Natural Language Processing

Lecture 13: The Chomsky Hierarchy

Formal Grammars

- Vocabulary of terminal symbols, Σ
- Set of nonterminal symbols, N
- Special start symbol $S \in N$
- Production rules
  - Restrictions on the rules determine what kind of grammar you have

- A formal grammar G defines a formal language, usually denoted L(G).

Regular Grammars

- Regular grammars can be either right-linear or left-linear.
- For right-linear grammars:
  - NT → T (any number of Ts on RHS)
  - NT → T NT (at most one NT on RHS; final)
- Left-linear grammars have the constraints on their RHS reversed
- All regular languages have both a right-linear and a left-linear grammar

Regular Grammars

- aaaa...bbbb...
  - S → a AS
  - S → a BS
  - AS → a AS
  - AS → a BS
  - BS → b
  - BS → b BS
- To what regular expression does this correspond?

Regular Grammars

- L(RG) can be recognized by FSM
  - Can be determinized
    - Each state has at most one arc per terminal
    - But might need 2^n new states
  - Can be minimized
    - Can find a minimal set of states/arcs that accept the same language
- Used in regular expressions

Context Free Grammars

- NT → NT NT T
- NT → T NT
- Only one non-terminal on left hand side
- No restriction on right hand side.
- Good for “bracketing”

CFGs

- $S \rightarrow S + S$
- $S \rightarrow S - S$
- $S \rightarrow (S)$
- $S \rightarrow a, b$
- For arithmetic expressions with a, b

Context Free Grammars

- L(G) recognized by push-down automata
- Can be normalized
  - Chomsky normal form
  - Only one terminal or two non-terminals on RHS
- Most programming languages are context free languages
  - Can you think of any exceptions?

Context Sensitive Grammars

- NT (NT) NT → T NT
- NT (NT) _ → T
- LHS can be more than one symbol
- Bracket symbol rewrites to RHS
- Often used for phonological rules
  - X a Y → X b Y
  - a → b / X __ Y
  - n → m / __ (p|b)
- Irony: almost all phonological rules can be implemented with **finite state** machinery

