Introduction to Information Sciences

Natural Language Processing
Formal Languages

mcuturi@i.kyoto-u.ac.jp

Reminder

- **Language**: "A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols." $L \subset \Sigma^\star, L \in \mathcal{P}(\Sigma^\star)$
- **Grammar**: "A grammar can be regarded as a **device** that enumerates the sentences of a language."
  - T, a finite set of terminal symbols,
  - N, a finite set of nonterminal symbols,
  - $S \in N$, a start symbol which is a nonterminal symbol,
  - P a finite set of production rules:

$$\text{Rule}: \quad \cdots \rightarrow \cdots$$

  where the dots are arbitrary symbols.

- A grammar of L can be regarded as a function whose range is exactly L

Summary of Today's Lecture

- Chomsky Hierarchy
- Turing Machines
- Type-0, Type-1 & Type-2 languages in more detail
- Relationships between Type-2 grammars and natural language.

Inspiration for some of the slides: A. McCallum's online lectures, Wikipedia, various online contents

Chomsky Hierarchy of Formal Languages

Reminder

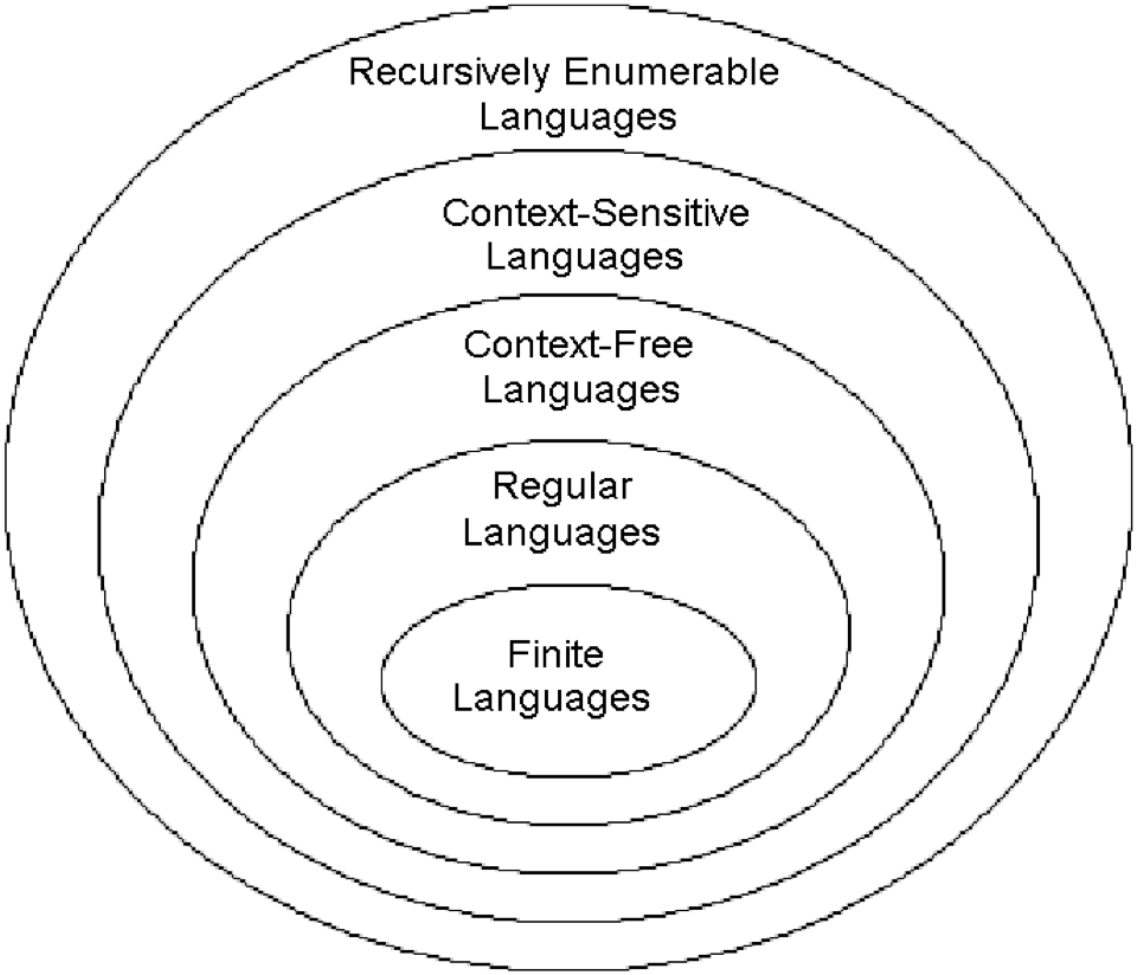
- Type-0 : all grammars.
- Type-1: $\alpha A \beta \rightarrow \alpha \gamma \beta$ where γ cannot be empty. $S \rightarrow \varepsilon$ is allowed iff S does not appear on the right side of a rule.
- Type-2 $A \rightarrow \gamma$ where γ a string of terminals and nonterminals.
- Type-3: Nonterminals can only appear on one side, $S \rightarrow \varepsilon$ is allowed iff S does not appear on the right side of a rule.

