And now for something completely different

Algorithms for NLP (11-711)
Fall 2018

Formal Language Theory

In one lecture

Robert Frederking

Now for Something Completely Different

- We will look at languages and grammars from a “mathematical” point of view
- But Discrete Math (logic)
  - No real numbers
  - Symbolic discrete structures, proofs
- Interested in complexity/power of different formal models of computation
  - Related to asymptotic complexity theory
- This is the source of many common CS algorithms/models

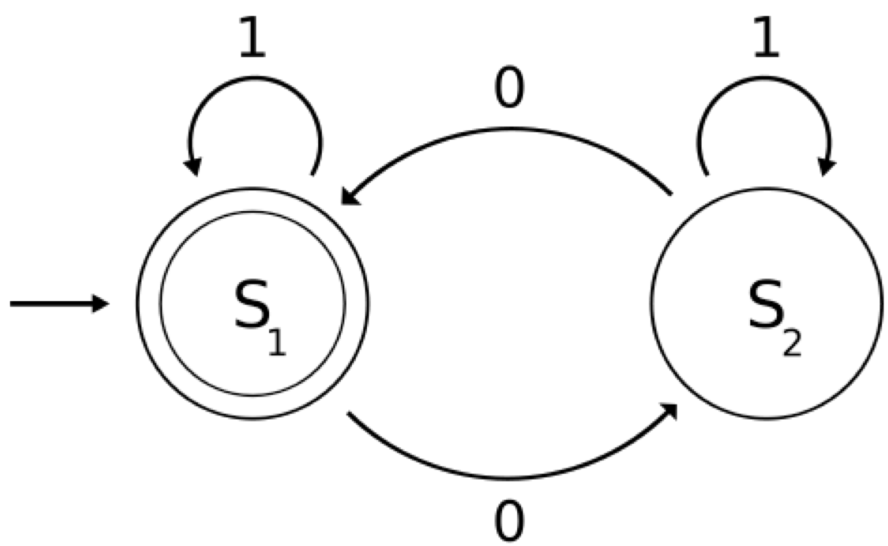
Two main classes of models

- Automata
  - Machines, like Finite-State Automata
- Grammars
  - Rule sets, like we have been using to parse
- We will look at each class of model, going from simpler to more complex/powerful
- We can formally prove complexity-class relations between these formal models

Simplest level: FSA/Regular sets

Finite-State Automata (FSAs)

- Simplest formal automata
- We've seen these with numbers on them as HMMs, etc.



(from Wikipedia)

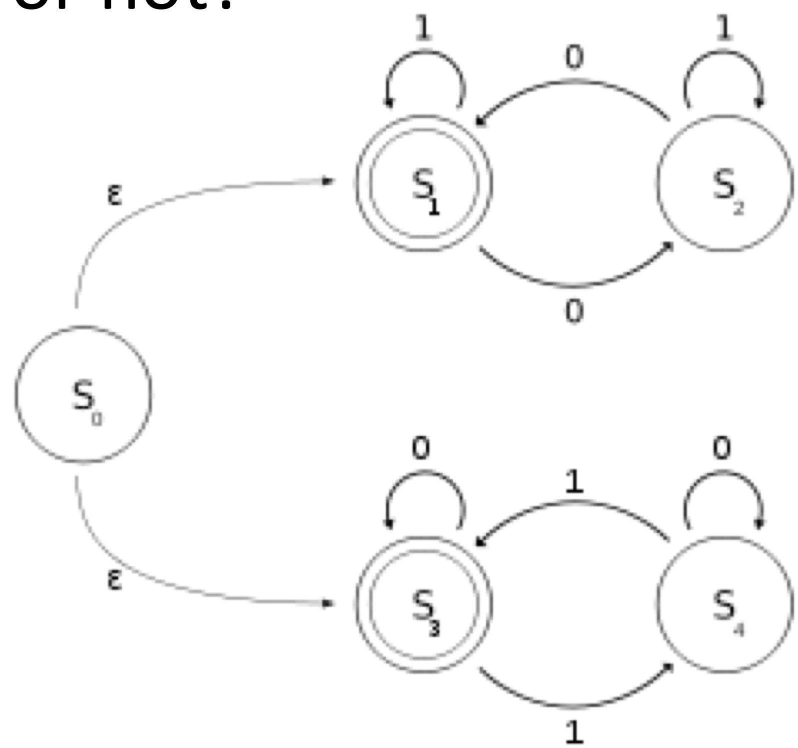
Formal definition of automata

- A finite set of states, Q
- A finite alphabet of input symbols, Σ
- An initial (start) state, $Q_0 \in Q$
- A set of final states, $F_i \in Q$
- A transition function, $\delta: Q \times \Sigma \rightarrow Q$

