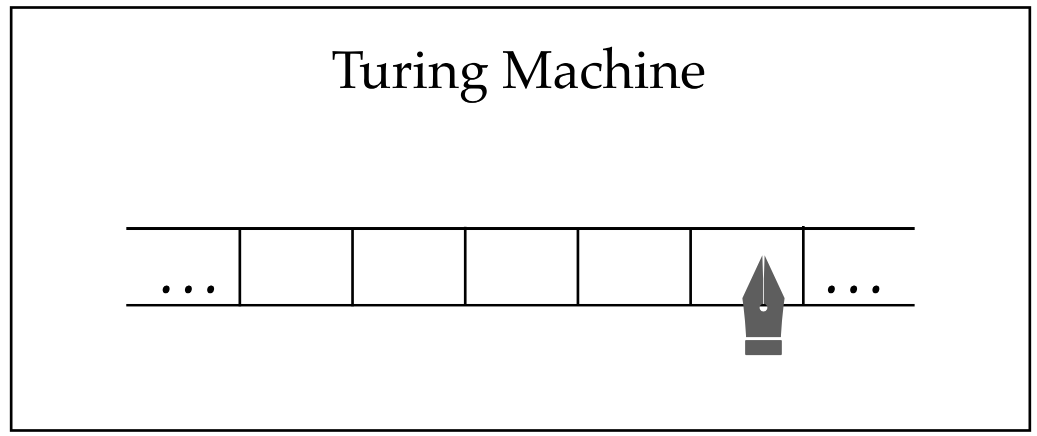
Model	Format	Notation
Type-1 context-sensitve	$\phi A \psi \to \phi \omega \psi$	ϕ, ω, ψ: non-empty string of either terminal or non-terminal
Type-2 context-free	$A \to \omega$	ω: non-empty string of either terminal or non-terminal
Type-3 regular	$A \to xB$ *or* $A \to x$	x: terminal, A/B: nonterminal

$a^n b^n c^n, n \geq 1$

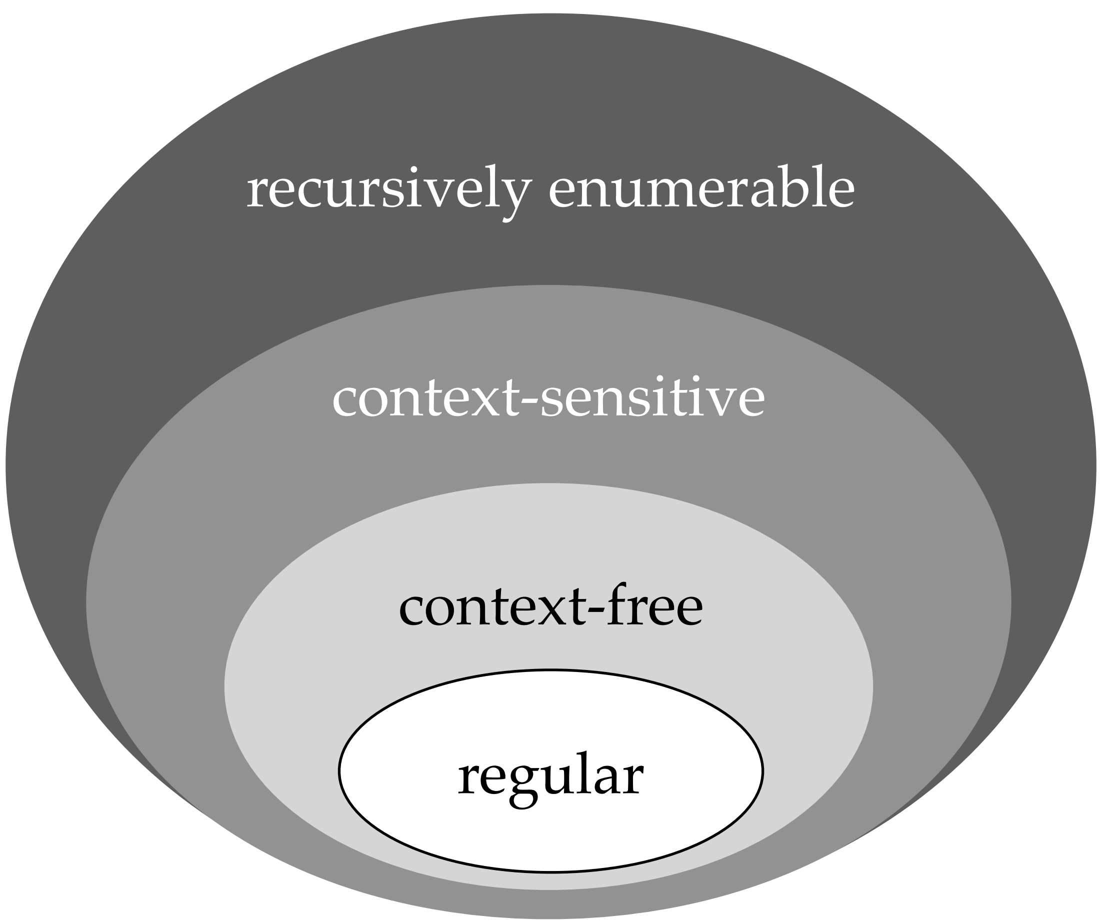
$a \rightarrow X$
$b \rightarrow Y$
$c \rightarrow Z$