Grammar	Languages	Automaton	**Production rules (constraints)**
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$(no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	**Context-free**	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$

taken from Noam Chomsky,**On Certain Formal Properties of Grammars**, *Information and Control*, Vol 2, 1959

Chomsky Hierarchy of Formal Languages

- This table defines a set of inclusions:



- Let's review more closely the meaning of the table line-by-line

Type-0 Languages

Turing machines

- A Turing machine is a 7-tuple $M = \{Q, \Gamma, b, \Sigma, \delta, q_0, F\}$ where
  - Q is a finite, non-empty set of **states**
  - Γ is a finite, non-empty set of **symbols**
  - $b \in \Gamma$ is the **blank symbol** (the only symbol allowed to occur on the tape infinitely often at any step during the computation)
  - $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of **input symbols**
  - $q_0 \in Q$ is the **initial state**
  - $F \subseteq Q$ is the set of **final** or **accepting states**.
  - $\delta : \Gamma \times \{Q \setminus F\} \rightarrow Q \times \{\Gamma \cup \{<, >\}\}$ is the **transition function**, where $<$ is left shift, $>$ is right shift.

- This is one of many possible formulations.
  ...*Why?* because a TM is a philosophical and a mathematical object
- Differently interpretations of **Turing's original formulation**.

in Alan Turing's mind

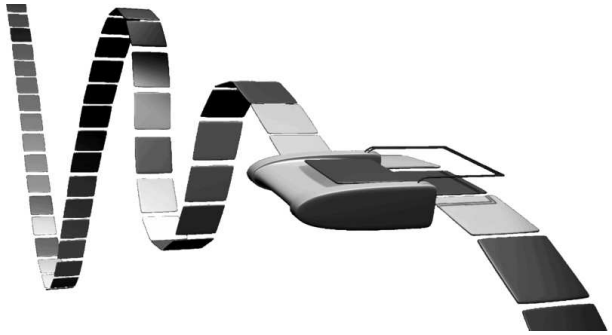
...an infinite memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings.

(A.M. Turing, Intelligent Machinery, 1948)

Turing machines

- A Turing machine is executed on a **tape** of infinite length. The tape only contains **symbols of** Γ.

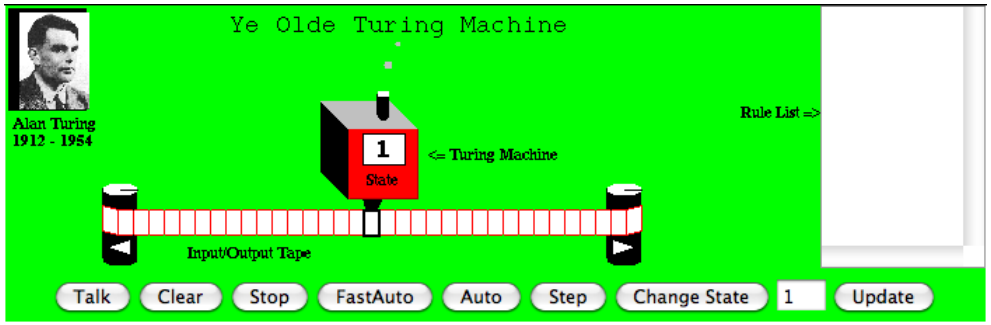
- The TM **accepts a tape** if it reaches a final/accepting state after a finite number of iterations.



