Linguistic Background

Lexical Category, Logic, Syntax, Grammar

Instructor: Nick Cercone - 3050 LAS - nick@cse.yorku.ca

Lexical category

- (also word class, lexical class, or in traditional grammar part of speech)
- a linguistic category of words (or more precisely *lexical items*), is generally defined by the *syntactic* or *morphological* behaviour of the lexical item in question.
- Common linguistic categories include *noun* and *verb*, among others.
- Open word classes constantly acquire new members
- Closed word classes acquire new members infrequently

Functional classification

- Open word classes:
  - adjectives
  - adverbs
  - interjections
  - nouns
  - verbs (except auxiliary verbs)
- Closed word classes:
  - auxiliary verbs
  - conjunctions
  - determiners (articles, quantifiers, demonstrative adjectives, and possessive adjectives)
  - particles
  - adpositions (prepositions, postpositions, and circumpositions)
  - pronouns
  - contractions
  - Misc (clitics, coverbs, measure words, preverbs, cardinal numbers)

POS tagging

Problem: Given a text where each word is associated with all its possible parts of speech, determine the most likely POS for the word with respect to its context.

- *who* PRON(int), PRON(rel)
- can AUX, V(inf), N(sg)
- it EXPLETIVE, PRON(3sg)
- *be* V(inf)
- ? PUNC

POS tagging

Russian/JJ and/CC Ukrainian//NNP energy/NN officials/NNS reached/VBD a/DT deal/NN Wednesday/NNP morning/NN ,/, ending/VBG a/DT dispute/NN over/IN the/DT price/NN of/IN Russian/JJ natural/JJ gas/NN that/WDT caused/VBD shortages/NNS in/IN the/DT Ukraine//NNP and/CC throughout/IN Western/NNP Europe/NNP ./.

Regular Expressions and Finite State Automata

- A **regular expression** is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e., "find and replace"- like operations. The concept arose in the 1950s, when the American mathematician Stephen Kleene formalized the description of a *regular language*, and came into common use with the Unix text processing utilities ed, an editor, and grep (global regular expression print), a filter.
- Each character in a regular expression is either understood to be a metacharacter with its special meaning, or a regular character with its literal meaning. Together, they can be used to identify textual material of a given pattern, or process a number of instances of it that can vary from a precise equality to a very general similarity of the pattern. The pattern sequence itself is an expression that is a statement in a language designed specifically to represent prescribed targets in the most concise and flexible way to direct the automation of text processing of general text files, specific textual forms, or of random input strings.

Regular Expressions and Finite State Automata

- A very simple use of a regular expression would be to locate the same word spelled two different ways in a text editor, for example seriali[sz]e. A wildcard match can also achieve this, but wildcard matches differ from regular expressions in that wildcards are limited to what they can pattern (having fewer metacharacters and a simple language-base), whereas regular expressions are not. A usual context of wildcard characters is in globbing similar names in a list of files, whereas regular expressions are usually employed in applications that pattern-match text strings in general. For example, the simple regexp **^[\t]+|[\t]+$** matches excess whitespace at the beginning and end of a
- A **regular expression processor** processes a regular expression statement expressed in terms of a grammar in a given formal language, and with that examines the target text string, parsing it to identify substrings that are members of its language, the regular expressions.

Regular Expressions and Finite State Automata

- Regular expressions are so useful in computing that the various systems to specify regular expressions have evolved to provide both a *basic* and *extended* standard for the grammar and syntax; *modern* regular expressions heavily augment the standard. Regular expression processors are found in several search engines, search and replace dialogs of several word processors and text editors, and in the command lines of text processing utilities, such as sed and AWK.
- Many programming languages provide regular expression capabilities, some built-in, for example Perl, Ruby, AWK, and Tcl, and others via a standard library, for example .NET languages, Java, Python and C++ (since C++11). Most other languages offer regular expressions via a library.
- Regular expressions describe regular languages in formal language theory. They have the same expressive power as regular grammars.

Regular Expressions and Finite State Automata – Formal Definition

Regular expressions consist of constants and operator symbols that denote sets of strings and operations over these sets, respectively. The following definition is standard, and found as such in most textbooks on formal language theory. Given a finite alphabet Σ, the following constants are defined as regular expressions:

- (*empty set*) ∅ denoting the set ∅.
- (*empty string*) ε denoting the set containing only the "empty" string, which has no characters at all.
- (*literal character*) *a* in Σ denoting the set containing only the character *a*.