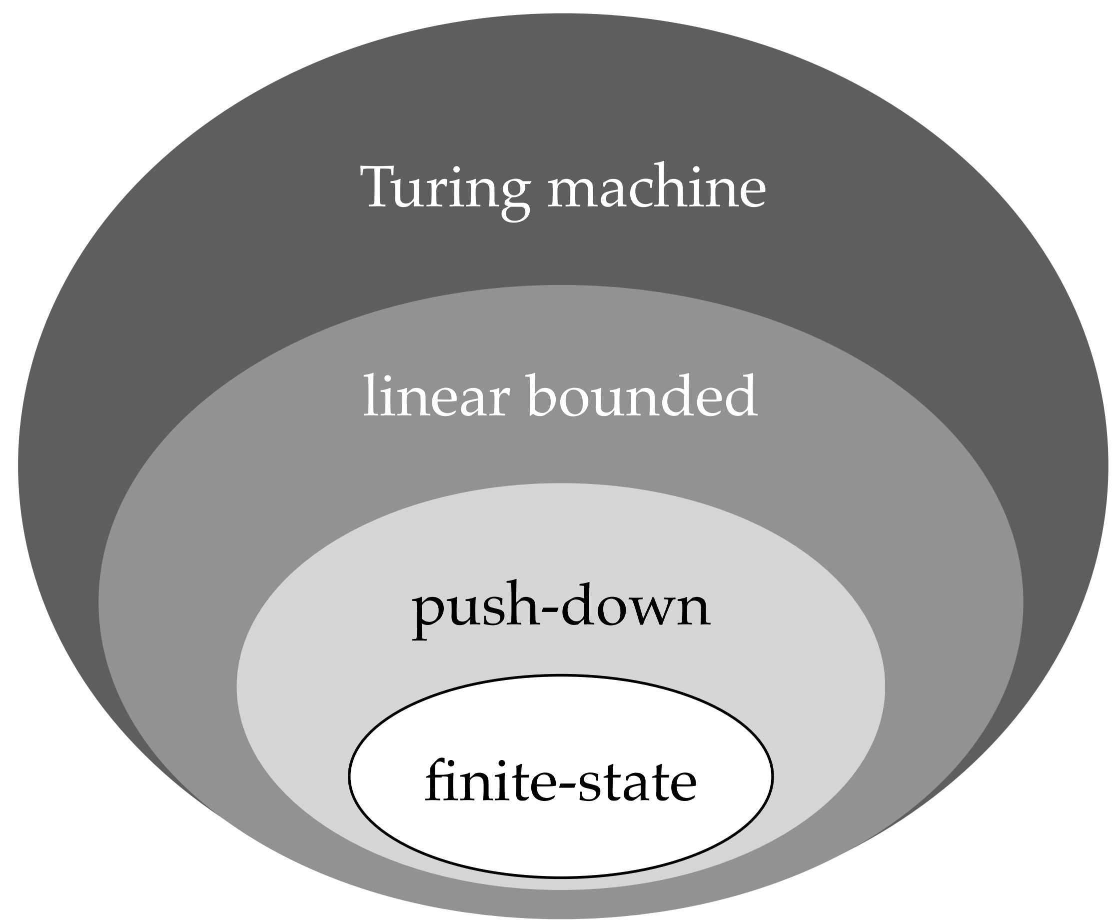


Recursively Enumerable

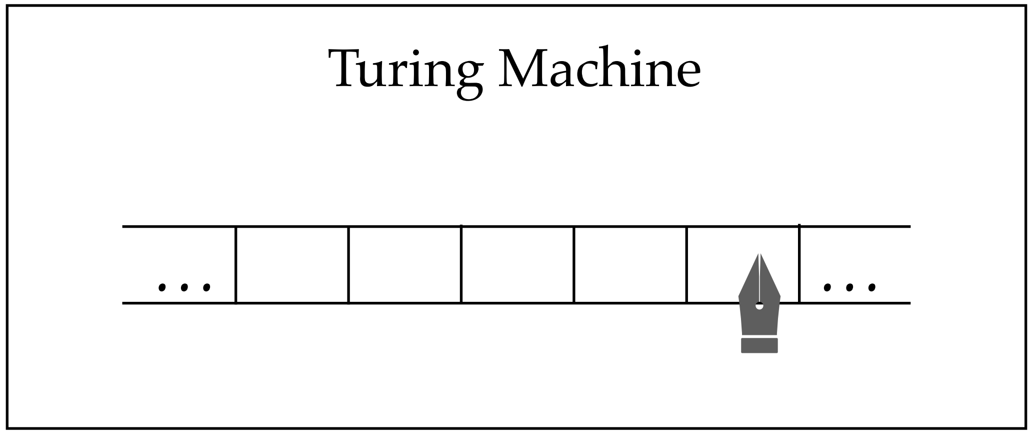


Turing (1936)



