9.591 - Parsing algorithms, sentence complexity

Ted Gibson

November 1, 2004

(with thanks to Florian Wolf for these slides)

1. What is parsing?
2. Parsing strategies
    1. Top-down
    2. Bottom-up
    3. Left-corner
    4. Chart parsing

What is parsing?

(1) The dog bites the man.

(2) The man bites the dog.

(3) *The dog bites man the.

(1) = boring, (2) = interesting,

(3) = not English

What is parsing?

- (1), (2), and (3) have the same words
- BUT: structure different ➔ different meaning for (1) and (2)
- Not every sentence structure is possible: (3)
- A **grammar** tells you what are possible sentence structures in a language:
  - S → NP VP
  - NP → Det N
  - etc.

Why is parsing hard?

- Infinite number of possible sentences
  - We can understand and say sentences we never heard before.
  - Therefore representations of sentences' meanings cannot be stored in and retrieved from memory.
- Ambiguity
  - *The man saw the woman on the hill with the telescope*.
  - Word-level ambiguity: *saw*
  - Phrase-level ambiguity: PP-attachments

What is parsing?

Parsing:

discover how the words in a sentence can combine, using the rules in a grammar.

What is parsing?

- Parser

  sentence → representation of meaning

- Generator

  representation of meaning → sentence

Parsing strategies - intro

- Our grammar:

S	→	NP VP
NP	→	Det Noun
VP	→	Verb NP
Det	→	the
Noun	→	man
Noun	→	woman
Verb	→	likes
Verb	→	meets

Parsing strategies - intro

- Our sentence:

  The man likes the woman

Parsing strategies - intro

- Our grammar is unambiguous: not the case in real life
  - VP → Verb — *I walk*
  - VP → Verb NP — *I eat the apple*
  - VP → VP PP — *I see you with the telescope*

Parsing strategies - intro

- How to deal with ambiguity in grammar:
  - Serial approach:
    - try one rule at a time, then backtrack if necessary
    - need to specify with which rule to start
  - Parallel approach:
    - work on all alternative rules
    - need data structure that can contain set of parse trees

Parsing strategies - intro

- Top-down parsing:
  - Start by looking at rules in grammar
  - See what of the input is compatible with the rules
- Bottom-up parsing:
  - Start by looking at input
  - See which rules in grammar apply to input
- Combination of top-down and bottom-up:
  - Only look at rules that are compatible with input

Top-down - intro

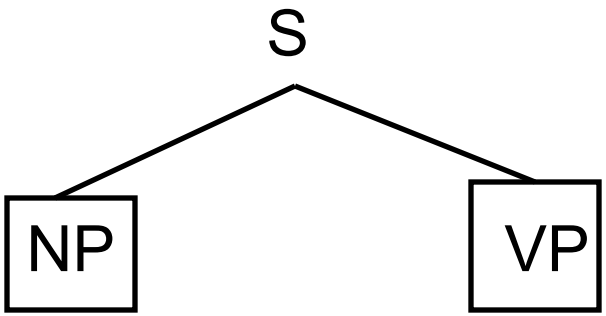
- Assume that input will eventually form a sentence
- Invoke S-node and all its possible extensions
- Keep expanding nodes until you find matching input
- Stack: keep track of what you still need to find in order to get grammatical sentence

Top-down - intro

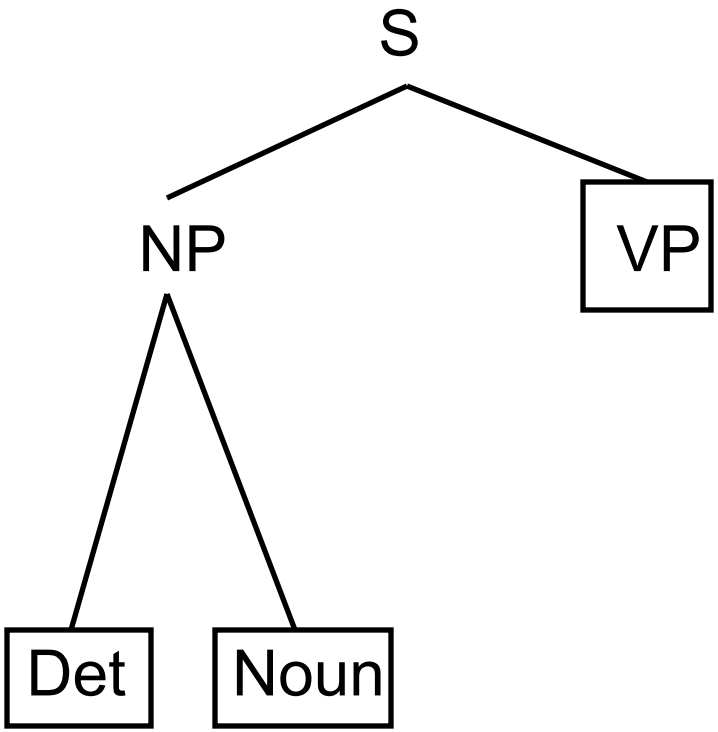
- If a LHS has more than one RHS:
  - **non-deterministic** = parser does not specify expansion order
  - **deterministic** = parser specifies expansion order