- This rigorously defines the FSAs we usually just draw as circles and arrows

DFSAs, NDFSAs

- Deterministic or Non-deterministic
  - Is δ function ambiguous or not?



  - For FSAs, weakly equivalent

Intersecting, etc., FSAs

- We can investigate what happens after performing different operations on FSAs:
  - Union: $L = L1 \cup L2$
  - Intersection
  - Negation
  - Concatenation
  - other operations: determinizing or minimizing FSAs

Regular Expressions

- For these “regular languages”, there’s a simpler way to write expressions: regular expressions:

  Terminal symbols

  (r + s)

  (r • s)

  r*

  ε

- For example: (aa+bbb)*

Regular Grammars

- Left-linear or right-linear grammars
- Left-linear template:

  $A \rightarrow Bw$ or $A \rightarrow w$

- Right-linear template:

  $A \rightarrow wB$ or $A \rightarrow w$

  (where *w* is a sequence of terminals)

- Example:

  $S \rightarrow aA \mid bB \mid \varepsilon$, $A \rightarrow aS$, $B \rightarrow bbS$

Formal Definition of a Grammar

- Vocabulary of terminal symbols, Σ   (e.g., *a*)
- Set of nonterminal symbols, N   (e.g., *A*)
- Special start symbol, S ∈ N
- Production rules, such as A → aB
  - Restrictions on the rules determine what kind of grammar you have

- A formal grammar G defines a **formal language**, L(G), the set of strings it generates

Amazing fact #1: FSAs are equivalent to RGs

- Proof: two constructive proofs:
  - 1: given an arbitrary FSA, construct the corresponding Regular Grammar
  - 2: given an arbitrary Regular Grammar, construct the corresponding FSA

Construct an FSA from a Regular Grammar

- Create a state for each nonterminal in grammar
- For each rule "A → wB" construct a sequence of states accepting *w* from A to B
- For each rule "A → w" construct a sequence of states accepting *w,* from A to a final state

- This shows right linear case; use L^R for left linear

Construct a Regular Grammar from a FSA

- Generate rules from edges
- For each edge from *Qi* to *Qj* accepting *a*:

  $Qi \rightarrow a\ Qj$

- For each ε transition from *Qi* to *Qj*:

  $Qi \rightarrow Qj$

- For each final state *Qf*:

  $Qf \rightarrow \varepsilon$

Proving a language is *not* regular

- So, what kinds of languages are *not* regular?

- Informally, a FSA can only *remember* a finite number of *specific* things. So a language requiring an unbounded memory won't be regular.

Proving a language is *not* regular

- So, what kinds of languages are *not* regular?

- Informally, a FSA can only *remember* a finite number of *specific* things. So a language requiring an unbounded memory won't be regular.

- How about $a^n b^n$? "equal count of *a*'s and *b*'s"

Pumping Lemma: argument:

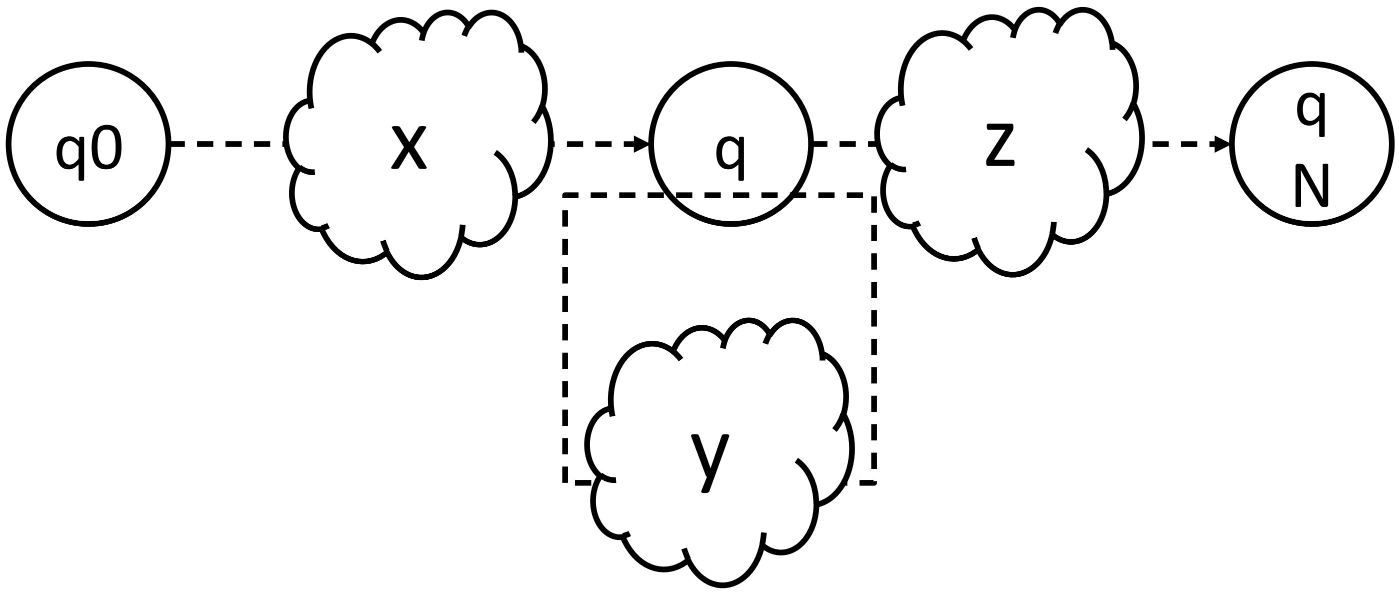
- Consider a machine with N states
- Now consider an input of length N; since we started in Q_0, we will now be in the (N+1)st state visited
- There *must* be a loop: we had to visit at least 1 state twice; let x be the string up to the loop, y the part in the loop, and z after the loop
- So it must be okay to also have M copies of y for any M (including 0 copies)

Pumping Lemma: formally:

- If L is an infinite regular language,
  then there are strings x, y, and z
  such that $y \neq \varepsilon$ and $xy^nz \in L$, for all $n \geq 0$.

- xyz being in the language requires also:
- xz, xyyz, xyyyz, xyyyyz, ..., xyyyyyyyyyyz, ...

Pumping Lemma: figure:



Example proof that a L is not regular

- What about $a^n b^n$?

  ab

  aabb

  aaabbb

  aaaabbbb

  aaaaabbbbb

  …

- Where do you draw the $xy^n z$ lines?