Given regular expressions R and S, the following operations over them are defined to produce regular expressions:

- (*concatenation*) *RS* denotes the set of strings that can be obtained by concatenating a string in R and a string in S. For example {"ab", "c"}{"d", "ef"} = {"abd", "abef", "cd", "cef"}.

Regular Expressions and Finite State Automata – Formal Definition

- (*alternation*) *R* | *S* denotes the set union of sets described by *R* and *S*. For example, if R describes {"ab", "c"} and S describes {"ab", "d", "ef"}, expression *R* | *S* describes {"ab", "c", "d", "ef"}.
- (*Kleene star*) *R** denotes the smallest superset of set described by *R* that contains ε and is closed under string concatenation. This is the set of all strings that can be made by concatenating any finite number (including zero) of strings from set described by *R*. For example, {"0","1"}* is the set of all finite binary strings (including the empty string), and {"ab", "c"}* = {ε, "ab", "c", "abab", "abc", "cab", "cc", "ababab", "abcab", ... }.

To avoid parentheses it is assumed that the Kleene star has the highest priority, then concatenation and then alternation. If there is no ambiguity then parentheses may be omitted. For example, (ab)c can be written as abc, and a|(b(c*)) can be written as a|bc*. Many textbooks use the symbols ∪, +, or ∨ for alternation instead of the vertical bar.

Regular Expressions and Finite State Automata: Examples

- a|b* denotes {ε, "a", "b", "bb", "bbb", ...}
- (a|b)* denotes the set of all strings with no symbols other than "a" and "b", including the empty string: {ε, "a", "b", "aa", "ab", "ba", "bb", "aaa", ...}
- ab*(c|ε) denotes the set of strings starting with "a", then zero or more "b"s and finally optionally a "c": {"a", "ac", "ab", "abc", "abb", "abbc", ...}

Expressive power and compactness

The formal definition of regular expressions is purposely parsimonious and avoids defining the redundant quantifiers ? and +, which can be expressed as follows: a+ = aa*, and a? = (a|ε). Sometimes the complement operator is added, to give a *generalized regular expression*; here R^c matches all strings over Σ* that do not match *R*. In principle, the complement operator is redundant, as it can always be circumscribed by using the other operators. However, the process for computing such a representation is complex, and the result may require expressions of a size that is double exponentially larger.

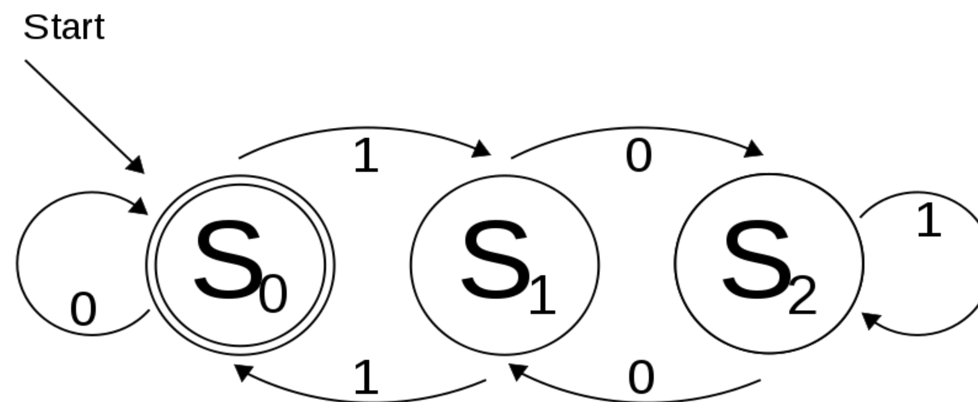
Regular Expressions and Finite State Automata

Regular expressions in this sense can express the regular languages, exactly the class of languages accepted by deterministic finite automata. There is, however, a significant difference in compactness. Some classes of regular languages can only be described by deterministic finite automata whose size grows exponentially in the size of the shortest equivalent regular expressions.

There is a simple mapping from regular expressions to the more general nondeterministic finite automata (NFAs) that does not lead to such a blowup in size; for this reason NFAs are often used as alternative representations of regular languages. NFAs are a simple variation of the type-3 grammars of the Chomsky hierarchy.