Top-down - example

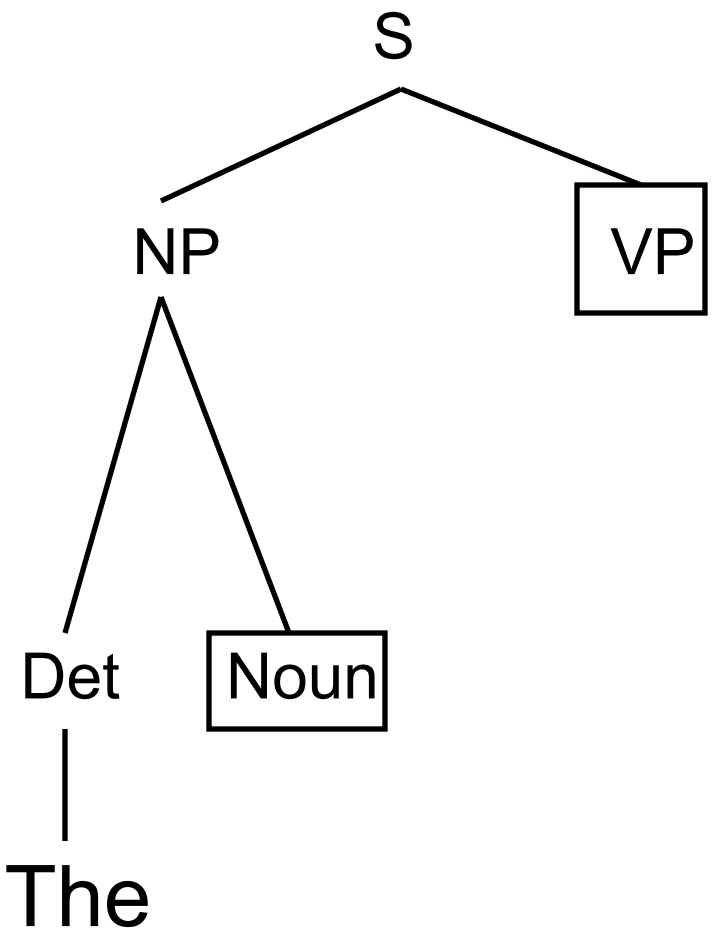
Top-down - example



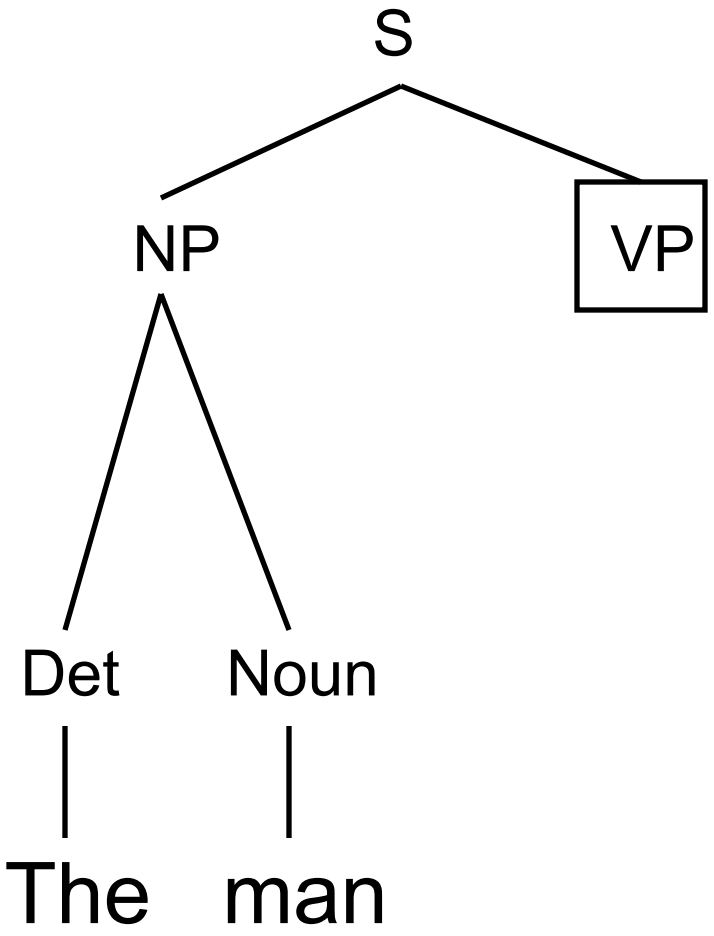
Top-down - example



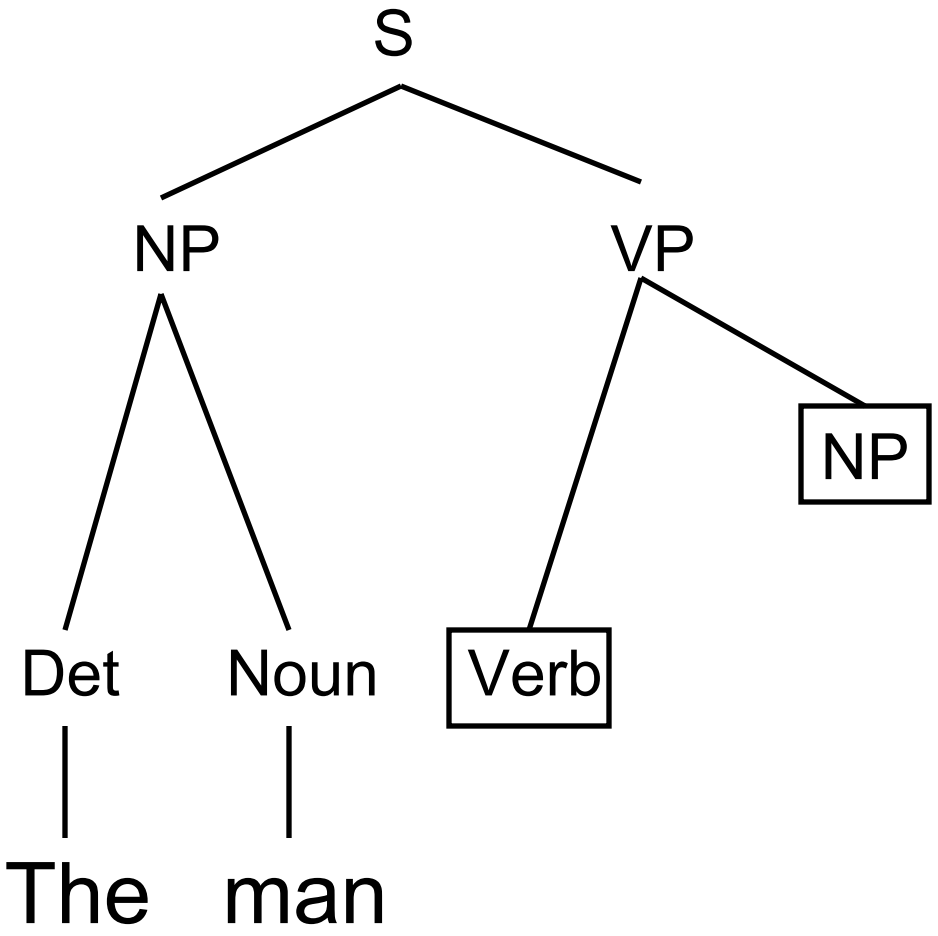
Top-down - example



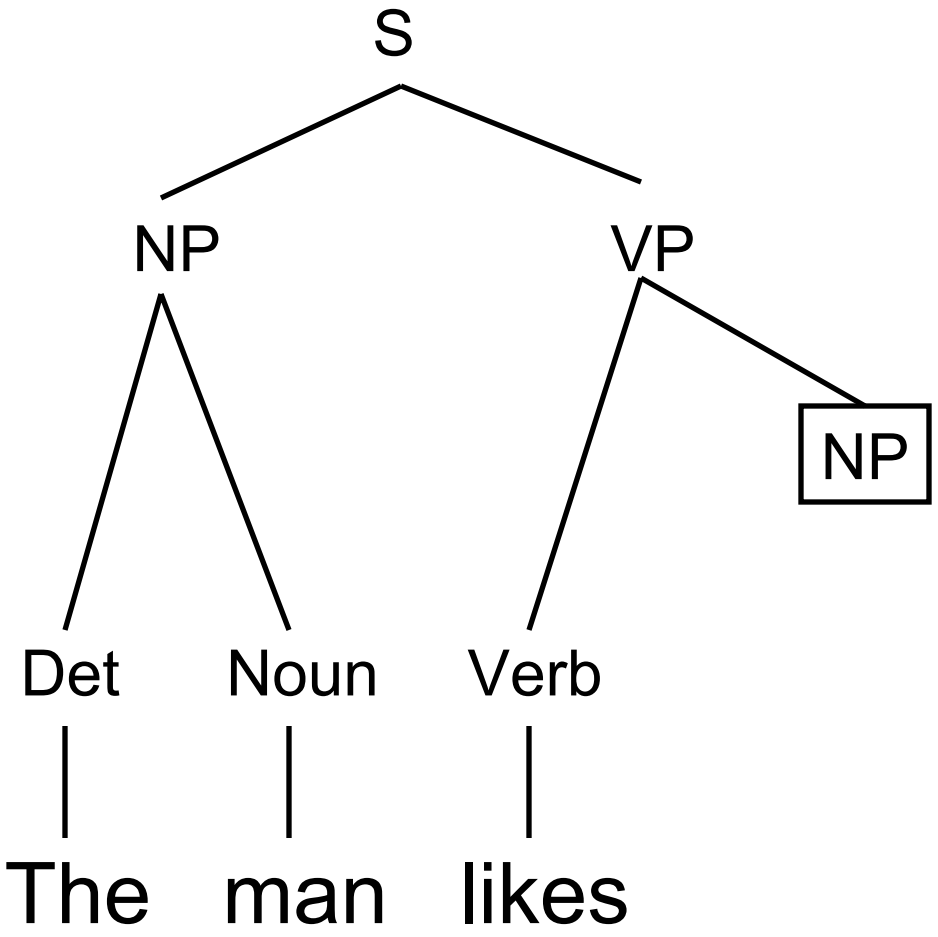
Top-down - example



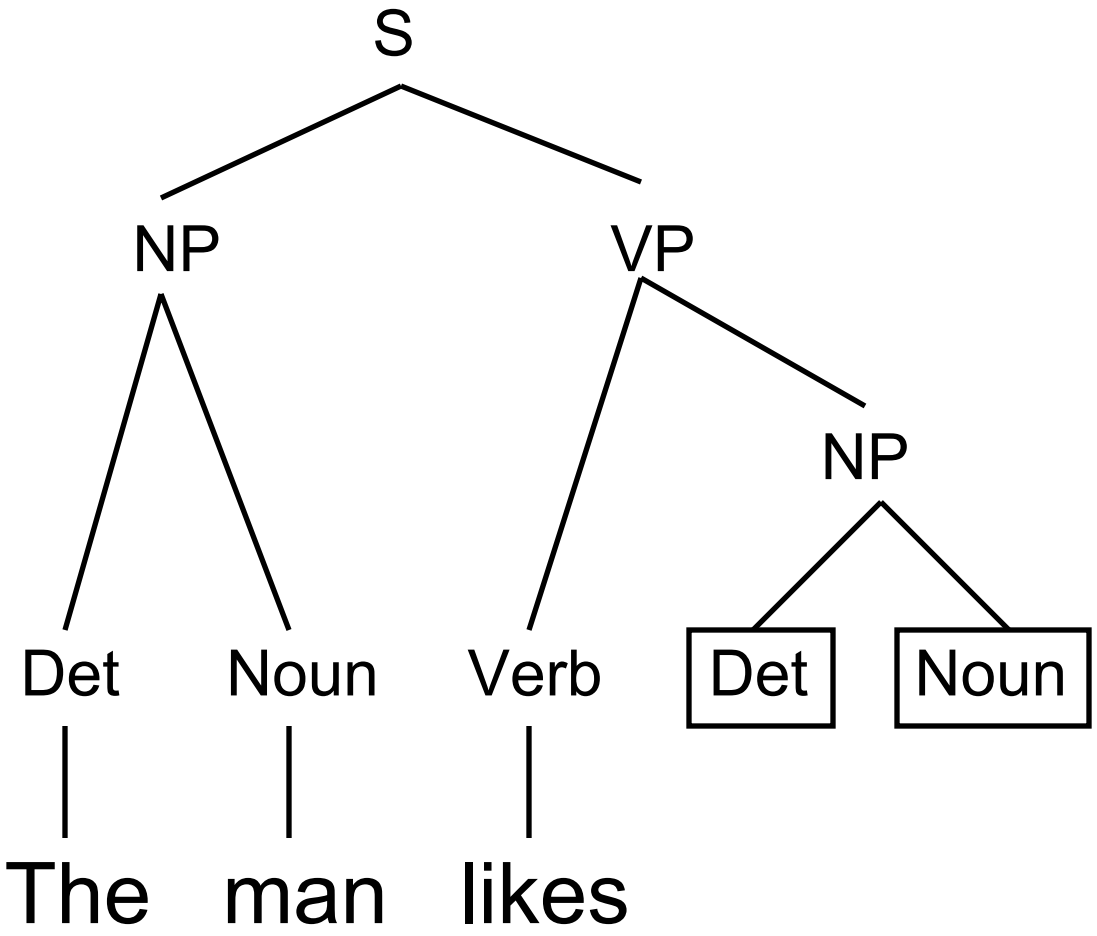
Top-down - example



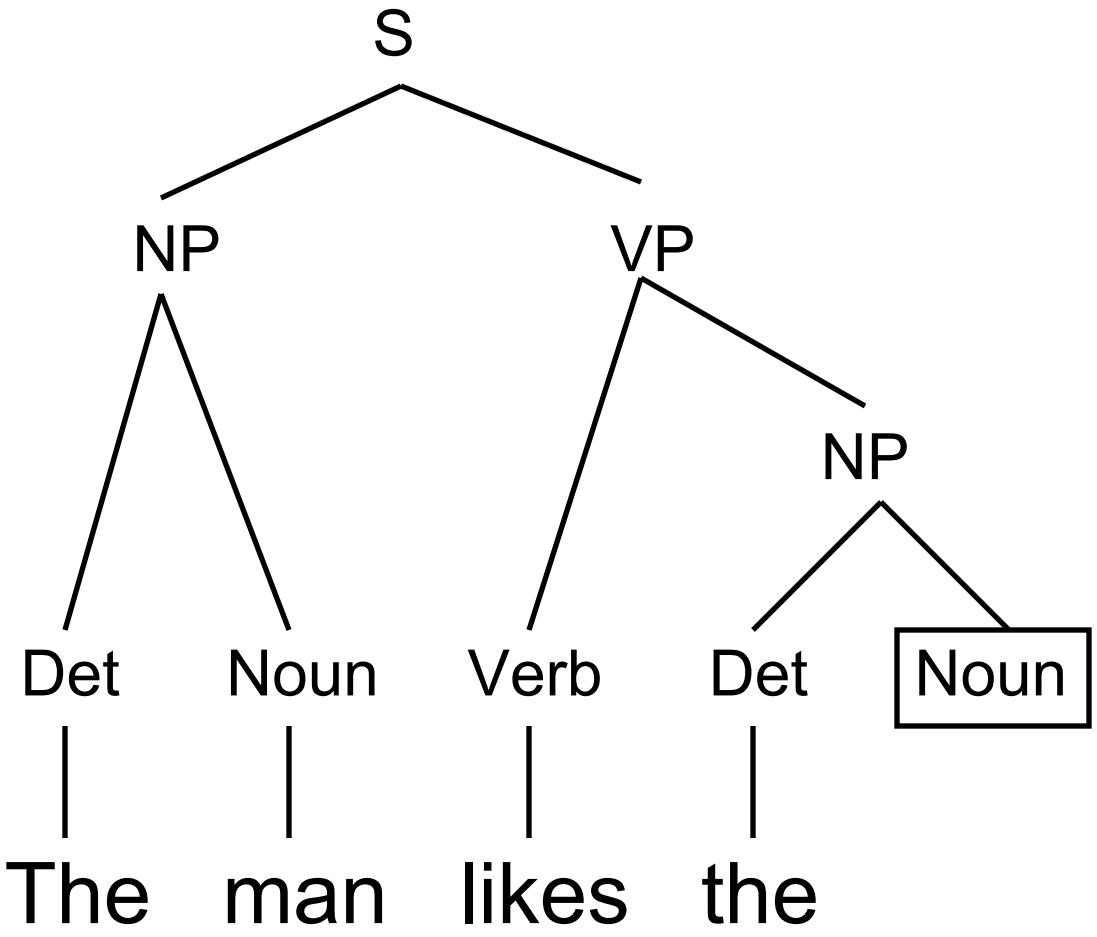
Top-down - example



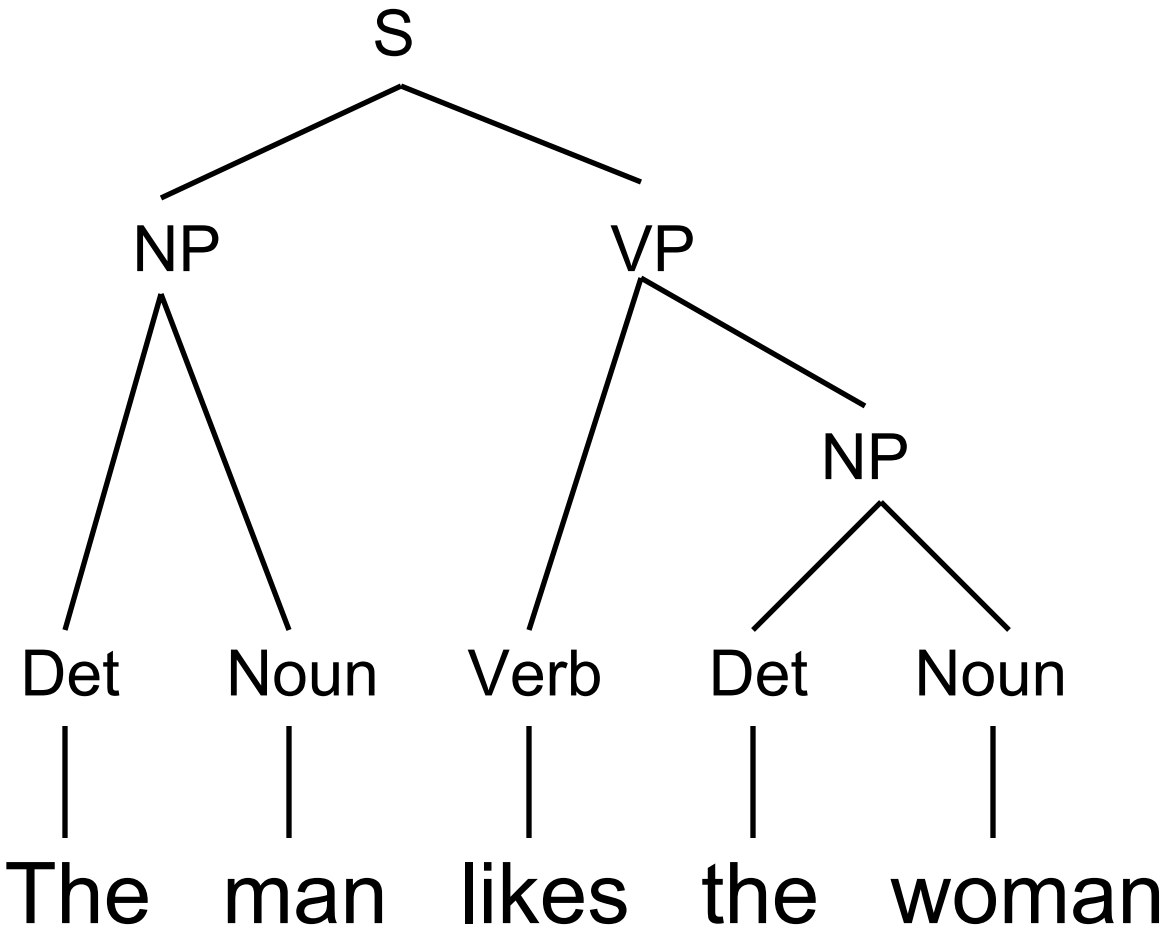
Top-down - example



Top-down - example



Top-down - example



Top-down - evaluation

- Advantage:
  - Starts with S-node, therefore never tries to form a structure that will never form an S
- Disadvantages:
  - Can try to build trees inconsistent with input (if we had rule VP → V PP)
  - Left-recursion

