English Syntax and Parsing

ANLP: Lecture 12

Shay Cohen

School of Informatics
University of Edinburgh

11 October 2019

Last class

- Constituents and their heads
- Context-free grammars
- Structural ambiguity

Today:

- Chomsky normal form for context-free grammars
- More on English grammar
- Agreement in context-free grammars
- If time left: a bit on parsing

Side Note: English not being Finite State

A question I was asked (paraphrase):

> *Why do we need to go through the complicated process of finding a regular language* $L = \{(\text{the N})^n\ \text{TV}^m\ \text{likes tuna fish} \mid n, m \geq 0\}$ *and intersect it with English to show we do not get a regular language? Is it not sufficient to just state that a subset of English is the language* $\{(\text{the N})^n\ \text{TV}^{n-1}\ \text{likes tuna fish} \mid n \geq 1\}$ *, which is not regular, and therefore English is not regular?*

- Hint: Is the language of all possible sequence of words Σ^* regular (finite state)?

Chomsky Normal Form

A context-free grammar is in Chomsky normal form if all productions are of the form $A \rightarrow B\ C$ or $A \rightarrow a$ where A, B, C are nonterminals in the grammar and a is a word in the grammar.

> *Disregarding the empty string, every CFG is equivalent to a grammar in Chomsky normal form (the grammars' string languages are identical)*

Why is that important?

- A normal form constrains the possible ways to represent an object
- Makes parsing efficient

Conversion to Chomsky Normal Form

- Replace all words in an RHS with a preterminal that rewrites to that word
- Break all RHSes into a sequence of RHSes with two nonterminals, possibly introducing new nonterminals:

$$S \rightarrow A_1 A_2 A_3$$

transforms into

$$S \rightarrow A_1 B$$

$$B \rightarrow A_2 A_3$$

.

Sentence types

Among the large number of constructions for English sentences, four are particularly common:

- Declarative
  I prefer a morning flight.
- Imperative
  Give me the newspaper.
- Yes-no question
  Do any of these flights have stops?
- Wh-questions
  What is your name?

Declarative

S $\rightarrow$ NP VP

- I want a flight from Ontario to Chicago.
- The flight should be eleven a.m. tomorrow.
- The return flight should leave at around seven.
- I will be back tomorrow.

Imperative

Often begin with a VP and have no subject.

S $\rightarrow$ VP

- Show the lowest fare.
- Give me Sunday's flights arriving in Las Vegas from New York City.
- List all flights between five and seven.
- Go home.

Yes-no questions

Often used to ask questions or requests.

S → Aux NP VP

- Do any of these flights have stops?
- Does American's flight eighteen twenty five serve dinner?
- Can you give me the same information for United?

Wh-subject-question

Identical to the declarative structure, except that the first NP contains a wh-word.

S → Wh-NP VP

- What airlines fly from Burbank to Denver?
- Which flights depart Burbank after noon and arrive in Denver by six?
- Whose flights serve breakfast?

Wh-non-subject-question

Wh-phrase is not the subject of the sentence, and so the sentence includes another subject with the auxiliary before the subject NP

S → Wh-NP Aux NP VP

- What flights do you have from Burbank to Tacoma?

Long-range dependencies and movement

Wh-non-subject-questions contain long-distance dependencies:

What flights do you have from Burbank to Tacoma?

Wh-NP what flights is separated from the predicate that it is related to the VP have.

Some annotations and linguistic theories (minimalism) contain a small marker called a "trace" or "empty category" that is inserted after the verb to indicate long-distance dependency.

This is to denote that the object has "moved" from the object position to the beginning of the sentence:

I have an 11am flight from Burbank to Tacoma

Noun phrases and determiners

NP → Det Nominal

NP can begin with simple determiners:

- a stop
- the flight
- this flight
- those flights
- any flights
- some flights

More complex expressions can act as determiners:

- United's flight
- United's pilot's union
- Denver's mayor's mother's cancelled flight

The determiner can be a possessive expression: Det → NP's
Determiners are not obligatory:

- I like water.
- I like apples.

