Phrase Structure and Parsing as Search

Informatics 2A: Lecture 16

Mirella Lapata

School of Informatics
University of Edinburgh

25 October 2011

Heads and Phrases

Noun (N): Noun Phrase (NP)
Adjective (A): Adjective Phrase (AP)
Verb (V): Verb Phrase (VP)
Preposition (P): Prepositional Phrase (PP)

- So far we have looked at terminal symbols.
- Today, we'll look at non-terminals, which are called phrases.
- The class that a word belongs to is closely linked to the name of the phrase it customarily appears in
- In a X-phrase (eg NP), the key occurrence of X (eg N) is called the head.
- In English, the head tends to appear in the middle of a phrase.

Heads and Phrases

English NPs are commonly of the form:

(Det) Adj* **Noun** (PP | RelClause)*
NP: *an angry duck that tried to bite me*, head: *duck*.

VPs are commonly of the form:

(Aux) Adv* **Verb** Arg* Adjunct*
Arg $\rightarrow$ NP | PP
Adjunct $\rightarrow$ PP | AdvP | ...
VP: *would usually eat artichokes for dinner*, head: *eat*.

In Japanese, Korean, Hindi, Urdu, and other head-final languages, the head is at the end of its associated phrase.

In Irish, Welsh, Scots Gaelic and other head-initial languages, the head is at the beginning of its associated phrase.

Desirable Properties of a Grammar

Chomsky specified two properties that make a grammar "interesting and satisfying":

- It should be a finite specification of the strings of the language, rather than a list of its sentences.
- It should be revealing, in allowing strings to be associated with meaning (semantics) in a systematic way.

We can add another desirable property:

- It should capture structural and distributional properties of the language. (E.g. where heads of phrases are located; how a sentence transforms into a question; which phrases can float around the sentence.)

Desirable Properties of a Grammar

- Context-free grammars (CFGs) provide a pretty good approximation.
- Some features of NLs are more easily captured using mildly context-sensitive grammars, as well see later in the course.
- There are also more modern grammar formalisms that better capture structural and distributional properties of human languages. (E.g. combinatory categorial grammar.)

A Tiny Fragment of English

Let's say we want to capture in a grammar the structural and distributional properties that give rise to sentences like:

A duck walked in the park.	**PP, NP, NP, verb**
The man walked with a duck.	**PP, NP, NP, verb**
You made a duck.	**PP, NP, NP, verb**
You made her duck.	**PP, NP, NP, verb**
A man with a telescope saw you.	**PP, NP, NP, verb**
A man saw you with a telescope.	**PP, NP, NP, verb**
You saw a man with a telescope.	**PP, NP, NP, verb**

We want to write grammatical rules that describe the phrases and lexical rules that describe what words appear in them.

Grammar for the Tiny Fragment of English

Grammar G1 generates the sentences on the previous slide:

Grammatical rules	**Lexical rules**
S → NP VP	Det → a | the | her (determiner)
NP → Det N	N → man | park | duck | telescope (noun)
NP → Det N PP	Pro → you (pronoun)
NP → Pro	V → saw | walked | made (verb)
VP → V NP PP	Prep → in | with | for (preposition)
VP → V NP	
VP → V	
PP → Prep NP	

Does G1 produce a finite or an infinite number of sentences?

Recursion

Recursion in a grammar makes it possible to generate an infinite number of sentences.

In direct recursion, a non-terminal on the LHS of a rule also appears on its RHS. The following rules add direct recursion to G1:

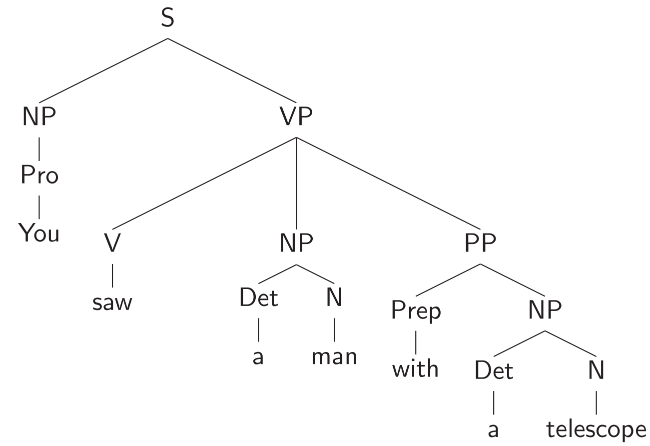
VP $\longrightarrow$ VP Conj VP
Conj $\longrightarrow$ and | or

In indirect recursion, some non-terminal can be expanded (via several steps) to a sequence of symbols containing that non-terminal:

NP $\longrightarrow$ Det N PP
PP $\longrightarrow$ Prep NP

Structural Ambiguity

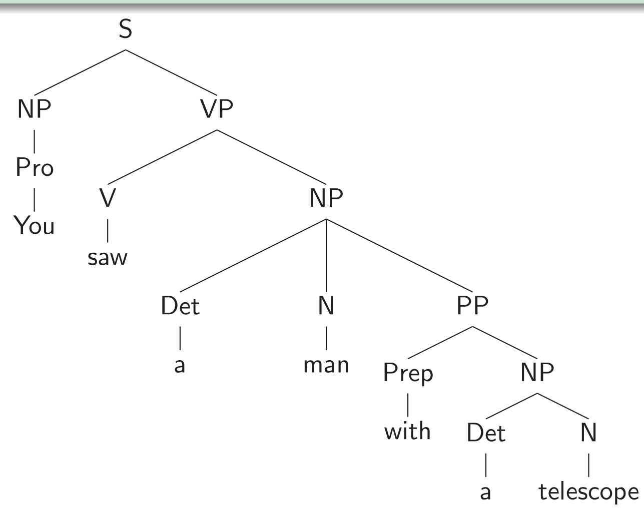
You saw a man with a telescope.



Phrase Structure
Grammars and Parsing
Recursion
Structural Ambiguity
Recursive Descent Parsing
Shift-Reduce Parsing

Structural Ambiguity

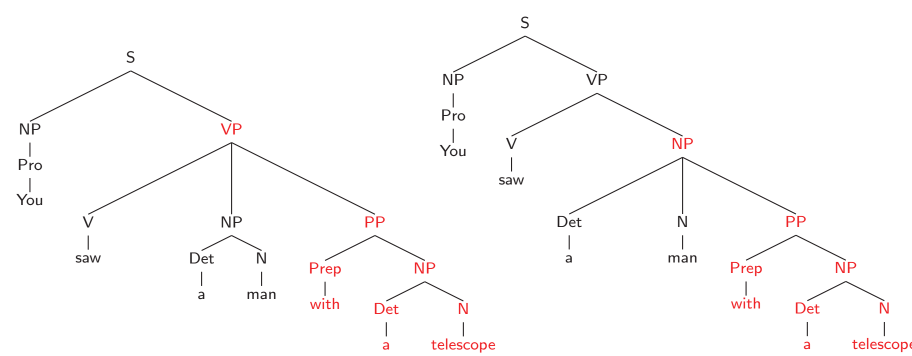
You saw a man with a telescope.



Phrase Structure
Grammars and Parsing
Recursion
Structural Ambiguity
Recursive Descent Parsing
Shift-Reduce Parsing

Structural Ambiguity

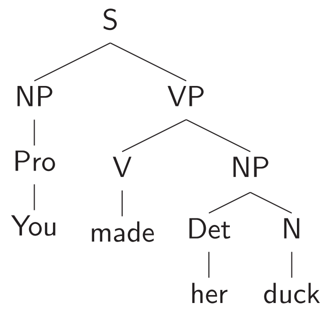
You saw a man with a telescope.



The trees illustrate attachment ambiguity: the PP can be a part of the VP or of the NP.

Structural Ambiguity

Grammar G1 only gives us one analysis of *you made her duck*.

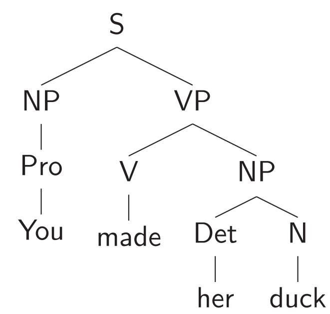


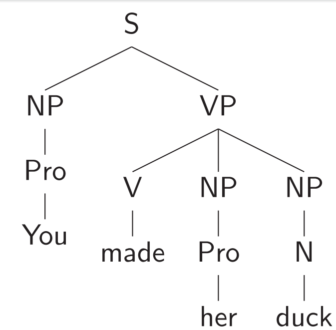
What did you make for her?
You made her duck.

Structural Ambiguity

For this, G1 also needs rules like:

NP → N
VP → V NP NP
Pro → her





Structural ambiguity is caused by part of speech ambiguity.

Structural Ambiguity

There is a third analysis as well, one that underlies the pair:

What did you make her do?
You made her duck. (move head or body quickly downwards)

Here, the small clause (*her duck*) is the direct object of a verb.

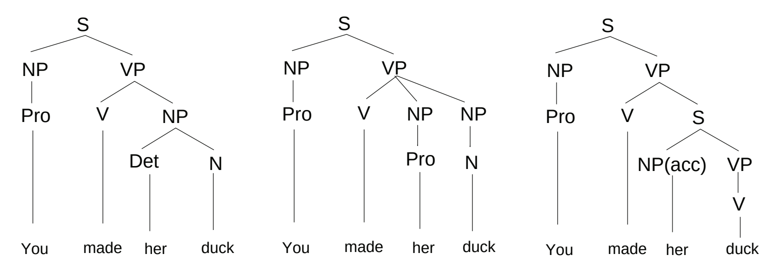
Similar small clauses are possible with verbs like *see*, *hear* and *notice*, but not *ask*, *want*, *persuade*, etc.

