# Introduction to Natural Language Engineering / Part 11: Parsing & Logical Representation

Udo Kruschwitz / Bernd Ludwig
Lehrstuhl für Informationswissenschaft

WS 2020/21

Universität Regensburg

# Einführung in die Informationslinguistik I / Teil 11: Syntaxanalyse & Einfache Satzsemantik

Udo Kruschwitz / Bernd Ludwig
Lehrstuhl für Informationswissenschaft

WS 2020/21

Universität Regensburg

# Module Overview

- Motivation
- Regular expressions
- Basic statistical natural language processing
- Part-of-speech tagging
- Context-free grammars
- Parsing principles
- Complexity
- Semantics
- Applications: IE, IR, QA, …

# Module Overview (more specific)

- Motivation
- Regular expressions
- Basic statistical natural language processing
- Part-of-speech tagging
- Text classification
- Lexical semantics (embeddings)
- Context-free grammars
- Parsing principles + Complexity
- Applications: IE, IR, QA, …

# Following on from last time …

- Formal grammars can be used to describe a language
- How do we find out whether a sentence is part of that language?
- That’s what a parser will do …

# Parsing

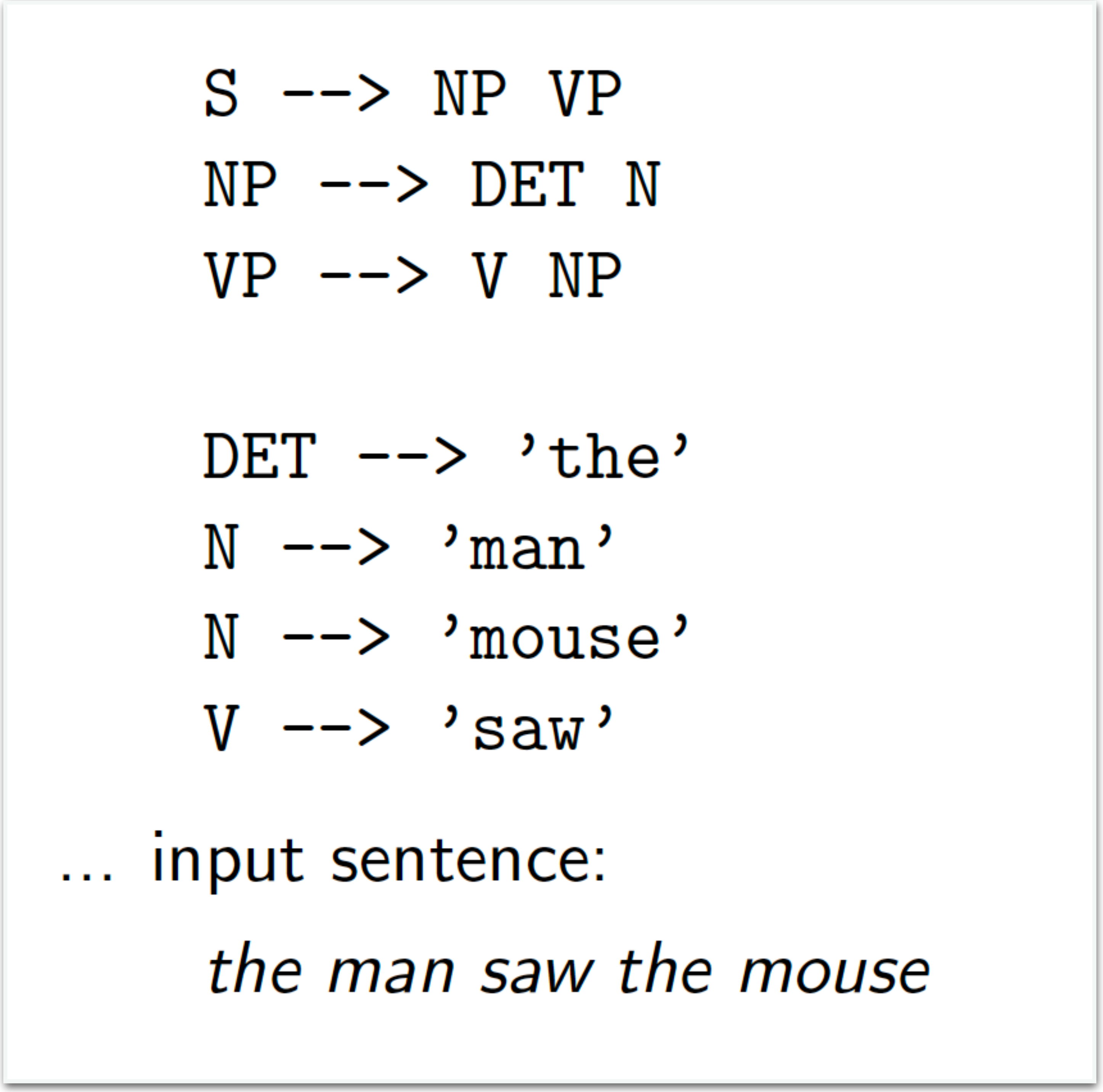
# Parsing: Overview

- Parser takes a grammar and an input string and returns possible analyses of that string
- Parsing is a search problem
- Three criteria for evaluating parsers:
  - Correctness
  - Completeness
  - Efficiency
- Parsing strategies:
  - Top-down vs. Bottom-Up
  - Breadth-first vs. Depth-first

# Top-Down Parsing - Strategy

- Start from S (*goal-driven*)
- Look for rules that have S as left-hand side and replace S by the right-hand side of the rule
- Progressively refine structures by performing this for the resulting string replacing *non-terminals* by right-hand sides of rules
- Finished when the result finally matches the input sentence

# Top-Down Parsing - Example Grammar



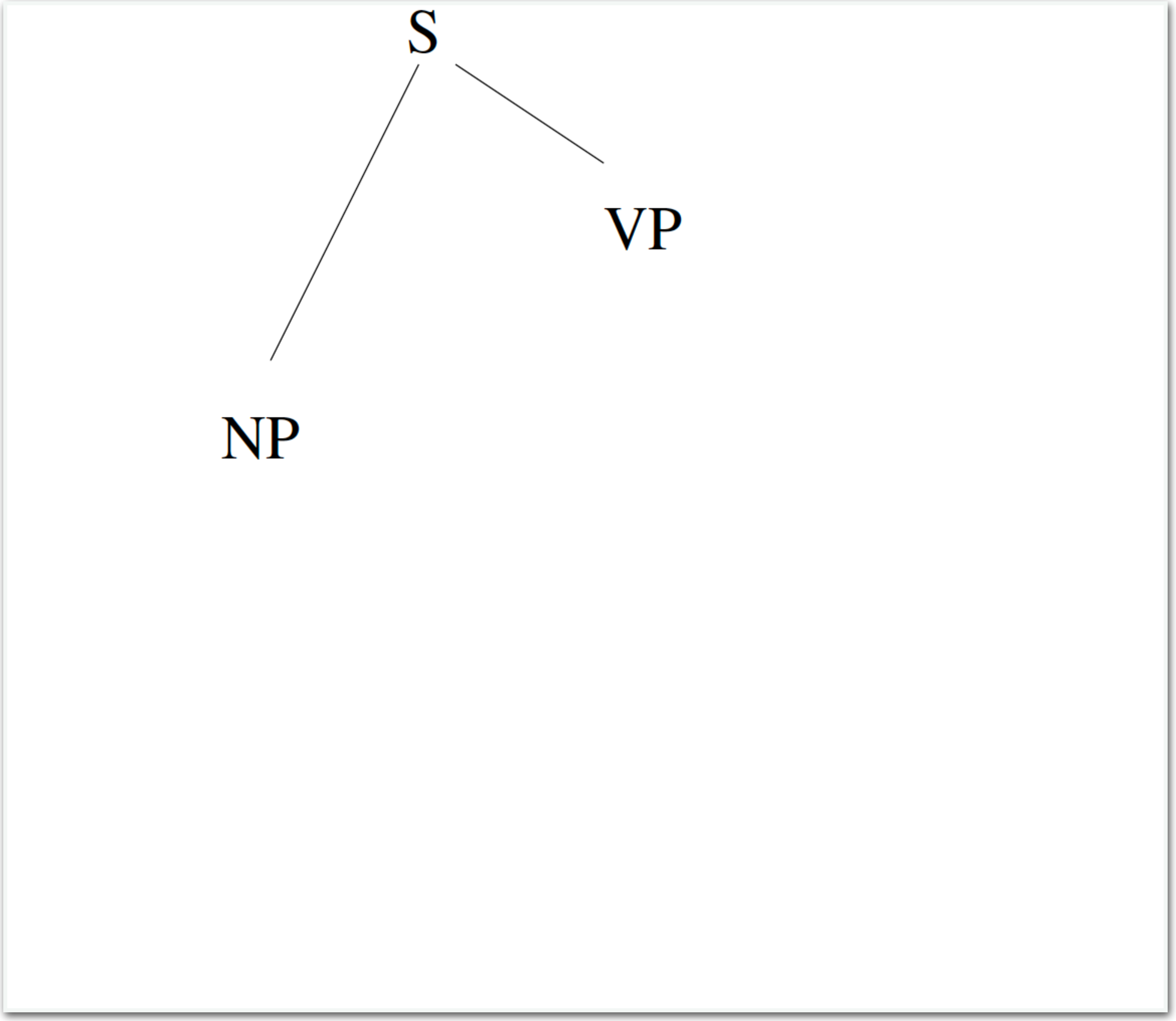
```
S --> NP VP
NP --> DET N
VP --> V NP

DET --> 'the'
N --> 'man'
N --> 'mouse'
V --> 'saw'
```

… input sentence:

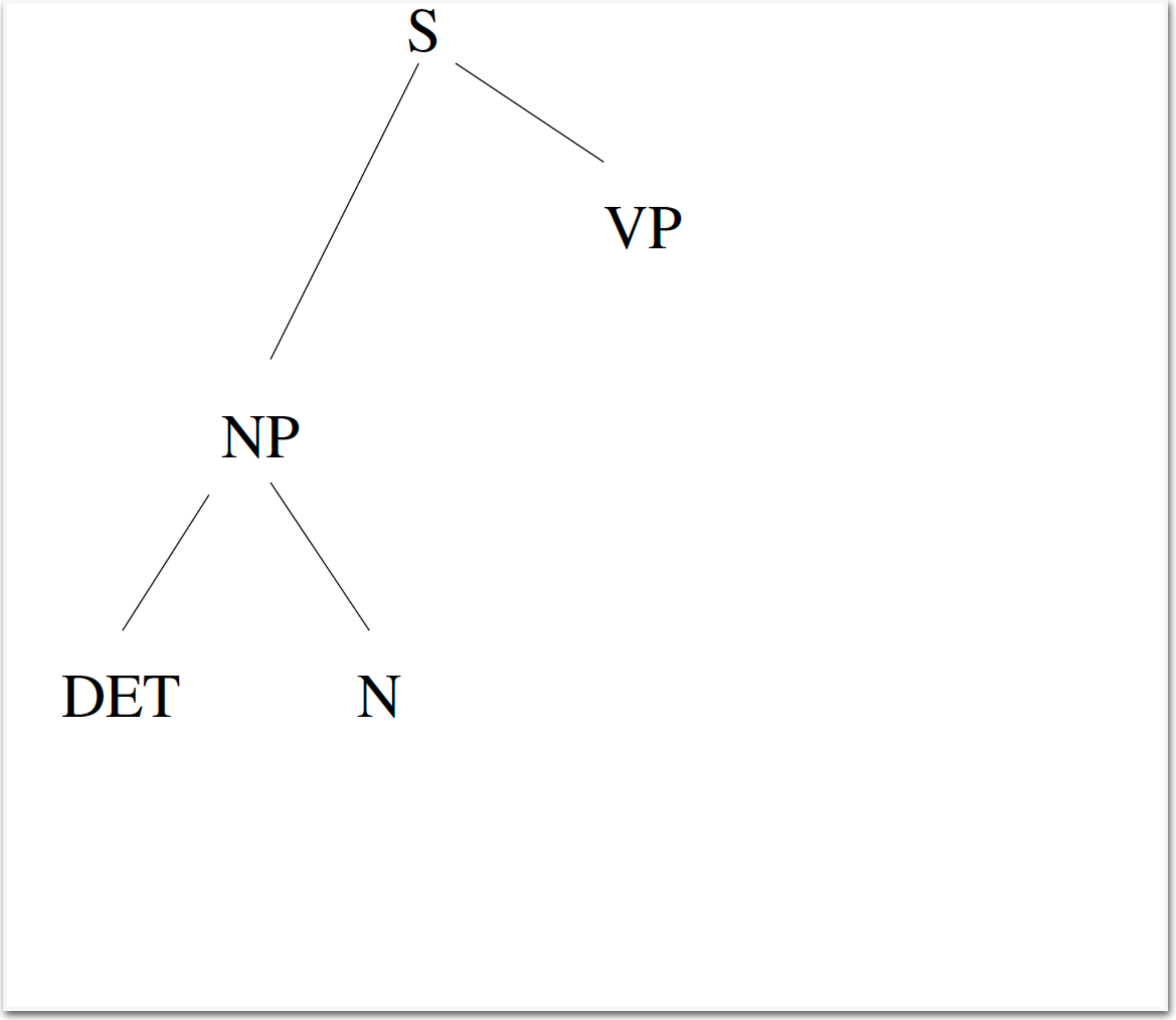
*the man saw the mouse*

# Top-Down Parsing - Example

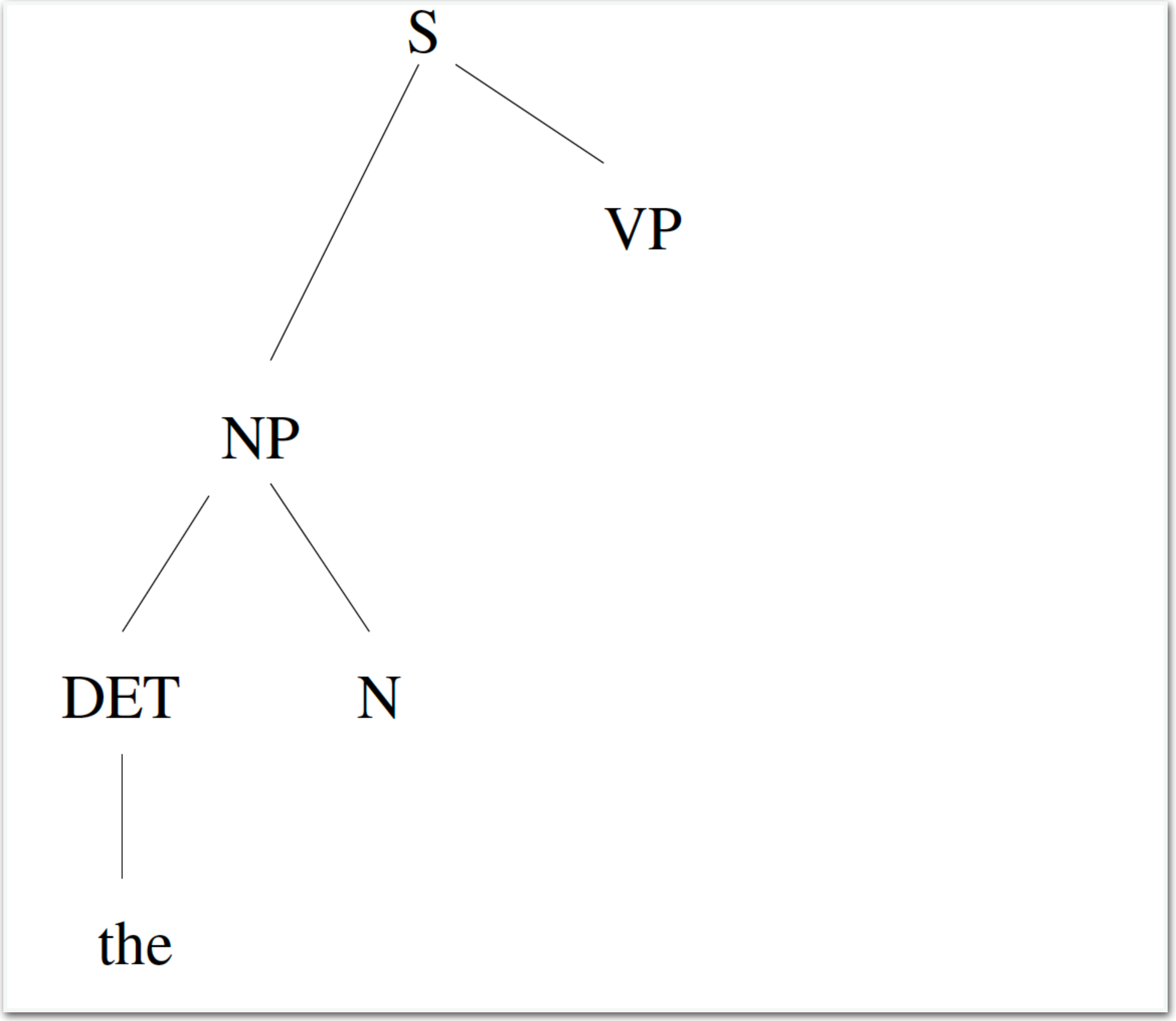
# Top-Down Parsing - Example



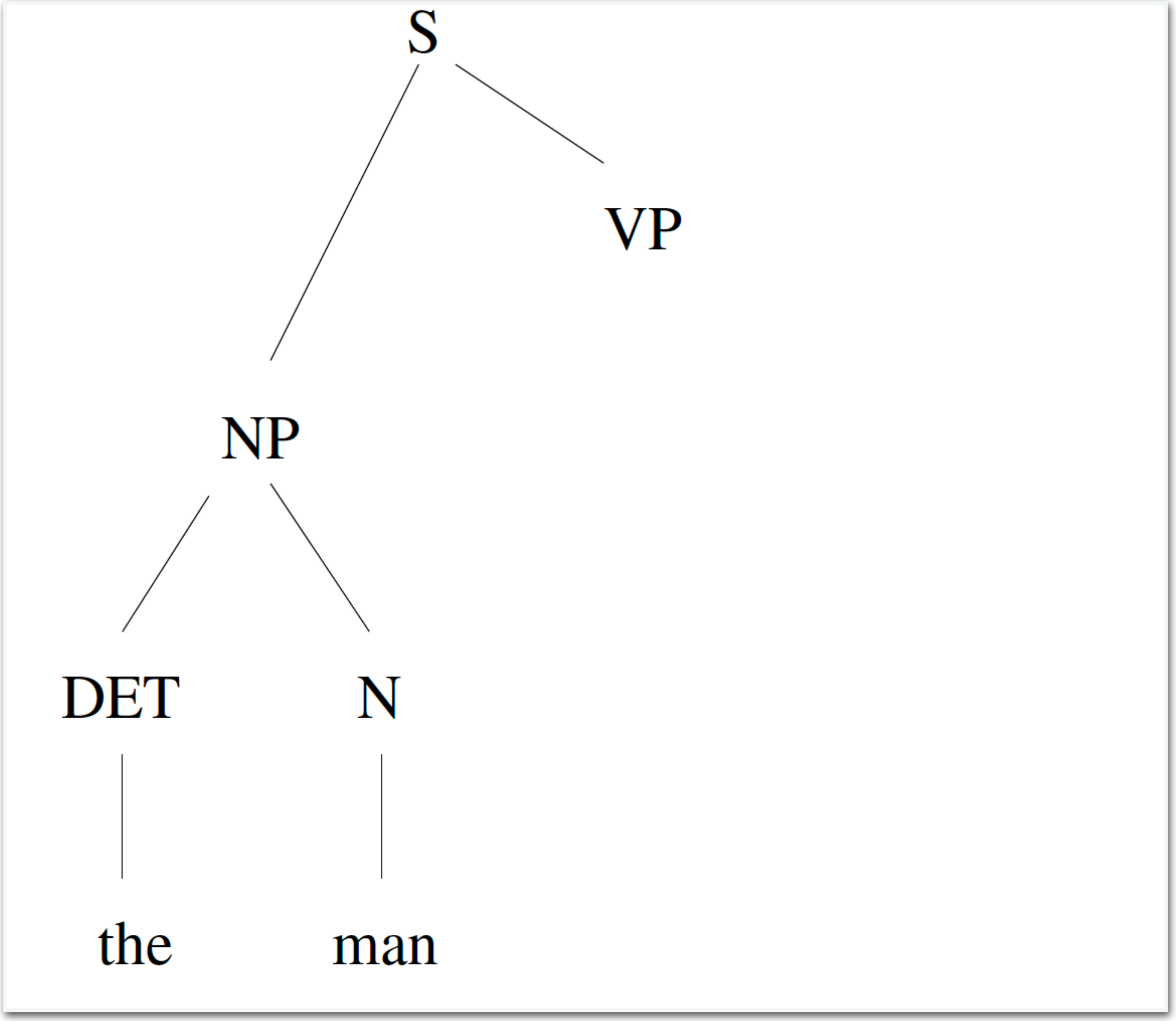
# Top-Down Parsing - Example



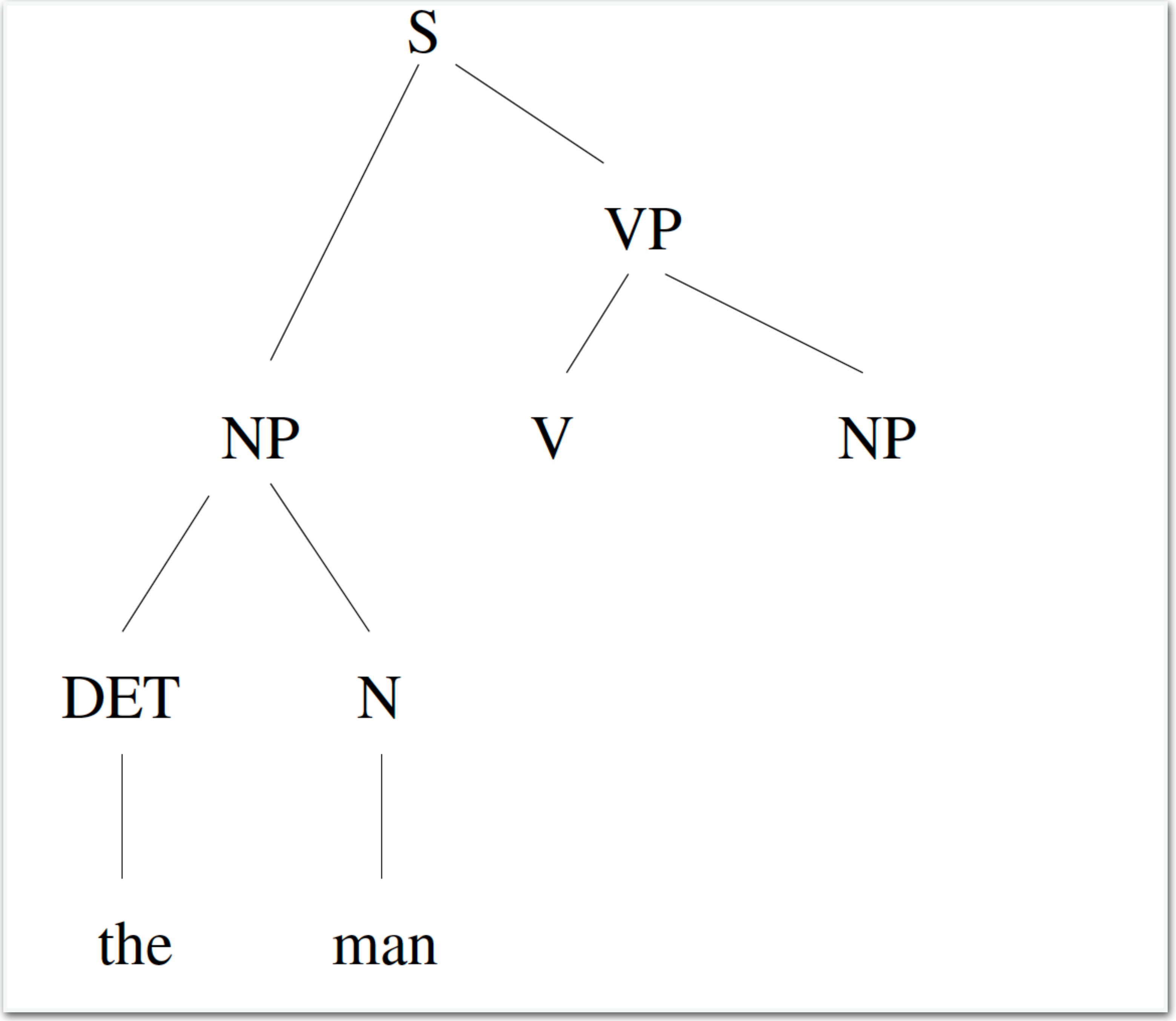