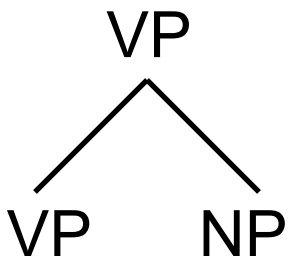
Top-down - evaluation

- Left-recursion, cont'd:
  - Left-recursion = some LHS can be expanded through series of rules such that left corner of one of these expansions is in same LHS category
  - VP → VP NP
- Parser gets caught in endless loop

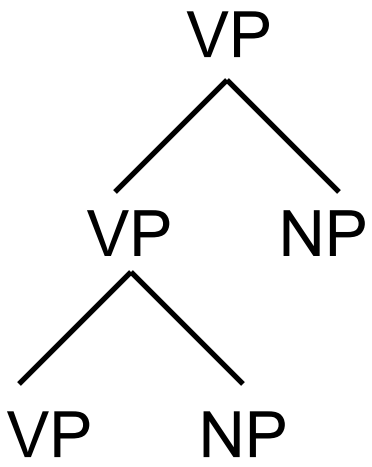
Top-down - left-recursion

VP

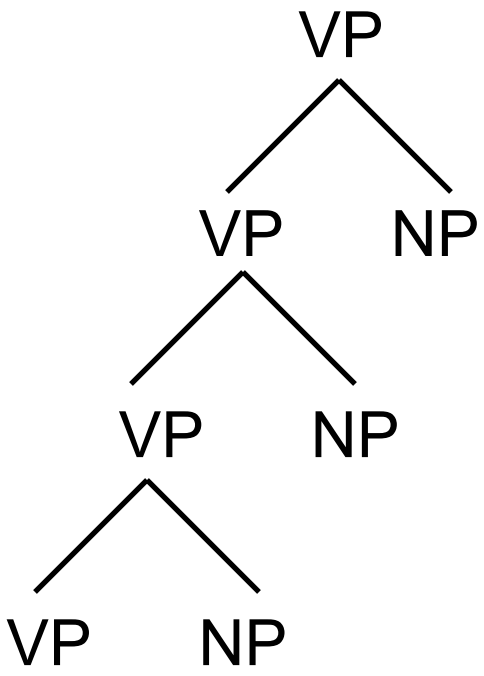
Top-down - left-recursion



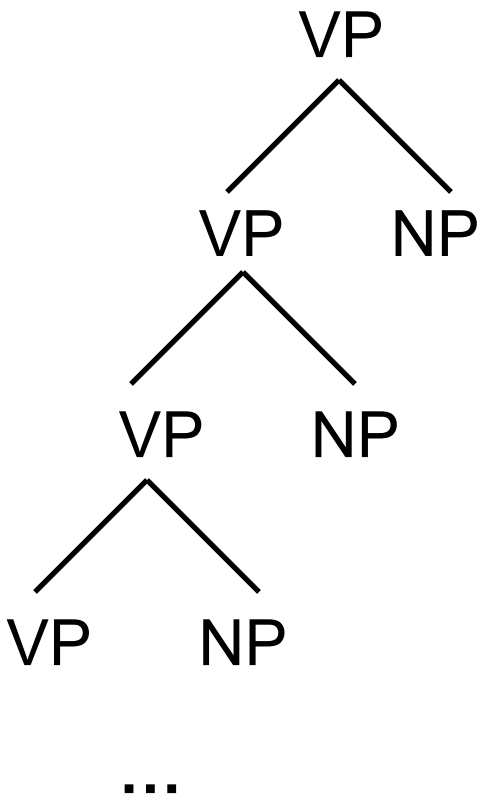
Top-down - left-recursion



Top-down - left-recursion



Top-down - left-recursion



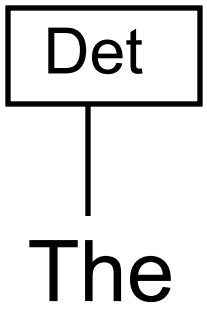
Bottom-up - intro

- Look at input, then try to find rules in grammar that apply to input
- Stack keeps track of what has been found so far and still needs to be integrated in parse tree