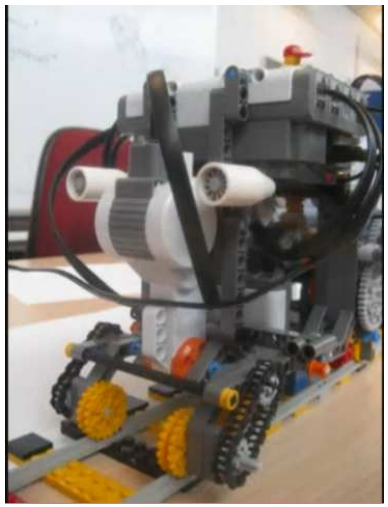
Turing Machines

- Let's see a Turing machine in action

  `http://web.bvu.edu/faculty/schweller/Turing/Turing.html`



- Compare with this one displayed on this video



Alternative Formulations of Turing Machines

- $\Gamma \times \delta : Q \setminus F \rightarrow Q \times \Gamma \times \{<, |, >\}$: replaces *and* moves (or stands still)
- **Multiple** (but finite number of) tapes $\Rightarrow$ boils down to single tape
  - Comparable to $\mathbb{N}$ countable, then $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Q}^n$ *etc.* are countable too.
  - Multiple tape can be expressed as single tapes, but far heavier notation.

$$\delta : \Gamma^k \times \{Q \setminus F\} \rightarrow Q \times (\Gamma \times \{<, >\})^k.$$

- **Nondeterministic**: action at each iteration is **one of many possible**
  - Different actions may apply for the same combination of state and symbol.

$$\delta \subseteq (\Gamma \times \{Q \setminus F\}) \times (Q \times \{\Gamma \cup \{<, >\}\})$$

    is now a "**transition relation**".
  - When a tape is fed to the TM, and this TM can reach an accepting state for **one out of all possible paths**, then the TM is said to accept that tape.

Type-0 Languages & Turing machines

unrestricted grammars = type 0 languages?

- proof $\Rightarrow$: build a **TM** that can **decide whether a word w is in $L(G)$**.
- consider a two-tape Turing machine.
  - tape t_1 contains the input word w to be tested,
  - tape t_2 is used by the machine to generate words from G. Contains starting symbol.
- The Turing machine then does the following:
  - *Nondeterministically* choose a symbol in non empty slot on t_2.
  - *Nondeterministically* choose a production $\beta \to \gamma$ from the productions in G.
  - If β appears at that position on t_2, replace β by γ at that point, shifting symbols if necessary.
  - Compare the resulting word on t_2 to the word on t_1:
    - they match, then the Turing machine accepts the word.
    - they don't: back to step 1.

Type-1 Languages

Type-1 Languages, Context-sensitive

- Some restriction on the rules:
  - $\alpha A \beta \to \alpha \gamma \beta$ where γ cannot be empty.
  - $S \to \varepsilon$ is allowed iff S does not appear on the right side of a rule.

Context-**sensitive**

$\Updownarrow$

transformation of **non-terminal symbols** may depend on their context.

