Introduction to Formal Languages

Carl Pollard

October 27, 2011

Review of Basic Concepts

- The members of A^n are called **A-strings of length** n.
- For any $n \in \omega$, there's a bijection from A^n to $A^{(n)}$ mapping each A-string of length n to an n-tuple of elements of A.
- $A^* =_{\text{def}} \bigcup_{i \in \omega} A_i$ is the set of all A-strings.
- For nonempty finite A:
  - A^* is countably infinite
  - The set $\wp(A^*)$ of A-**languages** (i.e. sets of A-strings) is nondenumerable (in fact, equinumerous with $\wp(\omega)$).

The Monoid of A-Strings

- For any set A, A^* forms a monoid with
  - $\frown$ (**concatenation**) as the associative operation
  - ϵ_A (the null A-string) as the identity for $\frown$.
- Here if $f \in A^m$ and $g \in A^n$, $f \frown g \in A^{m+n}$ is given by
  - $(f \frown g)(i) = f(i)$ for all $i < m$; and
  - $(f \frown g)(m + i) = g(i)$ for all $i < n$.

Note 1: Usually concatenation is expressed without the "$\frown$", by mere juxtaposition; e.g. fg for $f \frown g$.

Note 2: Because concatenation is an associative operation, we can write simply fgh instead of $f(gh)$ or $(fg)h$.

The Ordered Monoid of A-Languages

For any set A, $\wp(A^*)$ forms an ordered monoid with

- **A-languages** (i.e. sets of A-strings) as the elements
- subset inclusion as the order
- **language concatenation**, written $\bullet$, as the binary operation, where for any A-languages L and M, $L \bullet M$ is the set of all strings of the form $u \frown v$ where $u \in L$ and $v \in M$
- $1_A = \{\epsilon_A\}$ as the identity for $\bullet$.

One Way to Define a Language Recursively

1. Start with:

   a. a set L_0 of A-strings (the 'lexicon') which you know you want in the language you wish to define, and

   b. a unary operation R (the 'rules') on A-languages.

2. Then define L to be $\bigcup_{n \in \omega} L_n$, where where for each $k \in \omega$, $L_{k+1} = F(L_k)$.
3. This makes sense because of RT with $X = \wp(A^*)$, $x = L_0$, and $F = R$.

Example: the Mirror Image Language (1/2)

- Intuitively $\mathrm{Mir}(A)$ is the language consisting of all strings whose "second half is the reverse of its first half".
- Using a popular informal style of recursive definition, we 'define' the language $\mathrm{Mir}(A)$ as follows:

  1. $\epsilon \in \mathrm{Mir}(A)$;
  2. If $x \in \mathrm{Mir}(A)$ and $a \in A$, then $axa \in \mathrm{Mir}(A)$;
  3. Nothing else is in $\mathrm{Mir}(A)$.

Example: the Mirror Image Language (2/2)

- Formally, this definition is justified by RT with

  - $X = \wp(A^*)$
  - $x = 1_A$
  - F is the function that maps any A-language S to

$$F(S) = \{y \in A^* \mid \exists a \exists x[(a \in A) \wedge (x \in S) \wedge (y = axa)]\}$$

- RT then guarantees the existence of a function $h\colon \omega \to \wp(A^*)$ such that:
  - $h(0) = \{\epsilon\}$
  - for every $n \in \omega$, $h(n+1) = F(h(n))$.
- Finally, we define

$$\mathrm{Mir}(A) =_{\mathrm{def}} \bigcup_{n \in \omega} h(n).$$

- Note that $h(n)$ is the set of all mirror image strings of length $2n$.

Some Teeny Languages

- For any $a \in A$, $\underline{a}$ is the singleton A-language whose only member is the string of length one a.
- 1_A is the singleton language whose only member is the null A-string ϵ.
- $\emptyset$ as always is just the empty set, but for any A we can also think of this as the A-language which contains no strings!

  An alternative notation for this language is 0_A.

New Languages from Old (1/3)

We define some operations on $\wp(A^*)$. In these definitions L and M range over A-languages.

- The **concatenation** of L and M, written $L \bullet M$, is the set of all strings of the form $u \frown v$ where $u \in L$ and $v \in M$.
- The **right residual** of L by M, written L/M, is the set of all strings u such that $u \frown v \in L$ for every $v \in M$.
- The **left residual** of L by M, written $M \backslash L$, is the set of all strings u such that $v \frown u \in L$ for every $v \in M$.