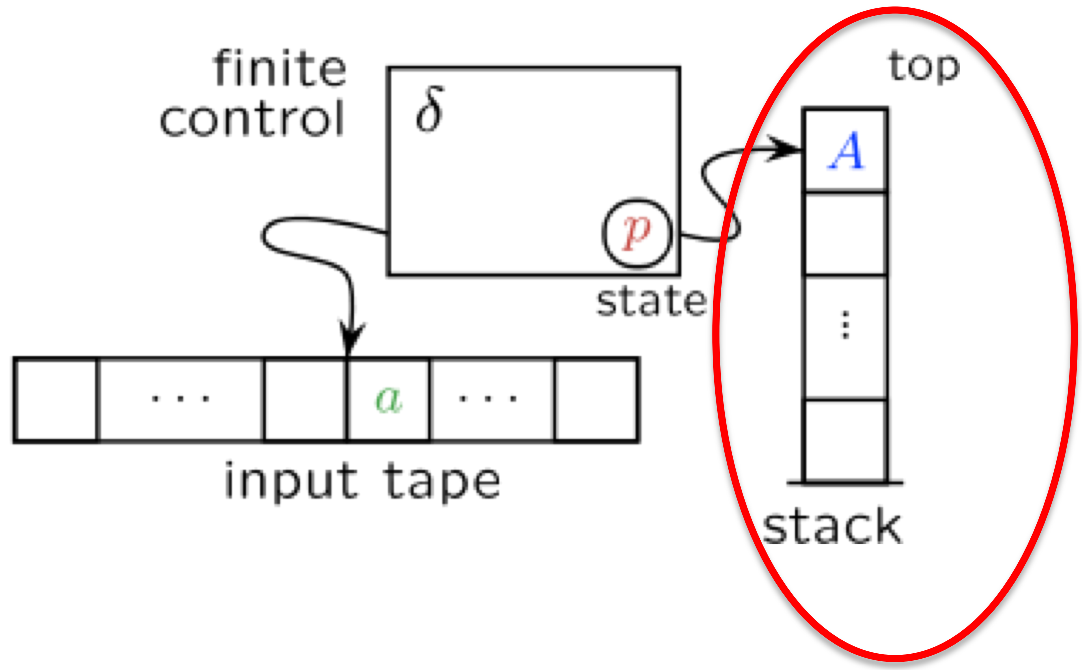
Example proof that a L is not regular

- What about $a^n b^n$? Where do you draw the lines?
- Three cases:
  - y is only a's: then $xy^n z$ will have too many a's
  - y is only b's: then $xy^n z$ will have too many b's
  - y is a mix: then there will be interspersed a's and b's
- So $a^n b^n$ cannot be regular, since it cannot be pumped

Next level:
PDA/CFG

Push-Down Automata (PDAs)

- Let’s add some unbounded memory, but in a limited fashion
- So, add a stack:



- Allows you to handle some non-regular languages, but not everything

Formal definition of PDA

- A finite set of states, Q
- A finite alphabet of input symbols, Σ
- A finite alphabet of stack symbols, Γ
- An initial (start) state, $Q_0 \in Q$
- An initial (start) stack symbol $Z_0 \in \Gamma$
- A set of final states, $F_i \in Q$
- A transition function, $\delta: Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$

Context-Free Grammars

- Rule template:

  $A \rightarrow \gamma$

  where γ is any sequence of terminals/non-terminals
- Example: $S \rightarrow a\ S\ b \mid \varepsilon$
- We use these a lot in NLP
  - Expressive enough, not too complex to parse.
    - We often add hacks to allow non-CF information flow.
  - It just really feels like the right level of analysis.
    - (More on this later.)

Amazing Fact #2: PDAs and CFGs are equivalent

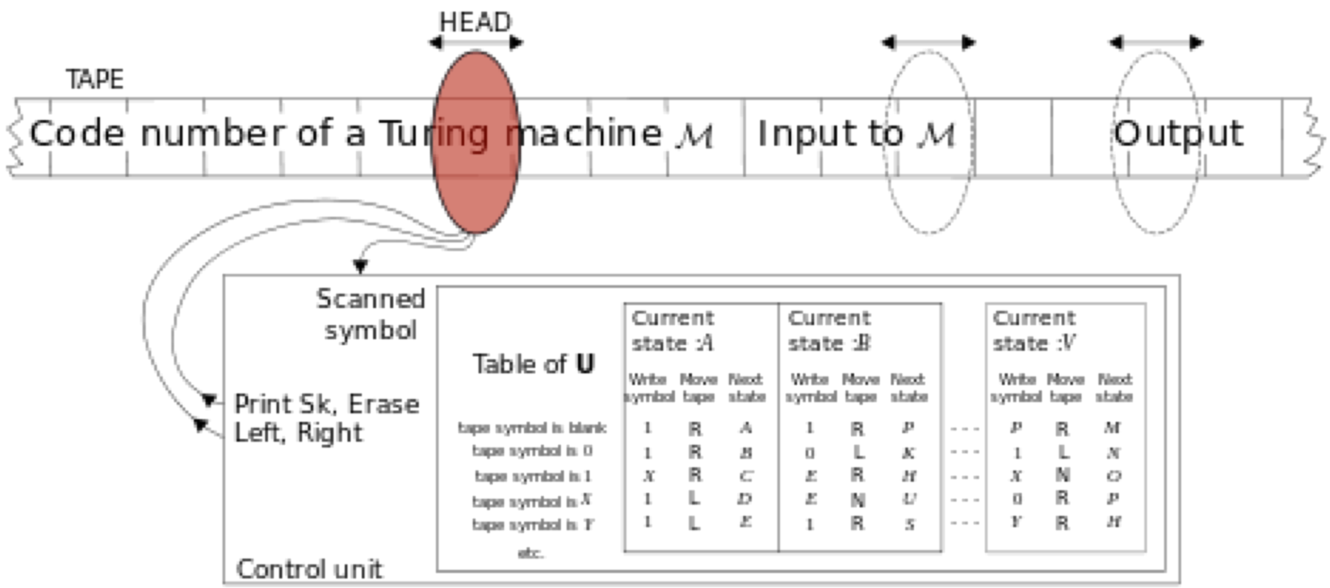
- Same kind of proof as for FSAs and RGs, but more complicated