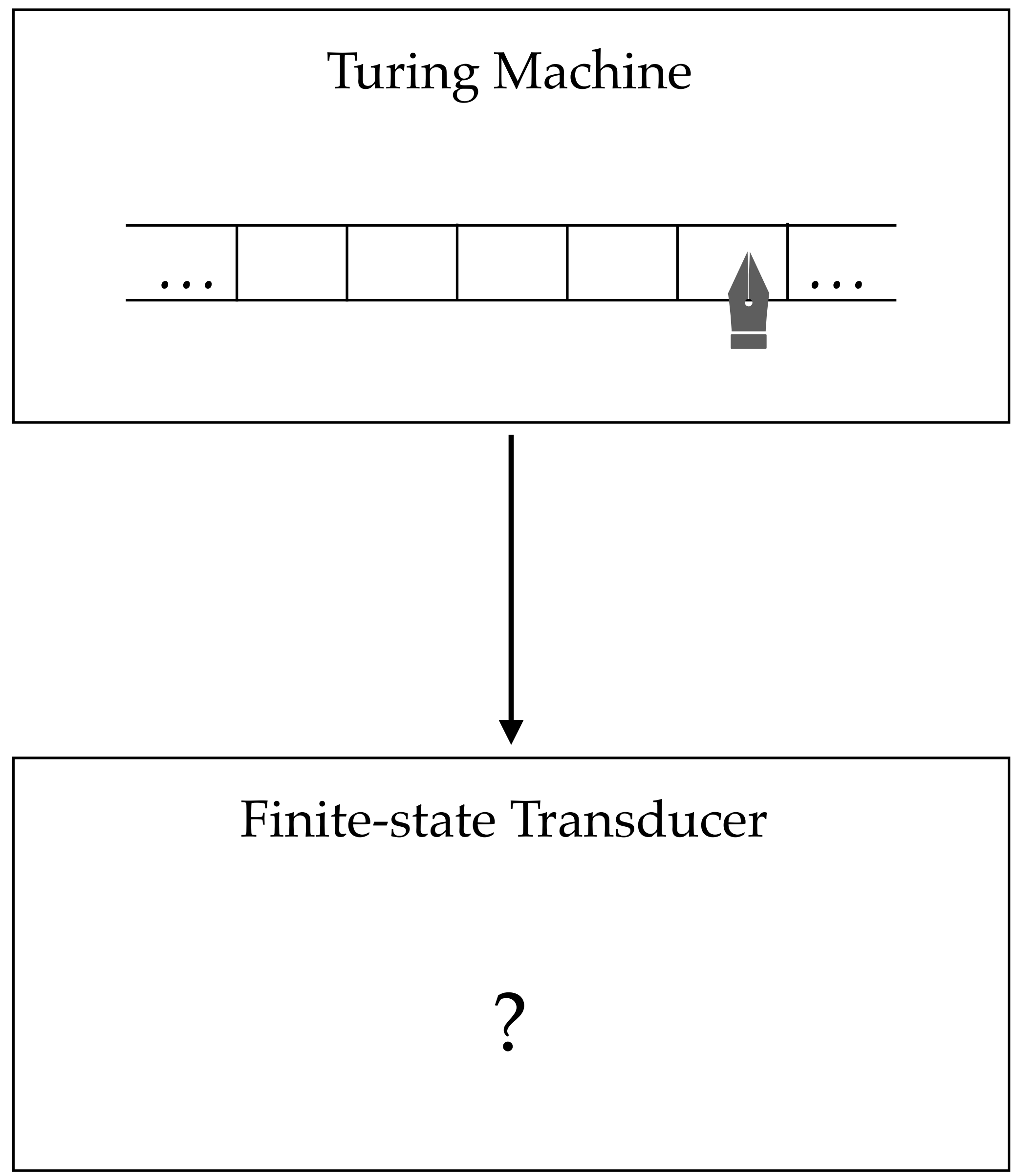


Physically…



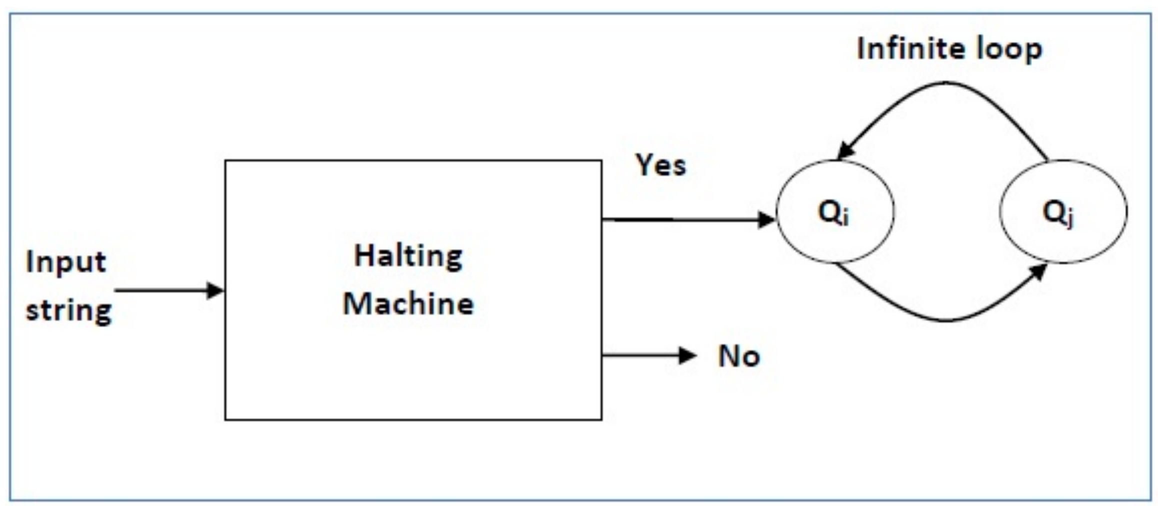
Demonstration

Tape

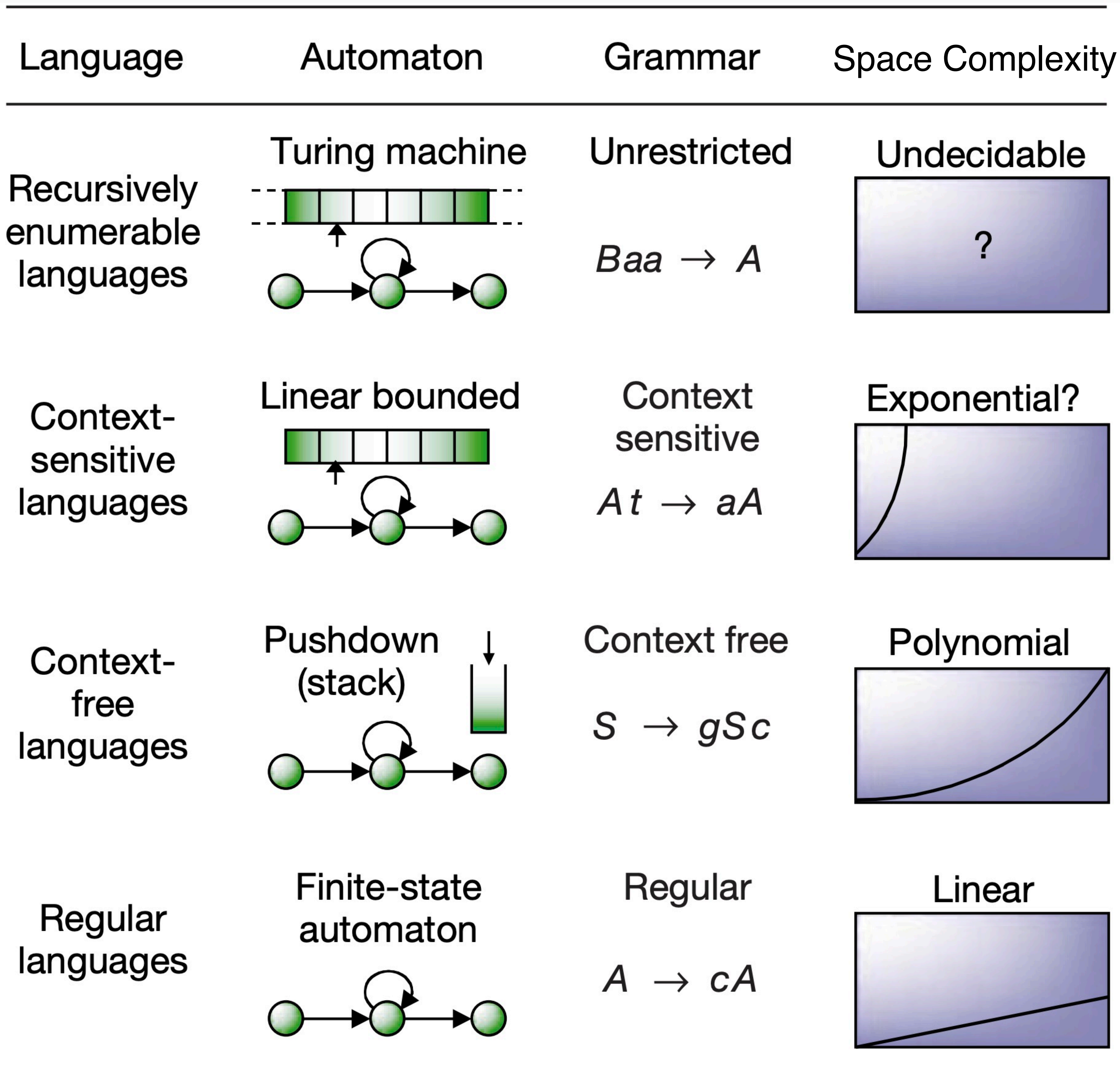


The Halting Problem

- Whether there exists an algorithm that can **always** determine, for any arbitrary computer program and its input, whether the program will eventually stop running (halt) or continue to run indefinitely.
- Here is a paradoxical program that halts if and only if it does not halt.



Why does it matter?



$O(2^n)$

$O(n^3)$

$O(n)$

from Searls (2012)

Space complexity

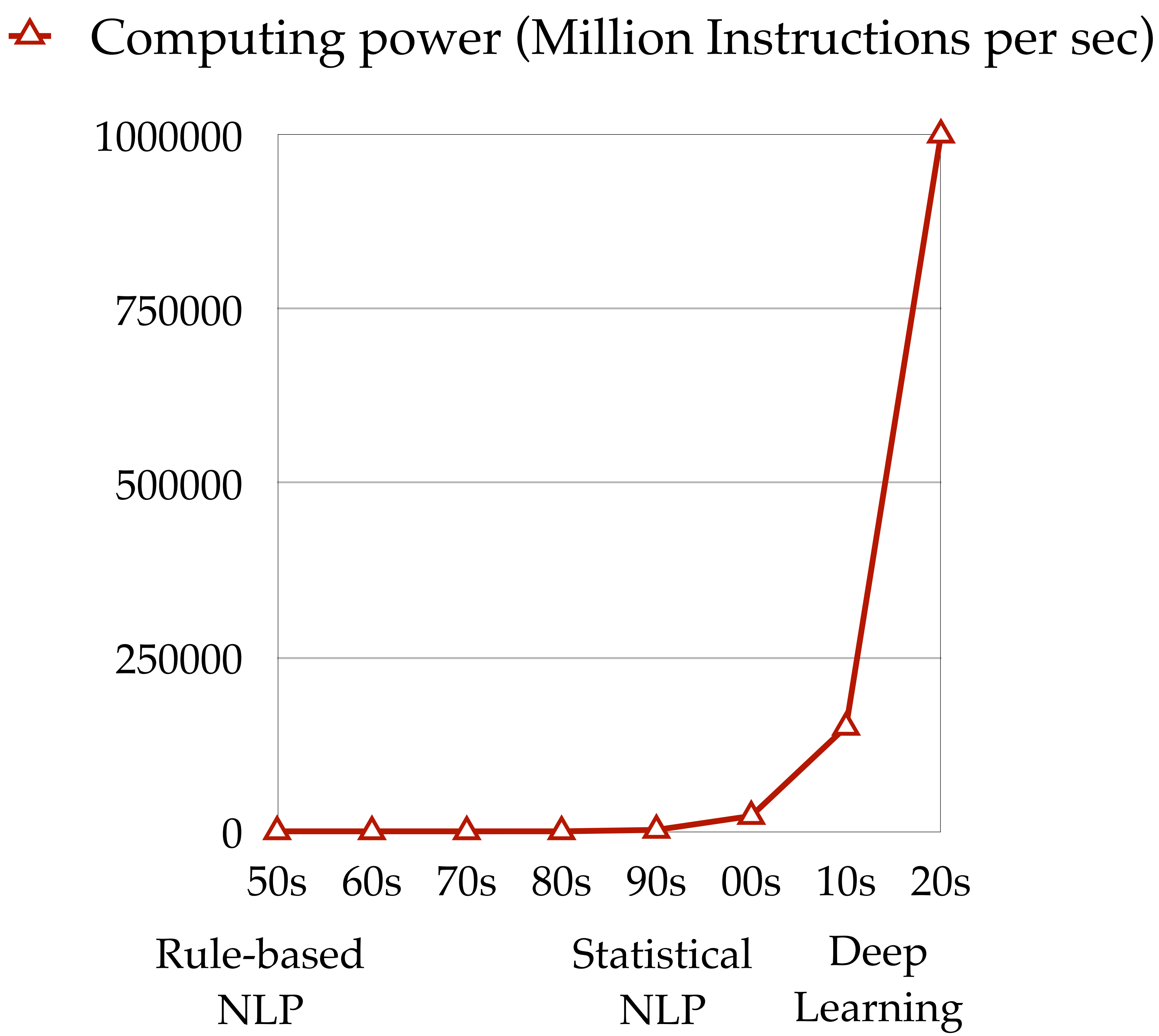
- Big O notation: *O(g(n))* reads "its complexity is bounded by *g(n)*"
- *g(n)* is a function that has its own growth rate, and the algorithm here doesn't grow faster than that ("bounded").