NP: the Nominal

The nominal follows the Det and may contain other modifiers. In its simplest form:

Nominal → Noun

Numbers and other quantifiers:

two friends	the second leg	the next day
many flights	the last flight	one stop

Adjectives:

a first-class fare	a non-stop flight
the longest layover	the earliest flight

Adjectives can be grouped into adjective phrase (AP): the least expensive fare

NP: Nominal

The head noun can also be followed by postmodifiers:
Prepositional phrase (PP):

all flights [from Cleveland]
all flights [from Cleveland] [to Newark]
arrival [in San Jose] [before seven]
a reservation [on flight sixty two] [from Tampa] [to Montreal]

A rule to account for postnominal PPs:

Nominal → Nominal PP

NP: Nominal

Non-finite clauses:

1. Gerundive(-ing) postmodifiers - VP that begins with the gerundive (-ing) form of the verb:

   Nominal → Nominal GerundVP

   any flights [arriving after eleven]
   flights [leaving on Thursday]
2. Infinitives:

   the last flight [to arrive in Boston]
3. -ed forms:

   I need to have dinner [served]
   Which is the aircraft [used by this flight]?

NP: Nominal

Relative clauses – a clause that begins with a relative pronoun (that or who). The relative pronoun functions as the subject of the embedded verb:

a flight [that serves breakfast]
flights [that leave in the morning]
the man [who arrived late]

Nominal → Nominal RelClause
RelClause → (who|that) VP

Various postnominal modifiers can be combined:

A flight [from Phoenix to Detroit] [leaving Monday evening]
Evening flights [from Nashville] [that serve dinner]

Verb Phrase

The verb phrase consists of the verb and a number of other constituents:

VP → Verb	disappear
VP → Verb NP	prefer [a morning flight]
VP → Verb NP PP	leave [Boston] [in the morning]
VP → Verb PP	leaving [on Thursday]

More complex constituents are also possible:

Another VP:

I want [VP to fly from Milwaukee to Orlando]

Sentential complement:

I [VP [V think] [S I would like to take the early flight]]

Conjunction

Major phrase types can be combined with conjunctions like and, or, but:

I need to know [NP [NP the aircraft] and [NP the flight number]]

NP → NP and NP

The ability to form coordinate phrases through conjunctions is used to test for constituency:

I need to know the [Nom [Nom aircraft] and [Nom flight number]].

Conjunction

VP conjunctions:

What flights do you have [VP [VP leaving Denver] and [VP arriving in San Francisco]]

S conjunctions:

[S [S I'm interested in a flight from Dallas to Chicago] and [S I'm also interested in going to Baltimore]]

VP → VP and VP
S → S and S

Meta-rule: X → X and X

Any non-terminal can be conjoined with the same non-terminal to yield a constituent of the same type.

Grammars in Practice

- Read off treebanks
- May contain thousands of rules
- We will talk more about treebanks when we add *probabilities* to grammars

Agreement phenomena

In programming languages, typing rules enforce type agreement between different (often separated) constituents of a program:

```
int i=0; ...; if (i>2) ...
```

There are somewhat similar phenomena in NL: constituents of a sentence (often separated) may be constrained to agree on an attribute such as person, number, gender.

- You, I imagine, are unable to attend.
- The hills are looking lovely today, aren't they?
- He came very close to injuring himself.

Agreement in various languages

These examples illustrate that in English:

- Verbs agree in person and number with their subjects;
- Tag questions agree in person, number, tense and mode with their main statement, and have the opposite polarity.
- Reflexive pronouns follow suit in person, number and *gender*.

French has much more by way of agreement phenomena:

- Adjectives agree with their head noun in gender and number.

  Le petit chien, La petite souris, Les petites mouches
- Participles of *être* verbs agree with their subject:

  Il est arrivé, Elles sont arrivées
- Participles of other verbs agree with preceding direct objects:

  Il a vu la femme, Il l'a vue

How can we capture these kinds of constraints in a grammar?

Agreement rules: why bother?

Modelling agreement is obviously important if we want to generate grammatically correct NL text.

But even for understanding input text, agreement can be useful for resolving ambiguity.
E.g. the following sentence is ambiguous . . .

The boy who eats flies ducks.

. . . whilst the following are less so:

The boys who eat fly ducks.
The boys who eat flies duck.