- Are there non-CF languages? How about $a^n b^n c^n$?

Highest level:
TMs/Unrestricted grammars

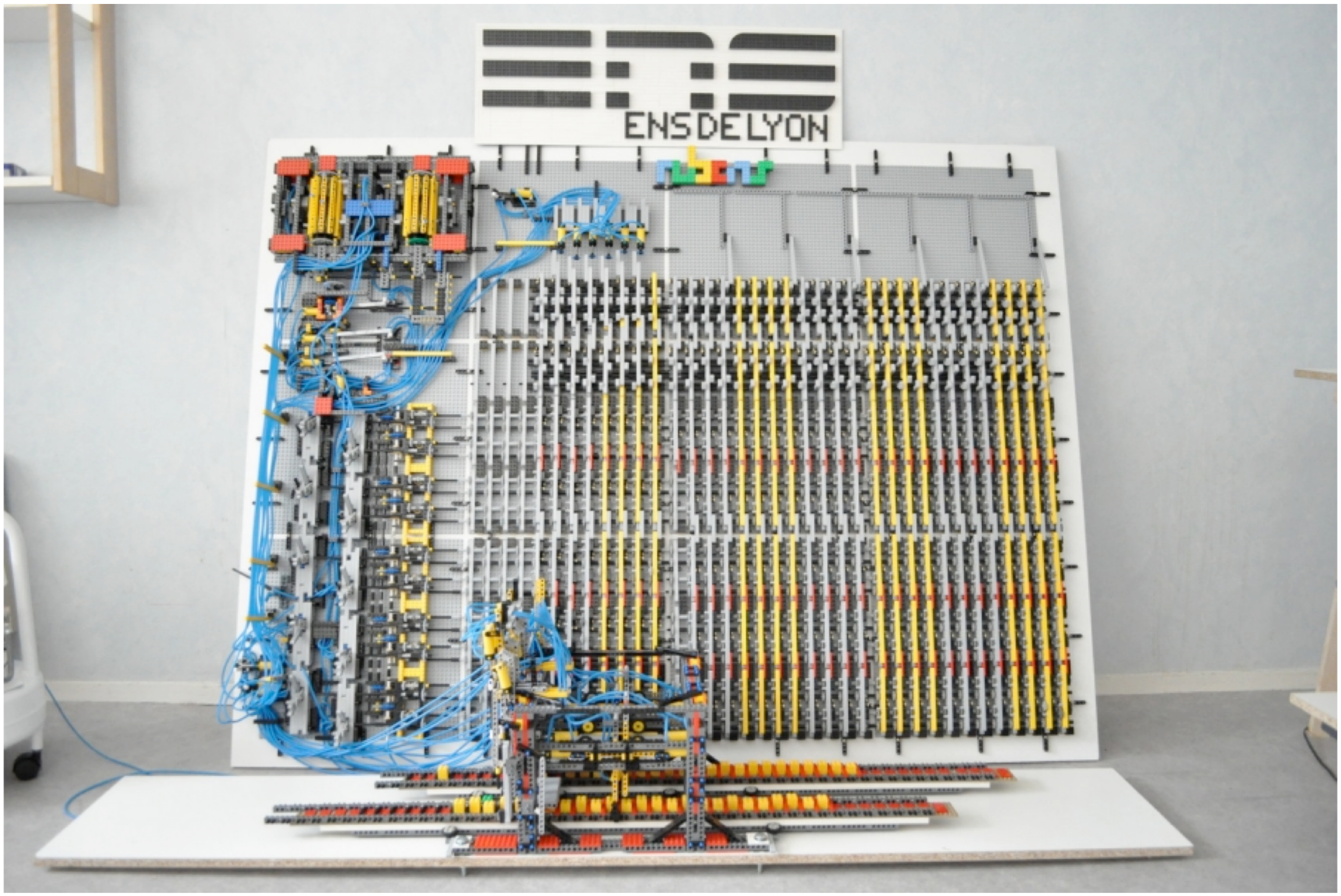
Turing Machines

- Just let the machine move and write on the tape:



