Formal Models of Language

Jon Dehdari

October 26, 2015

Introduction

Hi!

Formal Languages

- Once Upon a Time...

Formal Languages

- Once Upon a Time...
- Mathematicians started to think about language...

Formal Languages

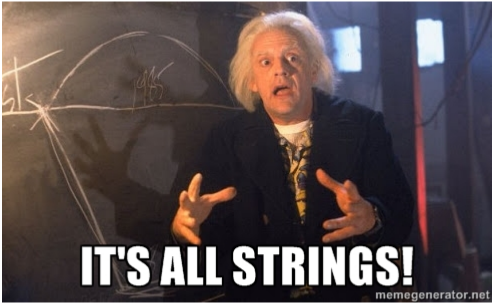
- Once Upon a Time...
- Mathematicians started to think about language...
- They used ideas from logic to represent linguistic objects...

Formal Languages

- Once Upon a Time...
- Mathematicians started to think about language...
- They used ideas from logic to represent linguistic objects...
- They had a craaaazy idea...

Formal Languages

- Once Upon a Time...
- Mathematicians started to think about language...
- They used ideas from logic to represent linguistic objects...
- They had a craaaazy idea...



Strings

What's a String?

- A **string** in this context is just a sequence of words

Strings

What's a String?

- A **string** in this context is just a sequence of words
- A **formal language** (L) is a subset of all the possible strings

Strings

What's a String?

- A **string** in this context is just a sequence of words
- A **formal language** (L) is a subset of all the possible strings
- An **vocabulary** (Σ, also sometimes called *alphabet*) here is a set of all the words in the language

Strings

What's a String?

- A **string** in this context is just a sequence of words
- A **formal language** (L) is a subset of all the possible strings
- An **vocabulary** (Σ, also sometimes called *alphabet*) here is a set of all the words in the language
- Words here don't need to correspond to words used for natural languages

Strings

What's a String?

- A **string** in this context is just a sequence of words
- A **formal language** (L) is a subset of all the possible strings
- An **vocabulary** (Σ, also sometimes called *alphabet*) here is a set of all the words in the language
- Words here don't need to correspond to words used for natural languages
- For example, this set:

  { ☺, 🌳, ☃ }

  is a perfectly valid vocabulary for a formal language.

Strings

What's a String?

- A **string** in this context is just a sequence of words
- A **formal language** (L) is a subset of all the possible strings
- An **vocabulary** (Σ, also sometimes called *alphabet*) here is a set of all the words in the language
- Words here don't need to correspond to words used for natural languages
- For example, this set:

  is a perfectly valid vocabulary for a formal language. But we usually use boring symbols like $\{$a, b, c$\}$

Strings

What's a String?

- A **string** in this context is just a sequence of words
- A **formal language** (L) is a subset of all the possible strings
- An **vocabulary** (Σ, also sometimes called *alphabet*) here is a set of all the words in the language
- Words here don't need to correspond to words used for natural languages
- For example, this set:

  {  }

  is a perfectly valid vocabulary for a formal language. But we usually use boring symbols like {a, b, c}
- (Similar to musical/poetic form analysis)

Formal Grammar

- A formal grammar is a way of telling what a valid string is in a formal language
- Formal grammars can also **generate** valid strings

Formal Grammar

- A formal grammar is a way of telling what a valid string is in a formal language
- Formal grammars can also **generate** valid strings
- If two different grammars can generate/accept the same formal languages, then they have the same **weak generative capacity**

Formal Grammar

- A formal grammar is a way of telling what a valid string is in a formal language
- Formal grammars can also **generate** valid strings
- If two different grammars can generate/accept the same formal languages, then they have the same **weak generative capacity**
- If two different grammars can generate/accept the same structures as well, then they have the same **strong generative capacity**

Formal Language Hierarchy

	Formal Language
	Non-Turing-acceptable
0:	**Recursively enumerable**
	Recursive/ Decidable
1:	Context-sensitive
	Indexed
	Mildly context-sensitive
2:	**Context-free**
	Deterministic context-free
3:	**Regular**
	Finite

Formal Language Hierarchy

	Formal Language
	Non-Turing-acceptable
0:	**Recursively enumerable**
	Recursive/ Decidable
1:	Context-sensitive
	Indexed
	Mildly context-sensitive
2:	**Context-free**
	Deterministic context-free
3:	**Regular**
	Finite

This is extended from the older *Chomsky hierarchy*.

Formal Language Hierarchy

	Formal Language
	Non-Turing-acceptable
0:	**Recursively enumerable**
	Recursive/ Decidable
1:	Context-sensitive
	Indexed
	Mildly context-sensitive
2:	**Context-free**
	Deterministic context-free
3:	**Regular**
	Finite

This is extended from the older *Chomsky hierarchy*. We'll discuss the ones in boldface, as they're relevant to natural languages.

Why is this Stuff Relevant??

- Knowing what types of formal languages a grammar/automaton can generate & accept will give you an idea of what phenomena in natural languages that they can handle

Why is this Stuff Relevant??