Node-splitting via attributes

One solution is to refine our grammar by splitting certain non-terminals according to various *attributes*. Examples of attributes and their associated values are:

- Person: 1st, 2nd, 3rd
- Number: singular, plural
- Gender: masculine, feminine, neuter
- Case: nominative, accusative, dative, ...
- Tense: present, past, future, ...

In principle these are language-specific, though certain common patterns recur in many languages.

We can then split phrase categories as the language demands, e.g.

- Split NP on person, number, case (e.g. NP[3,sg,nom]),
- Split VP on person, number, tense (e.g. VP[3,sg,fut]).

Parameterized CFG productions

We can often use such attributes to enforce agreement constraints. This works because of the head phrase structure typical of NLs. E.g. we may write parameterized rules such as:

$$\begin{aligned} \text{S} &\rightarrow \text{NP[p,n,nom] VP[p,n]} \\ \text{NP[3,n,c]} &\rightarrow \text{Det[n] Nom[n]} \end{aligned}$$

Each of these really abbreviates a finite number of rules obtained by specializing the attribute variables. (Still a CFG!)
When specializing, each variable must take the same value everywhere, e.g.

$$\begin{aligned} \text{S} &\rightarrow \text{NP[3,sg,nom] VP[3,sg]} \\ \text{S} &\rightarrow \text{NP[1,pl,nom] VP[1,pl]} \\ \text{NP[3,pl,acc]} &\rightarrow \text{Det[pl] Nom[pl]} \end{aligned}$$

Parsing algorithms can be adapted to work with this machinery: don't have to 'build' all the specialized rules individually.

Example: subject-verb agreement in English

$$\begin{aligned} \text{S} &\rightarrow \text{NP[p,n,nom] VP[p,n]} \\ \text{NP[p,n,c]} &\rightarrow \text{Pro[p,n,c]} \\ \text{Pro[1,sg,nom]} &\rightarrow \textit{I}\text{, etc.} \\ \text{Pro[1,sg,acc]} &\rightarrow \textit{me}\text{, etc.} \\ \text{NP[3,n,c]} &\rightarrow \text{Det[n] Nom[n] RelOpt[n]} \\ \text{Nom[n]} &\rightarrow \text{N[n]} \mid \text{Adj Nom[n]} \\ \text{N[sg]} &\rightarrow \textit{person}\text{, etc.} \\ \text{N[pl]} &\rightarrow \textit{people}\text{, etc.} \\ \text{RelOpt[n]} &\rightarrow \epsilon \mid \textit{who}\ \text{VP[3,n]} \\ \text{VP[p,n]} &\rightarrow \text{VV[p,n] NP[p',n',acc]} \\ \text{VV[p,n]} &\rightarrow \text{V[p,n]} \mid \text{BE[p,n] VG} \\ \text{V[3,sg]} &\rightarrow \textit{teaches}\text{, etc.} \\ \text{BE[p,n]} &\rightarrow \textit{is}\text{, etc.} \\ \text{VG} &\rightarrow \textit{teaching}\text{, etc.} \end{aligned}$$

(Other rules omitted.)

Example: subject-verb agreement in English

$$\begin{aligned} \text{S} &\rightarrow \text{NP[p,n,nom] VP[p,n]} \\ \text{NP[p,n,c]} &\rightarrow \text{Pro[p,n,c]} \\ \text{Pro[1,sg,nom]} &\rightarrow \textit{I}\text{, etc.} \\ \text{Pro[1,sg,acc]} &\rightarrow \textit{me}\text{, etc.} \\ \text{NP[3,n,c]} &\rightarrow \text{Det[n] Nom[n] RelOpt[n]} \\ \text{Nom[n]} &\rightarrow \text{N[n]} \mid \text{Adj Nom[n]} \\ \text{N[sg]} &\rightarrow \textit{person}\text{, etc.} \\ \text{N[pl]} &\rightarrow \textit{people}\text{, etc.} \\ \text{RelOpt[n]} &\rightarrow \epsilon \mid \textit{who}\ \text{VP[3,n]} \\ \text{VP[p,n]} &\rightarrow \text{VV[p,n] NP[p',n',acc]} \\ \text{VV[p,n]} &\rightarrow \text{V[p,n]} \mid \text{BE[p,n] VG} \\ \text{V[3,sg]} &\rightarrow \textit{teaches}\text{, etc.} \\ \text{BE[p,n]} &\rightarrow \textit{is}\text{, etc.} \\ \text{VG} &\rightarrow \textit{teaching}\text{, etc.} \end{aligned}$$

Recap: Syntax

Two reasons to care about syntactic structure (parse tree):

- As a guide to the semantic interpretation of the sentence
- As a way to prove whether a sentence is grammatical or not

But having a grammar isn't enough.