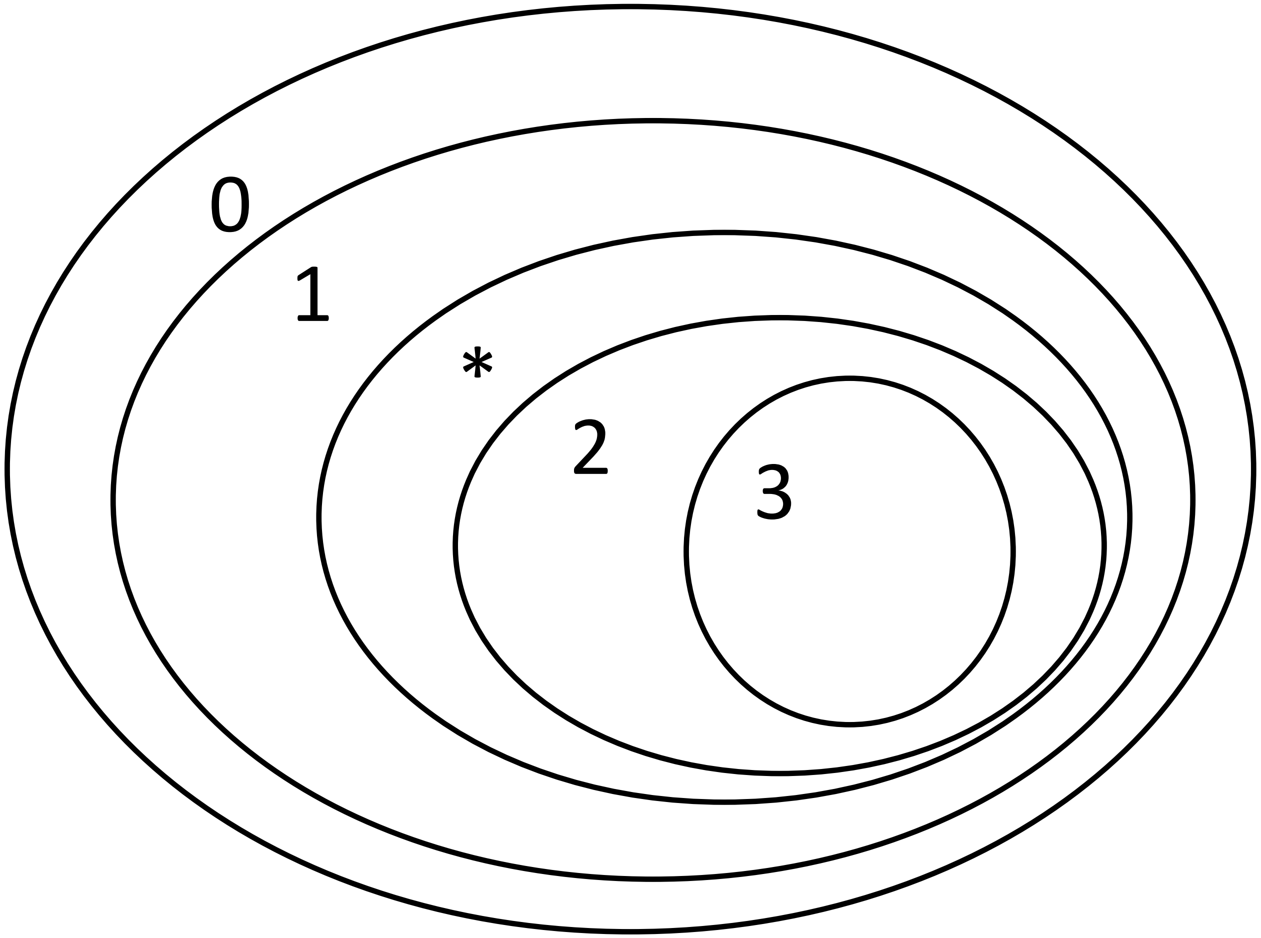
Context Sensitive Grammars

- L(G) recognized by linear bounded automata
- Can be harder to process
  - Parsing is expensive
  - Spurious ambiguity

Generalized Re-write Rules

- [T NT]* → [T NT]*
- Any number of symbols on either side
- Equivalent to Turing Machines
- Can be intractable
- Can be used to implement a new Android twitter client (though inefficiently)

Chomsky Hierarchy



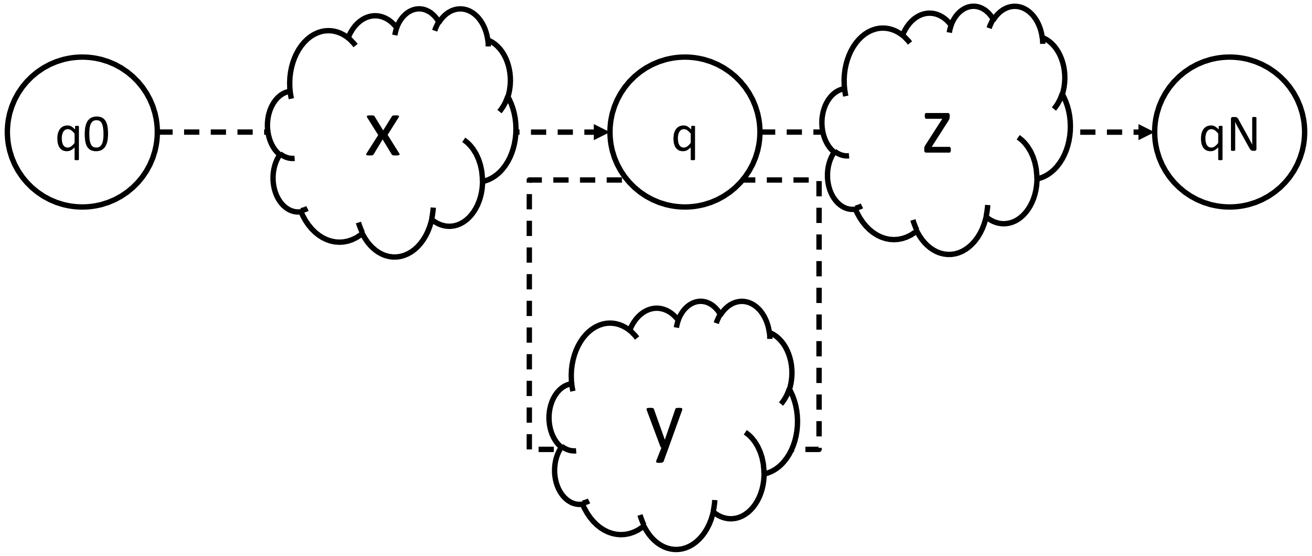
	language class	**automaton**
0	recursively enumerable	Turing machine
1	context-sensitive	linear bounded automaton
*	mildly context-sensitive	thread automaton
2	context-free	pushdown automaton
3	regular	finite-state automaton