- Knowing what types of formal languages a grammar/automaton can generate & accept will give you an idea of what phenomena in natural languages that they can handle
- For example: long-distance dependencies, complex reordering in machine translation, reduplication, etc.

Why is this Stuff Relevant??

- Knowing what types of formal languages a grammar/automaton can generate & accept will give you an idea of what phenomena in natural languages that they can handle
- For example: long-distance dependencies, complex reordering in machine translation, reduplication, etc.
- You can also get an idea of how fast or slow it will take for a computer (or human) to process sequential stuff (like natural language!)

Finite Languages

- In a finite language, there are a finite (ie not infinite) number of valid *sentences*.
- Time: constant (through hash-table lookup)
- Memory: constant (duh)

Finite Languages

- In a finite language, there are a finite (ie not infinite) number of valid *sentences*.
- Time: constant (through hash-table lookup)
- Memory: constant (duh)
- For natural language, this would correspond to having a finite number of possible sentences

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity
- If a natural language has a vocabulary of, say, 100 million words ...

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity
- If a natural language has a vocabulary of, say, 100 million words ...
- And a sentence can have, say, up to 10,000 words in it, ...

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity
- If a natural language has a vocabulary of, say, 100 million words ...
- And a sentence can have, say, up to 10,000 words in it, ...
- Then there would be $10^{80,000}$ possible sentences

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity
- If a natural language has a vocabulary of, say, 100 million words ...
- And a sentence can have, say, up to 10,000 words in it, ...
- Then there would be $10^{80,000}$ possible sentences
- This number sounds way too big to be practical for either humans or computers to deal with!

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity
- If a natural language has a vocabulary of, say, 100 million words ...
- And a sentence can have, say, up to 10,000 words in it, ...
- Then there would be $10^{80,000}$ possible sentences
- This number sounds way too big to be practical for either humans or computers to deal with!
- But it's much smaller than infinity.

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity
- If a natural language has a vocabulary of, say, 100 million words ...
- And a sentence can have, say, up to 10,000 words in it, ...
- Then there would be $10^{80,000}$ possible sentences
- This number sounds way too big to be practical for either humans or computers to deal with!
- But it's much smaller than infinity.
- Much much smaller.

Are Natural Languages Finite??!!

- It sounds crazy to think that you could ever list all of the possible sentences of a natural language
- Really, really crazy
- But...
- There's a big difference between a really large number and infinity
- If a natural language has a vocabulary of, say, 100 million words ...
- And a sentence can have, say, up to 10,000 words in it, ...
- Then there would be $10^{80,000}$ possible sentences
- This number sounds way too big to be practical for either humans or computers to deal with!
- But it's much smaller than infinity.
- Much much smaller.
- (There's more discussion on the interwebs if you're interested)

Regular Languages

- Ok, so maybe for now it's too difficult to list all possible sentences
- Let's assume that the vocabulary (Σ) is still fixed (or finite), but we can generate an infinite number of sentences from this fixed vocab
- Regular grammars have a fixed-length history, so they're limited in the types of long-distance phenomena they can handle

Regular Languages

- Ok, so maybe for now it's too difficult to list all possible sentences
- Let's assume that the vocabulary (Σ) is still fixed (or finite), but we can generate an infinite number of sentences from this fixed vocab
- Regular grammars have a fixed-length history, so they're limited in the types of long-distance phenomena they can handle
- For example: $\mathbf{a}\ \mathbf{a}'\ \mathbf{b}\ \mathbf{b}'\ \mathbf{c}\ \mathbf{c}'$

Regular Languages

- Ok, so maybe for now it's too difficult to list all possible sentences
- Let's assume that the vocabulary (Σ) is still fixed (or finite), but we can generate an infinite number of sentences from this fixed vocab
- Regular grammars have a fixed-length history, so they're limited in the types of long-distance phenomena they can handle
- For example: $\mathbf{a}\ \mathbf{a}'\ \mathbf{b}\ \mathbf{b}'\ \mathbf{c}\ \mathbf{c}'$
- Processing regular languages can be done in linear time ($\mathcal{O}(n)$), with a constant size of memory ($\mathcal{O}(1)$)

Deterministic Context-Free Languages

- Deterministic context-free (DCF) languages include longer-distance phenomena
- DCF grammars have a full-length history, as long as there's no ambiguity (ie. it can't backtrack)

Deterministic Context-Free Languages

- Deterministic context-free (DCF) languages include longer-distance phenomena
- DCF grammars have a full-length history, as long as there's no ambiguity (ie. it can't backtrack)
- Processing DCF languages can be done in linear time ($\mathcal{O}(n)$), with linear memory usage ($\mathcal{O}(n)$)

Context-Free Languages

- Context-free languages include phenomena like center embedding
- For example: $\mathbf{a}\ \mathbf{b}\ \mathbf{c}\ \mathbf{c}'\ \mathbf{b}'\ \mathbf{a}'$