Finite State Automata - Deterministic finite automaton

- A **deterministic finite automaton** (**DFA**)—also known as **deterministic finite state machine**—is a finite state machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string. 'Deterministic' refers to the uniqueness of the computation. In search of simplest models to capture the finite state machines, McCulloch and Pitts were among the first researchers to introduce a concept similar to finite automaton in 1943.
- The figure illustrates a deterministic finite automaton using a state diagram.



Finite State Automata - Deterministic finite automaton

In the automaton, there are three states: S0, S1, and S2 (denoted graphically by circles). The automaton takes a finite sequence of 0's and 1's as input. For each state, there is a transition arrow leading out to a next state for both 0 and 1. Upon reading a symbol, a DFA jumps *deterministically* from a state to another by following the transition arrow. For example, if the automaton is currently in state S0 and current input symbol is 1 then it deterministically jumps to state S1. A DFA has a *start state* (denoted graphically by an arrow coming in from nowhere) where computations begin, and a set of *accept states* (denoted graphically by a double circle) which help define when a computation is successful.

A DFA is defined as an abstract mathematical concept, but due to the deterministic nature of a DFA, it is implementable in hardware and software for solving various specific problems. For example, a DFA can model software that decides whether or not online user-input such as email addresses are valid. DFAs recognize exactly the set of regular languages which are, among other things, useful for doing lexical analysis and pattern matching. DFAs can be built from nondeterministic finite automata.

Finite State Automata - Deterministic finite automaton - Formal definition

A **deterministic finite automaton** M is a 5-tuple, $(Q, \Sigma, \delta, q0, F)$, consisting of

- a finite set of states (Q)
- a finite set of input symbols called the alphabet (Σ)
- a transition function ($\delta : Q \times \Sigma \rightarrow Q$)
- a start state ($q0 \in Q$)
- a set of accept states ($F \subseteq Q$)

Let $w = a_1a_2 \ldots a_n$ be a string over the alphabet Σ. The automaton M accepts the string w if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in Q with the following conditions:

- $r_0 = q0$
- $r_{i+1} = \delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$
- $r_n \in F$.

Finite State Automata - Deterministic finite automaton - Formal definition

In words, the first condition says that the machine starts in the start state q_0. The second condition says that given each character of string *w*, the machine will transition from state to state according to the transition function δ. The last condition says that the machine accepts *w* if the last input of *w* causes the machine to halt in one of the accepting states. Otherwise, it is said that the automaton *rejects* the string. The set of strings *M* accepts is the language *recognized* by *M* and this language is denoted by *L(M)*.

A deterministic finite automaton without accept states and without a starting state is known as a transition system or semiautomaton.

For more comprehensive introduction of the formal definition see automata theory.

English is not a regular language

- 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

$$(\textit{the} + \mathrm{N})^n \, \mathrm{Vt}^{n-1} \; \textit{likes tuna fish}$$

We'll return to this slide later

Linguistic complexity

Why are some sentences more difficult to understand than others?

The cat the dog the rat bit chased likes tuna fish

Limitations on stack size?

Formal Languages

Formal languages sprang to life around 1956 when Noam Chomsky gave a mathematical model of a grammar in connection with his study of natural language.

We define a formal language abstractly as a mathematical system, allowing us to make vigorous statements about formal languages and develop a body of knowledge which can be applied to them.

Formal Languages

We make the following definitions:

- alphabet (or vocabulary) - any finite set of symbols
- sentences - over an alphabet, a sentence is any string of finite length composed of symbols from the alphabet. Synonyms for sentences are *string* and *word*. The empty sentence, E, in the sentence consists of no symbols. If V is an alphabet, then V* denotes the set of all sentences composed of symbols of V, including E. We use V+ to denote the set V*- { E} . Thus, if V={ 0,1}, then V*={ E, 0, 1, 00, 01, 10, 11, 000, ...} and V+={ 0, 1, 00, 01, 10, 11, 000, ...).
- language - any set of sentences over an alphabet.

Formal Notion of a Grammar

- Formally, we denote a *grammar* G by (Vn, Vt, P, S). The symbols Vn, Vt, P, and S are, respectively, the *variables, terminals, productions*, and *start symbol*. Vn, Vt, and P are finite sets. We assume that Vn and Vt contain no elements in common; i.e.,

$$Vn \cap Vt = \varepsilon$$

where ε denotes the empty set and, conventionally

$$Vn \cup Vt = V$$

The set of productions P consists of expressions of the form a → b, where a is a string in V+ and b is a string in V*. Finally, S is always a symbol in Vn.

