INF4820

Formal Grammars and Parsing Strategies

Erik Velldal

University of Oslo

Oct. 13, 2009

Topics for Today

- Formal Grammars
  - Context Free Grammars
  - Treebanks
- Parsing
  - Basic strategies:
  - Bottom-Up
  - Top-Down

From Linear Order to Hierarchical Structure

- Some of the models we've looked at so far:
  - n-gram models. Purely linear and surface oriented.
  - HMMs. Adds one layer of abstraction; POS as hidden variables. Still only linear.
- Today; Formal grammar. Adds hierarchical structure.
  - In NLP, being a sub-discipline of AI, we'd like our programs to *understand* language use (on some level).
  - Finding the grammatical structure of sentences is a step towards understanding.
  - Shift focus from "sequences" to "sentences".

Why We Need Structure

Constituency

- Word tends to lump together into groups that behave like single units.
- Constituents of the same type are interchangeable in similar syntactic environments.

The decision
The controversial decision
The decision of the members
The decision of this year's Nobel committee

} *surprises most of us.*

Why We Need Structure

Constituency

- Word tends to lump together into groups that behave like single units.
- Constituents of the same type are interchangeable in similar syntactic environments.

The decision
The controversial decision
The decision of the members
The decision of this year's Nobel committee
} *surprises most of us.*

Long-Distance Dependencies

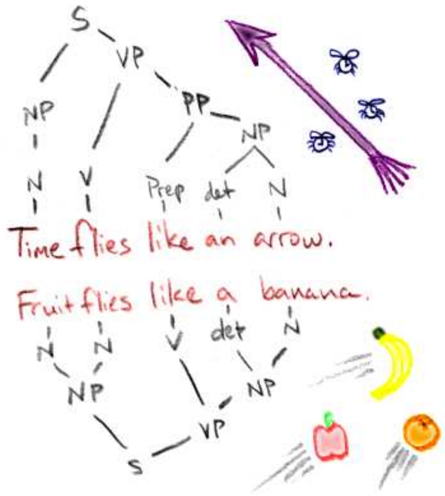
- *The decision of the Nobel committee members surprises most of us.*
- Why would a purely linear model have problems predicting this?
- Verb agreement reflects a hierarchical structure of the sentence, not just the linear order of words.

Why We Need Structure

Grammatical Roles

- *The committee gave the prize to Obama.*
- *Obama was given the prize by the committee.*
- *The prize was given to Obama by the committee.*
- Who gives what to whom?
- $give(committee, prize, obama)$

Why We Need Structure



(Courtesy of the *Speculative Grammarian*, *–the journal of satirical linguistics*.)

Context Free Grammars (CFGs)

- Phrase structure grammar
- Formal mathematical system for modeling constituent structure.
- Defined in terms of a lexicon and a set of rules
- Formal models of “language” in a broad sense
  - natural languages, programming languages, communication protocols...
  - Can be expressed in the “meta-syntax” of the Backus-Naur Form formalism.
  - When looking up macros and special forms in the Common Lisp HyperSpec, you've been reading (extended) BNF.
- Powerful enough to express sophisticated relations among words, yet in a computationally tractable way.

CFGs (Formally This Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

- C is the set of categories (aka *non-terminals*), e.g. $\{\text{S}, \text{NP}, \text{VP}, \text{V}\}$;
- Σ is the vocabulary (aka *terminals*), e.g. $\{\text{Kim}, \text{snow}, \text{saw}, \text{in}\}$;
- P is a set of category rewrite rules (aka *productions*), e.g.

$$
\begin{aligned}
&\text{S} \rightarrow \text{NP VP}\\
&\text{VP} \rightarrow \text{V NP}\\
&\text{NP} \rightarrow \text{Kim}\\
&\text{NP} \rightarrow \text{snow}\\
&\text{V} \rightarrow \text{saw}
\end{aligned}
$$

- $S \in C$ is the *start symbol*, a filter on complete ('sentential') results;
- for each rule '$\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n$' $\in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$; $1 \leq i \leq n$.

Derivation and Generation

- If we can use the rules in P to recursively rewrite S into a sequence w_i^n where each $w_i \in \Sigma$, we say that a w_i^n can be derived from S.
- Top-down view of generative grammars:
  - For a grammar G, the language $\mathcal{L}_G$ is defined as the set of strings that can be derived from S.
  - Grammatical strings $=_{def}$ strings generated by the grammar
- The "context-freeness" of CFGs refers to the fact that we rewrite non-terminals without regard to the overall context in which they occur.

Treebanks

- When training our HMM taggers we used corpus data annotated with POS.
- When a corpus is annotated with *grammatical structure*, we call it treebank.
- A treebank can define the grammar, or we can use a grammar to construct a treebank.
- Most important use: Inferring stochastic grammars, e.g. Probabilistic Context-Free Grammars (PCFGs).
  - Each production is associated with a probability.