# Top-Down Parsing - Example



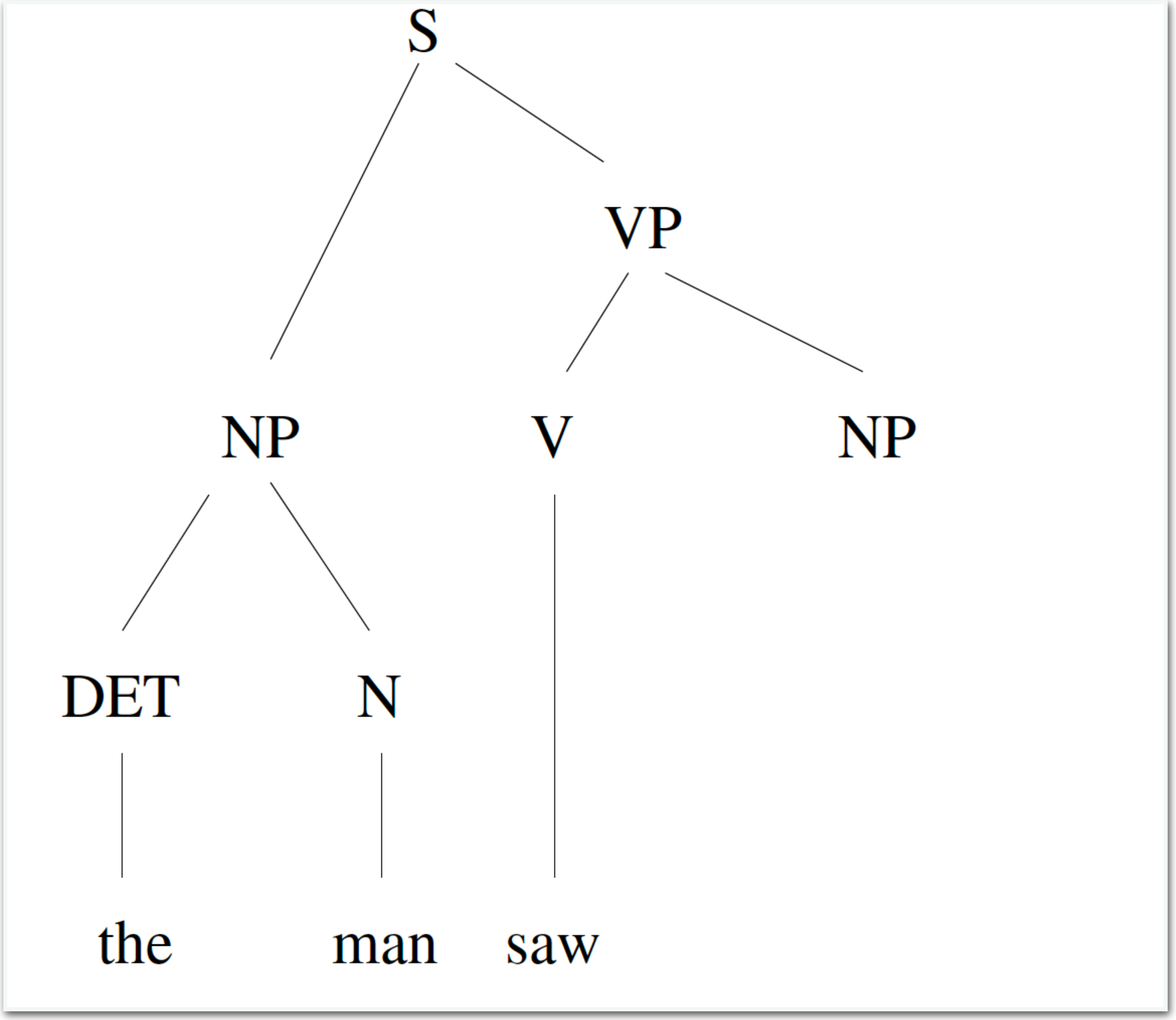
# Top-Down Parsing - Example



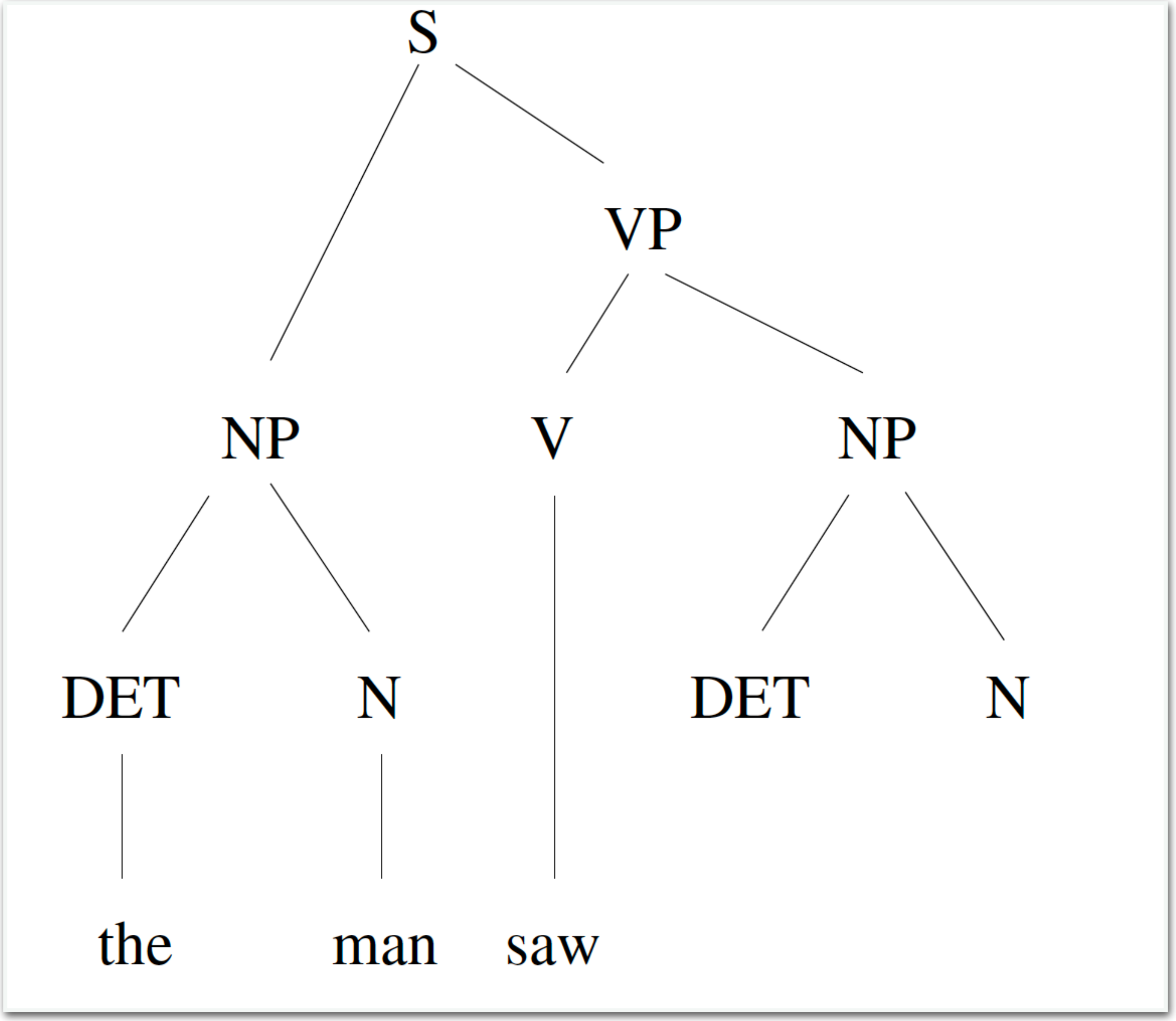
# Top-Down Parsing - Example



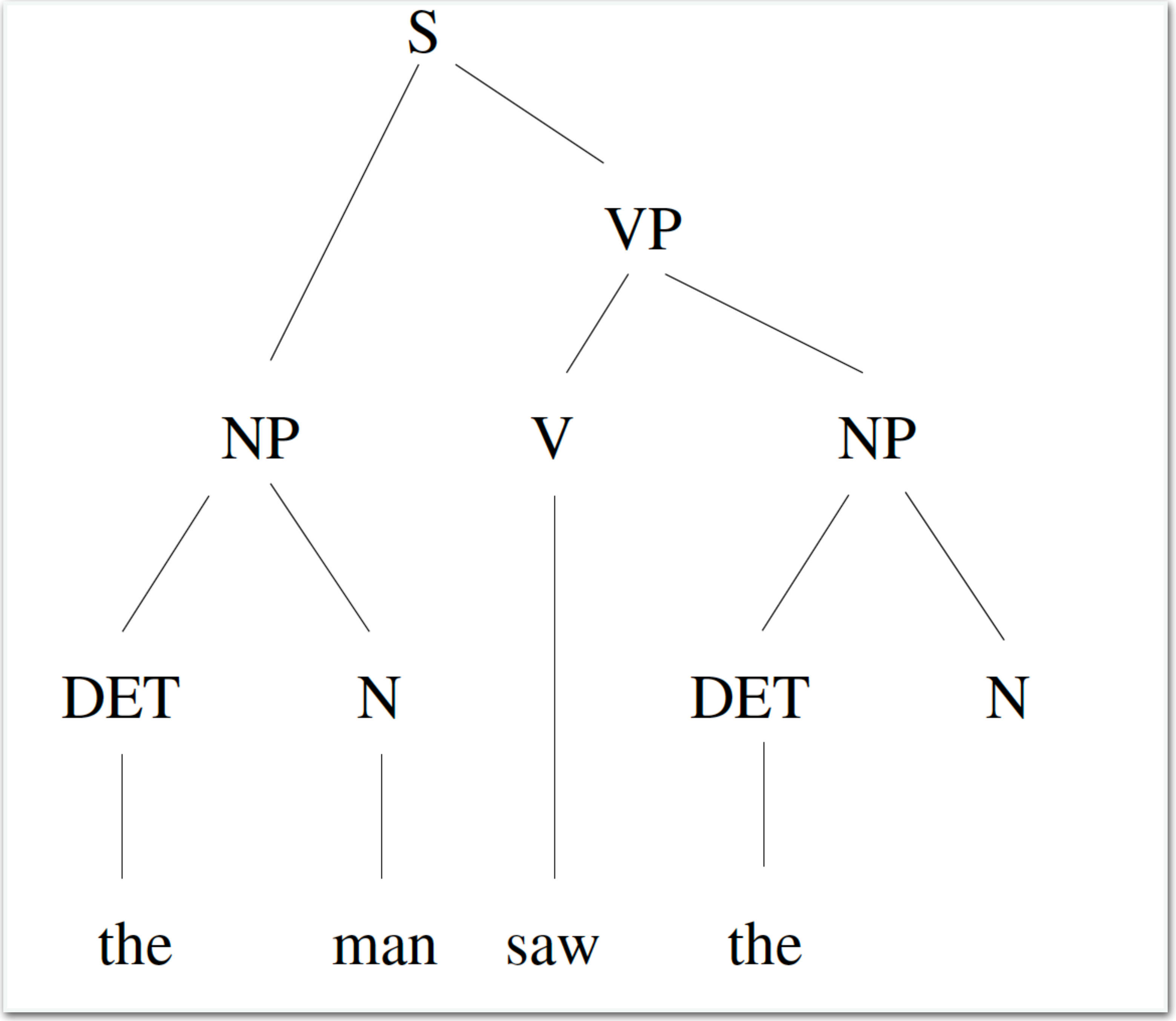
# Top-Down Parsing - Example



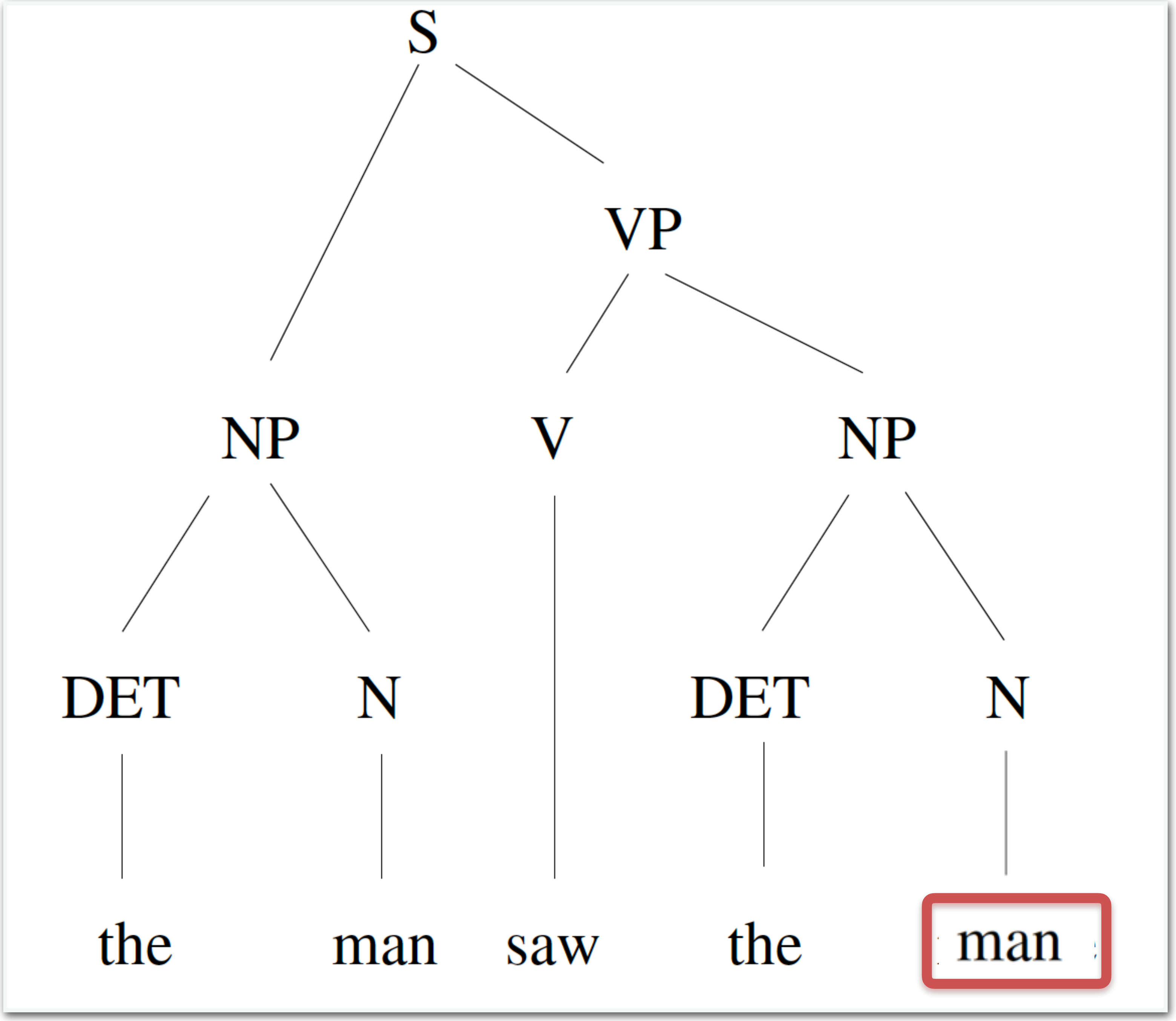