Pumping Lemma for Regular Languages

- An intuition (from Jurafsky & Martin, p. 533): "…if a regular language has any long strings (longer than the number of states in the automaton), there must be some sort of loop in the automaton for the language. We can use this fact by showing that if a language *doesn't* have such a loop, then it can't be regular."

Pumping Lemma for Regular Languages

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 $n \geq 0$.



Is English Regular?

- The cat likes tuna fish.
- The cat the dog chased likes tuna fish.
- The cat the dog the rat bit chased likes tuna fish.
- The cat the dog the rat the elephant admired bit chased likes tuna fish.

Is English Regular?

L1 =
(the cat|dog|mouse|...)* (chased|bit|ate|...)* likes tuna fish

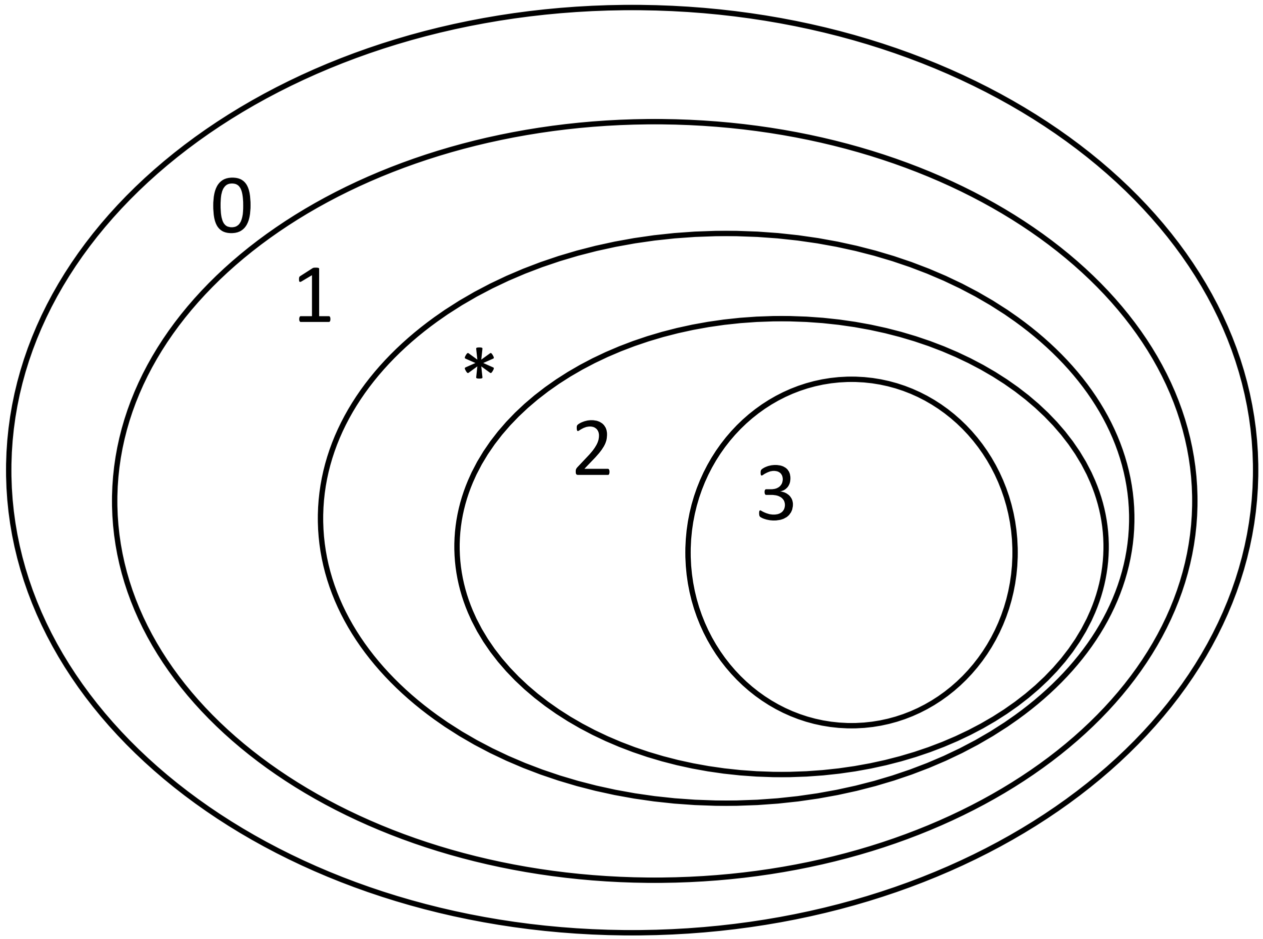
L2 = English

L1 ∩ L2 =
(the cat|dog|mouse|...)^n (chased|bit|ate|...)$^{n-1}$ likes tuna fish

More Examples

- The cat likes tuna fish
- The cat the dog chased likes tuna fish
- The cat the dog the mouse scared chased likes tuna fish
- The cat the dog the mouse the elephant squashed scared chased likes tuna fish
- The cat the dog the mouse the elephant the flea bit squashed scared chased likes tuna fish
- The cat the dog the mouse the elephant the flea the virus infected bit squashed scared chased likes tuna fish

Chomsky Hierarchy



	language class	**automaton**
0	recursively enumerable	Turing machine
1	context-sensitive	linear bounded automaton
*	mildly context-sensitive	thread automaton
2	context-free	pushdown automaton
3	regular	finite-state automaton

Swiss German

dative-Np accusative-Nq dative-taking-Vp accusative-taking-Vq

- 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'