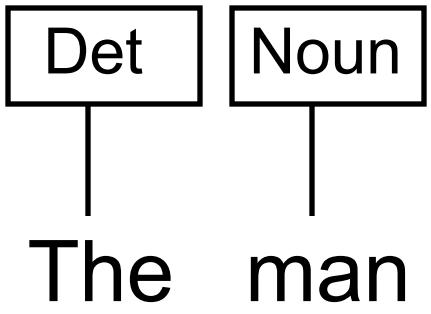
Bottom-up - intro

- **shift** = push categories on stack that still need to be integrated
- **reduce** = apply grammar rules to categories in stack
- Shift-reduce parser

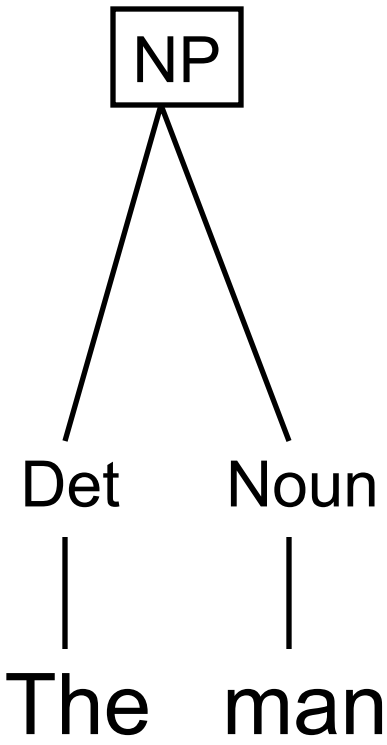
Bottom-up - example



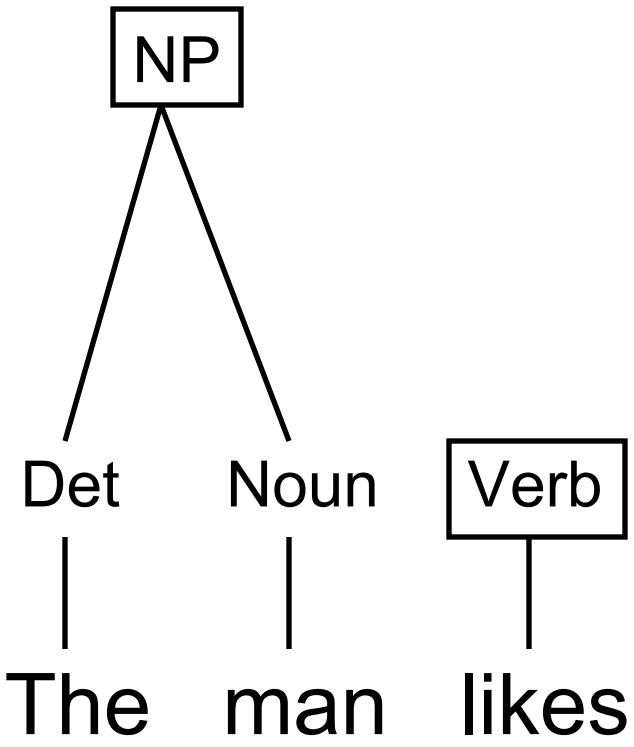
Bottom-up - example



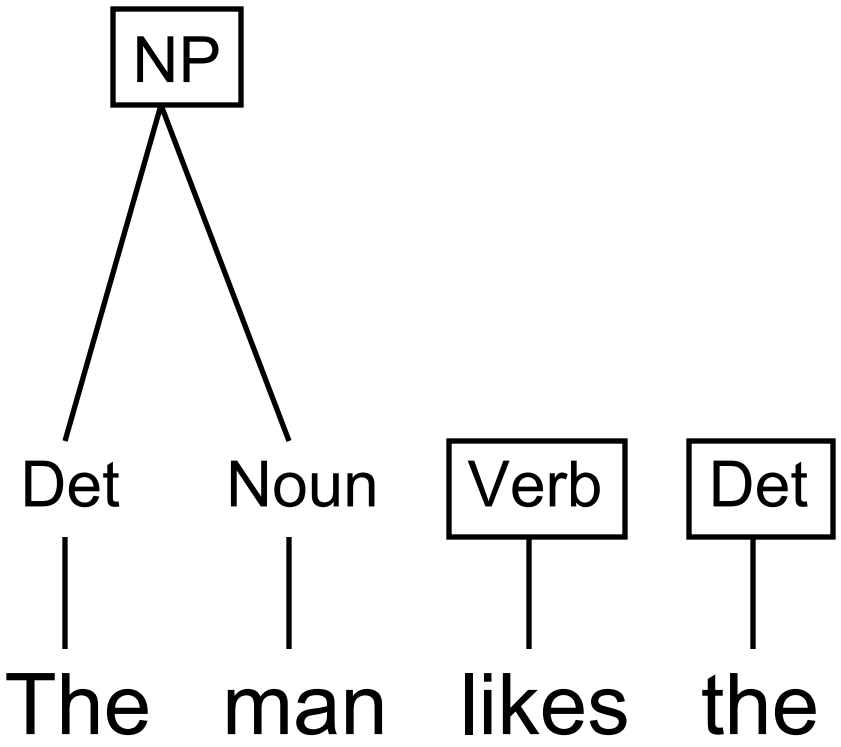
Bottom-up - example



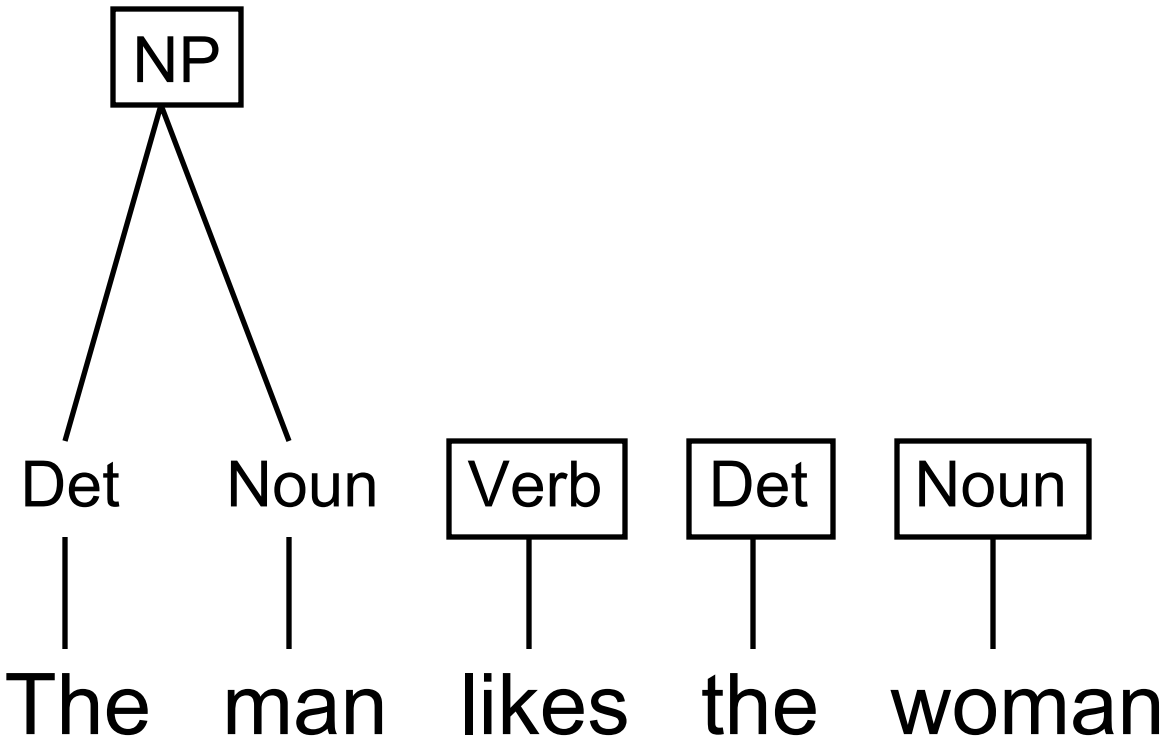
Bottom-up - example



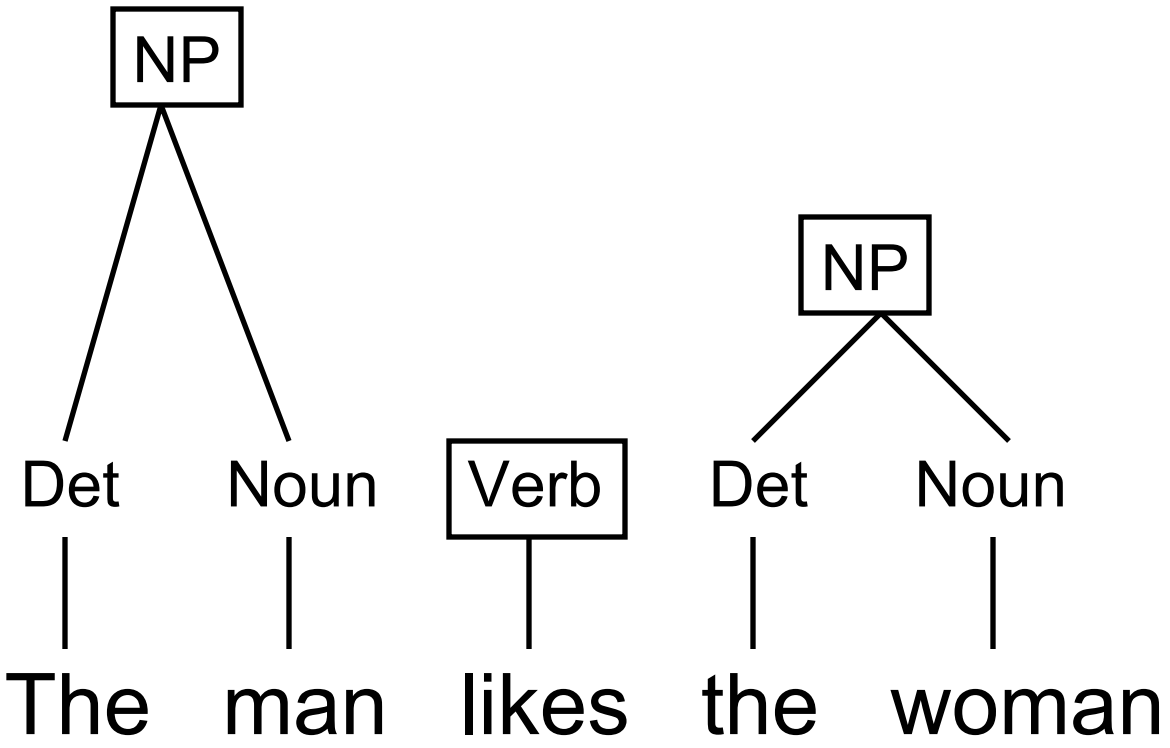
Bottom-up - example



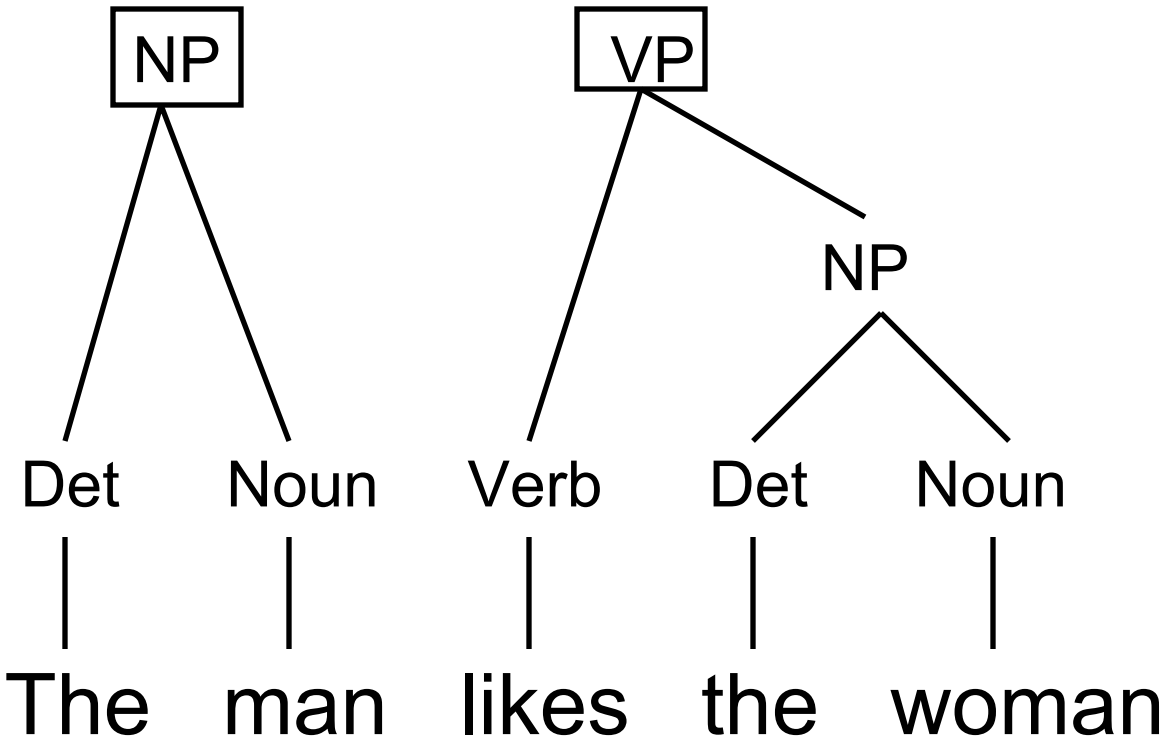
Bottom-up - example



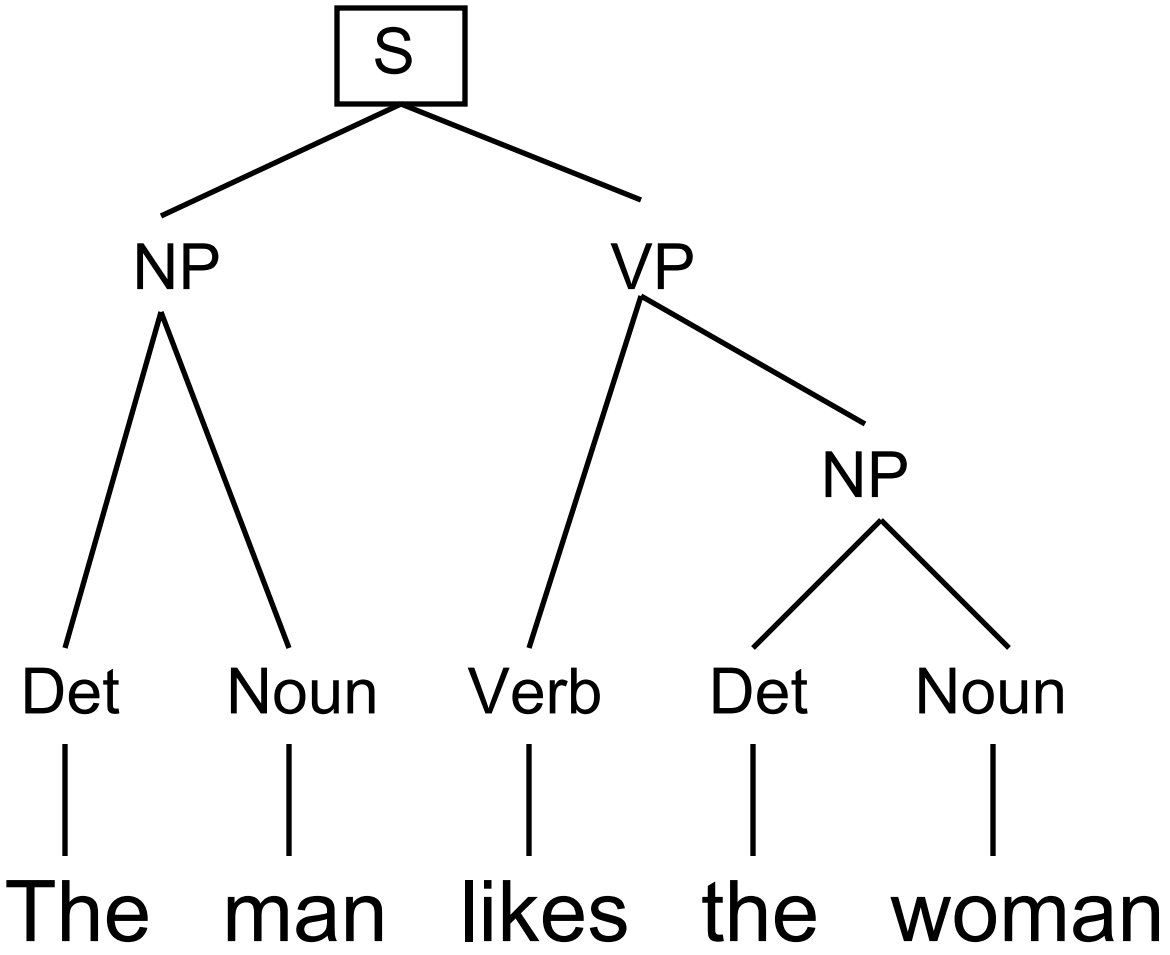
Bottom-up - example



Bottom-up - example



Bottom-up - example



Bottom-up - evaluation

- Advantages:
  - Does not predict trees that are inconsistent with input
  - Can handle left-recursion
- Disadvantage:
  - Can generate structures that never result in an S

Top-down vs. bottom-up

- Top-down
  - Good
    - Always produces S
  - Bad
    - Can generate structures inconsistent w/ input
    - No left-recursion
- Bottom-up
  - Good
    - Always generates structures consistent w/ input
    - Handles left-recursion
  - Bad
    - Can generate non-S structures

Processing complexity

- Processing complexity possibly related to storage space needed to parse a sentence
- The more unfinished stuff you have sitting around, the harder parsing becomes

- Storage space: the depth of the stack: the number of incomplete rules (top-down) or the number of constituents covering a subset of the target sentence (bottom-up)

Processing complexity

- A top-down strategy requires unbounded stack depth for left-branching sentences of arbitrary depth, e.g.,

  (1) John's brother's mother's uncle's friend's dog's tail fell off.

- People have no difficulty with arbitrary-depth left-branching (e.g., head-final languages). Therefore, a a fully top-down strategy may not be psychologically viable.

Processing complexity

- A bottom-down strategy requires unbounded stack depth for right-branching sentences of arbitrary depth, e.g.,

  (2) The father of the brother of the friend of the owner of the dog fell over.

- People have no difficulty with arbitrary-depth right-branching. Therefore, a a fully bottom-up strategy may not be psychologically viable.

Left-corner - intro

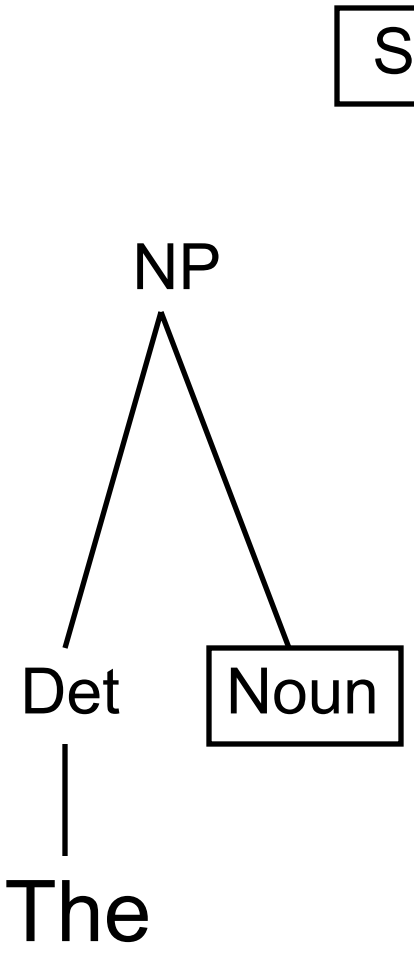
- Rule only predicted (top-down) if current input (bottom-up) matches leftmost corner (left-corner) of the RHS of a rule
  - VP → Verb NP
- Stack keeps track of what input is still needed to complete a predicted rule

Left-corner - complexity

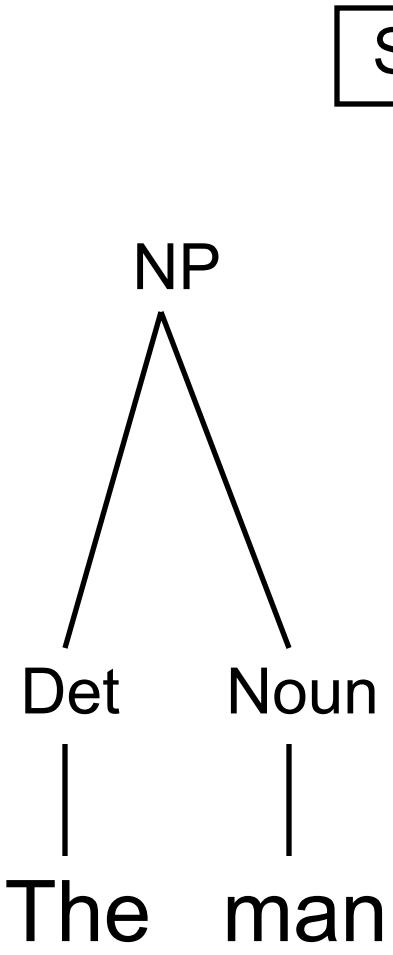
- Mirrors human performance closer than top-down or bottom-up