# Top-Down Parsing - Example



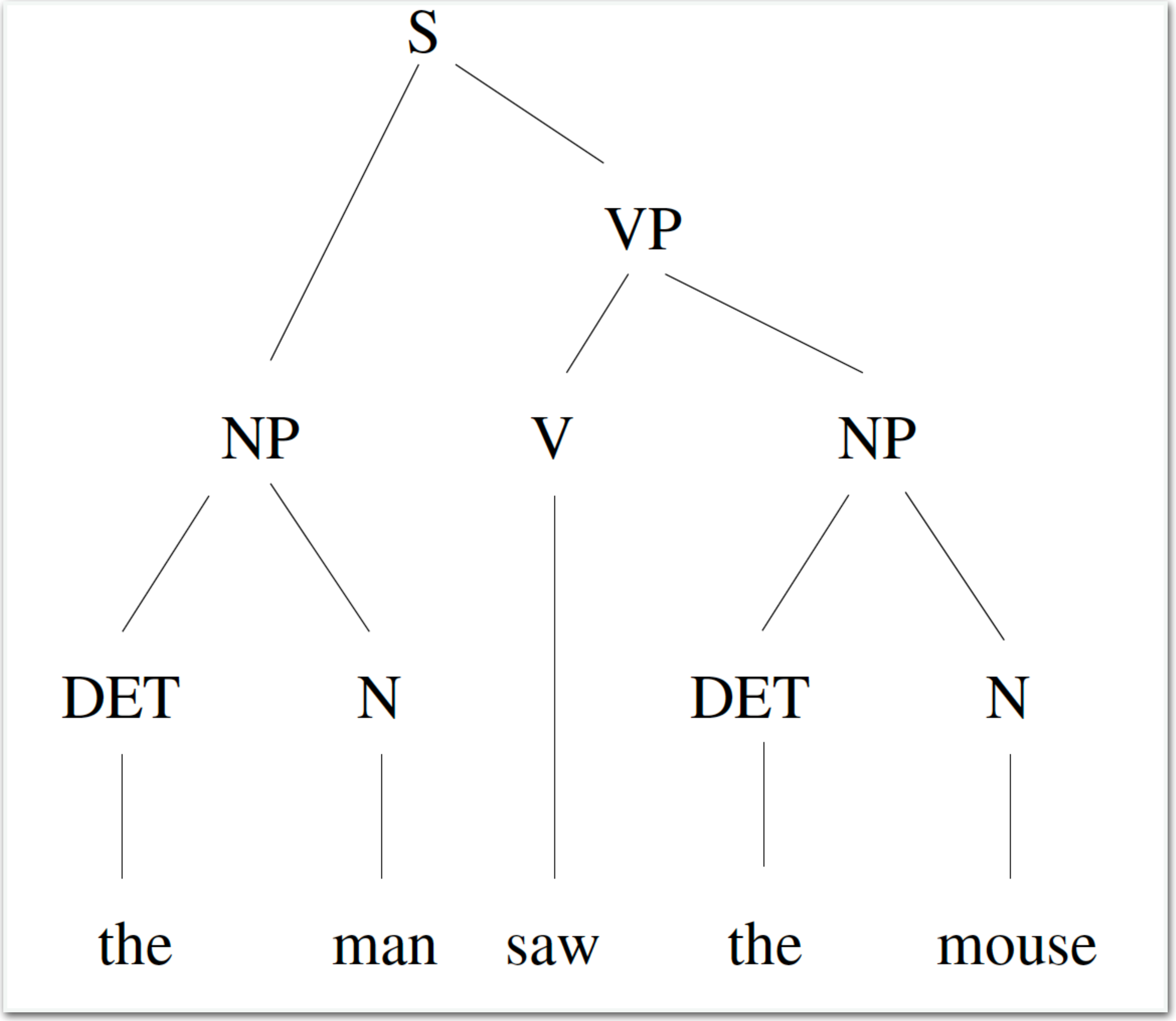
# Top-Down Parsing - Example



# Top-Down Parsing - Example



# Top-Down Parsing - Example



# Top-Down Parsing - Problems

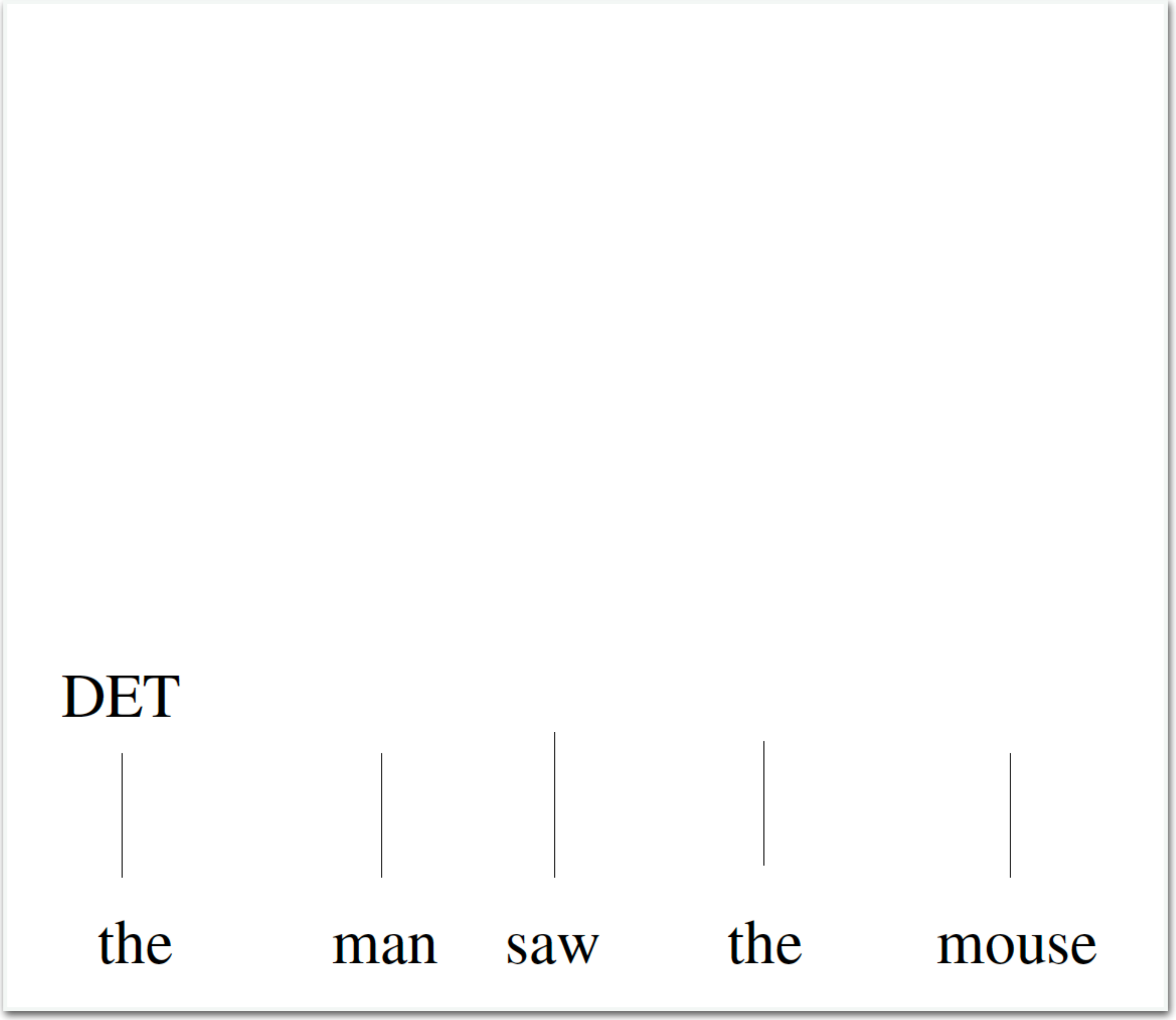
- Left recursion
- Structural ambiguity