Is Swiss German Context-Free?

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 (d’chind)^n (em Hans)^m es huus haend wele (laa)^n (hälfe)^m aastriiche

Swiss German Appears Not to Be Context Free

- Swiss German has sequences like [A B C A′ B′ C′] where A is licit just in case A′ appears and vice versa (and likewise for B/B′ and C/C′)
- Cross-serial dependencies
- These patterns cannot be recognized by a context free grammar
- Why?

Context Sensitive English

“respectively”

Alice, Bob and Carol will have a beer, a wine and a coffee respectively

A B C … a b c ...

Chomsky Hierarchy

- Natural Language is mildly context sensitive
  - This may not be true of English
    - English is largely context-free
    - There are some exceptional constructions, though
  - This is true of Swiss German, and some other languages
    - The frequency of context-sensitive constructions is relatively low
    - They tend to be bounded in depth by memory constraints

Are CFGs Sufficient?

- For your application, CFGs might be adequate, even if you are modeling Swiss German
  - They are more computationally tractable that context sensitive grammars, even weakly context sensitive grammars
  - They are easy to understand
  - Parsers are easy to implement
    - We will talk about this in the next three lectures
    - CYK/CKY algorithm, Earley algorithm, etc.

Are Regular Grammars Sufficient?

- For many applications, regular grammars are actually sufficient
  - Since recursive structures in syntax are bounded by working memory, actual sentences people say and write can usually be modeled by a sufficiently complex FSM
  - Once compiled, FSMs run in linear time
  - But they are “flat;” cannot parse sentences into nested constituents