Formal Notion of a Grammar

- Customarily, we shall use capital Latin alphabet letters for variables. Lower case letters at the beginning of the Latin alphabet are used for terminals. Strings of terminals are denoted by lower case letters near the end of the Latin alphabet, and strings of variables and terminals are denoted by lower case Greek letters.
- Given the grammar G = (Vn, Vt, P, S), we define the language it generates as follows. If a → b is a production of P and g and d are strings in V*, then gad → gbd.
- In English we say gad *directly derives* gbd in *grammar G.* We say that the production a → b is applied to the string gad to obtain gbd. Thus G relates two strings exactly when the second is obtained from the first by the application of a single production.

Complexity of natural languages

- Computational complexity: the expressive power of (and the resources needed to process) classes of languages
- Linguistic complexity: what makes individual constructions or sentences more difficult to understand
  - This is the dog, that worried the cat, that killed the rat, that ate the malt, that lay in the house that Jack built.
  - This is the malt that the rat that the cat that the dog worried killed ate.

The Chomsky hierarchy of languages

- A hierarchy of classes of languages, viewed as sets of strings, ordered by their "complexity". The higher the language is in the hierarchy, the more "complex" it is.
- In particular, the class of languages in one class properly includes the languages in lower classes.
- There exists a correspondence between the class of languages and the format of phrase-structure rules necessary for generating all its languages. The more restricted are the rules, the lower in the hierarchy are the languages they generate.

The Chomsky hierarchy of languages

- Recursively enumerable languages
- Context-sensitive languages
- Context-free languages
- Regular languages

The Chomsky hierarchy of languages

- Type-0 grammars (unrestricted grammars) include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. These languages are also known as the recursively enumerable languages.
- Type-1 grammars (context-sensitive grammars) generate the context-sensitive languages. These grammars have rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ with A a nonterminal and α, β and γ strings of terminals and nonterminals. The strings α and β may be empty, but γ must be nonempty. The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule. The languages described by these grammars are exactly all languages that can be recognized by a linear bounded automaton (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input.)

The Chomsky hierarchy of languages

- Type-2 grammars (context-free grammars) generate the context-free languages defined by rules of the form $A \rightarrow \gamma$ with A a nonterminal and γ a string of terminals and nonterminals. These languages are exactly all languages recognized by a non-deterministic pushdown automaton. Context-free languages are the basis for most programming languages.
- Type-3 grammars (regular grammars) generate the regular languages restricting its rules to a single nonterminal on the LHS and a RHS consisting of a single terminal, possibly followed (or preceded, but not both in the same grammar) by a single nonterminal. The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the RHS of any rule. These languages are those that can be decided by a finite state automaton. This family of languages can be obtained by regular expressions. Regular languages are commonly used to define search patterns and the lexical structure of programming languages.

The Chomsky hierarchy of languages

Every regular language is context-free, every context-free language, not containing the empty string, is context-sensitive and every context-sensitive language is recursive and every recursive language is recursively enumerable. These are all proper inclusions, meaning that there exist recursively enumerable languages which are not context-sensitive, context-sensitive languages which are not context-free and context-free languages which are not regular.

The Chomsky hierarchy of languages

The following table summarizes each of Chomsky's four types of grammars, the class of language it generates, the type of automaton that recognizes it, and the form its rules must have.'

Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	α _ β (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	αAβ _ αγβ
Type-2	Context-free	Non-deterministic pushdown automaton	A _ γ
Type-3	Regular	Finite state automaton	A _ a and A _ a B

Why is it interesting?

- The hierarchy represents some informal notion of the complexity of natural languages
- It can help accept or reject linguistic theories
- It can shed light on questions of human processing of language

What exactly is the question?

- When viewed as a set of strings, is English a regular language? Is it context-free? How about Hebrew?
- Competence vs. Performance
  - This is the dog, that worried the cat, that killed the rat, that ate the malt, that lay in the house that Jack built.
  - This is the malt that the rat that the cat that the dog worried killed ate

Where are natural languages located?

- Chomsky (1957): "English is not a regular language"
- As for context-free languages, "I do not know whether or not English is itself literally outside the range of such analyses"

How not to do it

An introduction to the principles of transformational syntax (Akmajian and Heny, 1976)

"Since there seem to be no way of using such PS rules to represent an obviously significant generalization about one language, namely, English, we can be sure that phrase structure grammars cannot possibly represent all the significant aspects of language structure. We must introduce a new kind of rule that will permit us to do so."