Context-Free Languages

- Context-free languages include phenomena like center embedding
- For example: $\mathbf{a}$ $\mathbf{b}$ $\mathbf{c}$ $\mathbf{c}'$ $\mathbf{b}'$ $\mathbf{a}'$
- Context-free grammars have a full-length history, and they can backtrack for ambiguous sentences

Context-Free Languages

- Context-free languages include phenomena like center embedding
- For example: **a b c c′ b′ a′**
- Context-free grammars have a full-length history, and they can backtrack for ambiguous sentences
- Processing DCF languages can be done in about cubic time ($\mathcal{O}(n^3)$), with linear memory usage ($\mathcal{O}(n)$)

Mildly Context-Sensitive Languages

- Mildly context-sensitive (MCS) languages include phenomena like reduplication and cross-serial dependencies.
- Example: $\mathbf{a}\ \mathbf{b}\ \mathbf{c}\ \mathbf{a}'\ \mathbf{b}'\ \mathbf{c}'$

Mildly Context-Sensitive Languages

- Mildly context-sensitive (MCS) languages include phenomena like reduplication and cross-serial dependencies.
- Example: **a b c a′ b′ c′**

… das	mer	d'chind	em Hans	es huus	lönd	hälfe	aastriiche
… that	we	the children-ACC	Hans-DAT	house-ACC	let	help	paint

'… that we let the children help Hans paint the house'

Mildly Context-Sensitive Languages

- Mildly context-sensitive (MCS) languages include phenomena like reduplication and cross-serial dependencies.
- Example: **a b c a′ b′ c′**

…das	mer	d'chind	em Hans	es huus	lönd	hälfe	aastriiche
…that	we	the children-ACC	Hans-DAT	house-ACC	let	help	paint

'…that we let the children help Hans paint the house'

- Processing MCS languages can be done in about $\mathcal{O}(n^6)$ time, with quadratic memory usage ($\mathcal{O}(n^2)$)

Mildly Context-Sensitive Languages

- Mildly context-sensitive (MCS) languages include phenomena like reduplication and cross-serial dependencies.
- Example: **a b c a′ b′ c′**

…das	mer	d'chind		em Hans	es huus		lönd	hälfe	aastriiche
…that	we	the	children-ACC	Hans-DAT	house-ACC	let		help	paint

'…that we let the children help Hans paint the house'

- Processing MCS languages can be done in about $\mathcal{O}(n^6)$ time, with quadratic memory usage ($\mathcal{O}(n^2)$)
- Some grammar formalisms that can handle MCS languages include:
  - Tree Adjoining Grammar (TAG)
  - Combinatory Categorial Grammar (CCG)
  - Linear Indexed Grammars (LIG)
  - Head Grammars (HG)

Recursively Enumerable Languages

- Recursively enumerable languages allow any string that a computer (or equivalent device) can generate/accept
- There's no guarantee that the computer will ever stop processing the sentence
- Essentially any word can occur in any place in the sentence

Recursively Enumerable Languages

- Recursively enumerable languages allow any string that a computer (or equivalent device) can generate/accept
- There's no guarantee that the computer will ever stop processing the sentence
- Essentially any word can occur in any place in the sentence
- Some grammar formalisms that allow recursively enumerable languages include:
  - Chomskyan grammars (due to transformations / moves)
  - Lexical Functional Grammar (LFG)
  - Head-driven Phrase Structure Grammar (HPSG)

Recursively Enumerable Languages

- Recursively enumerable languages allow any string that a computer (or equivalent device) can generate/accept
- There's no guarantee that the computer will ever stop processing the sentence
- Essentially any word can occur in any place in the sentence
- Some grammar formalisms that allow recursively enumerable languages include:
  - Chomskyan grammars (due to transformations / moves)
  - Lexical Functional Grammar (LFG)
  - Head-driven Phrase Structure Grammar (HPSG)
- Note that these grammar formalisms can place some restrictions on word order, but they still accept/generate recursively enumerable languages.

Recursively Enumerable Languages

- Recursively enumerable languages allow any string that a computer (or equivalent device) can generate/accept
- There's no guarantee that the computer will ever stop processing the sentence
- Essentially any word can occur in any place in the sentence
- Some grammar formalisms that allow recursively enumerable languages include:
    - Chomskyan grammars (due to transformations / moves)
    - Lexical Functional Grammar (LFG)
    - Head-driven Phrase Structure Grammar (HPSG)
- Note that these grammar formalisms can place some restrictions on word order, but they still accept/generate recursively enumerable languages. How is that so?

Recursively Enumerable Languages

- Recursively enumerable languages allow any string that a computer (or equivalent device) can generate/accept
- There's no guarantee that the computer will ever stop processing the sentence
- Essentially any word can occur in any place in the sentence
- Some grammar formalisms that allow recursively enumerable languages include:
  - Chomskyan grammars (due to transformations / moves)
  - Lexical Functional Grammar (LFG)
  - Head-driven Phrase Structure Grammar (HPSG)
- Note that these grammar formalisms can place some restrictions on word order, but they still accept/generate recursively enumerable languages. How is that so? Additional grammar rules can work around such restrictions to accept/generate the string.