We also need a parsing algorithm to compute the parse tree for a given input string and grammar.

Parsing algorithms

Goal: compute the structure(s) for an input string given a grammar.

- As usual, ambiguity is a huge problem.
  - For correctness: need to find the right structure to get the right meaning.
  - For efficiency: searching all possible structures can be very slow; want to use parsing for large-scale language tasks (e.g., used to create Google's "infoboxes").

Global and local ambiguity

- We've already seen examples of global ambiguity: multiple analyses for a full sentence, like I saw the man with the telescope
- But local ambiguity is also a big problem: multiple analyses for parts of sentence.
  - the dog bit the child: first three words could be NP (but aren't).
  - Building useless partial structures wastes time.
  - Avoiding useless computation is a major issue in parsing.
- Syntactic ambiguity is rampant; humans usually don't even notice because we are good at using context/semantics to disambiguate.

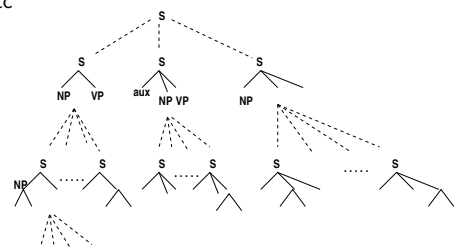
Parser properties

All parsers have two fundamental properties:

- Directionality: the sequence in which the structures are constructed.
  - top-down: start with root category (S), choose expansions, build down to words.
  - bottom-up: build subtrees over words, build up to S.
  - Mixed strategies also possible (e.g., left corner parsers)
- Search strategy: the order in which the search space of possible analyses is explored.

Example: search space for top-down parser

- Start with S node.
- Choose one of many possible expansions.
- Each of which has children with many possible expansions...
- etc



Search strategies

- depth-first search: explore one branch of the search space at a time, as far as possible. If this branch is a dead-end, parser needs to backtrack.
- breadth-first search: expand all possible branches in parallel (or simulated parallel). Requires storing many incomplete parses in memory at once.
- best-first search: score each partial parse and pursue the highest-scoring options first. (Will get back to this when discussing statistical parsing.)

Recursive Descent Parsing

- A recursive descent parser treats a grammar as a specification of how to break down a top-level goal (find `S`) into subgoals (find `NP VP`).
- It is a **top-down**, **depth-first** parser:
  - Blindly expand nonterminals until reaching a terminal (word).
  - If multiple options available, choose one but store current state as a backtrack point (in a **stack** to ensure depth-first.)
  - If terminal matches next input word, continue; else, backtrack.

RD Parsing algorithm

Start with subgoal = `S`, then repeat until input/subgoals are empty:

- If first subgoal in list is a **non-terminal** `A`, then pick an expansion `A` $\rightarrow$ `B C` from grammar and replace `A` in subgoal list with `B C`
- If first subgoal in list is a **terminal** `w`:
  - If input is empty, backtrack.
  - If next input word is different from `w`, backtrack.
  - If next input word is `w`, match! i.e., consume input word `w` and subgoal `w` and move to next subgoal.

If we run out of backtrack points but not input, no parse is possible.

Recursive descent parsing pseudocode

In the background: a CFG G, a sentence $x_1 \cdots x_n$

Function RecursiveDescent(t, v, i) where

- t is a partially constructed tree
- v is a node in t
- i is a sentence position
- Let N be the nonterminal in v
- For each rule with LHS N:
  - If the rule is a lexical rule $N \rightarrow w$, check whether $x_i = w$, if so increase i by 1 and call RecursiveDescent(t, u, $i+1$) where u is the lowest point above v that has a nonterminal
  - If the rule is a grammatical rule, Let t' be t with v expanded using the rule $N \rightarrow A_1 \cdots A_n$. For each $j \in \{1 \cdots n\}$, call RecursiveDescent(t', u_j, i) where u_j is the node for nonterminal A_j in t'.