- Example: generative grammar of CS language $\{a^n b^n c^n : n \geq 1\}$:

$$
\begin{array}{lll}
1: & S & \to aSBC \\
2: & S & \to aBC \\
3: & CB & \to HB \\
4: & HB & \to HC \\
5: & HC & \to BC \\
6: & aB & \to ab \\
7: & bB & \to bb \\
8: & bC & \to bc \\
9: & cC & \to cc
\end{array}
$$

Type-1 Languages, Context-sensitive

- The generation chain for aaa bbb ccc is:

$$
\begin{aligned}
& S \\
\rightarrow_1 \; & aSBC \\
\rightarrow_1 \; & a\boldsymbol{aSBC}BC \\
\rightarrow_2 \; & aa\boldsymbol{aBC}BCBC \\
\rightarrow_3 \; & aaaB\boldsymbol{HB}CBC \\
\rightarrow_4 \; & aaaB\boldsymbol{HC}CBC \\
\rightarrow_5 \; & aaaB\boldsymbol{BC}CBC \\
\rightarrow_3 \; & aaaBBC\boldsymbol{HB}C \\
\rightarrow_4 \; & aaaBBC\boldsymbol{HC}C \\
\rightarrow_5 \; & aaaBBC\boldsymbol{BC}C \\
\rightarrow_3 \; & aaaBB\boldsymbol{HB}CC \\
\rightarrow_4 \; & aaaBB\boldsymbol{HC}CC \\
\rightarrow_5 \; & aaaBB\boldsymbol{BC}CC \\
\rightarrow_6 \; & aa\boldsymbol{ab}BBCCC \\
\rightarrow_7 \; & aaa\boldsymbol{bb}BCCC \\
\rightarrow_7 \; & aaab\boldsymbol{bb}CCC \\
\rightarrow_8 \; & aaabb\boldsymbol{bc}CC \\
\rightarrow_9 \; & aaabbb\boldsymbol{cc}C \\
\rightarrow_9 \; & aaabbbc\boldsymbol{cc}
\end{aligned}
$$

Type-1 Languages, Context-sensitive

A few remarks

- $|\alpha A \beta| \leq |\alpha \gamma \beta|$
- At each iteration of a production rule, the word's length can only **grow**.
- Hence CSGs are called **non-contracting** grammars.
- As usual, same language can be generated with simpler grammar
  - Example: consider the following CSG

$$\begin{aligned} S &\rightarrow aSBc \\ S &\rightarrow abc \\ cB &\rightarrow Bc \\ bB &\rightarrow bb \end{aligned}$$

  - It also generates $\{a^n b^n c^n : n \geq 1\}$

Type-1 Languages & Linear Bounded Automaton

- A LBA is a restricted form of TM that satisfies the following conditions.

  1. The set of symbols Γ has two **special symbols**: **left and right endmarkers**.
  2. **endmarkers** can only appear once.
  3. The LBA **cannot move beyond the left & right endmarkers**, and it cannot modify the endmarkers

- More realistic to represent a computer (finite memory).
- "Context-sensitive grammars $\Leftrightarrow$ LBA" was proved by Kuroda in 60's.

  S.-Y. Kuroda, **Classes of languages and linear-bounded automata**, *Information and Control*, 7(2): 207223, June 1964.

- Basic idea of the proof: comes from non-contracting property.
- To check if **x** is in the language,
  - generate all possible words of length $\leq |\mathbf{x}|$
  - check if they are equal to **x**

Type-2 Languages

Type-2 Languages

- Defined by context-**free** grammars
- Production rules $A \to \gamma$ where γ is a string of terminals and nonterminals.
- **"free"** : **non-terminal** symbols can only have straightforward transformations

A few examples

- **Well-formed parentheses**
  - two terminal symbols (and), one nonterminal symbol S.
  - The production rules are

$$S \to SS;\ S \to (S);\ S \to ()$$

  - The first rule allows S's to multiply; the second rule allows Ss to become enclosed by matching parentheses; and the third rule terminates the recursion.
  - Example: $S \to SS \to SSS \to (S)SS \to ((S))SS \to ((SS))S(S) \to (((()S))S(S) \to (((())))S(S) \to (((()))))()(S) \to (((()))))()(())$

Type-2 Languages

- **Well-formed nested parentheses and square brackets**
  - terminal { symbols $[]()$ and nonterminal S.
  - production rules:

$$S \to SS, S \to (),$$
$$S \to (S), S \to [],$$
$$S \to [S]$$

  - Example:

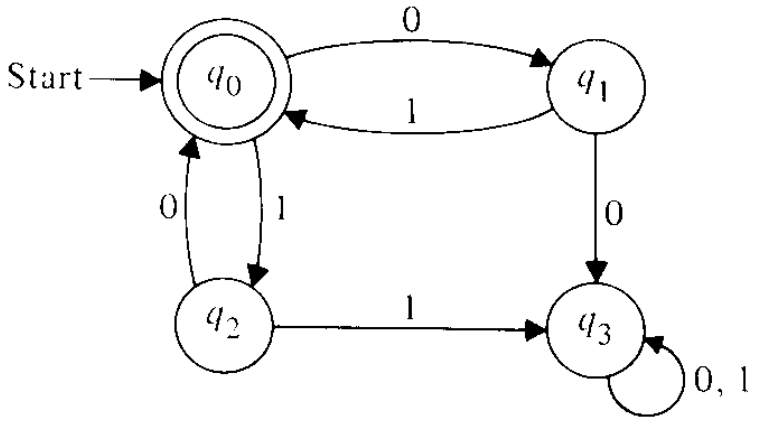
$$([[[()()[][]]]([])])$$

- There is **no context-free grammar** that generates all balanced sequences of two different types of parentheses, as for example:

$$[[[[((([]]]))))(([])(([])([])([])([])$$

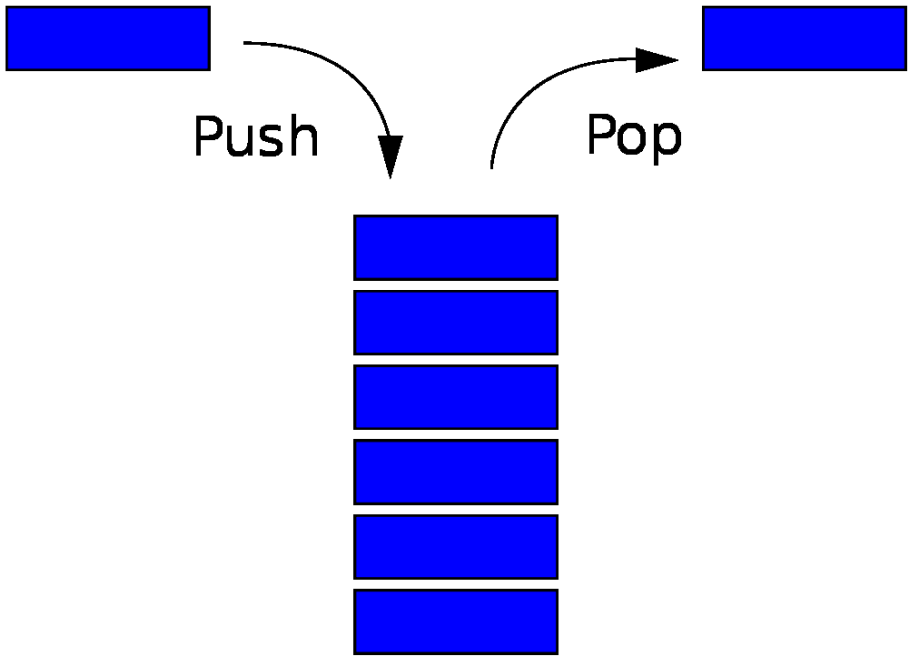
Parenthesis: Finite State Machines/Automaton

- A **finite state machine** 5-uple $(\Sigma, Q, q_0, \delta, F)$, where:
  - Σ is the input alphabet (a finite, non-empty set of symbols).
  - Q is a finite, non-empty set of states.
  - q_0 is the initial state, an element of Q.
  - δ is the state-transition function: $\delta : \Sigma \times Q \rightarrow Q$
  - F is the set of final states, a (possibly empty) subset of Q.
- Simply put, a **finite-state machine** is a Turing machine **without** the ability to
  - **modify** an infinite tape,
  - move freely in the infinite tape, can only go from left to right on the tape.