G1 needs a rule that requires accusative case-marking on the subject of a small clause and no tense on its verb.:

VP → V S1
S1 → NP(acc) VP(untensed)
NP(acc) → her | him | them

Structural Ambiguity

Now we have three analyses for *you made her duck:*



How can we compute these analyses automatically?

Clicker Question (1)

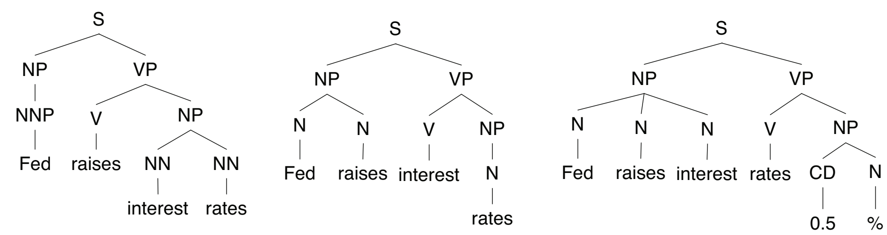
What are all the possible POS tags for the sentence *Fed raises interest rates*?

1. *Fed* is a verb, *raises* is a noun, *interest* is a verb and *rates* is a proper noun.
2. *Fed* is a proper noun, *raises* is a noun or a verb, *interest* is a verb and *rates* is a noun.
3. *Fed* is a proper noun, *raises* is a verb, or a *noun*, *interest* is a verb or a noun and *rates* is a verb or a noun.
4. *Fed* is a proper noun, *raises* is a noun or a verb, *interest* is a verb or a noun and *rates* is a noun.

Clicker Question (2)

How many syntactic trees does the sentence *Fed raises interest rates* give rise to?

1. 1
2. 2
3. 3
4. 4



Parsing Algorithms

A parser is an algorithm that computes a structure for an input string given a grammar. All parsers have two fundamental properties:

- Directionality: the sequence in which the structures are constructed (e.g., top-down or bottom-up).
- Search strategy: the order in which the search space of possible analyses is explored (e.g., depth-first, breadth-first).

Coming up soon: A zoo of parsing algorithms

Have already seen LL(1) parsing. We shall also look at:

- Recursive descent parsers (top-down). Simple and very general, but inefficient. Other problems
- Shift-reduce parsers (bottom-up).
- The Cocke-Younger-Kasami algorithm (bottom up). Works for any CFG with reasonable efficiency.
- The Early algorithm (top down). Chart parsing enhanced with prediction.

Recursive Descent Parsing

A recursive descent parser treats a grammar as a specification of how to break down a top-level goal into subgoals. Therefore:

- Parser searches through the trees licensed by the grammar to find the one that has the required sentence along its yield.
- **Directionality** = top-down: It starts from the start symbol of the grammar, and works its way down to the terminals.
- **Search strategy** = depth-first: It expands a given terminal as far as possible before proceeding to the next one.

Algorithm Sketch: Recursive Descent Parsing

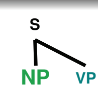
1. The top-level goal is to derive the start symbol (S).
2. Choose a grammatical rule with S as its LHS (e.g, S $\longrightarrow$ NP VP), and replace S with the RHS of the rule (the subgoals; e.g., NP and VP).
3. Choose a rule with the leftmost subgoal as its LHS (e.g., NP $\longrightarrow$ Det N). Replace the subgoal with the RHS of the rule.
4. Whenever you reach a lexical rule (e.g., Det $\longrightarrow$ the), match its RHS against the current position in the input string.
   - If it matches, move on to next position in the input.
   - If it doesn't, try next lexical rule with the same LHS.
   - If no rules with same LHS, backtrack to most recent choice of grammatical rule and choose another rule with the same LHS.
   - If no more grammatical rules, back up to the previous subgoal.
5. Iterate until the whole input string is consumed, or you fail to match one of the positions in the input. Backtrack on failure.