Start with: RecursiveDescent(S, topnode, 1)

Quick quiz: this algorithm has a bug. Where? What do we need to add?

Recursive descent example

Consider a very simple example:

- Grammar contains only these rules:

S → NP VP	VP → V	NN → bit	V → bit
NP → DT NN	DT → the	NN → dog	V → dog

- The input sequence is the dog bit

Recursive descent example

- Operations:
  - Expand (E)
  - Match (M)
  - Backtrack to step n (Bn)

Step	Op.	Subgoals	Input
0		S	the dog bit
1	E	NP VP	the dog bit
2	E	DT NN VP	the dog bit
3	E	the NN VP	the dog bit
4	M	NN VP	dog bit
5	E	bit VP	dog bit
6	B4	NN VP	dog bit
7	E	dog VP	dog bit
8	M	VP	bit
9	E	V	bit
10	E	bit	bit
11	M		

Recursive Descent Parsing

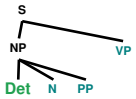
S

the dog saw a man in the park

Recursive Descent Parsing

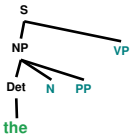
the dog saw a man in the park

Recursive Descent Parsing



the dog saw a man in the park

Recursive Descent Parsing

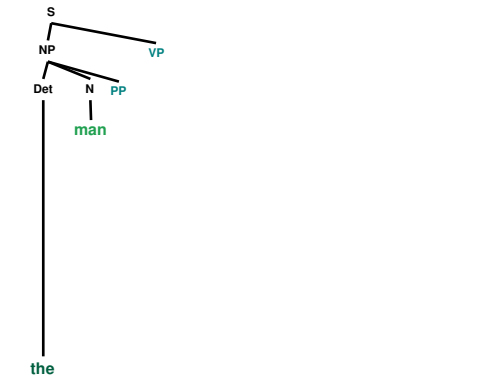


the dog saw a man in the park

Recursive Descent Parsing

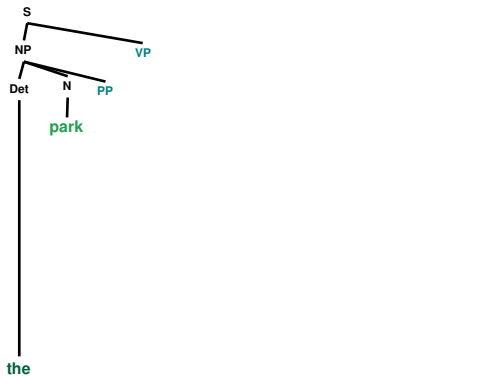