New Languages from Old (2/3)

The **Kleene closure** of L, written $\mathbf{kl}(L)$, has the following informal recursive definition:

1. (base clause) $\epsilon \in \mathbf{kl}(L)$
2. (recursion clause) if $u \in L$ and $v \in \mathbf{kl}(L)$, then $uv \in \mathbf{kl}(L)$
3. nothing else is in $\mathbf{kl}(L)$.

Intuitively: the members of $\mathbf{kl}(L)$ are the strings formed by concatenating zero or more strings of L.

New Languages from Old (3/3)

The **positive Kleene closure** of L, written $\mathbf{kl}^+(L)$, has the following informal recursive definition:

1. (base clause) If $u \in L$, then $u \in \mathbf{kl}^+(L)$
2. (recursion clause) if $u \in L$ and $v \in \mathbf{kl}^+(L)$, then $uv \in \mathbf{kl}^+(L)$
3. nothing else is in $\mathbf{kl}^+(L)$.

Intuitively: the members of $\mathbf{kl}^+(L)$ are the strings formed
by concatenating one or more strings of L.

The Set $\mathrm{Reg}(A)$ of Regular A-Languages

The following (informally) recursively defined set of languages is important in computational linguistics applications:

1. (Base clauses)

   a. For each $a \in A$, $\underline{a} \in \mathrm{Reg}(A)$
   b. $0_A \in \mathrm{Reg}(A)$
   c. $1_A \in \mathrm{Reg}(A)$

2. (Recursion clauses)

   a. for each $L \in \mathrm{Reg}(A)$, $\mathrm{kl}(L) \in \mathrm{Reg}(A)$
   b. for each $L, M \in \mathrm{Reg}(A)$, $L \cup M \in \mathrm{Reg}(A)$
   c. for each $L, M \in \mathrm{Reg}(A)$, $L \bullet M \in \mathrm{Reg}(A)$

3. nothing else is in $\mathrm{Reg}(A)$.

Context-Free Grammars (CFGs)

A CFG is an ordered quadruple $\langle T, N, D, P\rangle$ where

- T is a finite set called the **terminals**;
- N is a finite set called the **nonterminals**
- D is a finite subset of $N \times T$ called the **lexical entries**;
- P is a finite subset of $N \times N^+$ called the **phrase structure rules** (PSRs).

CFG Notation

- '$A \rightarrow t$ ' means $\langle A, t\rangle \in D$.
- '$A \rightarrow A_0 \ldots A_{n-1}$' means $\langle A, A_0 \ldots A_{n-1}\rangle \in P$.
- '$A \rightarrow \{s_0, \ldots s_{n-1}\}$' abbreviates $A \rightarrow s_i$ $(i < n)$.

A 'Toy' CFG for English (1/2)

$T = \{$**Fido**, **Felix**, **Mary**, **barked**, **bit**, **gave**, **believed**, **heard**, **the**, **cat**, **dog**, **yesterday**$\}$

$N = \{$S, NP, VP, TV, DTV, SV, Det, N, Adv$\}$

D consist of the following lexical entries:

NP $\rightarrow$ $\{$**Fido**, **Felix**, **Mary**$\}$

VP $\rightarrow$ **barked**

TV $\rightarrow$ **bit**

DTV $\rightarrow$ **gave**

SV $\rightarrow$ $\{$**believed**, **heard**$\}$

Det $\rightarrow$ **the**

N $\rightarrow$ $\{$**cat**, **dog**$\}$

Adv $\rightarrow$ **yesterday**

A 'Toy' CFG for English (2/2)

P consists of the following PSRs:

S $\rightarrow$ NP VP

VP $\rightarrow$ $\{$TV NP, DTV NP NP, SV S, VP Adv$\}$

NP $\rightarrow$ Det N

Context-Free Languages (CFLs)

- Given a CFG $\langle T, N, D, P\rangle$, we can define a function C from N to T-languages (we write C_A for $C(A)$) as described below.
- The C_A are called the **syntactic categories** of the CFG (and so a nointerminal can be thought of as a name of a syntactic category).
- A language is called **context free** if it is a syntactic category of some CFG.