Left-corner - example

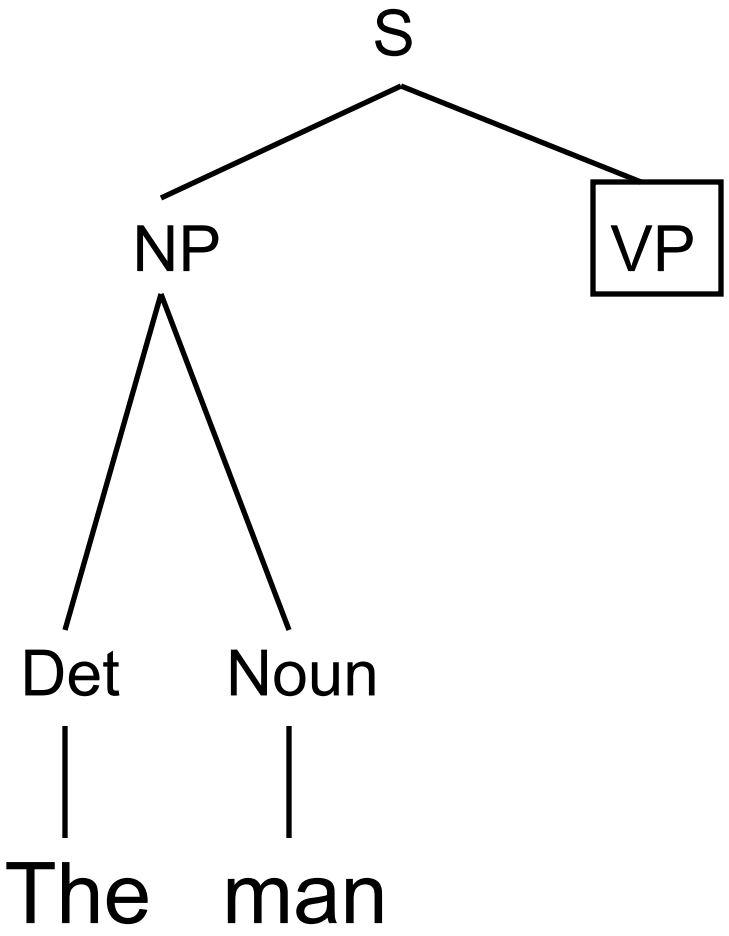
Left-corner - example



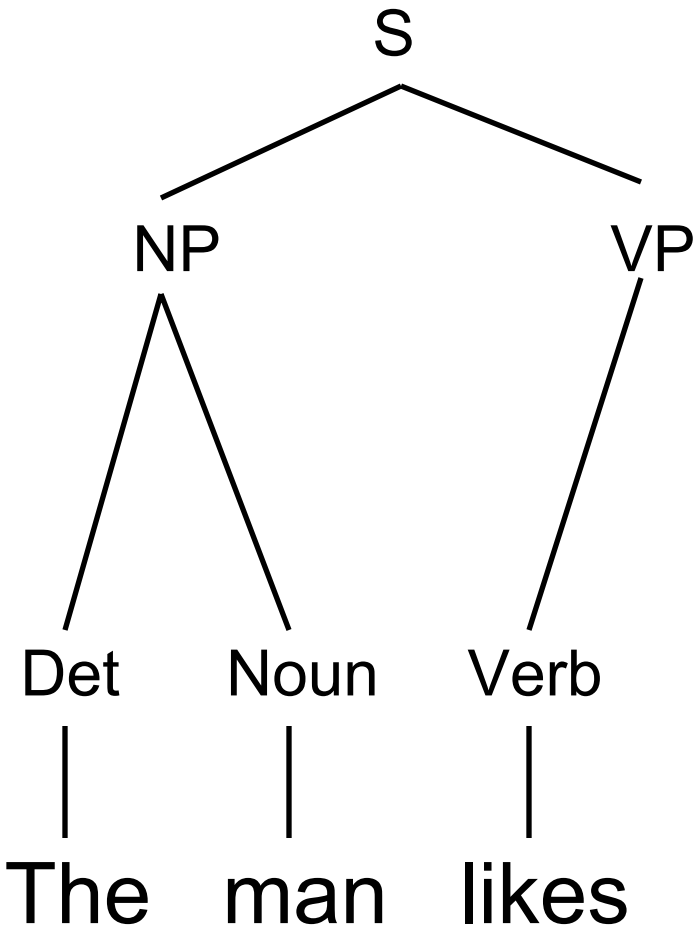
Left-corner - example



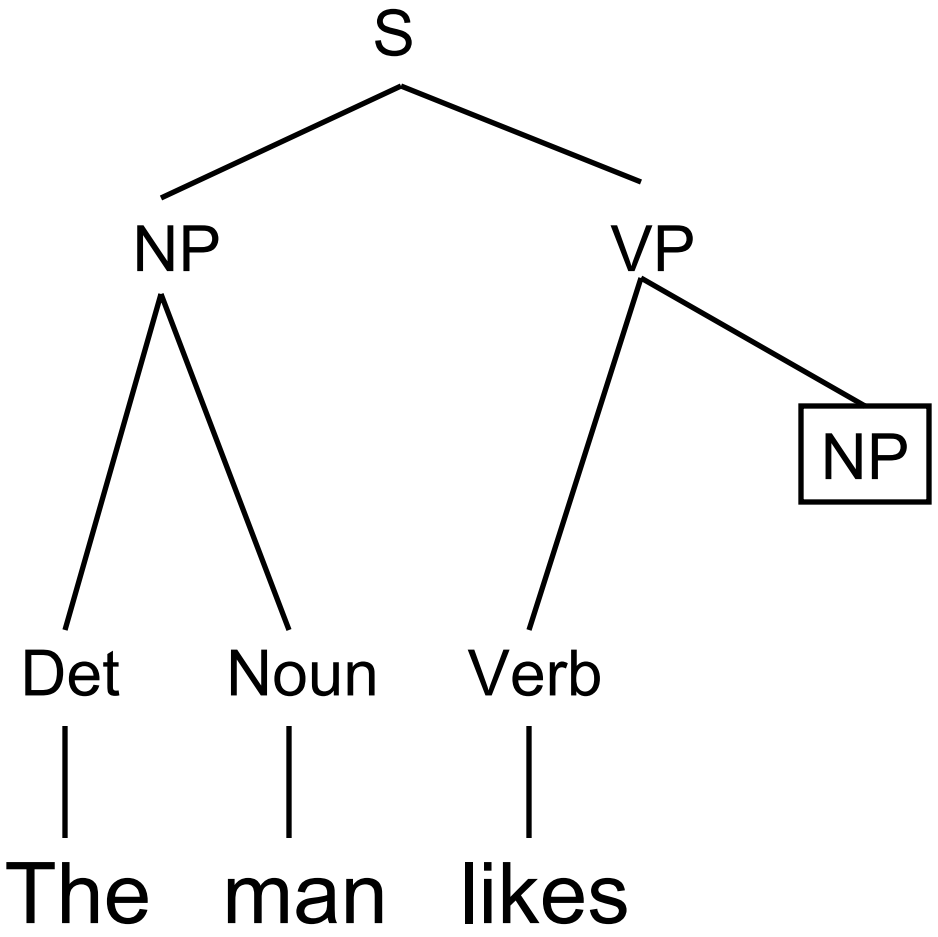
Left-corner - example



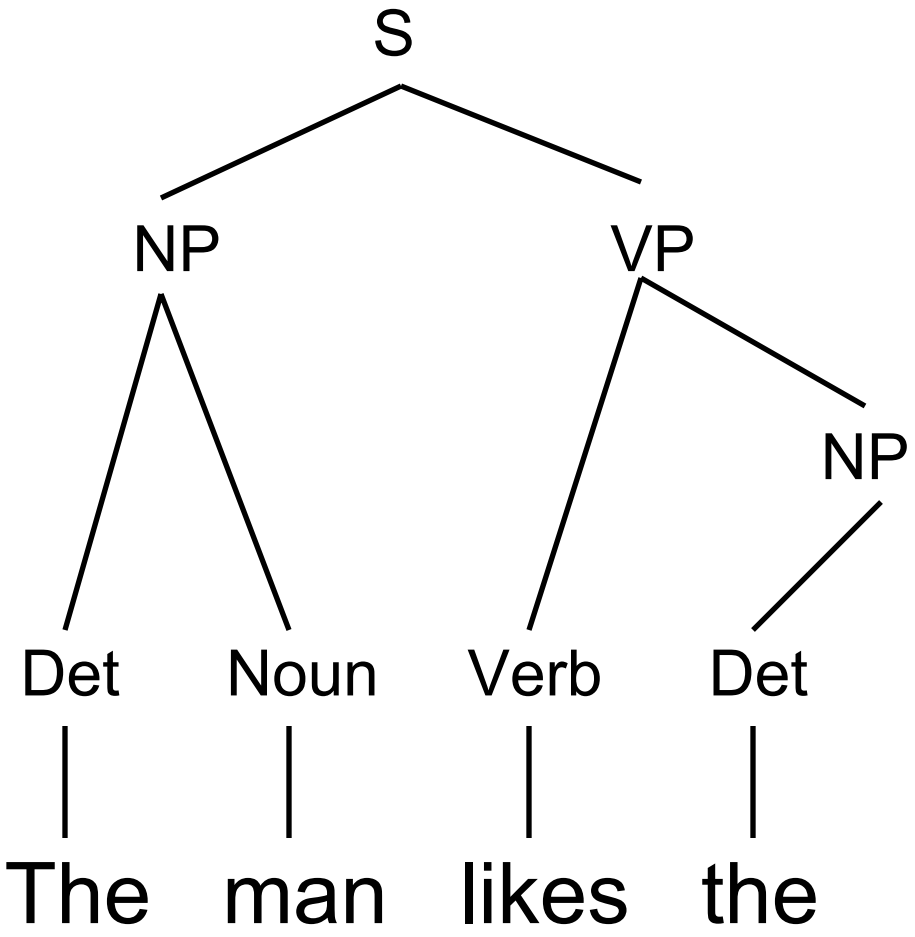
Left-corner - example



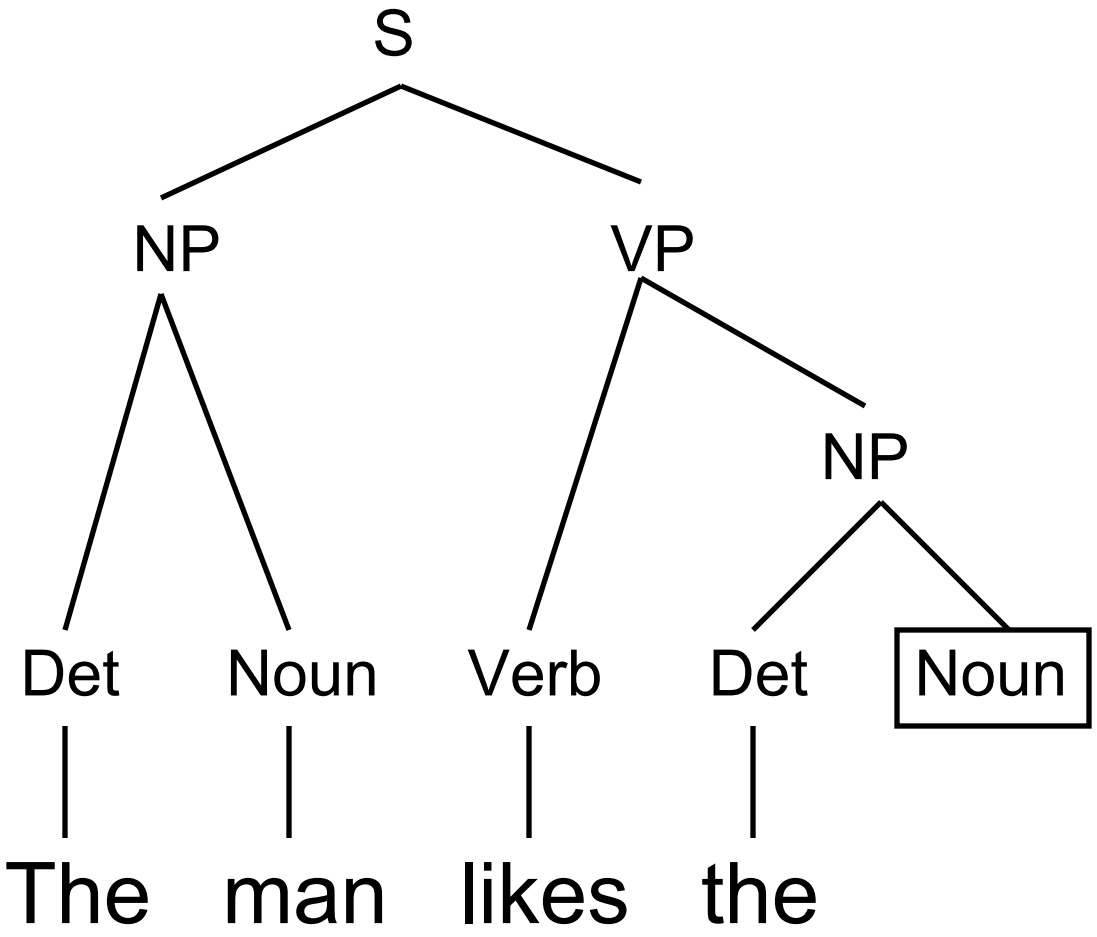
Left-corner - example



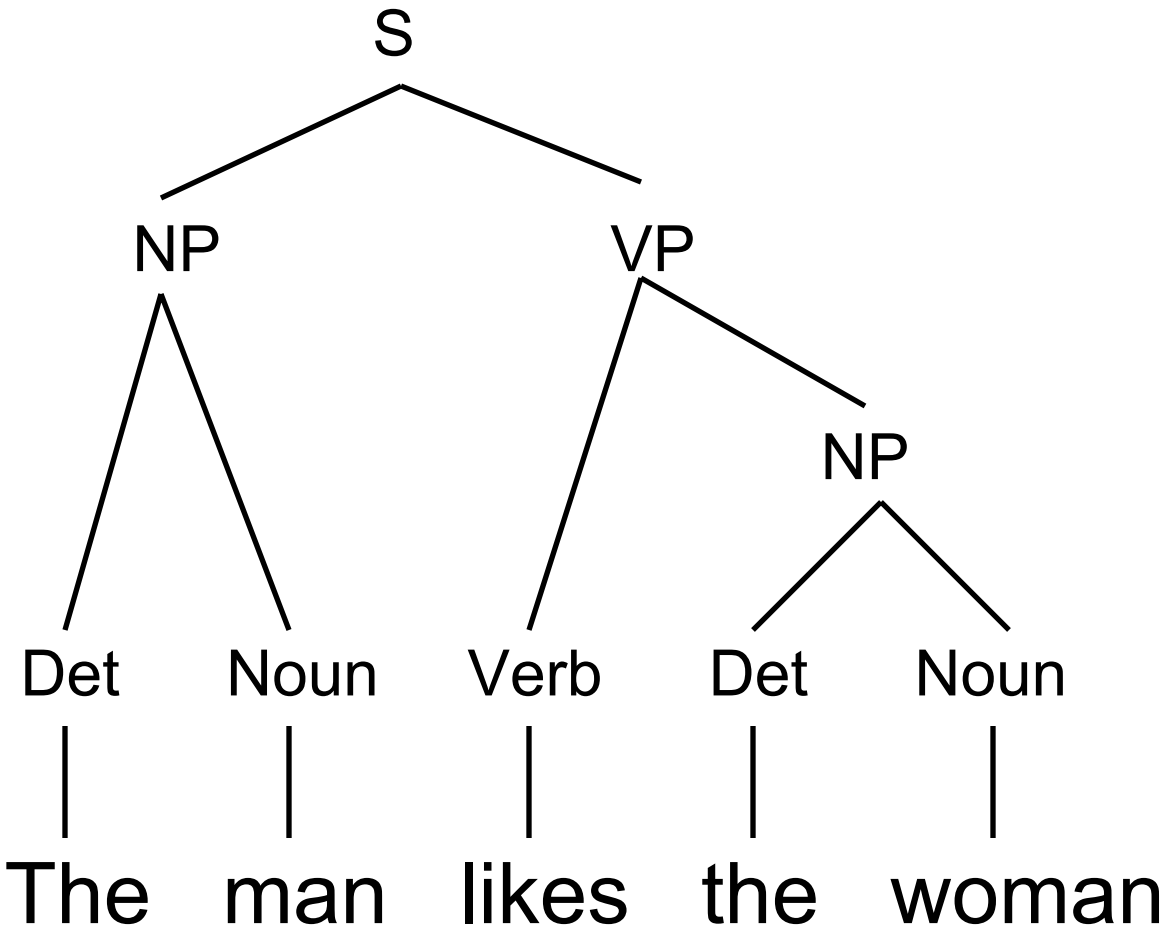
Left-corner - example



Left-corner - example



Left-corner - example



Processing complexity

- A left-corner strategy requires bounded stack depth for both left- and right-branching sentences of arbitrary depth.

- A left-corner strategy requires unbounded stack depth only for nested / center-embedded structures.

  (3) # The man who the woman who John met likes is tall.

- **This pattern best fits human judgments.** But it's only a rough cut:
- Problematic cases:
  - SC/RC vs RC/SC
  - Embedded pronouns in nested positions
  - Etc.

Processing complexity

	sentence structure		
	LB	**CE**	**RB**
TD stack size	Unbounded	Unbounded	Bounded
BU stack size	Bounded	Unbounded	Unbounded
LC stack size	Bounded	Unbounded	Bounded
humans	Easy	Hard	Easy

Chart parsing - intro

- Still a problem: ambiguity

  The tall man with brown hair saw the short woman with blond hair.

- *"saw"* is ambiguous
- *The tall man with brown hair* has to be parsed twice, for Noun and Verb meaning
- In realistic applications: enormous efficiency problem

Chart parsing - intro

- Solution: give the parser a memory!
- Keep track of partially and completely parsed rules from grammar
- Look up parsed rules instead of reparsing them
- **chart** = memory for parser (Earley, 1970)

Chart parsing - definitions

- Chart:
  - **edges** = rules from the grammar
  - **incomplete / active edges** = partially parsed edges
  - **complete / inactive edges** = completely parsed edges
  - Once an edge is entered into chart, it stays there

Chart parsing - edges

- Information about edges in chart:
  - Syntactic category of the edge (e.g. *NP*)
  - Where in the sentence the edge begins (left end)
  - Where in the sentence the edge ends (right end)
  - Pointers to further inactive edges (e.g. to *Det Noun* for *NP*)
  - For active edges: list of what categories are still needed to complete the edge (make it inactive)

Chart parsing - edges

- Edge notation:
  - [category] / [what's there] **.** [what's needed]
- Examples of edges:
  - S / NP **.** VP
  - NP / Det Noun **.**

Chart parsing - cont'd

- Fundamental Rule
  - Look for matches between categories needed in an active edge and set of inactive edges
- Top-down
  - Initialize chart w/ empty active *S* edge
- Bottom-up
  - Initialize chart w/ inactive edges for words