- Exponential complexity is usually considered infeasible for human cognition.
- Caveat: complexity is the **worst-case analysis**. e.g., Dyck language requires $O(n^3)$ but some other non-regular languages can still be computed in linear complexity. e.g., $a^n b^n, n \geq 1$ corresponds to $O(n)$.

Complexity-learnability correlation

- This correlation seems like an common assumption in previous FLT and AGL works.
- Known facts: *Some* more complex patterns are indeed harder to learn in AGL, such as first-last harmony vs. regular harmony in natural languages.
- What do you think? How should we measure learnability?

Does complexity still matter?



Hardware Comparison

	Supercomputer	Personal Computer	Human Brain
Computational units	10^6 GPUs + CPUs	8 CPU cores	10^6 columns
	10^{15} transistors	10^{10} transistors	10^{11} neurons
Storage units	10^{16} bytes RAM	10^{10} bytes RAM	10^{11} neurons
	10^{17} bytes disk	10^{12} bytes disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{18}	10^{10}	10^{17}

A crude comparison of a leading supercomputer, Summit; a typical personal computer of 2019; and the human brain.

Russel & Norvig (2022) *AI, A Modern Approach*; Fig 1.2

~355 years to train GPT-3 on a *single* NVIDIA Tesla V100 GPU, which is even more powerful than personal computer in terms of hardware.

Open question: is it feasible for children?

Nihilists' FAQ

"But is ___ psychologically real?"
"Do you really believe ___ exist in our brain?"
plug in any formal/computational models for cognition.
e.g., State machines, Rewrite rules, Optimality-Theoretic grammar, Bayesian networks, Neural networks, …

"What about ___?"
plug in anything that is not in your simplified data
e.g., noise, substance, gradience, variability, prosody, …

All are important questions that kept me awake at night.

Marr (1982) three levels of an information processing system

Computational: what are the problems and the goals? what are harder to compute?

Algorithmic/Representational: what representations the system uses and how it manipulates those representations?

Implementational: how can the system be realized physically?

- Most linguists are on the first two levels;
- In practice, we need many working hypothese for all three levels to work together;
- The final product is only an *approximation* of human cognition that we use to ask scientific questions and test against real-world data.

Marr (1982) three levels of an information processing system

Computational: what are the problems and the goals? what are harder to compute?

MathLing: logical definition of types of languages (regular, context-free, context-sensitive) and structure of formalisms (automata, rewrite rules, …).

Algorithmic/Representational: what representations the system uses and how it manipulates those representations?

The content of specific formal / computational models

Implementational: how can the system be realized physically?

Working and long-term memory

- Quick experiment: what are the words you read in last slide and what’s the last five sentence I just said and what’s the weather like today and who’s your favorite linguist and what’s the answer of 54321 * 12345…

Timeless insights of FLT

- Essentially, it touches on fundamental questions in any types of computing:
  1. what do you need to store in the memory during the computation?
  2. how much memory do you need?
  3. what is a good grammar for the data we have?
- Same questions for human cognition: natural language (Faculty of Language Narrow). and other computation, such as problem-solving.

What are good grammars?

Two extremes:

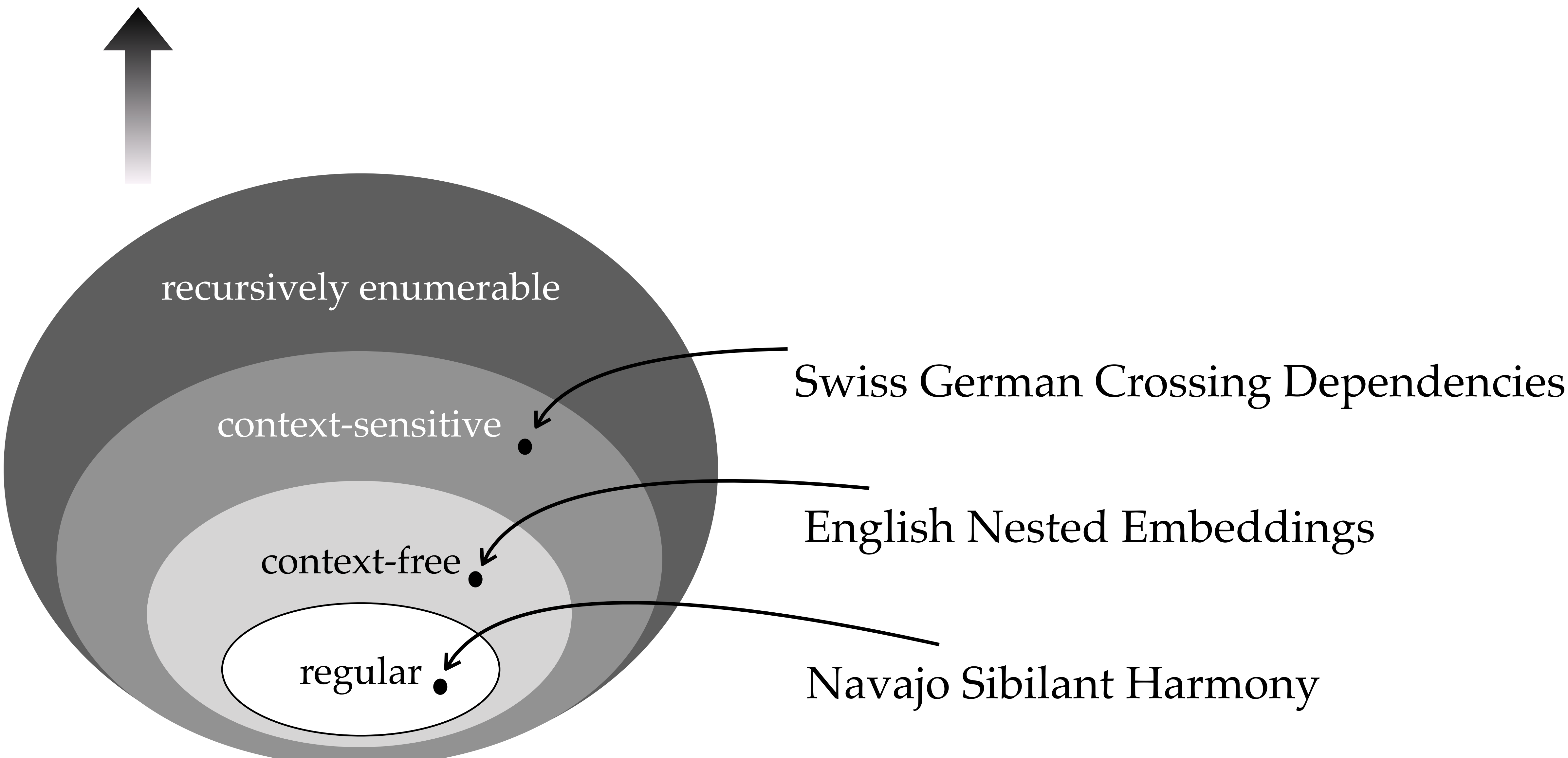
- Store nothing:
  $G = \{\,\}$
- Everything:
  $G = \{s \in S\}$, *S includes all encoutered strings*