# Bottom-Up Parsing - Strategy

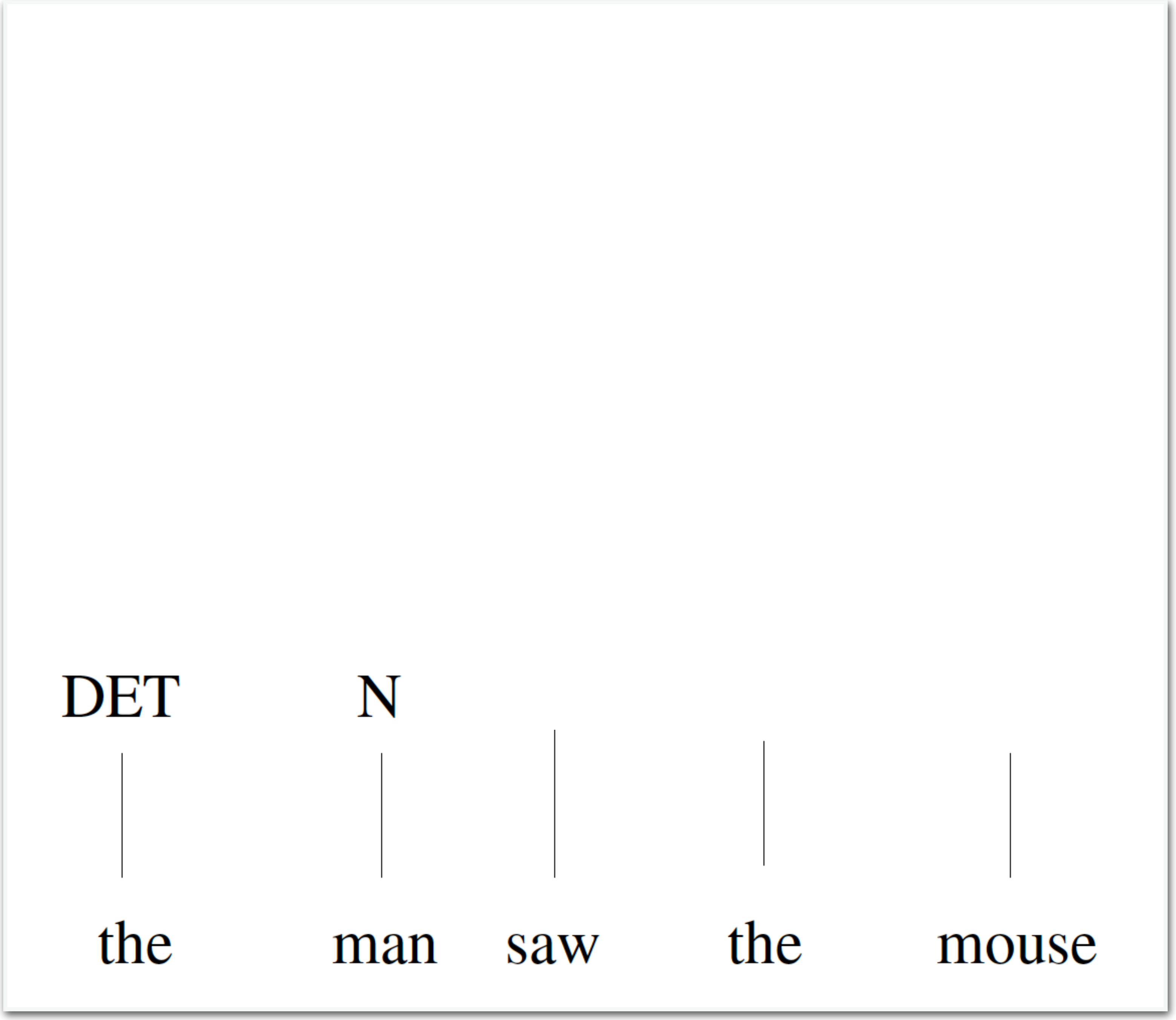
- Start from word level (*data-driven*)
- Progressively building up structures
- Find strings in the input that are right-hand sides of rules and can be replaced by the corresponding left-hand side
- Finished when the result is S