Chart parsing - example

bottom-up strategy

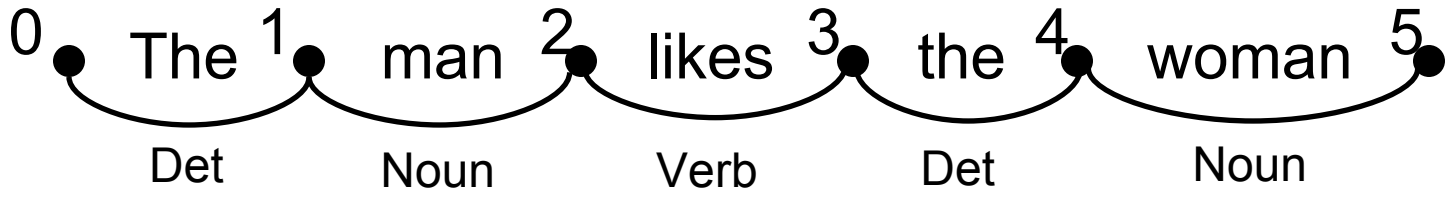


Chart parsing - example

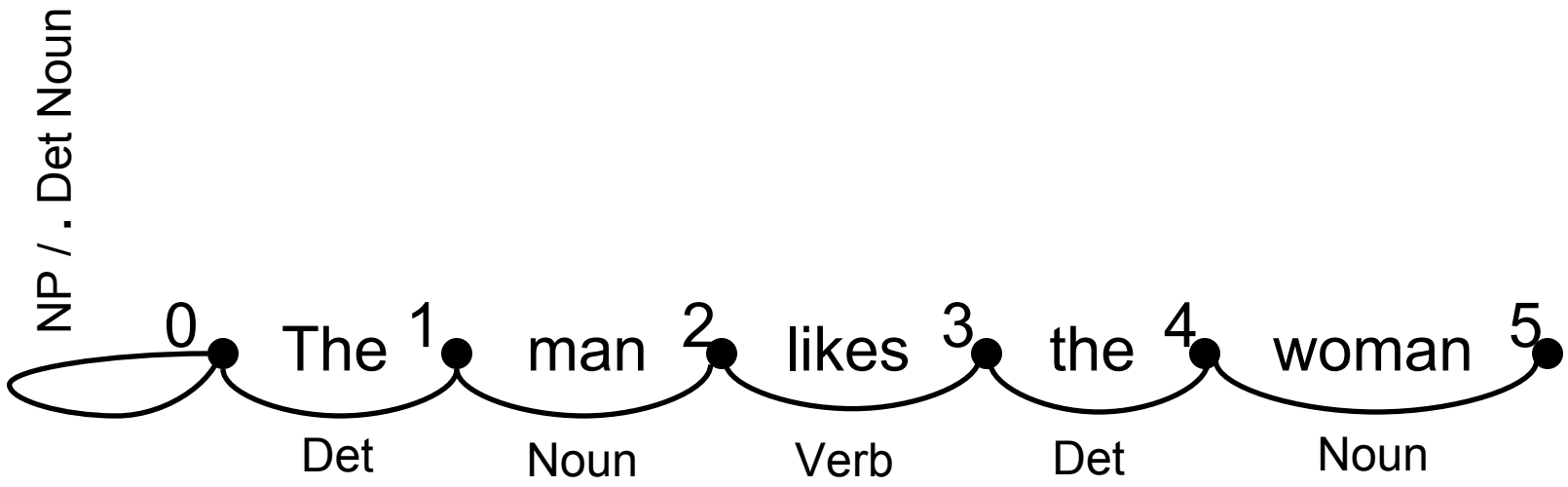


Chart parsing - example

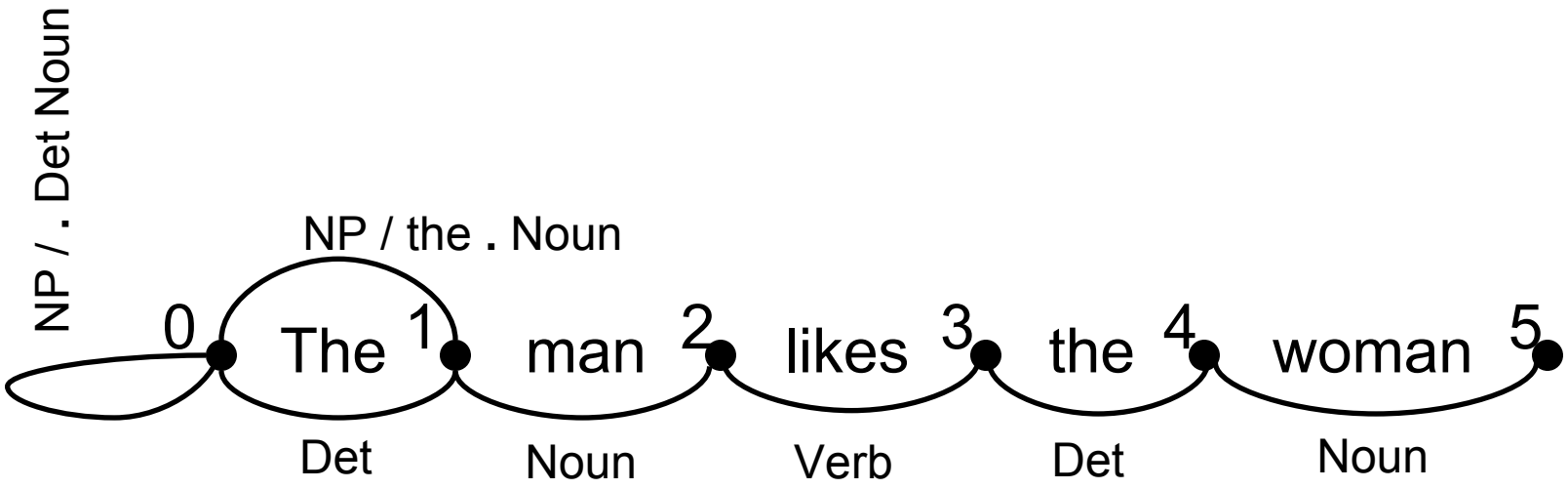


Chart parsing - example

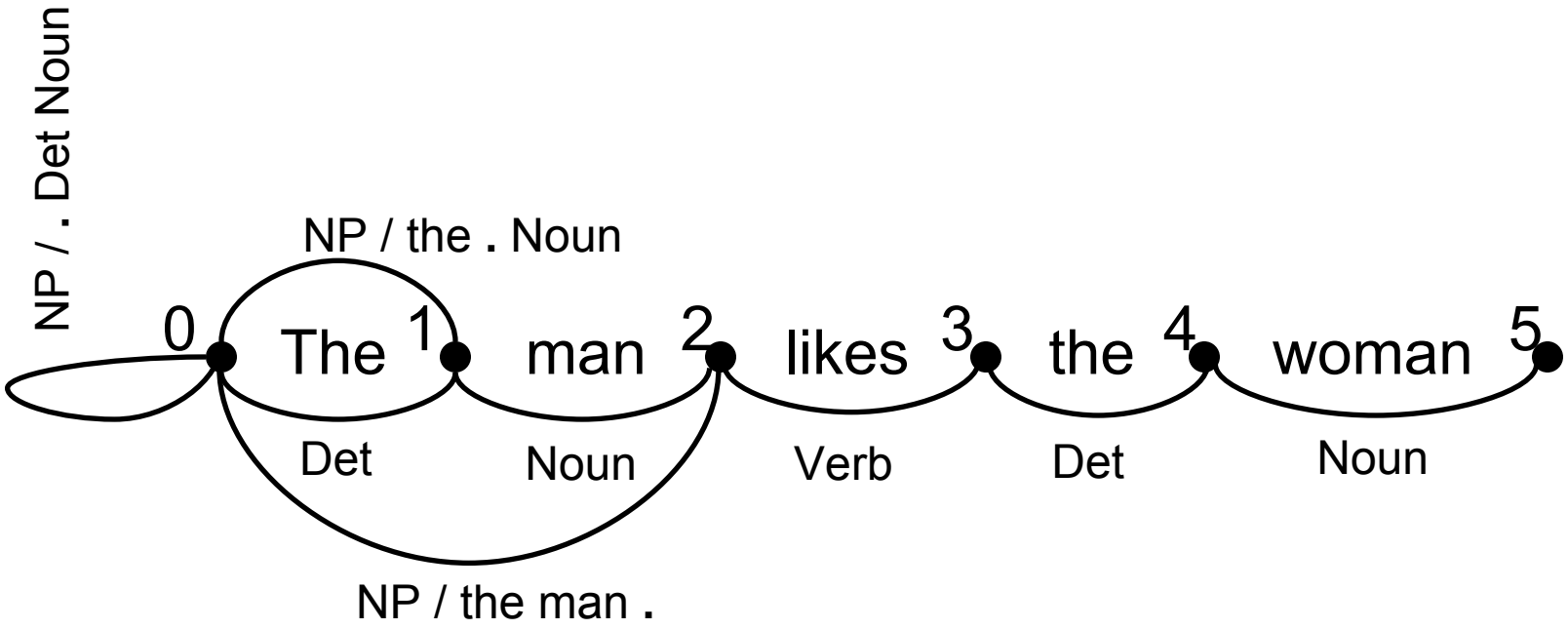


Chart parsing - example

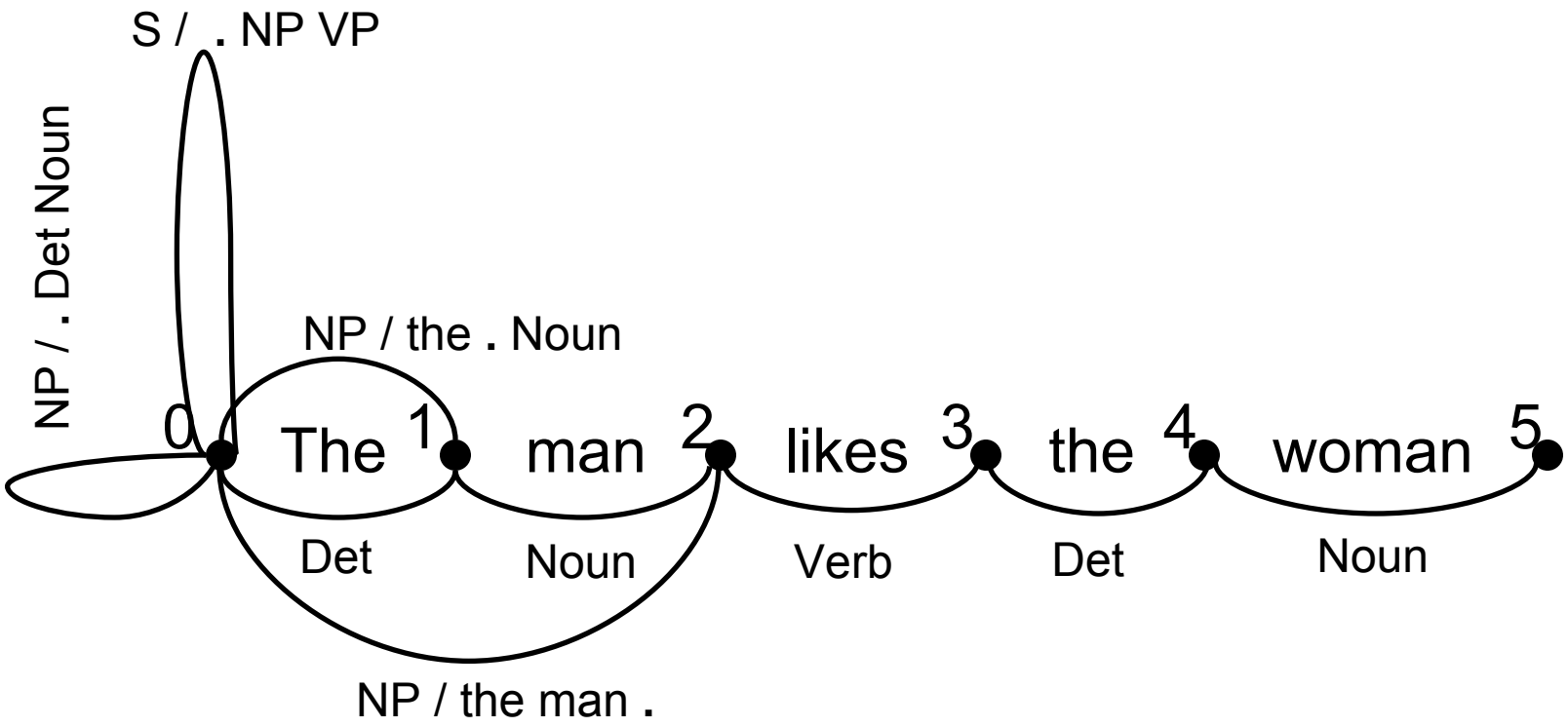


Chart parsing - example

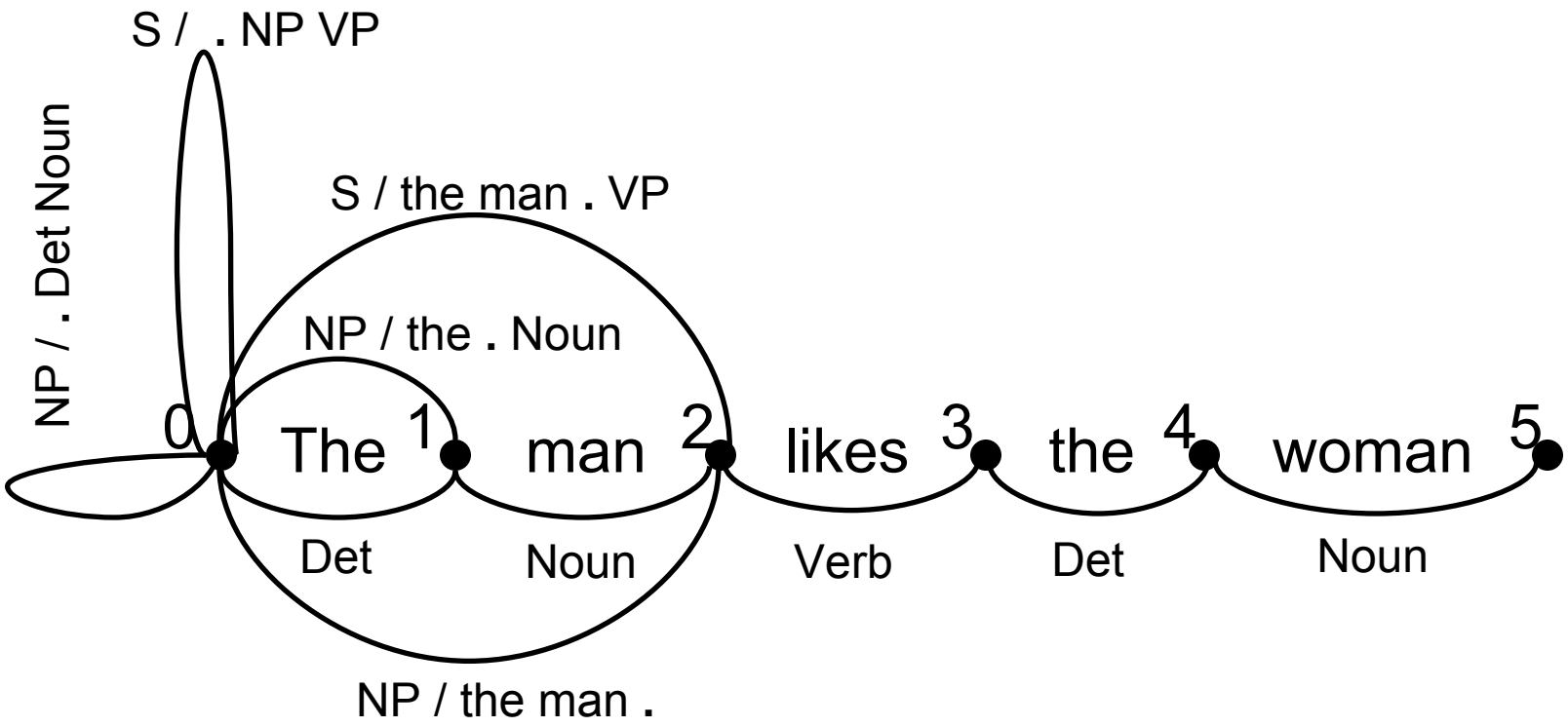


Chart parsing - example

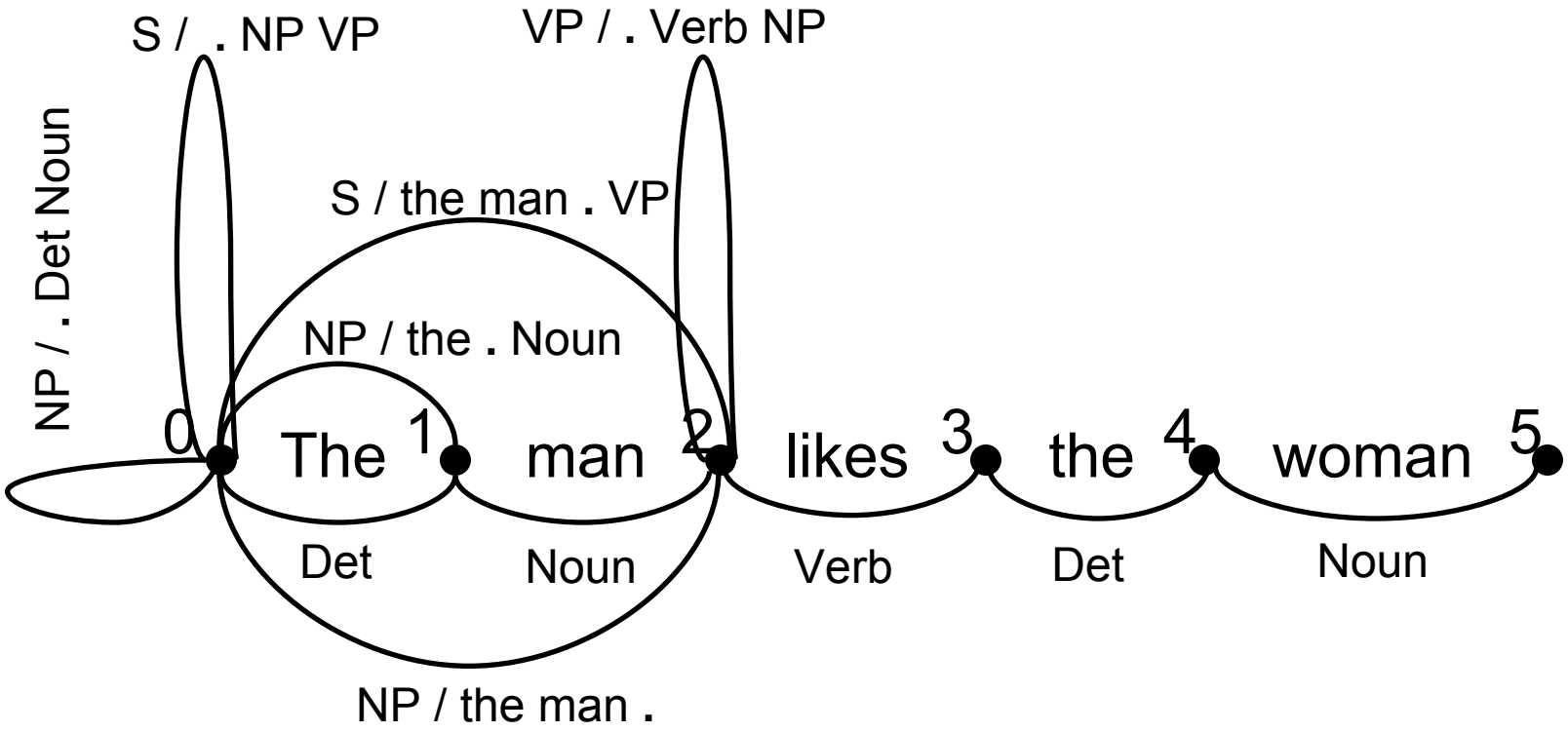


Chart parsing - example

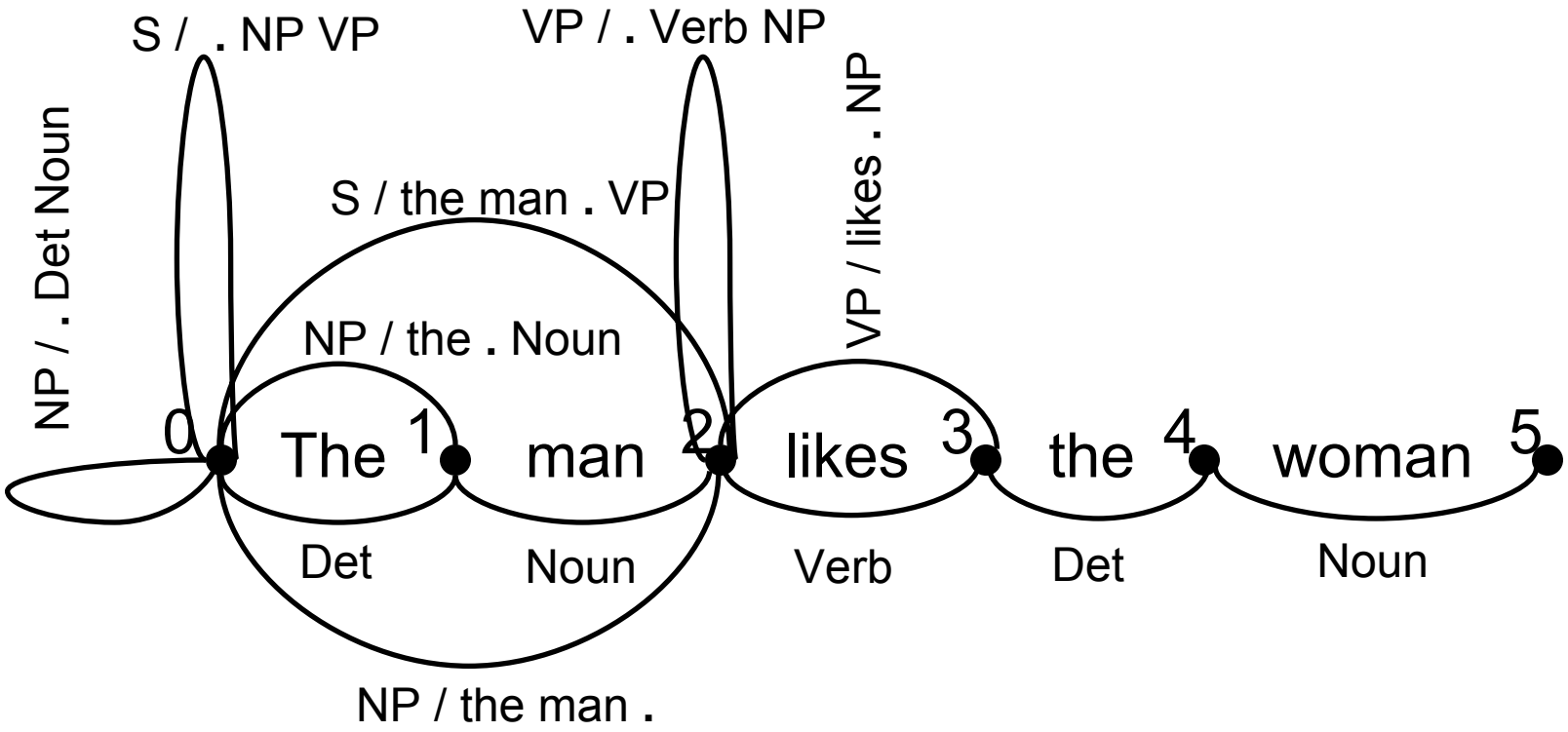


Chart parsing - example

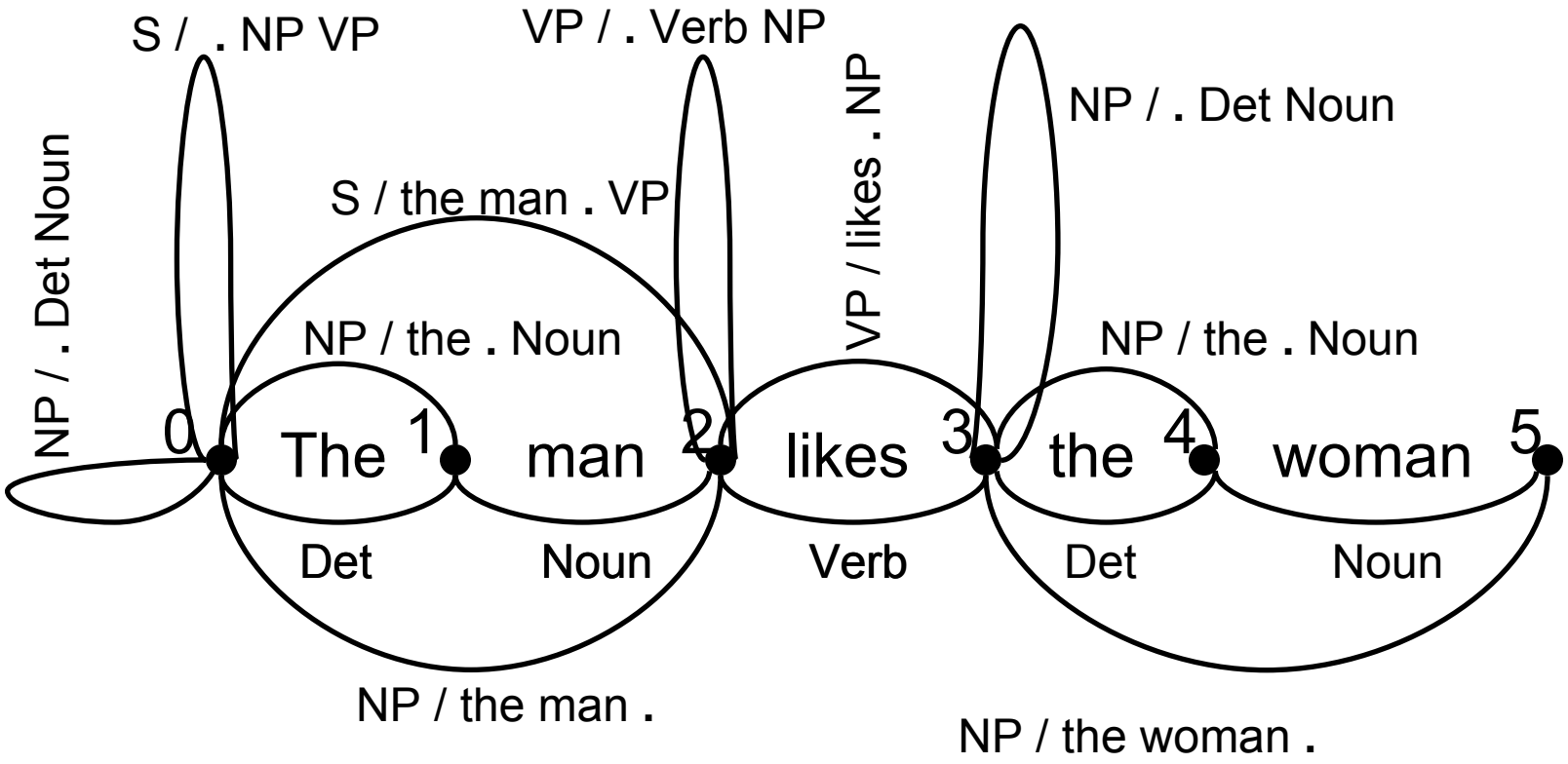
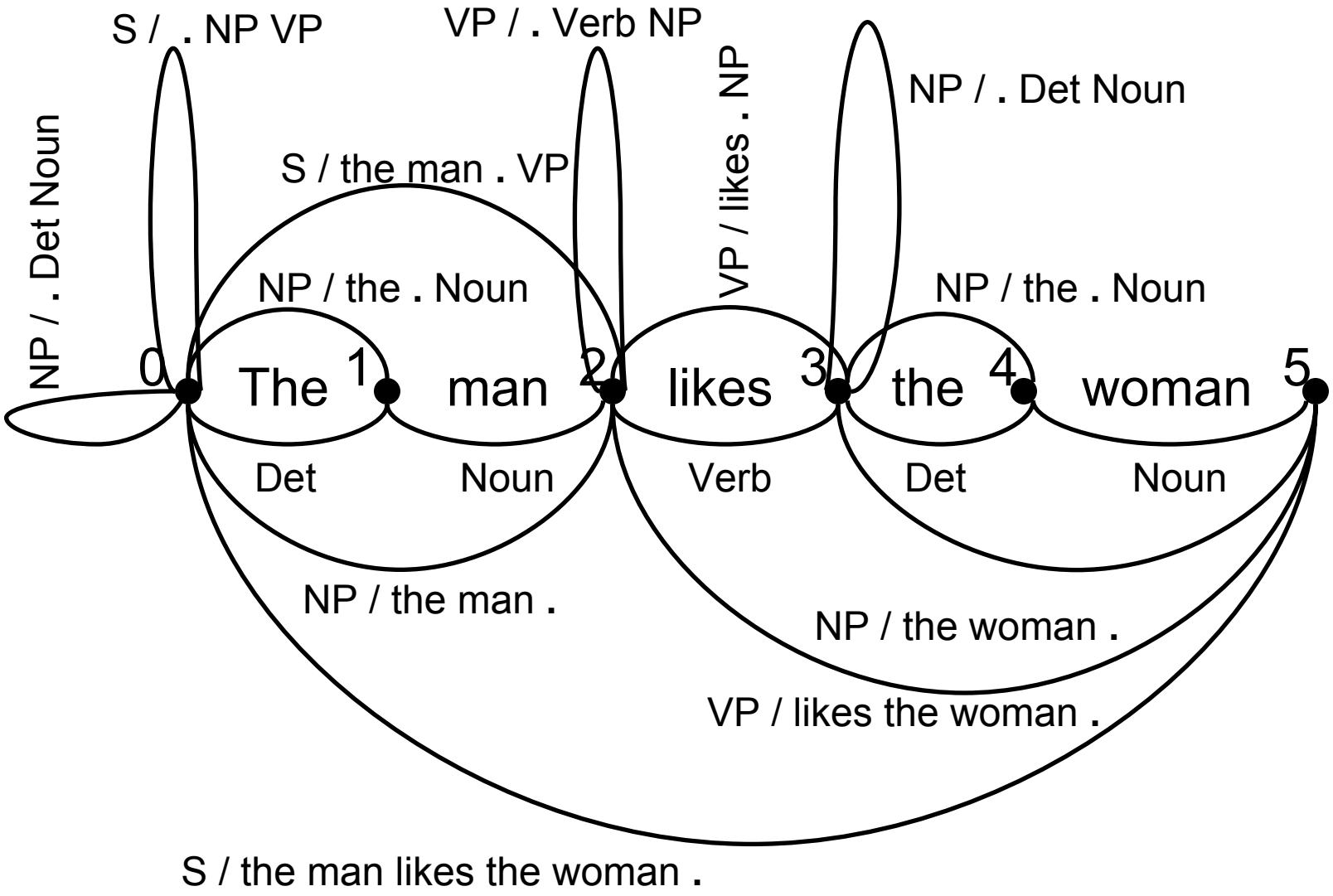


Chart parsing - example



Parsing algorithms - Summary

- Parsing is a crucial aspect of establishing **meaning** in language processing
- Basic **constraints**:
  - **Top-down:** grammar constrains which structures sentences can have in a language
  - **Bottom-up:** input constrains which rules from the grammar can apply
- **Ambiguity**
  - Structural and lexical ambiguities
  - Avoid reparsing by using a chart
  - Try out a real parser:

  http://www.link.cs.cmu.edu/link/submit-sentence-4.html