"An interesting grammar is one that sits in between these two extremes, yielding constrained productivity."
—Hunter (2020) "The Chomsky Hierarchy"

Natural Languages

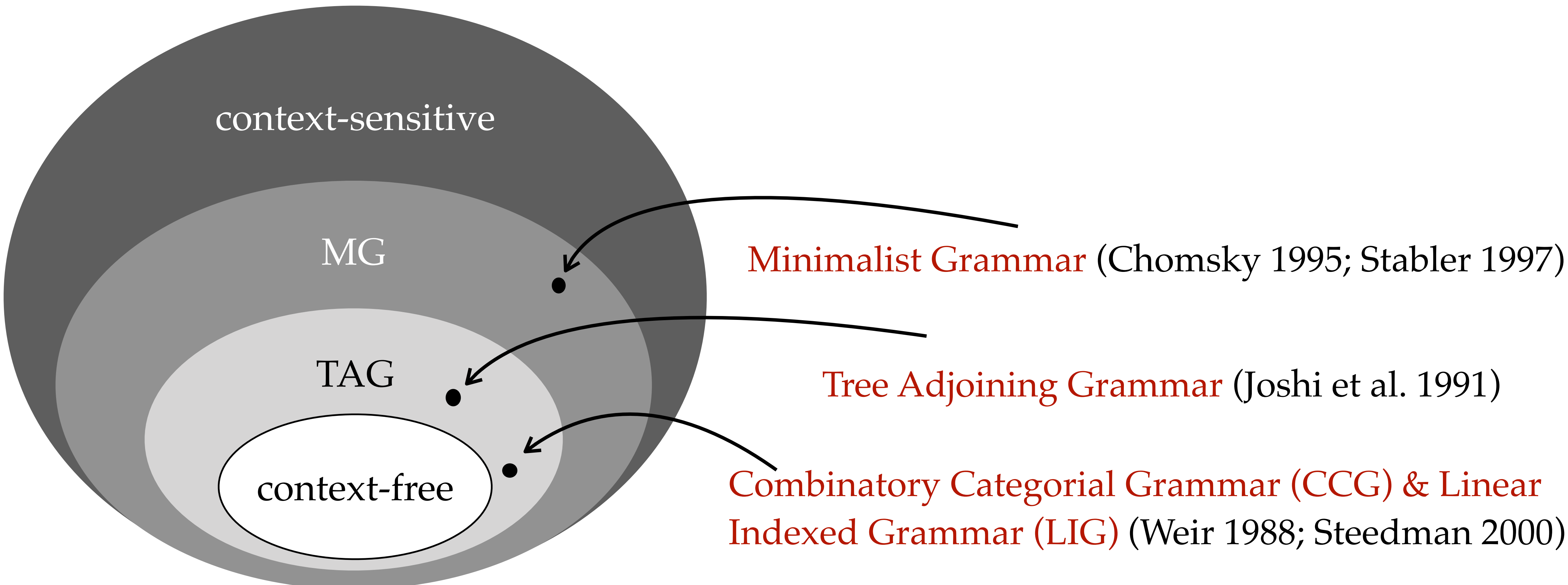
Where are natural languages located?

more working memory
more complex
more logical power
less restricted



Mildly Context-Sensitive

- 1980s: Find the classes that superset context-free, but have polynomial complexity
- Tim Hunter's tutorials
  https://timhunter.humspace.ucla.edu/lsa2023



(Sub-)regular Syntax

- Finite-State Tree Automata
- Graf (2023) SCiL paper: “Subregular Tree Transductions, Movement, Copies, Traces, and the Ban on Improper Movement the Ban on Improper Movement”

Subregular Hierarchy

Reference Text

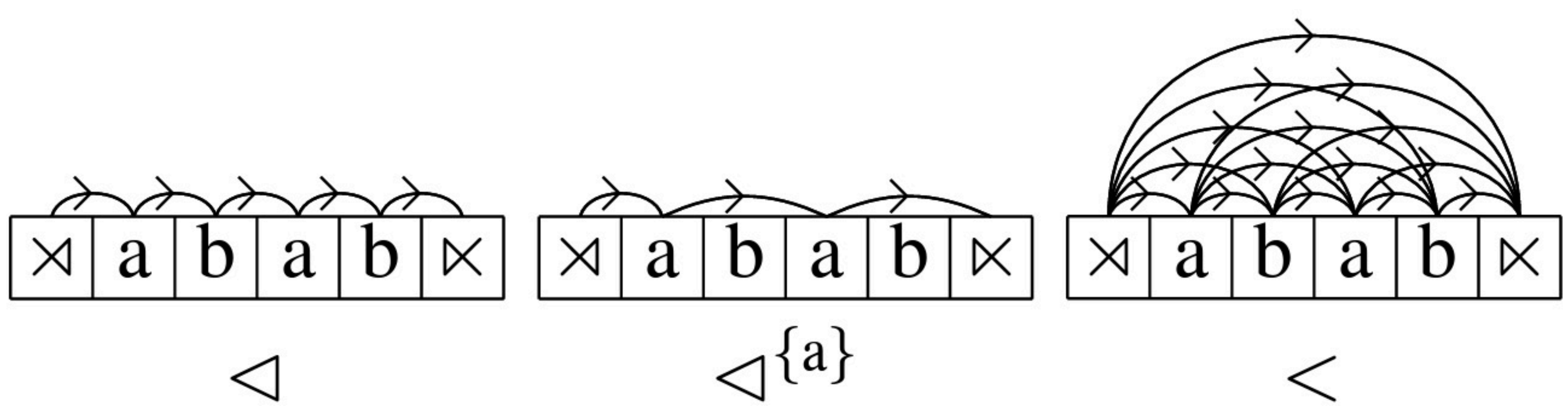
There are some discrepancies between the definitions in Jäger & Rogers (2012) and Heinz (2018).
I follow the later for consistency. It is also more intuitive for phonologists.

k-factors

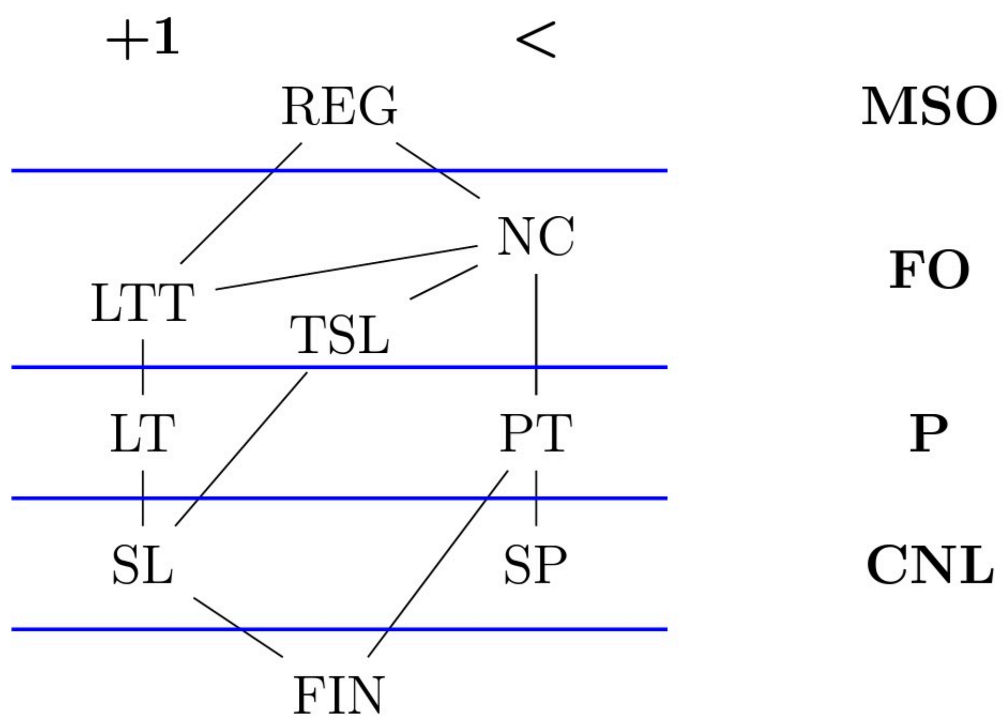
Assume the alphabet = {a, b}

Subregular languages always keep track of some window of width k in a string:

1. Successor relation (+1 or $\lhd$), e.g., ab, ba, …
2. Predecessor relation ($<$), e.g., a…b, b…a, …



Heinz (2018)



Names of the classes of stringsets			
REG	Regular	FIN	Finite
LTT	Locally Threshold Testable	NC	Non-Counting
LT	Locally Testable	PT	Piecewise Testable
SL	Strictly Local	SP	Strictly Piecewise
TSL	Tier-based Strictly Local		

Representational Primitives (order)		Logical Power	
+1	Successor	MSO	Monadic Second Order
$<$	Precedence	FO	First Order
		P	Propositional
		CNL	Conjunction of Negative Literals

Figure 2: Subregular hierarchies of stringsets.