the dog saw a man in the park

Recursive Descent Parsing



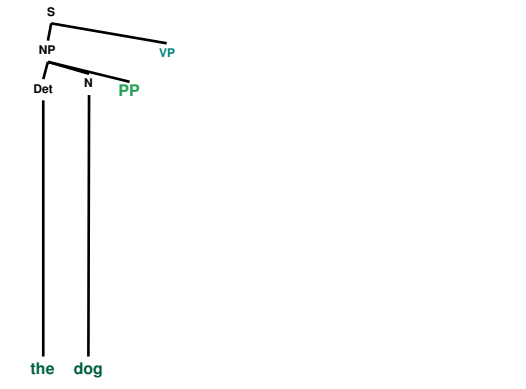
the dog saw a man in the park

Recursive Descent Parsing



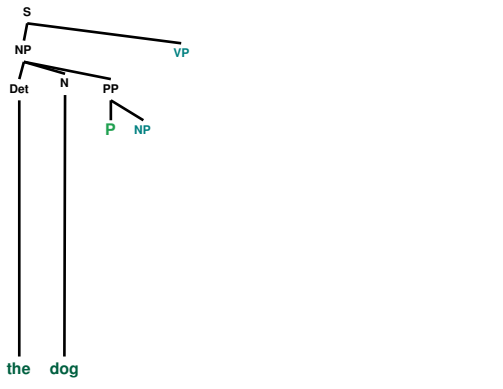
the dog saw a man in the park

Recursive Descent Parsing



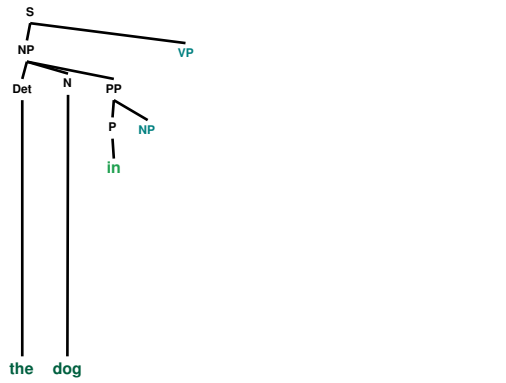
the dog saw a man in the park

Recursive Descent Parsing



the dog saw a man in the park

Recursive Descent Parsing

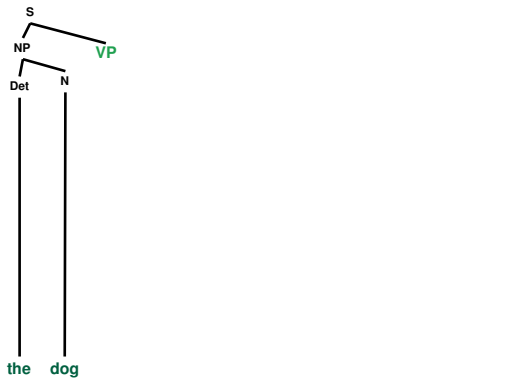


the dog saw a man in the park

Recursive Descent Parsing

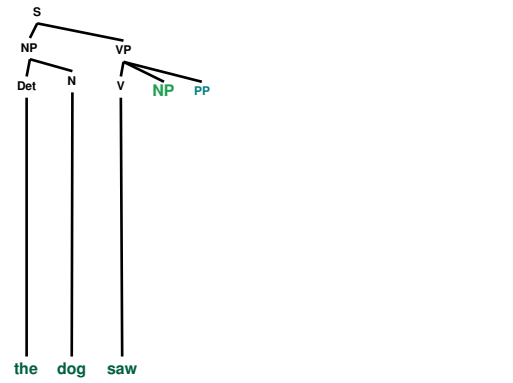
the dog saw a man in the park

Recursive Descent Parsing



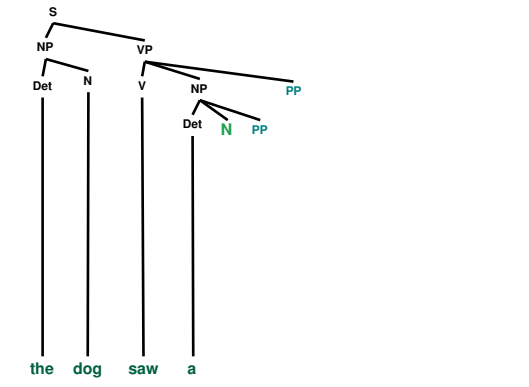
the dog saw a man in the park

Recursive Descent Parsing



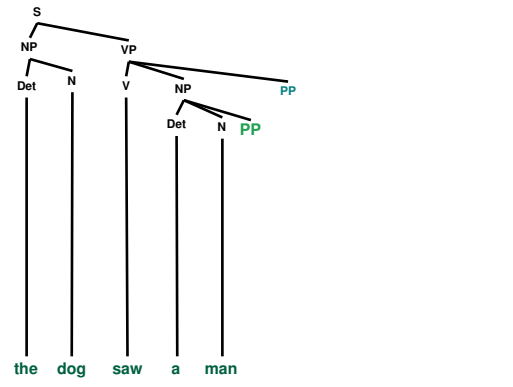
the dog saw a man in the park

Recursive Descent Parsing



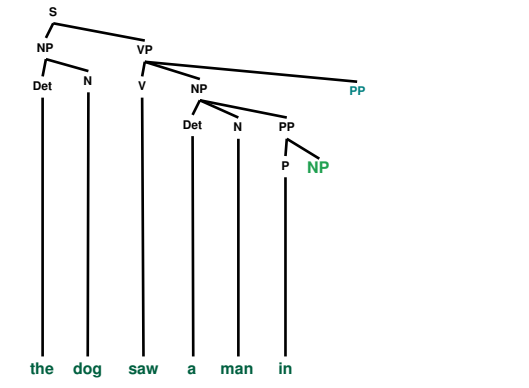
the dog saw a man in the park

Recursive Descent Parsing



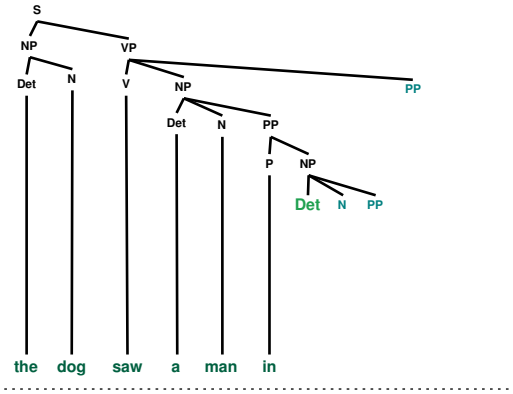
the dog saw a man in the park

Recursive Descent Parsing



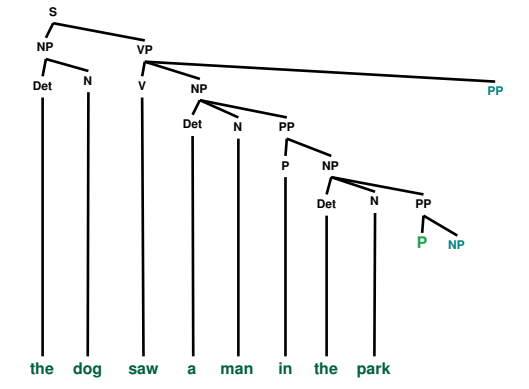