How not to do it

Example: Syntax (Peter Culicover, 1976)

In general, for any phrase structure grammar containing a finite number of rules it will always be possible to construct a sentence that the grammar will not generate. In fact, because of recursion there will always be an infinite number of such sentences. Hence, the phrase structure analysis will not be sufficient to generate English.

How not to do it

Example: Transformational grammar (Grinder & Elgin, 1973)

the girl saw the boy

*the girl kiss the boy

this well-known syntactic phenomenon demonstrates clearly the inadequacy of ... context-free phrase-structure grammars...

How not to do it

The defining characteristic of a context-free rule is that the symbol to be rewritten is to be rewritten without reference to the context in which it occurs. By definition, one cannot write a context-free rule that will expand the symbol *V* into *kiss* in the context of being immediately preceded by the sequence *the girls* and that will expand the symbol *V* into *kisses* in the context of being immediately preceded by the sequence *the girl*. Any set of context-free rules that generate (correctly) the sequences *the girl kisses the boy* and *the girls kiss the boy* will also generate (incorrectly) the sequences *the girl kiss the boy* and *the girls kisses the boy.*

How not to do it

The grammatical phenomenon of Subject-Predicate agreement is sufficient to guarantee the accuracy of: "English is not a context-free language".

How not to do it

Example: Syntactic theory (Bach 1974)

> These examples show that to describe the facts of English number agreement is literally impossible using a simple agreement rule of the type given in a phrase-structure grammar, since we cannot guarantee that the noun phrase that determines the agreement will precede (or even be immediately adjacent) to the present-tense verb.

How not to do it

Example: A realistic transformational grammar (Bresnan, 1978)

in many cases the number of a verb agrees with that of a noun phrase at some distance from it... this type of syntactic dependency can extend as far as memory or patience permits... The distant type of agreement... cannot be adequately described even by context-sensitive phrase-structure rules, for the possible context is not correctly describable as a finite string of phrases.

How to do it right - Proof techniques:

The pumping lemma for regular languages

- for a particular language to be a member of a language class, any sufficiently long string in the language contains a section, or sections, that can be removed, or repeated any number of times, with the resulting string remaining in that language.

Closure under intersection

- A set has **closure** under an operation if performance of that operation on members of the set always produces a member of the same set.

Closure under homomorphism

- A homomorphism is a function from strings to strings. Its output on a multi-character string is just the concatenation of its outputs on each individual character in the string. Or, equivalently, $h(xy) = h(x)h(y)$ for any strings x and y. If S is a set of strings, then h(S) is $\{w : w = h(x)$ for some x in S$\}$.
- To show that regular languages are closed under homomorphism, choose an arbitrary regular language L and a homomorphism h. It can be represented using a regular expression R. But then h(R) is a regular expression representing h(L). So h(L) must also be regular.

English is not a regular language

- Center embedding
- The following is a sequence of grammatical English sentences:

  A white male hired another white male.

  A white male – whom a white male hired – hired another white male.

  A white male – whom a white male, whom a white male hired, hired – hired another white male.

- Therefore, the language L_{trg} is a subset of English:

  Ltrg = { A white male (whom a white male)^n (hired)^n hired another white male | n > 0}

English is not a regular language

- L_{trg} is not a regular language. L_{trg} is the intersection of the natural language English with the regular set

  Lreg = { A white male (whom a white male) (hired) hired another white male }

- L_{reg} is regular, as it is defined by a regular expression.
- Since the regular languages are closed under intersection, and since L_{reg} is a regular language, then if English were regular, its intersection with L_{reg} , namely L_{trg} , would be regular.
- Since L_{trg} is trans-regular, English is not a regular language.

English is not a regular language

Similar constructions:

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

$(the + N)^{n}\ Vt^{n-1}$ *likes tuna fish*

Conclusion

Is English contex-free?

Other Concluding Remarks

A PSYCHOLOGICAL TIP

Whenever you're called on to make up your mind,
and you're hampered by not having any,
the best way to solve the dilemma, you'll find,
is simply by spinning a penny.
No -- not so that chance shall decide the affair
while you're passively standing there moping;
but the moment the penny is up in the air,
you suddenly know what you're hoping.