Strictly *k*-Local

Conjunction of Negative Literals: $\neg$, $\wedge$

e.g. No aa and no bb

$G = \neg aa \wedge \neg bb$

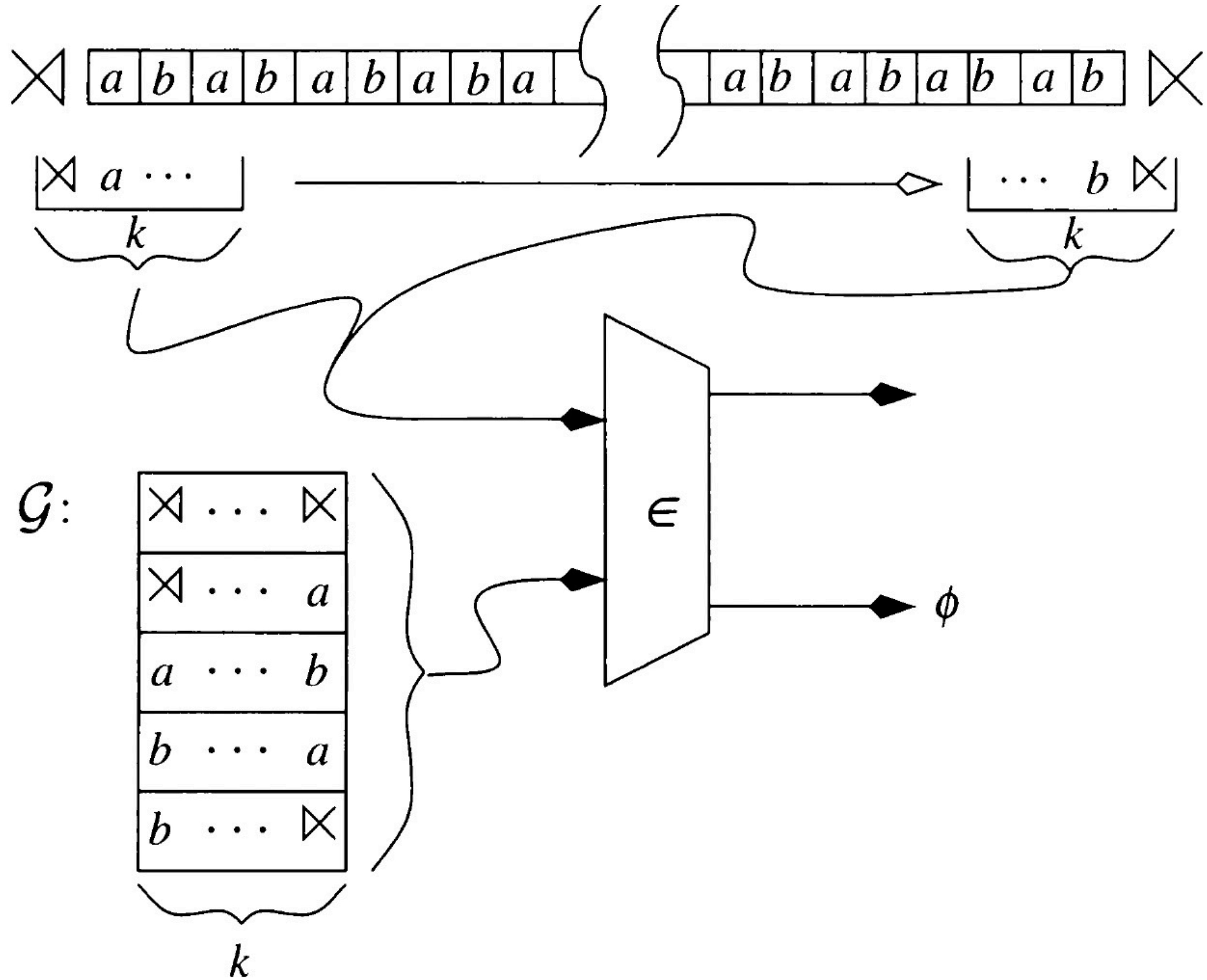


Figure 7. Scanners have a sliding window of width k, a parameter, which moves across the string stepping one symbol at a time, picking out the k-factors of the string. For SL languages the string is accepted if and only if each of these k-factors is included in a look-up table.

Suffix Substitution Closure

A stringset L is SL if there is a k such that for all strings u_1, v_1, u_2, v_2, (and a substring) x with the length equal to $k{-}1$, it is the case that if u_1xv_1 and u_2xv_2 belong to L then u_1xv_2 belongs to L as well.

Rogers & Pullum (2011)

"Content before previous *k-1* symbols won't affect the well-formedness of the string."