# Bottom-Up Parsing - Example

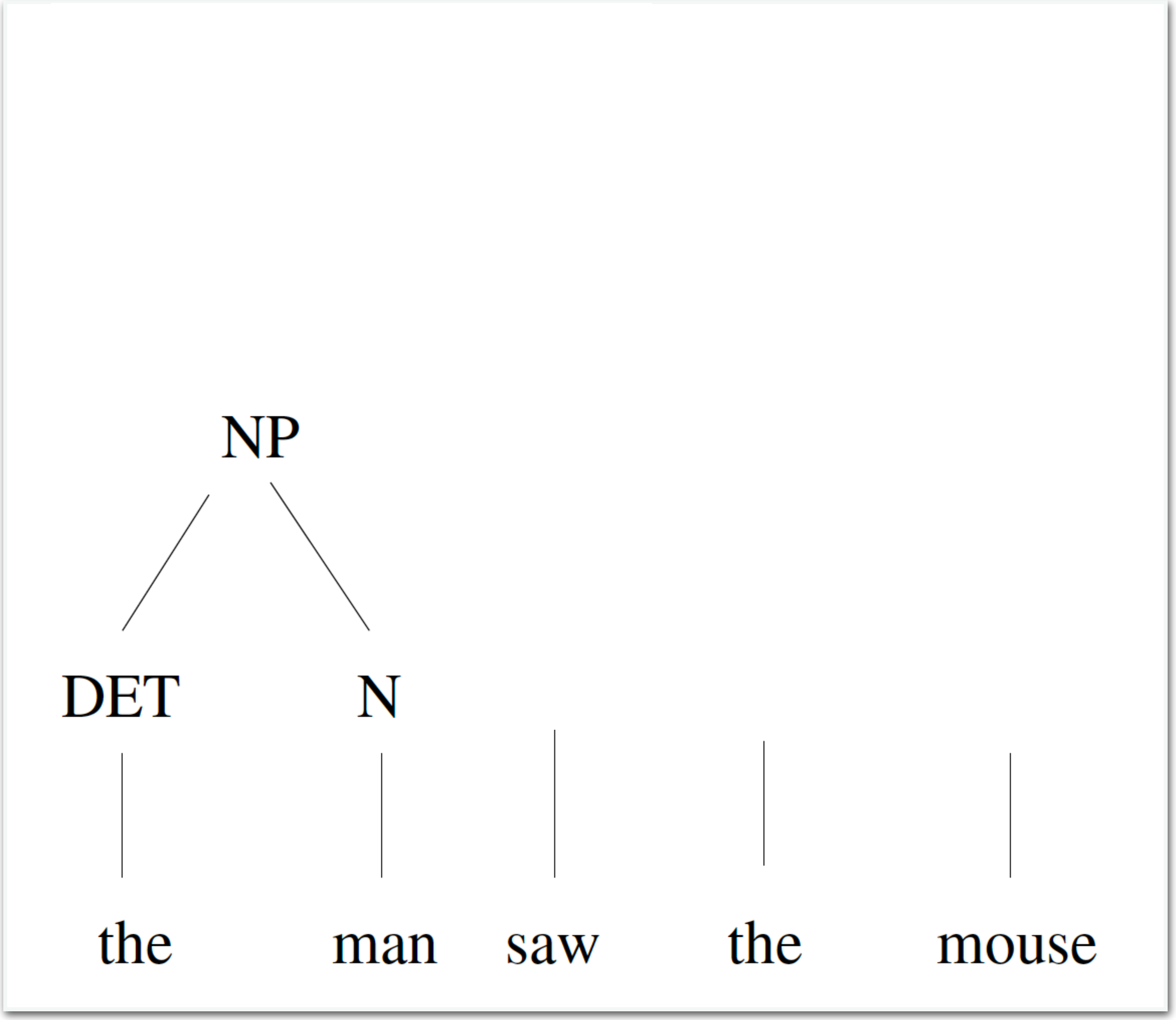
# Bottom-Up Parsing - Example



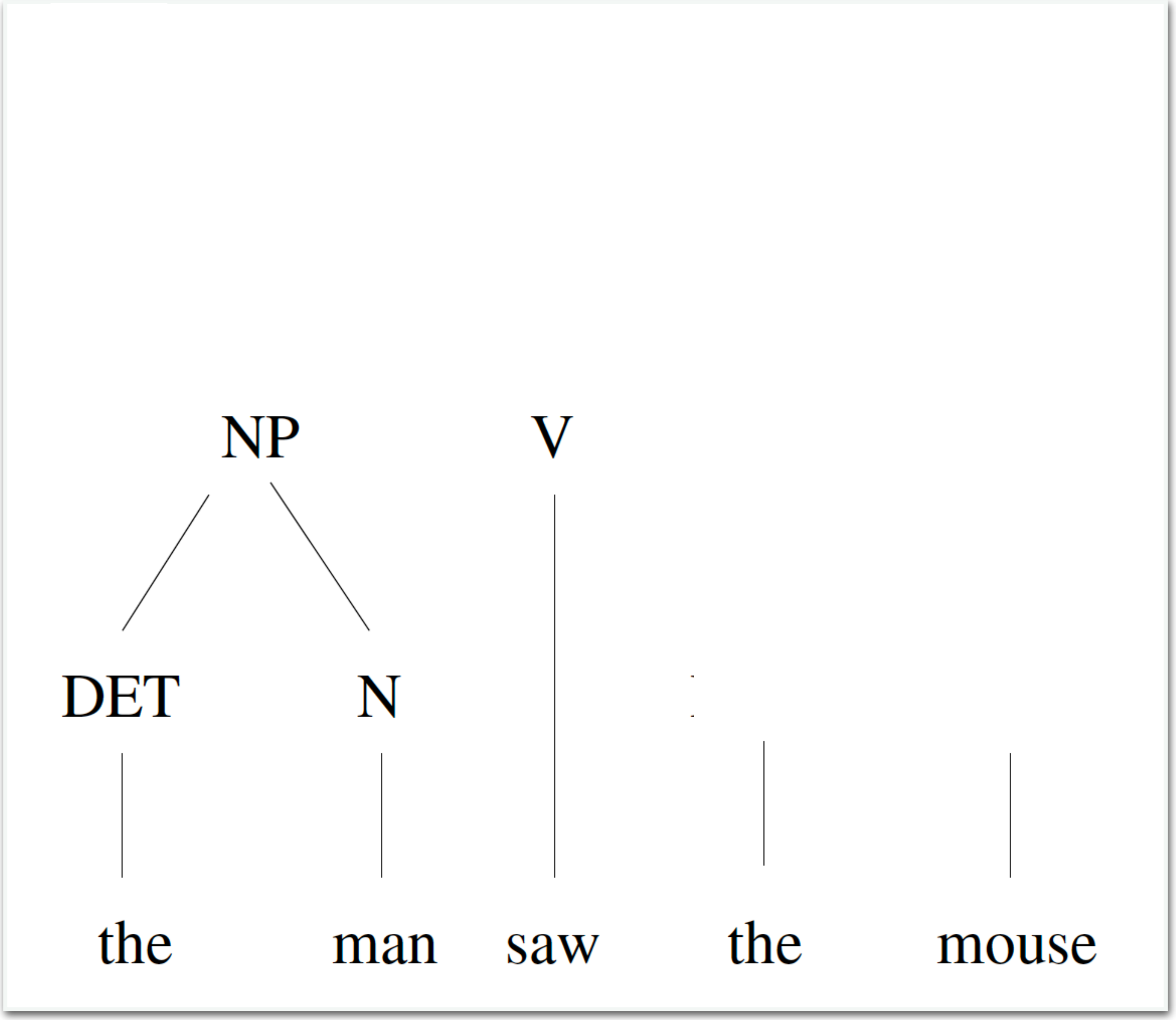
# Bottom-Up Parsing - Example



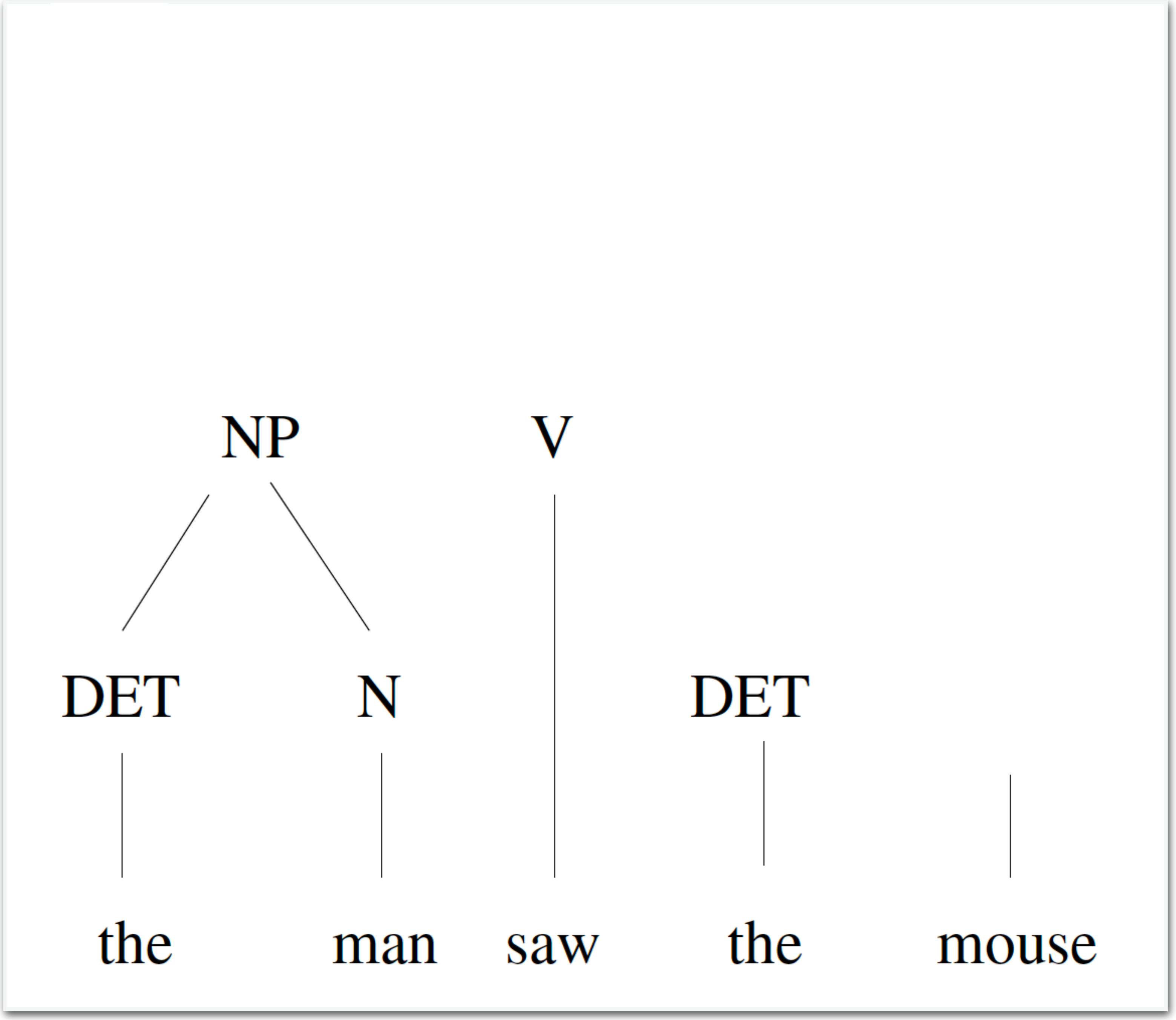
# Bottom-Up Parsing - Example



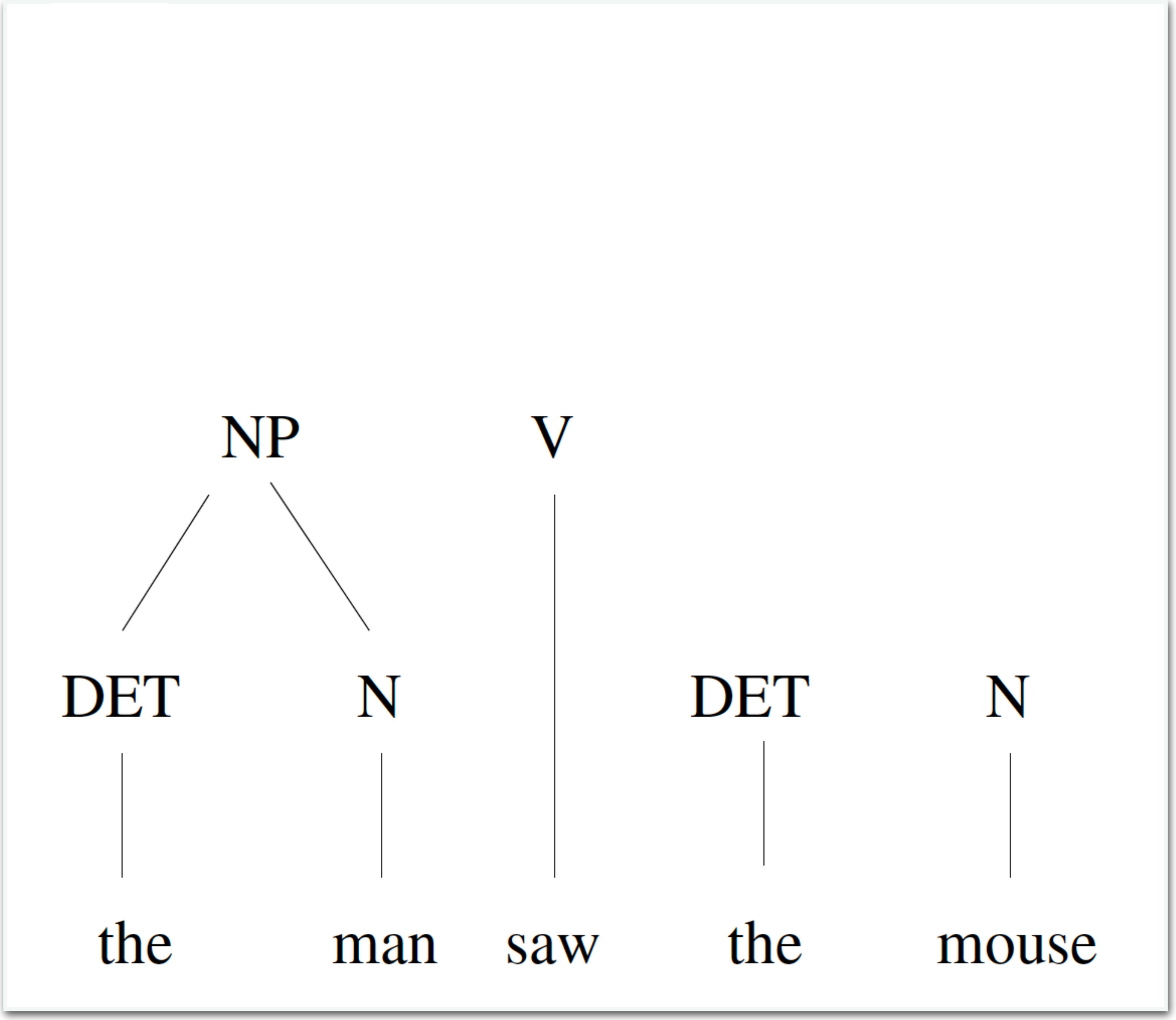