Recursive Descent Parsing

S

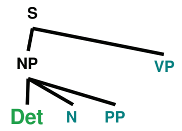
the dog saw a man in the park

Recursive Descent Parsing



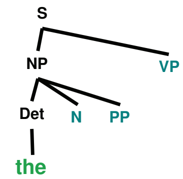
the dog saw a man in the park

Recursive Descent Parsing



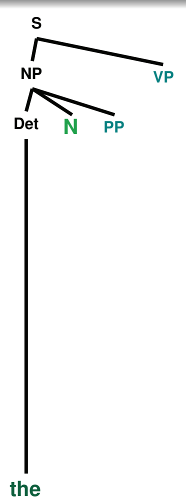
the dog saw a man in the park

Recursive Descent Parsing



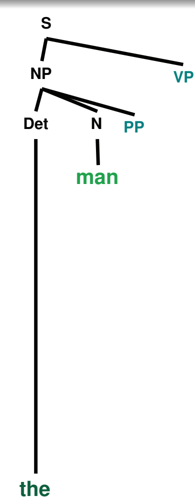
the dog saw a man in the park

Recursive Descent Parsing



the dog saw a man in the park

Recursive Descent Parsing

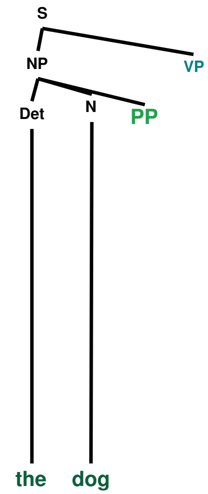


the dog saw a man in the park

Recursive Descent Parsing

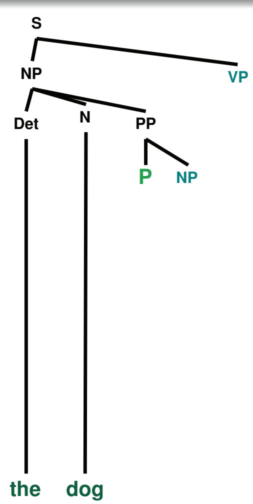
the dog saw a man in the park

Recursive Descent Parsing



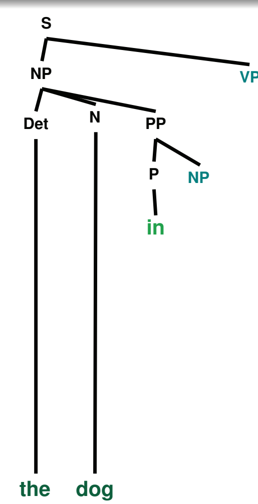
the dog saw a man in the park

Recursive Descent Parsing



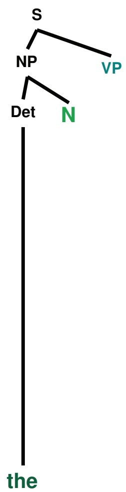
the dog saw a man in the park

Recursive Descent Parsing



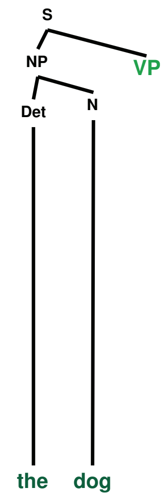