say we keep track of $x = a$

$$u_1 = ab,\ \ v_1 = ba,\ u_2 = abab,\ v_2 = baba$$

$$u_1xv_1 = ababa,\ \ u_2xv_2 = ababababa,\ u_1xv_2 = abababa$$

Locally *k*-Testable

Proposition Logic: ¬, ∧, ∨

we got implication →, biconditional ↔ for free

$$A \to B \Leftrightarrow \neg A \vee B$$

e.g. No ab and Some b words

$$G = \neg ab \wedge (ba \vee bb \vee b\ltimes \vee \rtimes b)$$

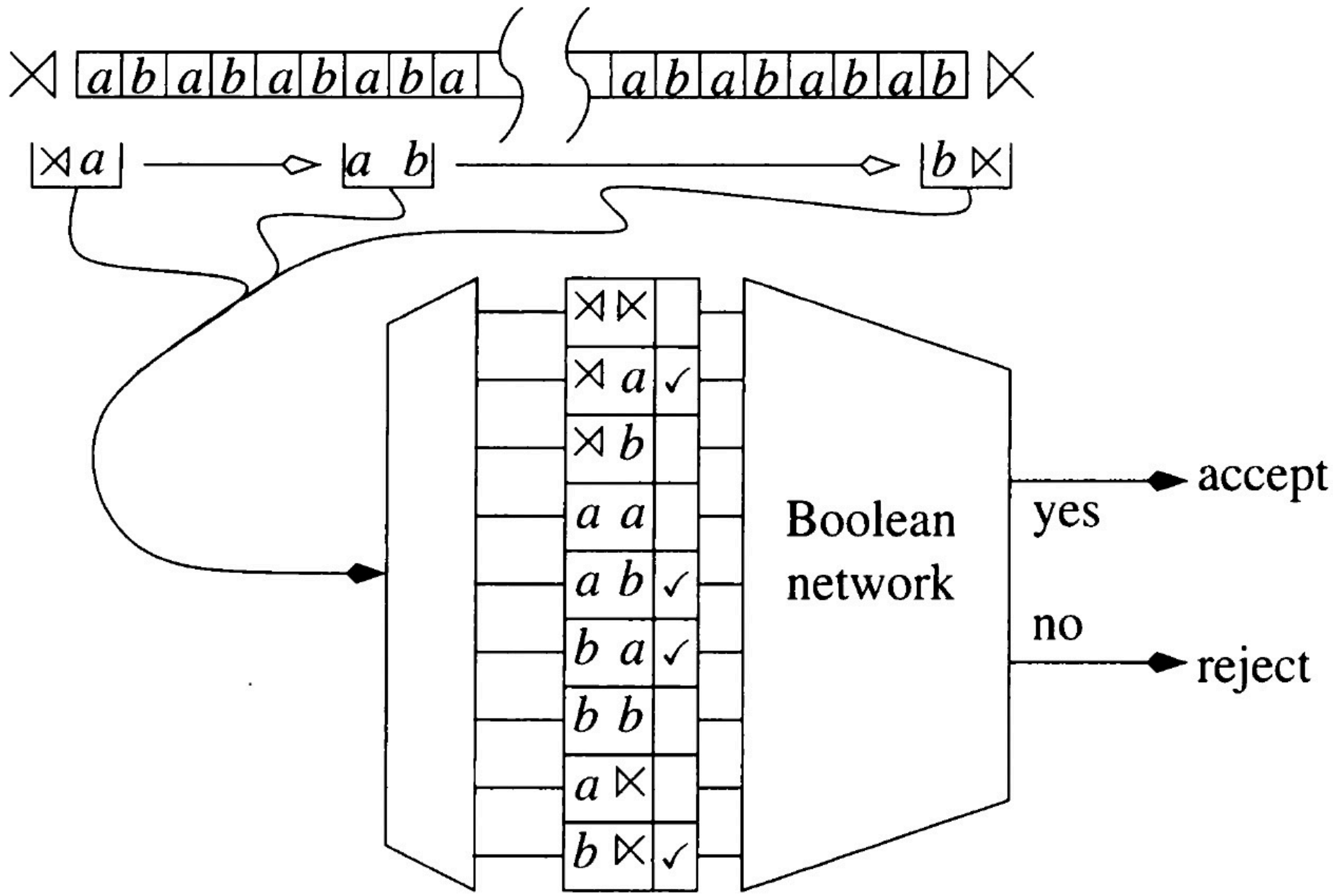


Figure 8. LT automata keep a record of which *k*-factors occur in a string and feed this information into a Boolean network. The automaton accepts if, once the entire string has been scanned, the output of the network is ‘yes’, and rejects otherwise.

Substring Equivalence

A stringset L is LT if there is a k such that for all strings u and v, if u and v have the same set of substrings of length k then either both u and v belong to L or both u and v do not belong to L.

e.g. $G = \neg ab \wedge (ba \vee bb \vee b\ltimes \vee \rtimes b)$
$L = \{ba, bba, bbba, \ldots\}$

say we keep track of $k = 2$

$u = bba, \; v = bbba$ *both belong*

Locally Threshold Testable

First-Order Logic: $\neg, \wedge, \vee, \rightarrow, \forall x, \exists x$

e.g. One b words

$$G_{\text{one-b}} = (\exists x)[b(x) \wedge (\neg \exists y)[b(y) \wedge \neg x \approx y]]$$

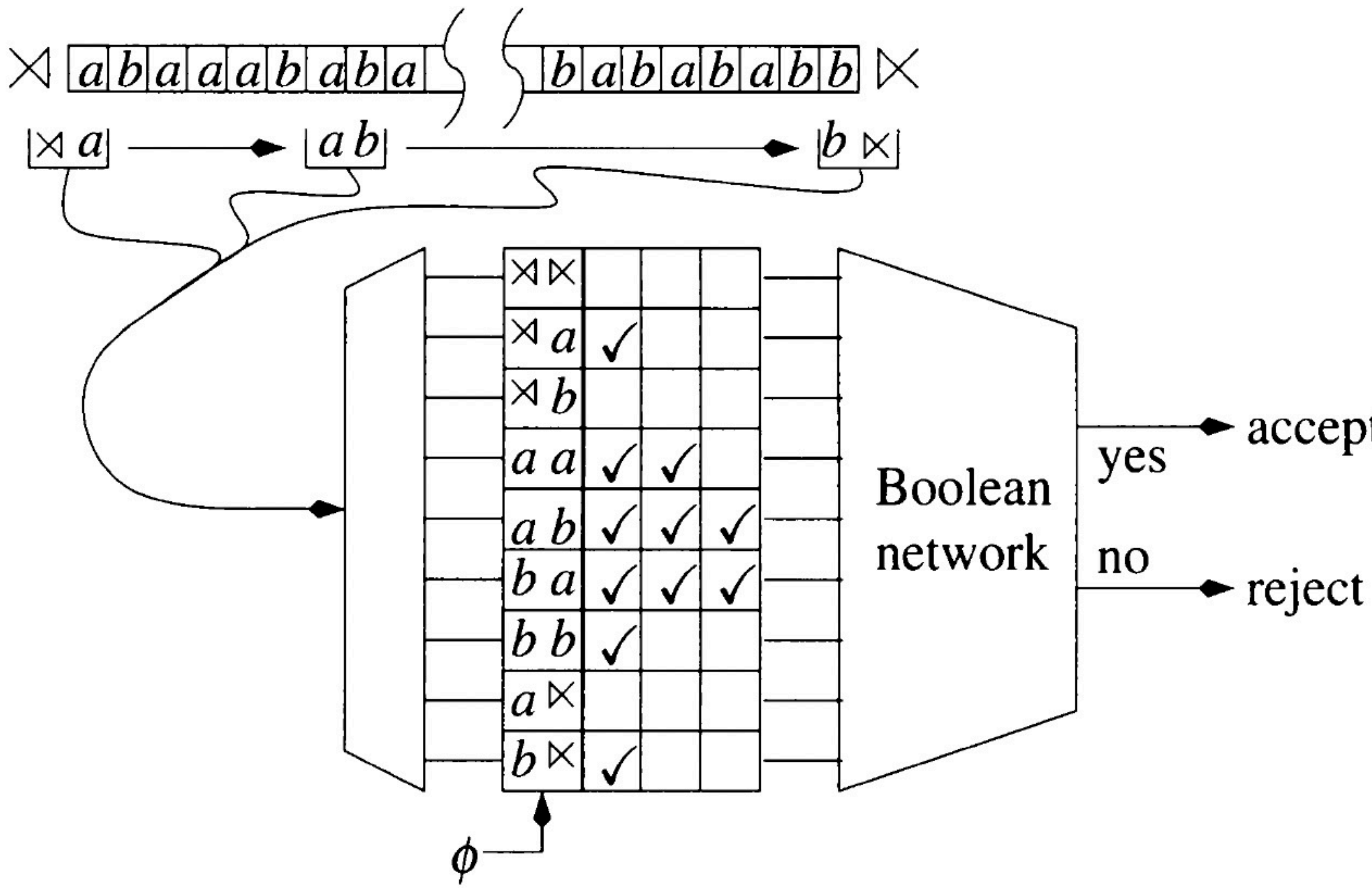


Figure 9. LTT automata count the number of k-factors that occur in a string up to some bound and feed this information into a Boolean network. The automaton accepts if, once the entire string has been scanned, the output of the network is 'Yes', and rejects otherwise.