Push-down Automaton

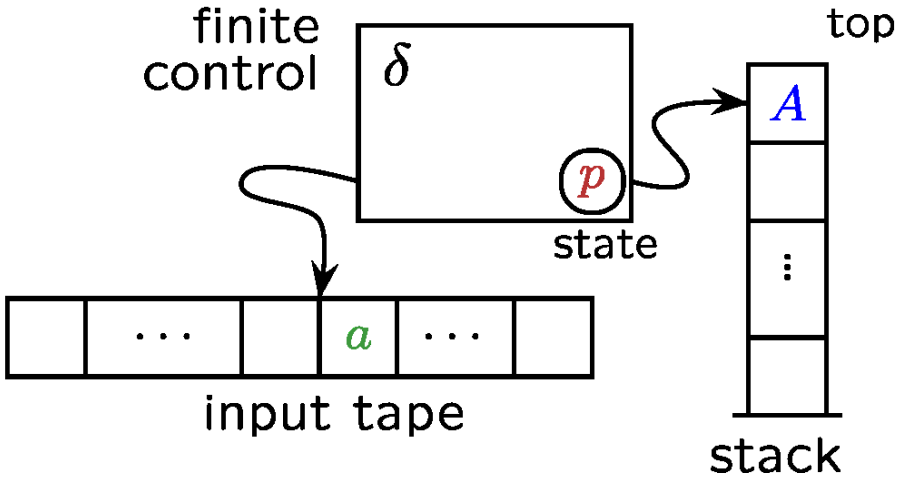
- A push-down automaton is a FSM with an additional tool to play with: a **stack**
- A stack is
  - a structure to hold data,
  - where data can be piled in and recovered according to 2 simple rules:
    - The data can be **push**ed in the stack,
    - The data can be **pop**ed out from the **top** the stack and recovered.



Push-down Automaton

- The push-down automaton can use that stack to modify its behaviour (namely to change the output of δ):

  1. They can use the **top of the stack** to decide which transition to take.
  2. They can **manipulate** (**push/pop**) the stack as part of performing a transition.



- The stack can be compated to a **restricted** kind of tape for TM's.

Push-down Automaton

- A PDA is a 7-uple, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ where
  - Q is a finite set of **states**,
  - $\Sigma$is a finite set of symbols, the **input alphabet**,
  - Γ is a finite set which is called the **stack alphabet**,
  - δ is a mapping $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$, the **transition relation**, where ε is the *empty string*.
  - $q_0 \in Q$ is the **start state**
  - $Z \in \Gamma$ is the **initial stack symbol**
  - $F \subseteq Q$ is the set of **final/accepting states**
- An element $(p, a, A, q, \alpha) \in \delta$ is a transition of M.

Type-2 Languages & Push-down Automaton

- Every context-free grammar can be transformed into an equivalent pushdown automaton
- given a context-free grammar, the PDA is constructed as follows.
  - $(1, \varepsilon, A, 1, \alpha)$ for each rule $A \to \alpha$ *(expand)*.
  - $(1, a, a, 1, \varepsilon)$ for each terminal symbol a *(match)*.
- The other way round is more difficult to prove.

Type-3 Languages

Type-3 Language

- Type-3 languages are called regular languages.
  - Nonterminals can only appear on one side, $A \to a$ and $A \to aB$.
  - $S \to \varepsilon$ is allowed iff S does not appear on the right side of a rule.
- Regular languages produce words which can be recognized by FSM.
- A slighlty more algebraic perspective:
  - a mapping f from a monoid[1] (M,+,0) to a monoid $(P, \times, 1)$ is a **monoid homomorphism** if

$$\forall a, b \in M, f(a+b) = f(a) \times f(b); \quad f(0) = 1,$$

  - Every regular language can be generated as the set

$$\{f^{-1}(s), s \in S\} \subset \Sigma^\star,$$

    where $S \subset M$, M being a *finite* monoid.
- Finite languages are a subset of regular languages.