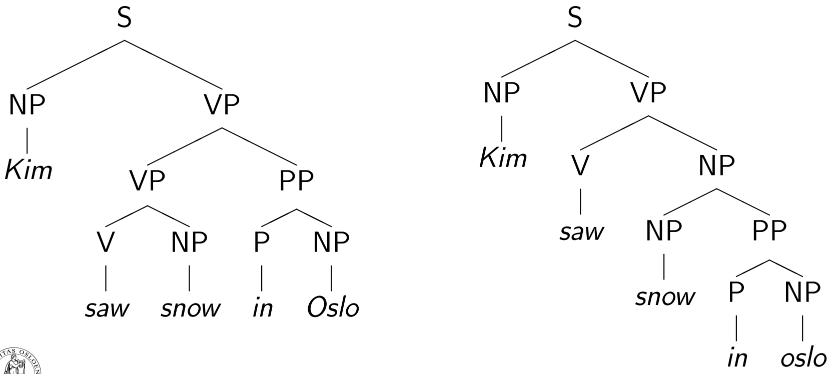
Parsing

- We now move from a declarative to a procedural view.
- Parsing = mapping a string to the derivation sequence(s) that could have generated it.
  - In parsing a sentence we attempt to recognize it wrt a grammar by assigning syntactic structure to it.
- We define the task as a search problem.
  - Find trees whose root is S and whose leafs cover exactly the words in the input.
  - Two basic constraints.

Parsing: Assigning Structure

S → NP VP
VP → V | V NP | VP PP
NP → NP PP
PP → P NP

NP → Kim | snow | Oslo
V → saw
P → in



Parsing Strategy: Top-Down

Goal-directed search

- Starting from the root S, we try to build a tree down to the leafs, matching the input.

Recursive Descent Parsing

- For a given parsing goal α, apply all rules in P where where α is the LHS;
- Successively try to expand the RHS of each rule;
  - For each β_i in the RHS of each rule, working from left to right, recursively attempt to parse β_i;
  - Termination: when α is a prefix of the input string, parsing succeeds.
- We successfully parse a string if a parsing goal S terminates consuming the full input string.
- (Note; we have implicitly formulated this as a depth-first search, using a stack and backtracking.)

Parsing Strategy: Top-Down

Advantages

- Never wastes time building trees that don't result in an S.

Disadvantages

- Wastes time on exploring trees that are inconsistent with the input.
- Duplicated effort; When backtracking we may discard parsed constituents that will need to be rebuilt again later.
  - Exponential complexity
  - Good candidate for memoization
- Doesn't terminate in the case of rules that are directly left recursive;
  - i.e. if the first symbol on the RHS is identical to the LHS non-terminal.
  - It is possible to transform the grammar to remove left recursive rules.

Parsing Strategy: Bottom-Up

Data-directed search

- Starting with the input, we try to build a tree upwards that is rooted in S and covers the entire input.

Shift-Reduce Parsing

- If a prefix of the symbols on top of the stack matches the RHS of a grammar rule, reduce the RHS of the rule to its LHS, replacing the RHS symbols on top of the stack with the non-terminal occurring on the LHS of the rule.
- If not, shift (push) the next input token onto the stack.
- We have successfully recognized a string if the stack can be reduced to the root symbol S when we get to the end of the input.
- What's missing from our formulation so far?

Parsing Strategy: Bottom-Up

Data-directed search

- Starting with the input, we try to build a tree upwards that is rooted in S and covers the entire input.

Shift-Reduce Parsing

- If a prefix of the symbols on top of the stack matches the RHS of a grammar rule, reduce the RHS of the rule to its LHS, replacing the RHS symbols on top of the stack with the non-terminal occurring on the LHS of the rule.
- If not, shift (push) the next input token onto the stack.
- We have successfully recognized a string if the stack can be reduced to the root symbol S when we get to the end of the input.
- What's missing from our formulation so far? A mechanism for backtracking to handle ambiguity and non-determinism!

Parsing Strategy: Bottom-Up

Advantages and disadvantages

- Never wastes time building trees that are not locally grounded in the input.
- Wastes time on exploring trees that cannot lead to an S (or be joined by their neighbors in an intermediate tree).
- (However, availability of partial analyses desirable for, at least, some applications.)
- Unary left-recursive rules (e.g. 'NP $\rightarrow$ NP') would still be problematic.

Next Week

- Local and global syntactic ambiguity
  - E.g. attachment ambiguity
  - Increased coverage = increased ambiguity
  - The backtracking approach is too inefficient
- Instead of just recognizing strings (accept/reject) or getting *a* parse, we would like to be able to (efficiently) extract *all* possible parses.
- Dynamic programming for more efficient parsing:
  - CKY
  - Earley
  - Chart parsing