Historical Notes

- Up until the mid 1980's an open research questions was whether NLs (considered as sets of word strings) were context-free languages (CFLs).
- Chomsky maintained they were not, and his invention of transformational grammar (TG) was motivated in large part by the perceived need to go beyond the expressive power of CFGs.
- Gazdar and Pullum (early 1980's) refuted all published arguments that NLs could not be CFLs.
- Together with Klein and Sag, they developed a context-free framework, generalized phrase structure grammar (GPSG), for syntactic theory.
- But in 1985, Shieber published a paper arguing that Swiss German cannot be a CFL.
- Shieber's argument is still generally accepted today.

Defining the Syntactic Categories of a CFG (1/2)

- We will recursively define a function $h : \omega \to \wp(T^*)^N$.
- Intuitively, for each nonterminal A, the sets $h(n)(A)$ are successively larger approximations of C_A.
- Then C_A is defined to be $C_A =_{\text{def}} \bigcup_{n \in \omega} h(n)(A)$.

Defining the Syntactic Categories of a CFG (2/2)

- We define h using the Recursion Theorem (RT) with X, x, F set as follows:
  - $X = \wp(T^*)^N$
  - x is the function that maps each $A \in N$ to the set of length-one strings t such that $A \to t$.
  - F is the function from X to X that maps a function $L : N \to \wp(T^*)$ to the function that maps each nonterminal A to the union of $L(A)$ with the set of all strings that can be obtained by applying a PSR $A \to A_0 \ldots A_{n-1}$ to strings $s_0, \ldots, s_{n-1}$, where, for each $i < n$, s_i belongs to $L(A_i)$. I.e. $F(L)(A) =$

    $L(A) \cup \bigcup \{L(A_0) \bullet \ldots \bullet L(A_{n-1}) \mid A \to A_0 \ldots A_{n-1}\}$.
  - Given these values of X, x, and F, the RT guarantees the existence of a unique function h from ω to functions from N to $\wp(T^*)$.

Proving that a String Belongs to a Category (1/2)

- With the C_A *formally* defined as above, the following two clauses amount to an (informal) simultaneous recursive definition of the syntactic categories:
  - (Base Clause) If $A \to t$, then $t \in C_A$.
  - (Recursion Clause) If $A \to A_0 \ \ldots A_{n-1}$ and for each $i < n$, $s_i \in C_{A_i}$, then $s_0 \ldots s_{n-1} \in C_A$.
- This in turn provides a simple-minded way to prove that a string belongs to a syntactic category (if in fact it does!).

Proving that a String Belongs to a Category (2/2)

- By way of illustration, consider the string

  $s =$ **Mary heard Fido bit Felix yesterday**.
- We can (and will) prove that $s \in C_{\mathrm{S}}$.
- But most syntacticians would say that s corresponds to *two different sentences*, one roughly paraphrasable as *Mary heard yesterday that Fido bit Felix* and another roughly paraphrasable as *Mary heard that yesterday, Fido bit Felix*.
- Of course, these two sentences mean different things; but more relevant for our present purposes is that we can also characterize the difference between the two sentences purely in terms of *two distinct ways of proving* that $s \in C_{\mathrm{S}}$.

First Proof

- From the lexicon and the base clause, we know that **Mary, Fido, Felix** $\in C_{\mathrm{NP}}$, **heard** $\in C_{\mathrm{SV}}$, **bit** $\in C_{\mathrm{TV}}$, and **yesterday** $\in C_{\mathrm{Adv}}$.
- Then, by repeated applications of the recursion clause, it follows that:
  1. since **bit** $\in C_{\mathrm{TV}}$ and **Felix** $\in C_{\mathrm{NP}}$, **bit Felix** $\in C_{\mathrm{VP}}$;
  2. since **bit Felix** $\in C_{\mathrm{VP}}$ and **yesterday** $\in C_{\mathrm{Adv}}$, **bit Felix yesterday** $\in C_{\mathrm{VP}}$;
  3. since **Fido** $\in C_{\mathrm{NP}}$ and **bit Felix yesterday** $\in C_{\mathrm{VP}}$, **Fido bit Felix yesterday** $\in C_{\mathrm{S}}$;
  4. since **heard** $\in C_{\mathrm{SV}}$ and **Fido bit Felix yesterday** $\in C_{\mathrm{S}}$, **heard Fido bit Felix yesterday** $\in CP_{\mathrm{VP}}$; and finally,
  5. since **Mary** $\in C_{\mathrm{NP}}$ and **heard Fido bit Felix yesterday** $\in C_{\mathrm{VP}}$, **Mary heard Fido bit Felix yesterday** $\in C_{\mathrm{S}}$.