[1] a set equipped with an associate binary operation and a neutral element for that operation

Back to NLP

NLP = Type 0,1,2 or 3 language?

Language structure and meaning

- The question is open... somewhere between type-1 and type-3.
- We'll settle for type-2 for the end of the lecture.
- Our target: map the **meaning** of a sentence is onto **language** structures.
- e.g. in English, a few examples of such mappings

  {**Thing** The dog} is {**Place** in the garden}
  {**Thing** The dog} is {**Property** fierce}
  {**Action**{**Thing** The dog} is chasing {**Thing** the cat}}
  {**State**{**Thing** The dog} was sitting {**Place** in the garden} {**Time** yesterday}}
  {**Action**{**Thing** We} ran { **Path** out into the water}}
  {**Action**{**Thing** The dog} barked {**Property/Manner** loudly}}
  {**Action**{**Thing** The dog} barked {**Property/Amount** nonstop for five hours}}

Segmenting meaning: use a typology for parts of speech

- Example of such a typology

Name	Concept	Example
Noun	Names of things	boy, cat, truth
Verb	Action or state	become, hit
Pronoun	Used for noun	become, hit
Adverb	Modifies V, Adj, Adv	sadly, very
Adjective	Modifies noun	happy, clever
Conjunction	Joins things	and, but, while
Preposition	Relation of N	to, from, into
Interjection	An outcry	ouch, oh, alas, psst

- Intuitively, we use a "substitution test" to guess whether the classification makes sense

> The {**sad**, **intelligent**, **green**, **fat**, ...} one is in the corner.

Constituency

Sentences have parts, some of which appear to have subparts.
These groupings of words that go together are called constituents.

- Consider the sentences

He ran into the man with a car
He ran into {the man with a car}
He ran into {the man} {with a car}

You could not go to her party
You {could not} go to her party
You could {not go} to her party

- We need **tools** that can highlight such **constituents**.

Constituent phrases

- Constituents are named based on the **word that heads them**.
- Examples:
  - *the man from Amherst*: Noun Phrase (NP) $\rightarrow$ the head *man* is a noun
  - *extremely clever*: Adjective Phrase (AP) $\rightarrow$ the head *clever* is an adjective
  - *down the river*: Prepositional Phrase (PP) $\rightarrow$ the head *down* is a preposition
  - *killed the rabbit*: Verb Phrase (VP) $\rightarrow$ the head *killed* is a verb

- Note that a word is a constituent, albeit a little one.
- Sometimes words also act as phrases. In:
  - *Joe grew potatoes.*
  - Compared with: *The man from Amherst grew beautiful russet potatoes.*

Constituents appears in language

They appear in similar environments ($e.g.$ before a verb)

- **Kermit the frog** comes on stage
- **They** come to Massachusetts every summer
- **December twenty-sixth** comes after Christmas
- **The reason he is running for president** comes out only now.

...but not each individual word in the constituent

- **The** comes our... **is** comes out... **for** comes out...

Constituents appears in language

Constituents can be placed in different locations

- Consider the Prepositional phrase (PP): **On December twenty-sixth**
  - **On December twenty-sixth** Id like to fly to Florida.
  - Id like to fly **on December twenty-sixth** to Florida.
  - Id like to fly to Florida **on December twenty-sixth**.

- ...*but not split apart*
  - On December Id like to fly twenty-sixth to Florida.
  - On Id like to fly December twenty-sixth to Florida.

Context Free Grammars and Constituency

CFG's are the most common way of modeling constituency.