the dog saw a man in the park

Recursive Descent Parsing



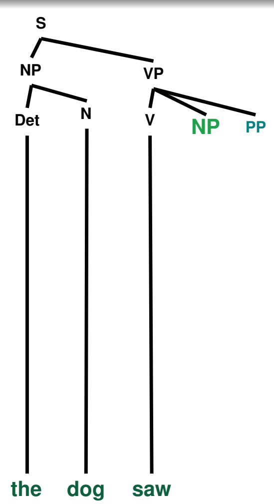
the dog saw a man in the park

Recursive Descent Parsing



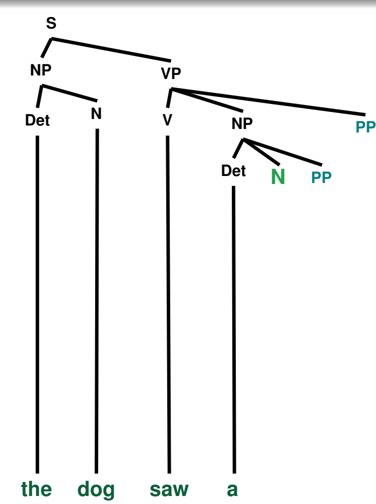
the dog saw a man in the park

Recursive Descent Parsing



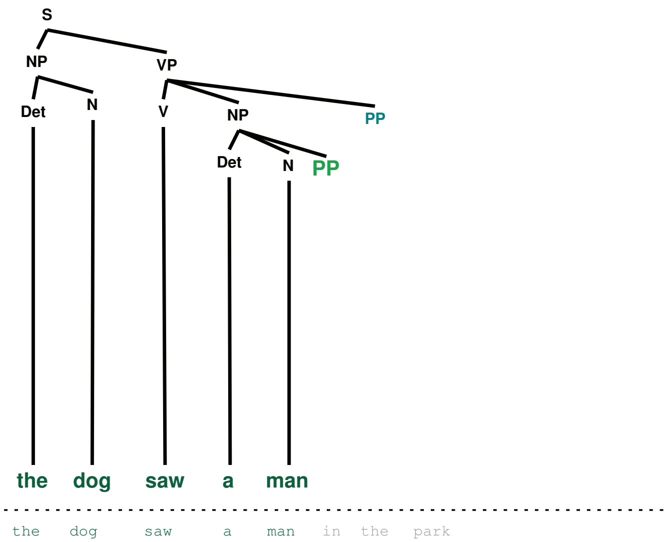
the dog saw a man in the park

Recursive Descent Parsing

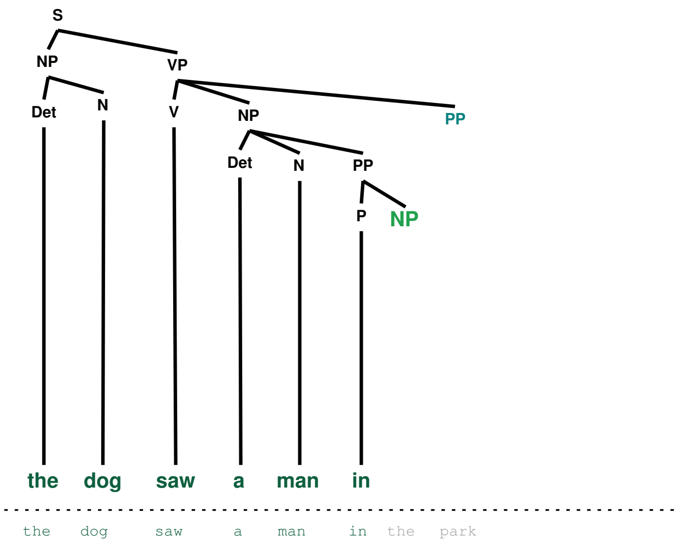


the dog saw a man in the park

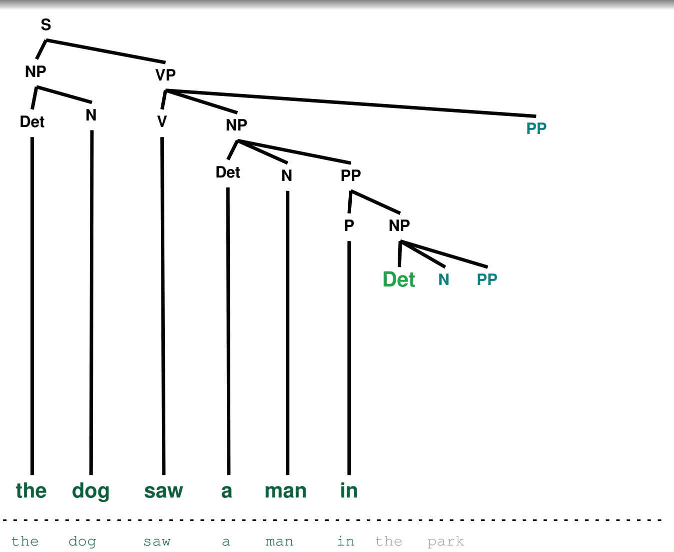
Recursive Descent Parsing



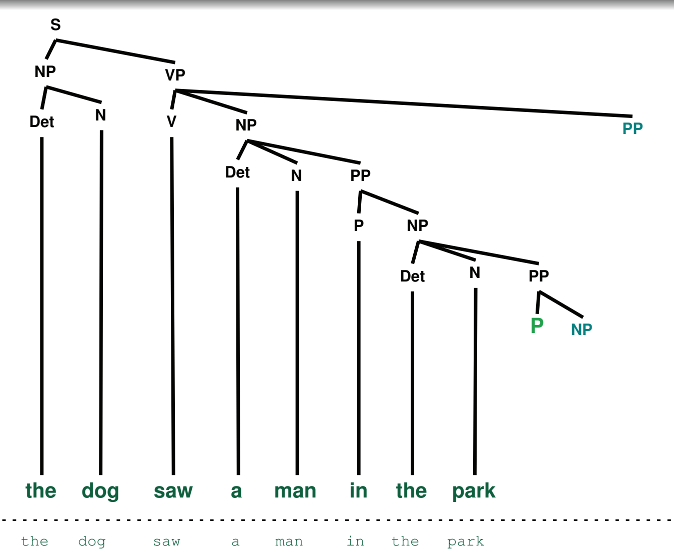
Recursive Descent Parsing



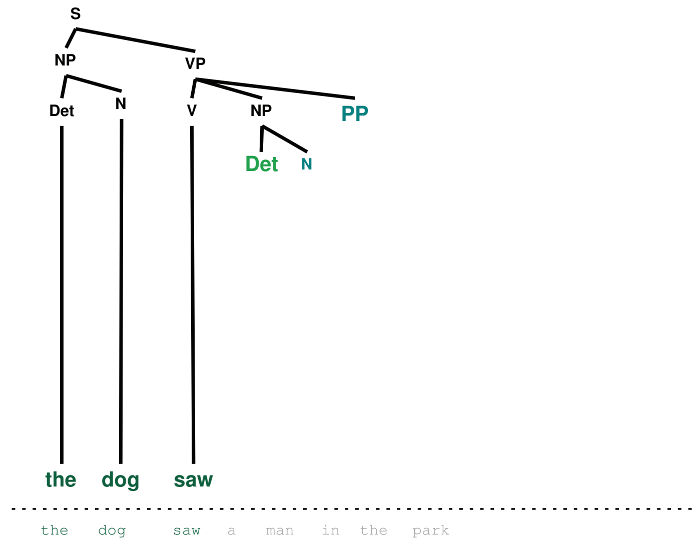
Recursive Descent Parsing



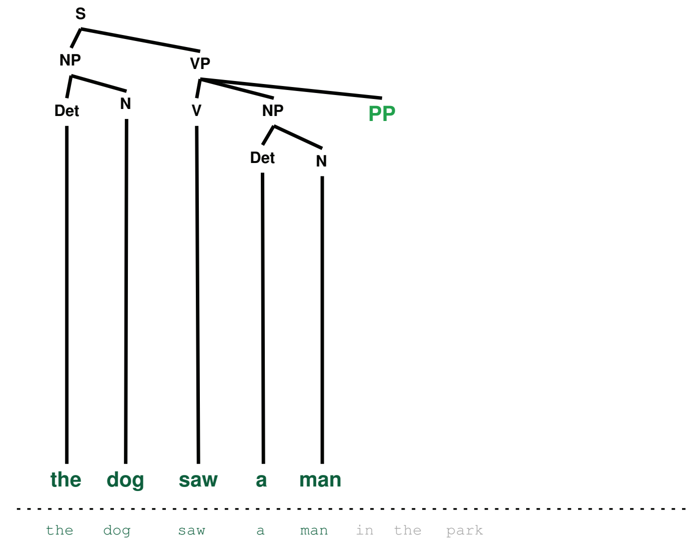
Recursive Descent Parsing



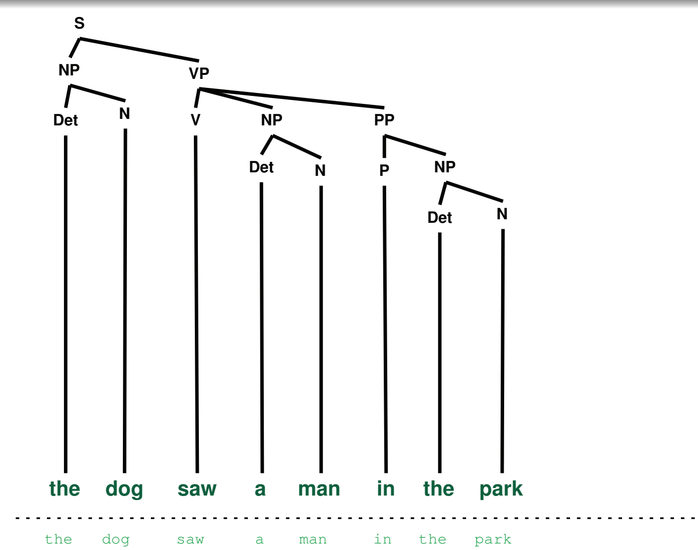
Recursive Descent Parsing



Recursive Descent Parsing



Recursive Descent Parsing



Shift-Reduce Parsing

A Shift-Reduce parser tries to find sequences of words and phrases that correspond to the righthand side of a grammar production and replace them with the lefthand side:

- **Directionality** = bottom-up: starts with the words of the input and tires to build trees from the words up.
- **Search strategy** = breadth-first: starts with the words, then applies rules with matching right handsides, and so on until the whole sentence is reduced to an S.

Algorithm Sketch: Shift-Reduce Parsing

Until the words in the sentences are substituted with S:

- Scan through the input until we recognise something that corresponds to the RHS of one of the production rules (shift)
- Apply a production rule in reverse; i.e., replace the RHS of the rule which appears in the sentential form with the LHS of the rule (reduce)

A shift-reduce parser implemented using a stack:

1. start with an empty stack
2. a shift action pushes the current input symbol onto the stack
3. a reduce action replaces *n* items with a single item