the dog saw a man in the park

Recursive Descent Parsing



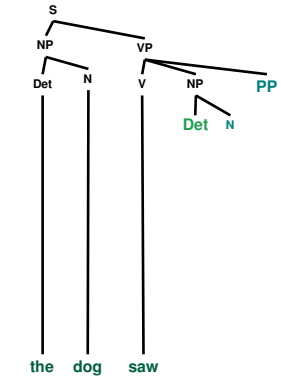
the dog saw a man in the park

Recursive Descent Parsing



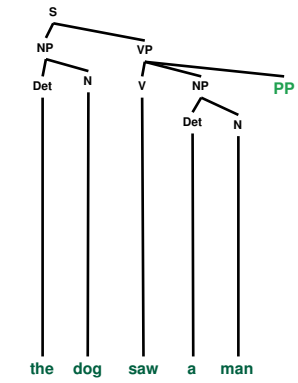
the dog saw a man in the park

Recursive Descent Parsing



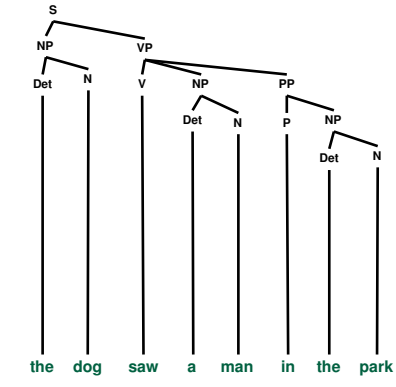
the dog saw a man in the park

Recursive Descent Parsing



the dog saw a man in the park

Recursive Descent Parsing



the dog saw a man in the park

Further notes

- The above sketch is actually a recognizer: it tells us whether the sentence has a valid parse, but not what the parse is. For a parser, we'd need more details to store the structure as it is built.
- We only had one backtrack, but in general things can be much worse!
  - If we have left-recursive rules like `NP` $\rightarrow$ `NP PP`, we get an infinite loop!

Questions to Ask Yourselves

- Can we hand-write a grammar to cover all of English? What are the alternatives?
- What is the complexity of recursive descent parsing?
- Is a recursive descent parser guaranteed to terminate on every grammar and string to begin with?