CFG = Context-Free Grammar = Phrase Structure Grammar = BNF = Backus-Naur Form

- Parenthesis: Backus-Naur is another notation for CFG.
- Here is a BN form for adresses in the US.

```
<postal-address> ::= <name-part> <street-address> <zip-part>
<name-part> ::= <personal-part> <last-name> <opt-jr-part> <EOL>
                    | <personal-part> <name-part>
<personal-part> ::= <first-name> | <initial> "."
<street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>
<zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>
<opt-jr-part> ::= "Sr." | "Jr." | <roman-numeral> | ""
```

Example of a Context Free Grammar in a Natural Language Setting

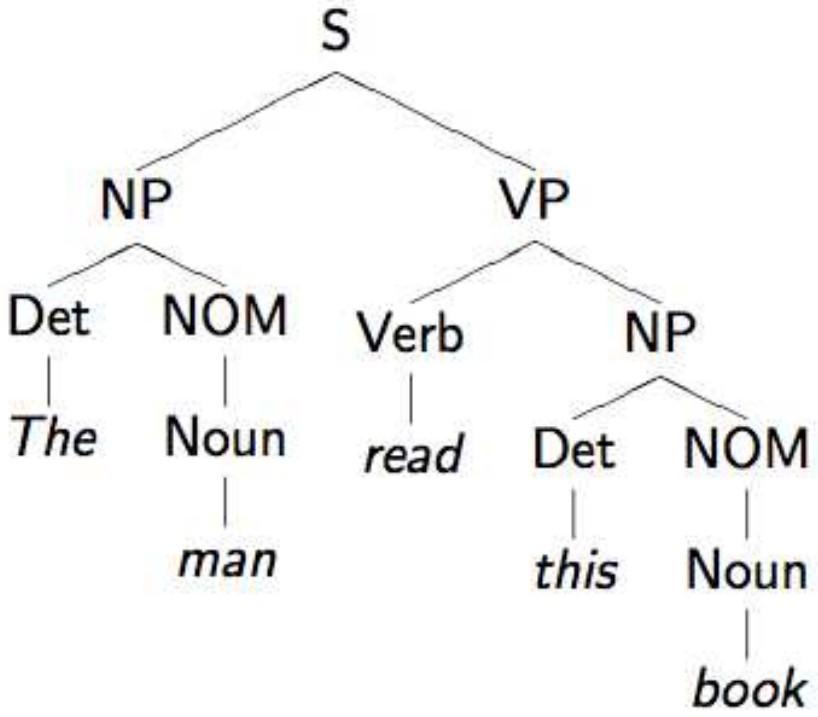
- a CFG G defined by $\{T, N, S, R\}$ where

$T =$ {that, this, a, the, man, book, flight, meal, include, read, does}
$N =$ {S, NP, NOM, VP, Det, Noun, Verb, Aux}
$S =$ S
$R =$ {
S $\rightarrow$ NP VP
S $\rightarrow$ Aux NP VP
S $\rightarrow$ VP
NP $\rightarrow$ Det NOM
NOM $\rightarrow$ Noun
NOM $\rightarrow$ Noun NOM
VP $\rightarrow$ Verb
VP $\rightarrow$ Verb NP
Det $\rightarrow$ that / this / a / the
Noun $\rightarrow$ book / flight / meal / man
Verb $\rightarrow$ book / include / read
Aux $\rightarrow$ does}

A simple application of the rules gives

$$S \rightarrow NP\ VP \rightarrow Det\ NOM\ VP \rightarrow The\ NOM\ VP$$
$$\rightarrow The\ Noun\ VP \rightarrow The\ man\ VP \rightarrow The\ man\ Verb\ NP$$
$$\rightarrow The\ man\ read\ NP \rightarrow The\ man\ read\ Det\ NOM \rightarrow The\ man\ read\ this\ NOM$$
$$\rightarrow The\ man\ read\ this\ Noun \rightarrow The\ man\ read\ this\ book$$

- The parse tree which corresponds to this example:



CFG's can capture some notion of recursion

- Example of seemingly endless recursion of embedded prepositional phrases:
  PP $\rightarrow$ Prep NP
  NP $\rightarrow$ Noun PP
- {S The mailman ate his {N P lunch {P P with his friend {P P from the cleaning staff {P P of the building {P P at the intersection {P P on the north end {P P of town}}}}}}}.
- Automaton to parse Type-2 languages: push-down automaton
- A lot of attention given to these, as well as languages that are closer to Type-1.