The Chomsky Hierarchy
(see Jurafsky & Martin, 2000, chapter 13)

- Regular languages: $A \Rightarrow xB$ or $A \Rightarrow x$
  - A, B non-terminals (NP, VP); x a terminal (words)
  - Linguistic example: Finite State Automata

The Chomsky Hierarchy

- Context-free languages: A $\Rightarrow$ any sequence of terminals and non-terminals
  - Linguistic example: Phrase structure grammars
  - E.g., S $\Rightarrow$ NP VP; etc.

The Chomsky Hierarchy

- Context-sensitive languages:

  $\alpha A \beta \Rightarrow \alpha \gamma \beta$, such that γ is not null

  Linguistic example: Tree-adjoining grammars (TAGs)

The Chomsky Hierarchy

- Turing-equivalent languages:

  $\alpha \Rightarrow \beta$, such that α is not null

  Linguistic example: Augmented Transition Networks (ATNs, Woods, 1970)

The Chomsky Hierarchy

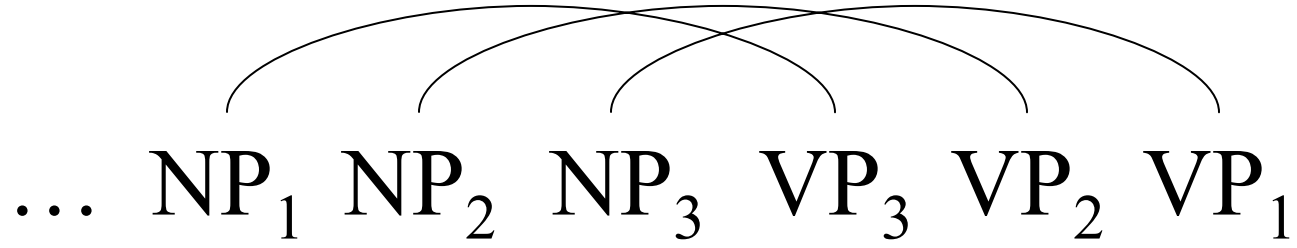
- Most linguistic phenomena can be generated by a context-free grammar or a finite-state automaton.
- Thus, most linguistic phenomena are **context-free** or **regular.**

The Chomsky Hierarchy

- Exception: Context-sensitive linguistic phenomena: Cross-serial dependencies in Swiss-German or Dutch.