Substring Threshold Equivalence

A stringset L is LTT if there is a k and a t such that for all strings u and v, if u and v have **the same number, up to some threshold *t*, of substrings** of length k then either both u and v belong to L or both u and v do not belong to L.

e.g. $G_{one\text{-}b} = (\exists x)[b(x) \wedge (\neg \exists y)[b(y) \wedge \neg x \approx y]]$

say we keep track of $k = 2$, *and* $t = 2$

{bb, ba} for $u = bba$, {ba, ab} for $v = bab$, neither belong to L, meaning L might be LTT

{ab, ba} for $u = aba$, {ba, ab} for $v = bab$, u belong but v doesn't, meaning L is not LTT

Monadic Second-Order Logic

First-Order Logic: $\neg, \wedge, \vee, \rightarrow, \forall x, \ \exists x$

Monadic Second-Order Logic: $\neg, \wedge, \vee, \rightarrow, \forall x, \ \exists x, \forall X, \ \exists X$

First-Order:

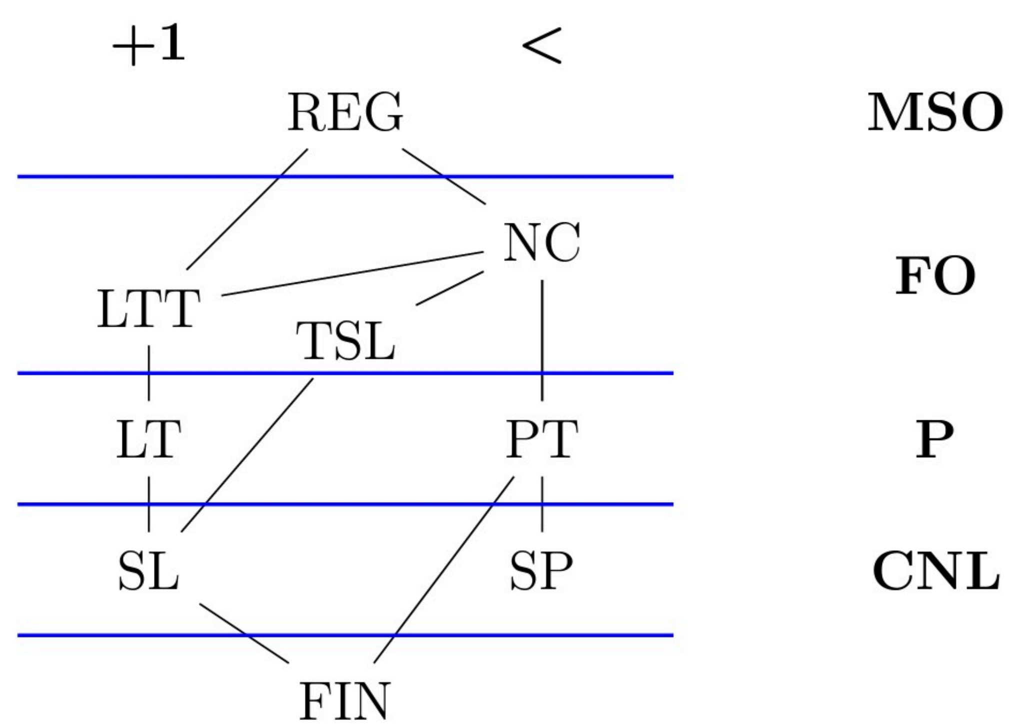
$$G_{\text{one-b}} = (\exists x)[b(x) \wedge (\neg \exists y)[b(y) \wedge \neg x \approx y]]$$

Monadic Second-Order Logic:

$$\exists X(\forall x(X(x) \leftrightarrow b(x)) \wedge \exists x(X(x)) \wedge \forall y \forall z((X(y) \wedge X(z)) \rightarrow y = z))$$

"Create X for b symbol {b}, and if a symbol in the string is already in X, others cannot be"

Heinz (2018)



Names of the classes of stringsets			
REG	Regular	FIN	Finite
LTT	Locally Threshold Testable	NC	Non-Counting
LT	Locally Testable	PT	Piecewise Testable
SL	Strictly Local	SP	Strictly Piecewise
TSL	Tier-based Strictly Local		

Representational Primitives (order)		Logical Power	
+1	Successor	MSO	Monadic Second Order
<	Precedence	FO	First Order
		P	Propositional
		CNL	Conjunction of Negative Literals

Figure 2: Subregular hierarchies of stringsets.

FLT and Artificial Grammar Learning

AGL setting

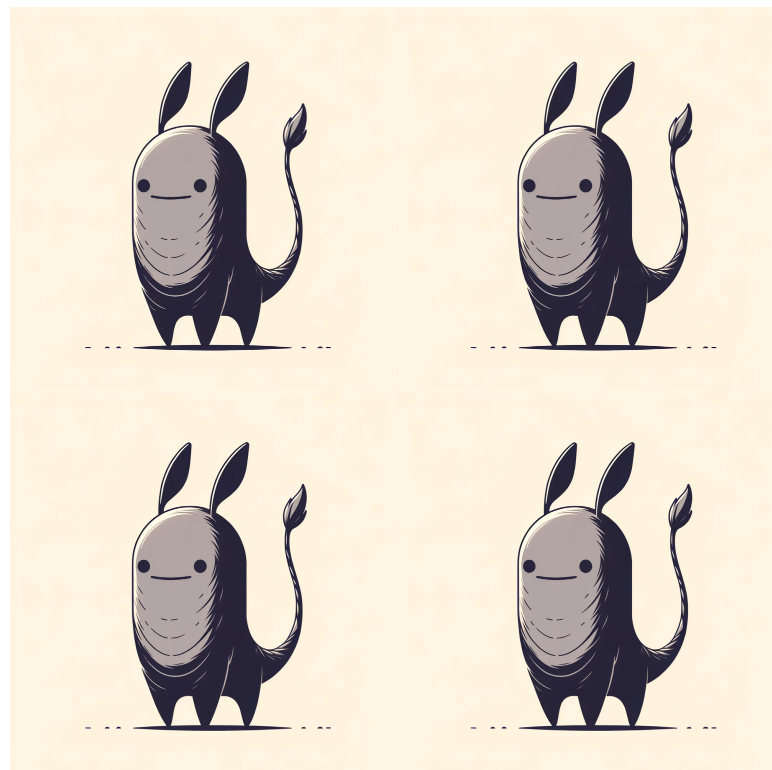
- Intended set (I): Target language
- Familiarization set (F): Training data, subset of I
- Discrimination set (D): some in I, some not.

- Jäger & Rogers had a graph to show the subset problem
- Stimuli seem a bit unreliable for more complex classes.

Familiarization set 1

gogi

gogimi



Familiarization set 2

mipu

mipumu

Discrimination set 1

gipumu gipumi

Discrimination set 2

dopimi dopimu

Doing AGL

- Intended set (I): Tier-based Strictly 2-Local vowel harmony pattern {*[+round][-round], *[-round][+round]}
- *vs*. First-Last harmony in testing phase

First-last		**TSL**	
Stem	**Stem-plural**	**Stem**	**Stem-plural**
mipu	mipumu	**mipu**	mipumu
gogi	gogimi	**gogi**	gogimi
...	...	...	...

- I wrote a Python code for creating the audio stimuli based on the given wordlist, using Google Text-to-Speech library.
- The visual stimuli are created using OpenAI DALL-E 3, "create a picture of a cute creature called gogi"

Homework

Describe Japanese syllable structure in **the most restrictive logic and automata** you learned. Most syllables in Japanese conform to (C)V(N), where C is a consonant, V is a vowel, and N is a nasal consonant that can appear at the end of syllables.

Japanese	**English**	**Syllable**
sakura さくら	Cherry blossom	CV.CV.CV
tomodachi ともだち	Friend	CV.CV.CV.CV
shinbun しんぶん	Newspaper	CVN.CVN
nihongo にほんご	Japanese Language	CV.CVN.CV
tsukue つくえ	Desk	CV.CV.V

Optional: write a code to recognize Japanese syllable structure!

Vote for topics

- Transducers and morphophonology
- Model-theoretic phonology (Heinz 2018)
- Computational syntax (Hunter 2023)
- Myhill-Nerode and *intersubstitutability* (Hunter 2023)
- Connection to typology and learnability
- Weighted / probabilistic finite-state automata
- Connection to machine learning: Jeff Heinz and colleagues's MLRegTest; Svete & Cotterel 2023