- This simple change produces general-purpose computer

TM made of LEGOs



Unrestricted Grammars

- $\alpha \rightarrow \beta$, where each can be any sequence (α not empty)

- Thus, there is *context* in the rules:

  aAb → aab

  bAb → bbb

- No surprise at this point: equivalent to TMs
  – Church-Turing Hypothesis

Even more amazing facts: Chomsky hierarchy

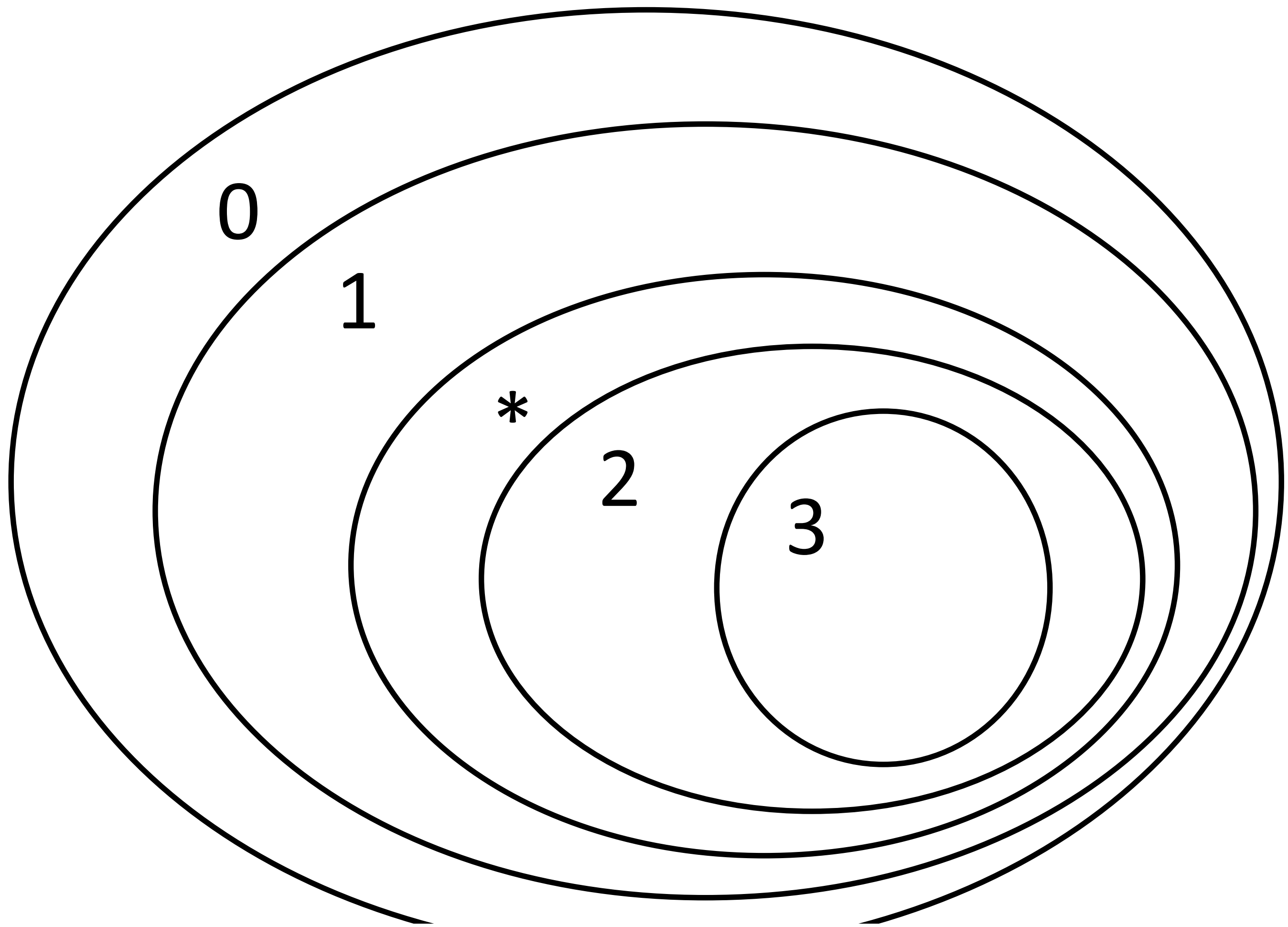
- Provable that each of these four classes is a proper subset of the next one:

Type 0: TM

Type 1: CSG

Type 2: CFG

Type 3: RE



Type 1: Linear-Bounded Automata/ Context-Sensitive Grammars

- TM that uses space linear in the input
- $\alpha A \beta \rightarrow \alpha \gamma \beta$ (γ not empty)

- We mostly ignore these; they get no respect
- Correspond to each other
- Limited compared to full-blown TM
  - But complexity can already be undecidable

Chomsky Hierarchy: proofs

- Form of hierarchy proofs:
  - For each class, you can prove there are languages not in the class, similar to Pumping Lemma proof
  - You can easily prove that the larger class really does contain all the ones in the smaller class

Intersecting, etc., Ls

- We can again investigate what happens with Ls in these various classes under different operations on Ls:
  - Union
  - Intersection
  - Concatenation
  - Negation
  - other operations

Chomsky hierarchy: table

Type	Common Name	Rule Skeleton	Linguistic Example
0	Turing Equivalent	$\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$	HPSG, LFG, Minimalism
1	Context Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$, s.t. $\gamma \neq \epsilon$	
–	Mildly Context Sensitive		TAG, CCG
2	Context Free	$A \rightarrow \gamma$	Phrase-Structure Grammars
3	Regular	$A \rightarrow xB$ or $A \rightarrow x$	Finite-State Automata

Mildly Context-Sensitive Grammars

- We really like CFGs, but are they in fact expressive enough to capture all human grammar?
- Many approaches start with a "CF backbone", and add registers, equations, etc., that are ***not*** CF.
- Several non-hack extensions (CCG, TAG, etc.) turn out to be weakly equivalent!
  - "Mildly context sensitive"
    - So CSFs get even less respect…
    - And so much for the Chomsky Hierarchy being such a big deal

Trying to prove human languages are *not* CF

- Certainly true of semantics. But NL syntax?
- Cross-serial dependencies seem like a good target:
  - *Mary, Jane, and Jim like red, green, and blue, respectively.*
  - But is this syntactic?
- Surprisingly hard to prove

Swiss German dialect!

dative-NP accusative-NP dative-taking-VP accusative-taking-VP

- Jan säit das mer em Hans es huus hälfed aastriiche
- Jan says that we Hans the house helped paint
- "Jan says that we helped Hans paint the house"

- Jan säit das mer d'chind em Hans es huus haend wele laa hälfe aastriiche
- Jan says that we the children Hans the house have wanted to let help paint
- "Jan says that we have wanted to let the children help Hans paint the house"

(A little like "The cat the dog the mouse scared chased likes tuna fish")

Is Swiss German Context-Free?

Shieber's complex argument…

L1 =
Jan säit das mer (d'chind)* (em Hans)* es huus haend wele (laa)* (hälfe)* aastriiche

L2 = Swiss German

L1 ∩ L2 =
Jan säit das mer $(\text{d'chind})^n$ $(\text{em Hans})^m$ es huus haend wele $(\text{laa})^n$ $(\text{hälfe})^m$ aastriiche

Why do we care? (1)

- Math is fun?
- Complexity:
  - If you can use a RE, don't use a CFG.
  - Be careful with anything fancier than a CFG.
- Safety: harder to write correct systems on a Turing Machine.
- Being able to use a weaker formalism may have explanatory power?

Why do we care? (2)

- Probably a source for future new algorithms
- Probably *not* how humans actually process NL
- Might not matter as much for NLP now that we know about real numbers?
  - But we don't want your friends making fun of you