Nested vs. Cross-serial dependencies

Cross-serial (Dutch)



Jeanine heeft de mannen Hans die paarden helpen leren voeren.

Joanna has the men Hans the horses helped teach feed

“Joanna helped the men teach Hans to feed the horses.”

The Chomsky Hierarchy and Complexity

- It's not clear what the relationship is between complexity and the Chomsky hierarchy.
- Context-free structures can be very hard to parse:
- S $\Rightarrow$ If S then S: If S_1 then S_2
- S $\Rightarrow$ Either S or S Either S_3 or S_4
- S $\Rightarrow$ NP who said S VP

  The man who said S_5 is arriving today.

- *If either the man who said S_5 is arriving today or the man who said S_5 is arriving tomorrow, then the man who said S_6 is arriving the day after tomorrow.*

The Chomsky Hierarchy and Complexity

- # Because if when the baby is crying the mother gets upset, the father will help, the grandmother can rest easily.
- If when the baby is crying the mother gets upset, the father will help, so the grandmother can rest easily.

  (Gibson, 1998)

2 Predictions of storage cost theory (Gibson et al., in press):

Prediction 1: People should read (1) faster than the same region in (2):

(1) **The reporter who the senator attacked ignored the president**.

(2) The fact that **the reporter who the senator attacked ignored the president** bothered the editor.

Prediction 2:

Object-modifying RCs as in (2) should be processed faster than subject-modifying RCs as in (1):

(1) The reporter **who the senator attacked** ignored the president.

(2) The president ignored the reporter **who the senator attacked.**

All theories of nesting make this prediction.

Experiment 1

Three factors:

(1) Modifier position: subject-modifying, object-modifying

(2) Extraction type: subject-extracted, object-extracted

(3) Embedding: Embedded, not embedded

Experiment 1 materials

(a) Subject modifier, object-extracted (SO), not embedded

The reporter **who the senator attacked** on Tuesday ignored the president.

(b) Object modifier, object-extracted (OO), not embedded

The president ignored the reporter **who the senator attacked** on Tuesday.

(c) Subject modifier, subject-extracted (SS), not embedded

The reporter **who attacked the senator** on Tuesday ignored the president.

(d) Object modifier, subject-extracted (OS), not embedded

The president ignored the reporter **who attacked the senator** on Tuesday.

Experiment 1 materials (continued)

(e) Subject modifier, object-extracted (SO), embedded

The fact that the reporter **who the senator attacked** on Tuesday ignored the president bothered the editor.

(f) Object modifier, object-extracted (OO), embedded

The fact that the president ignored the reporter **who the senator attacked** on Tuesday bothered the editor.

(g) Subject modifier, subject-extracted (SS), embedded

The fact that the reporter **who attacked the senator** on Tuesday ignored the president bothered the editor.

(h) Object modifier, subject-extracted (OS), embedded

The fact that the president ignored the reporter **who attacked the senator** on Tuesday bothered the editor.

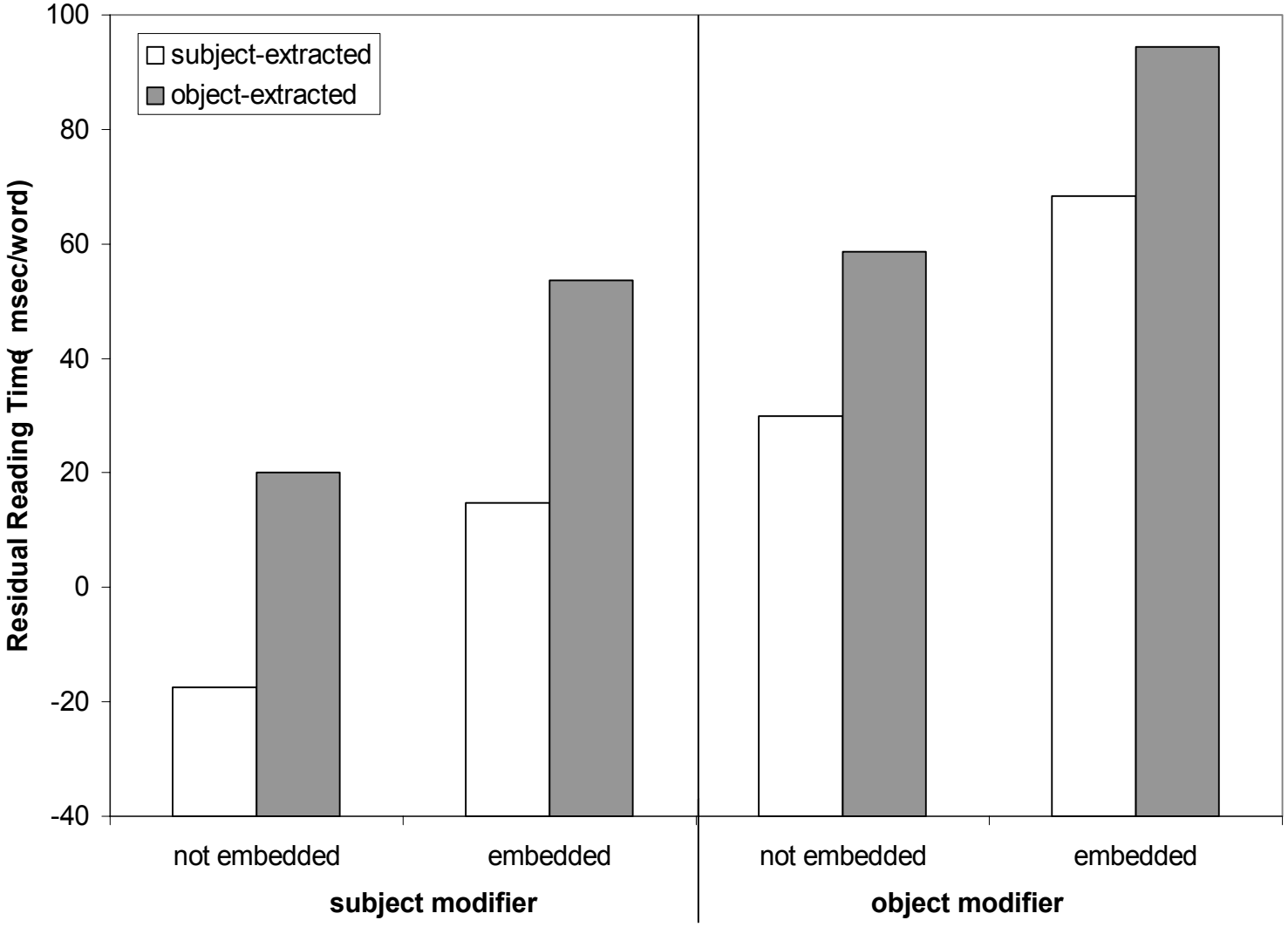
Experiment 1 predictions

(1) Subject-extractions will be processed faster than object-extractions

(2) Object-modifiers will be processed faster than subject-modifiers

(3) Unembedded RCs will be processed faster than embedded RCs

Experiment 1 residual reading times in the RC **who the senator attacked / attacked the senator**, as a function of modifier type, extraction type and embedding



Three main effects:

- Correct prediction of integration costs: subject-extractions are processed faster than object-extractions

- Correct prediction of storage costs: unembedded RCs are processed faster than embedded RCs

- **Incorrect** prediction of storage costs: subject-modifiers are processed **faster** than object-modifiers

How to account for the final result? This result is totally unpredicted by the DLT and all other resource theories.

What is a relative clause?

There are at least two kinds of RCs:

(1) The student that studied for the exam aced the test.

Restrictive RC: no intonational boundaries separating RCs

(2) Mary, who studied for the exam, aced the test.

Non-restrictive RC: intonational boundaries separating RCs

What is a relative clause?

Restrictive modifiers (e.g., (1)) usually serve a contrastive function: picking an element from a set.

I.e., They identify a particular referent from among a group of entities that contrast along the dimension denoted by the modifier.

E.g., “the student that studied for the exam” is usually used when there is a set of students out of which one can be identified by the restrictive modifier.

Consequently, the information in a restrictive RC is usually **background** information.

(More about non-restrictive RCs in a moment…)

The information flow hypothesis

Old, background information is comprehended more easily early in a sentence, such as in a position modifying the subject; New, foreground material is processed more easily later in a sentence, such as in a position in the main predicate of the sentence.

(Chafe, 1976, 1987; Du Bois, 1987; Givon, 1979, 1983, 1984; Prince, 1981)

By the information flow hypothesis, restrictive RCs should therefore be processed faster at the beginning of a sentence, where background information usually appears.

Hence, according to the information flow hypothesis, subject-modifying RCs will be processed faster that object-modifying RCs, as observed in Experiment 1.

Non-restrictive modifiers

(2) Mary, who studied for the exam, aced the test.

Non-restrictive modifiers add extra "aside" information. They do **not** function as identifying information in a set denoted by the head noun:

(3) My father, who ate ham this morning, became extremely ill.

(4) The sun, which rises in the east, can be used to orient oneself.

Non-restrictive modifiers

Cues to a restrictive RC in English: no pauses (commas) around the RC; the use of the complementizer “that”

Cues to a non-restrictive RC in English: pauses (commas) around the RC; the use of wh-pronouns “who” or “which”

(3) My father, who ate ham this morning, became extremely ill.

(5) * My father that ate ham this morning became extremely ill.

Prediction:

Non-restrictive RCs, which introduce some new material, and are not typically background material (not really foreground either: more like asides), should be processed differently than restrictive RCs.

Experiment 2 materials:

(4) (a) Subject-modifier, restrictive

A group of film critics praised a director at a banquet and another director at a film premiere. The director **that the critics praised at a banquet** insulted an actor from a big action movie during an interview.

(b) Object-modifier, restrictive

A group of film critics praised a director at a banquet and another director at a film premiere. An actor from a big action movie insulted the director **that the critics praised at a banquet** during an interview.

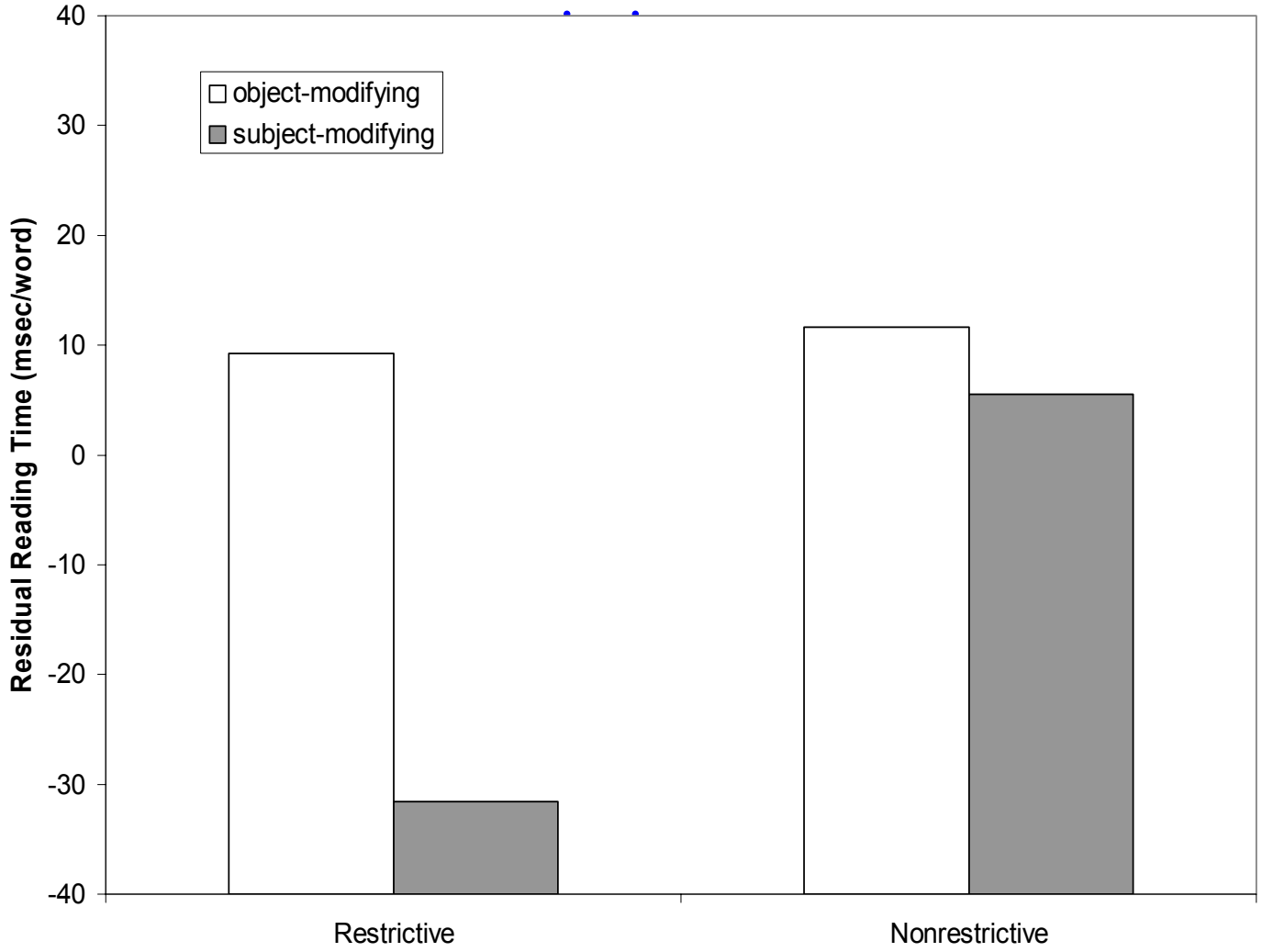
(c) Subject-modifier, non-restrictive

A group of film critics praised a director and a producer. The director, **who the critics praised at a banquet**, insulted an actor from a big action movie during an interview.

(d) Object-modifier, non-restrictive

A group of film critics praised a director and a producer. An actor from a big action movie insulted the director, **who the critics praised at a banquet**, during an interview.

Experiment 2 residual reading times in the RC, as a function of modifier type and



Results

- Replication of Experiment 1: subject-modifying restrictive RCs are read faster than object-modifying restrictive RCs

- Non-restrictives do not show the same advantage: no difference between the two (interaction between RC type (restrictive, non-restrictive) and modifier position (subject, object)

Local Conclusion

Information flow is another important factor in determining sentence complexity.

Background information is processed more easily early in a sentence. Foreground / new information is processed more easily later in a sentence.