Shift-Reduce Parsing

Stack	Remaining
	my dog saw a man in the park

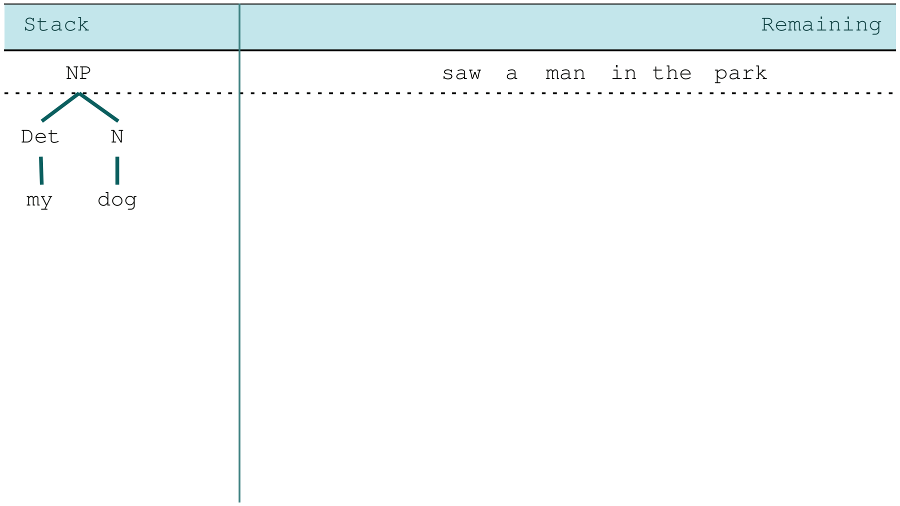
Shift-Reduce Parsing

Stack	Remaining
Det	dog saw a man in the park
my	

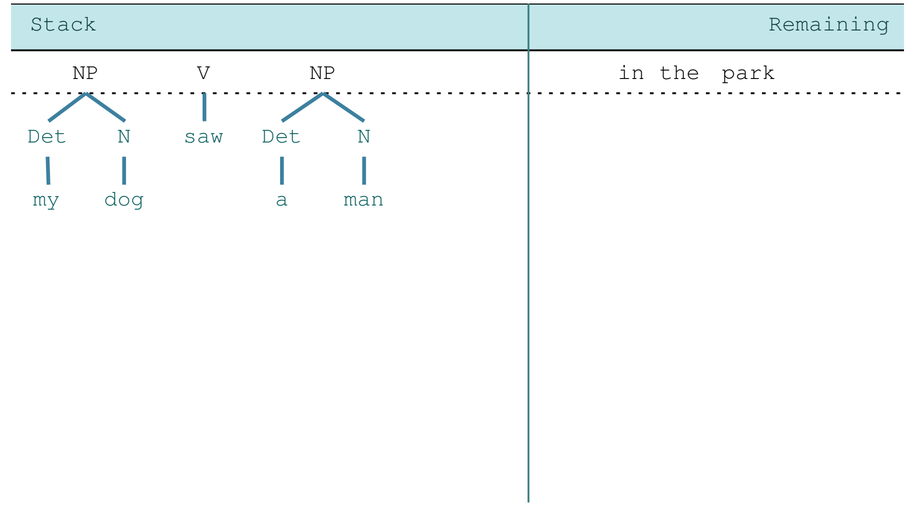
Shift-Reduce Parsing

Stack	Remaining
Det N	saw a man in the park
my dog	

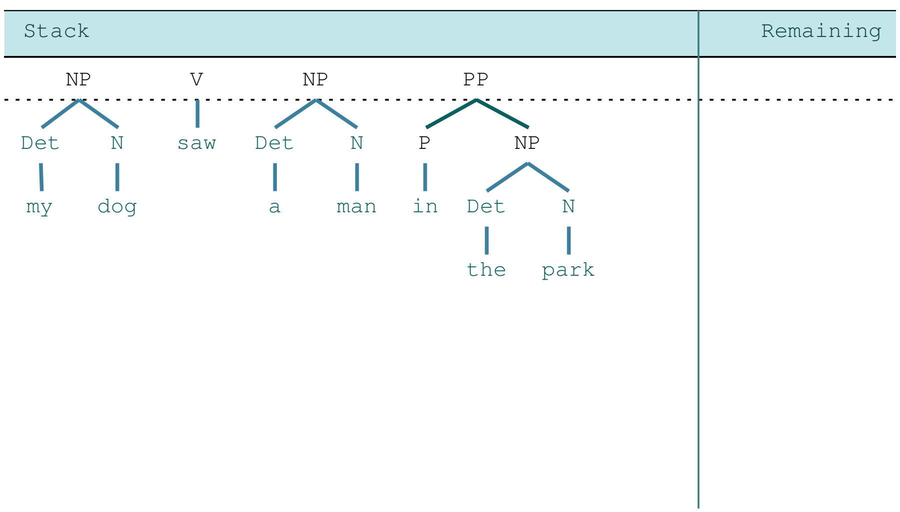
Shift-Reduce Parsing



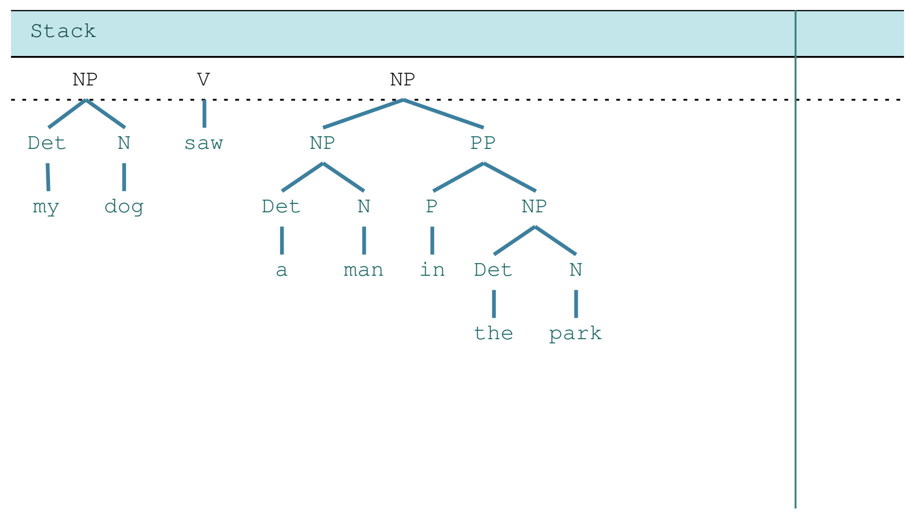
Shift-Reduce Parsing



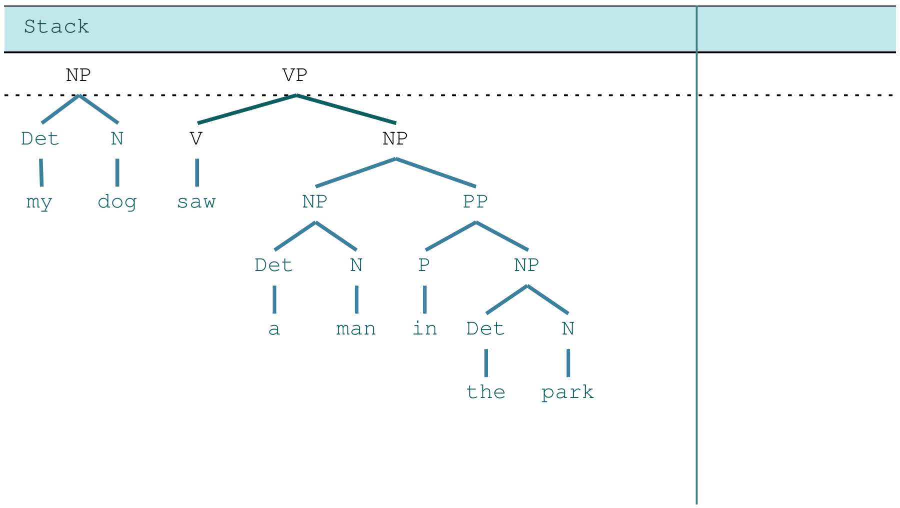
Shift-Reduce Parsing



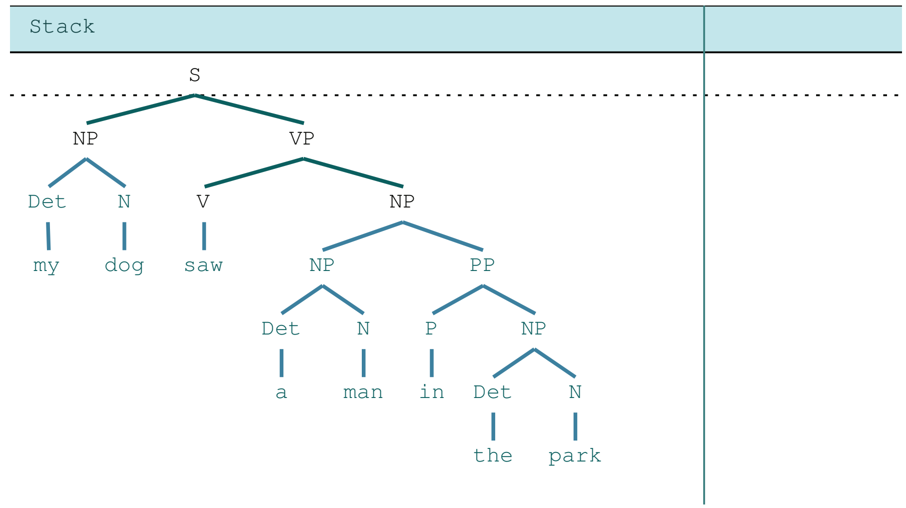
Shift-Reduce Parsing



Shift-Reduce Parsing



Shift-Reduce Parsing



Try it out Yourselves!

Recursive Decent Parser

```
>>> from nltk.app import rdparser
>>> rdparser()
```

Shift-Reduce Parser

```
>>> from nltk.app import srparser
>>> srparser()
```


Summary

- We use CFGs to represent NL grammars
- Grammars need recursion to produce infinite sentences
- Most NL grammars have structural ambiguity
- A parser computes structure for an input automatically
- Recursive descent and shift-reduce parsing
- We'll examine more parsers in Lectures 17–22

Reading: J&M (2nd edition) Chapter 12 (intro – section 12.3), Chapter 13 (intro – section 13.3)

Next lecture: The CYK algorithm