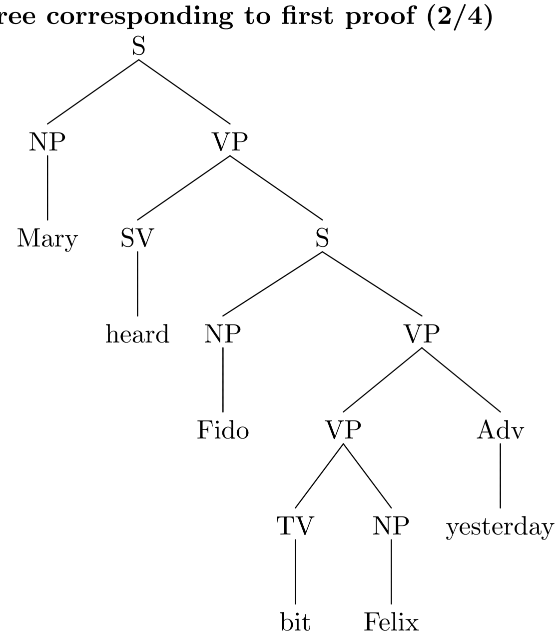
Second Proof

- Same as for first proof.
- Then, by repeated applications of the recursion clause, it follows that:
  1. since **Fido** $\in$ C_{NP} and **bit Felix** $\in$ C_{VP}, **Fido bit Felix** $\in$ C_{S};
  2. since **heard** $\in$ C_{SV} and **Fido bit Felix** $\in$ C_{S}, **heard Fido bit Felix** $\in$ C_{VP};
  3. since **heard Fido bit Felix** $\in$ C_{VP} and **yesterday** $\in$ C_{Adv}, **heard Fido bit Felix yesterday** $\in$ C_{VP}; and finally,
  4. since **Mary** $\in$ C_{NP} and **heard Fido bit Felix yesterday** $\in$ C_{VP}, **Mary heard Fido bit Felix yesterday** $\in$ C_{S}.

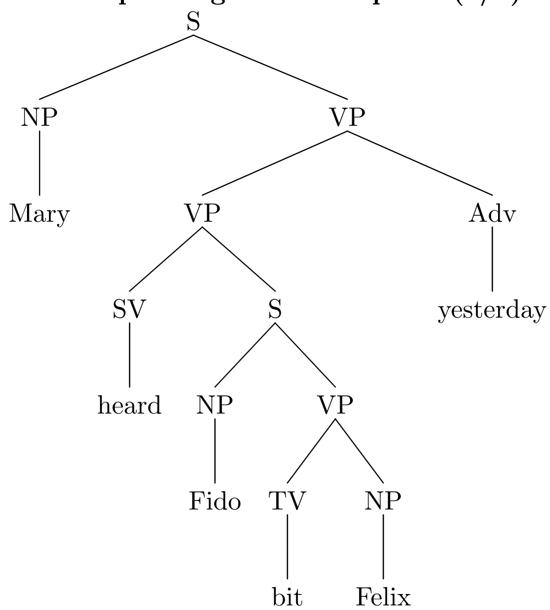
Proofs vs. Trees (1/4)

- The analysis of NL syntax in terms of proofs is characteristic of the family of theoretical approaches collectively known as **categorial grammar**, initiated by Lambek (1958).
- But the most widely practiced approaches (sometimes referred to as **mainstream generative grammar**) analyze NL syntax in terms of *trees*, which will be introduced presently.
- For now, we just note that the two proofs above would correspond in a more 'mainstream' syntactic approach to the two trees represented informally by diagrams on the next two slides.

Tree corresponding to first proof (2/4)



Tree corresponding to second proof (3/4)



Proofs vs. Trees (4/4)

- Intuitively, it seems clear that there is a close relationship between the proof-based approach and the tree-based one, but the nature of the relationship cannot be made precise till we know more about trees and about proofs.