# Bottom-Up Parsing - Example



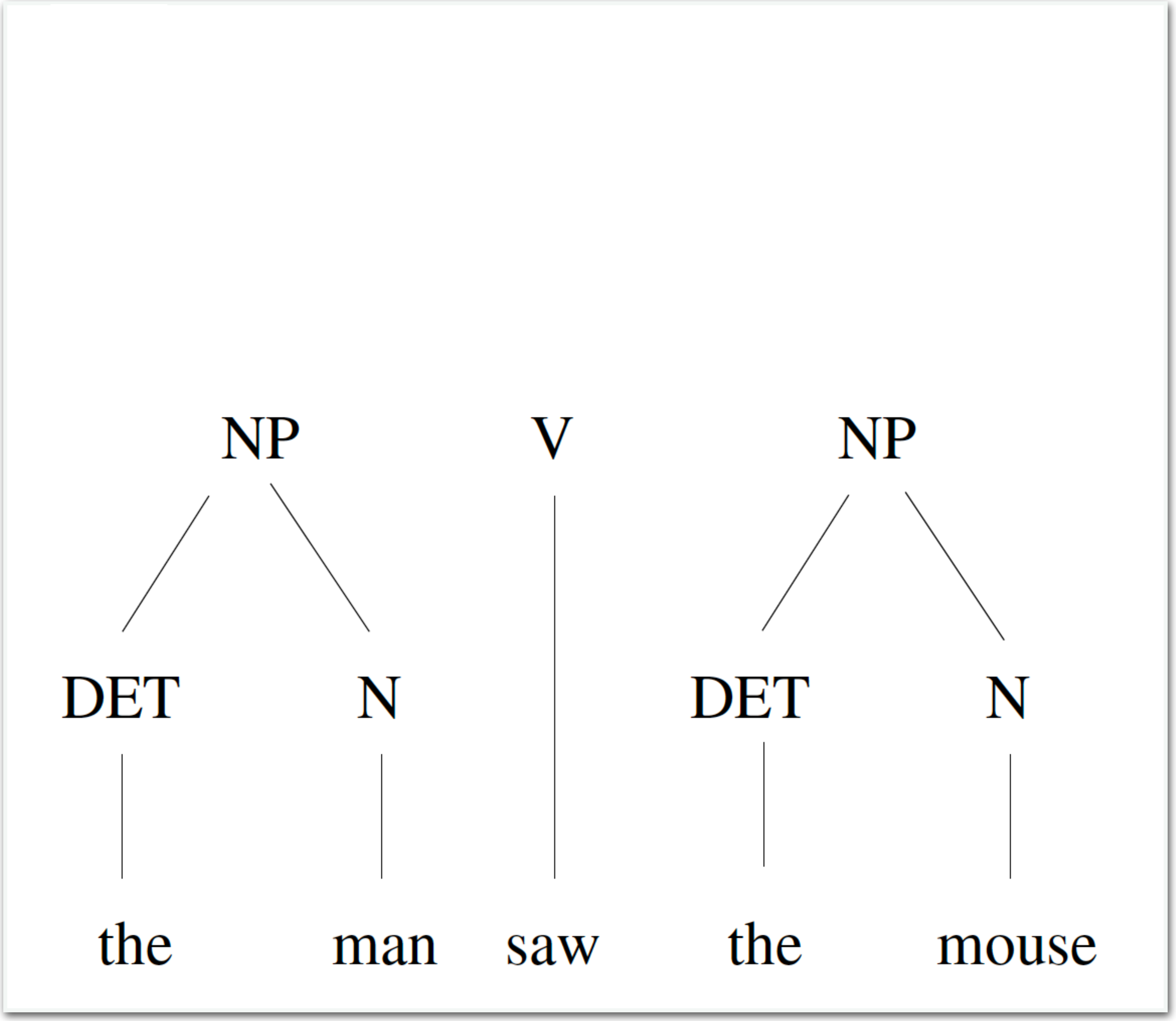
# Bottom-Up Parsing - Example



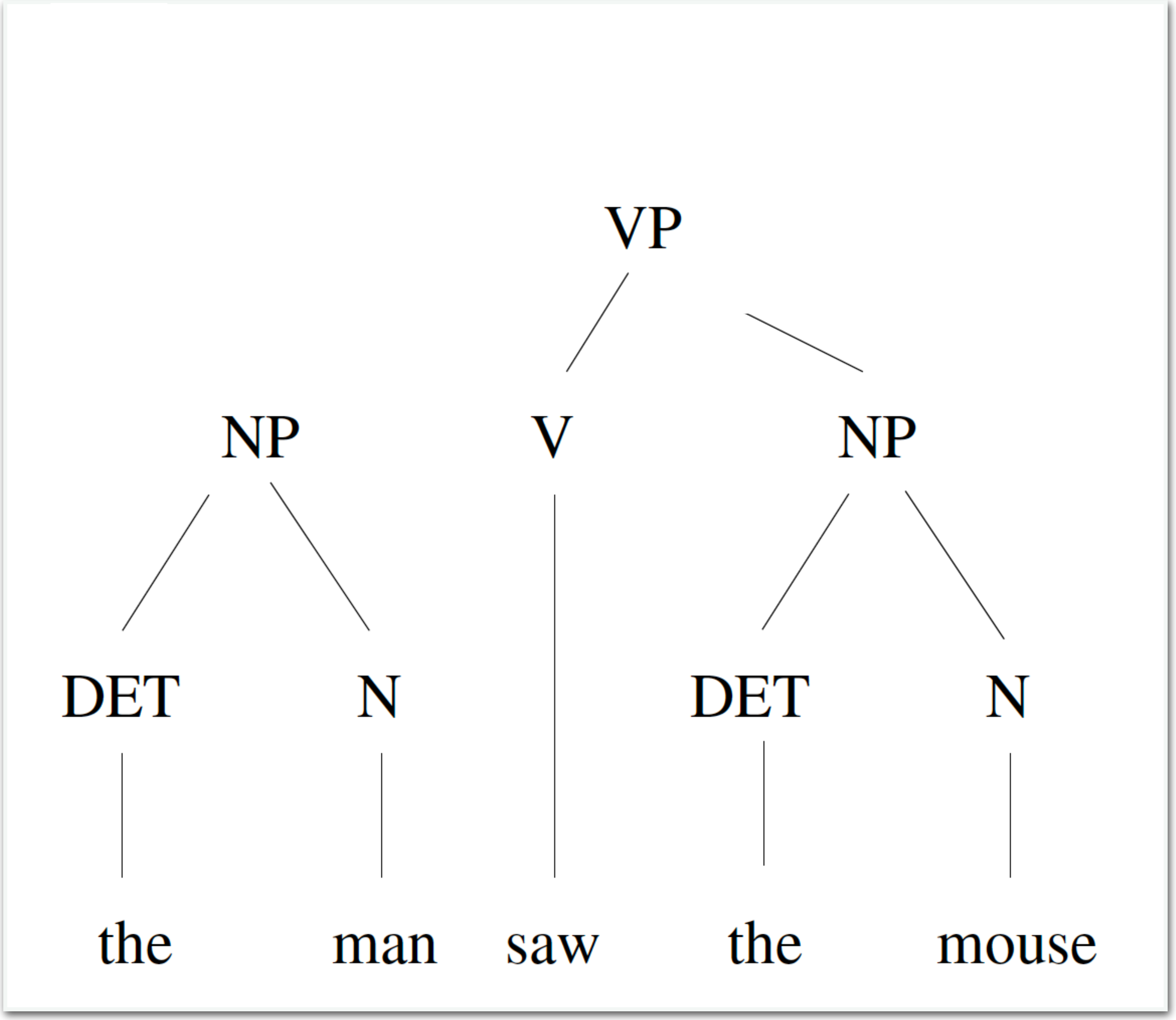
# Bottom-Up Parsing - Example



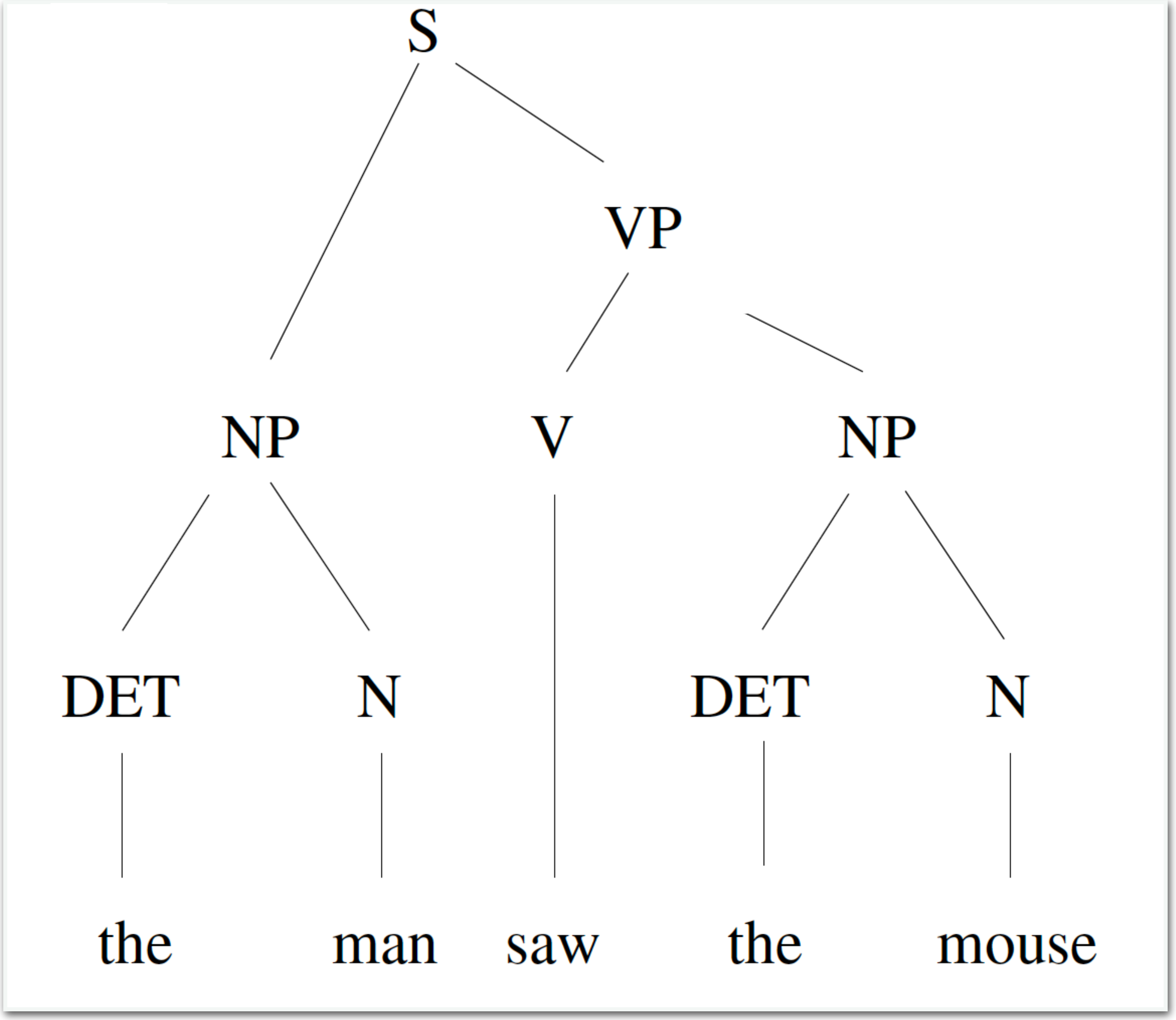
# Bottom-Up Parsing - Example



# Bottom-Up Parsing - Example



# Bottom-Up Parsing - Example



# Bottom-Up Parsing - Problems

- ε-production
- Lexical ambiguity

# Chart Parsing - Motivation

- Problems with top-down and bottom-up parsers, e.g.:
    - Left recursion
    - Ambiguity (structural, lexical category etc.)
    - Inefficiency (backtracking)

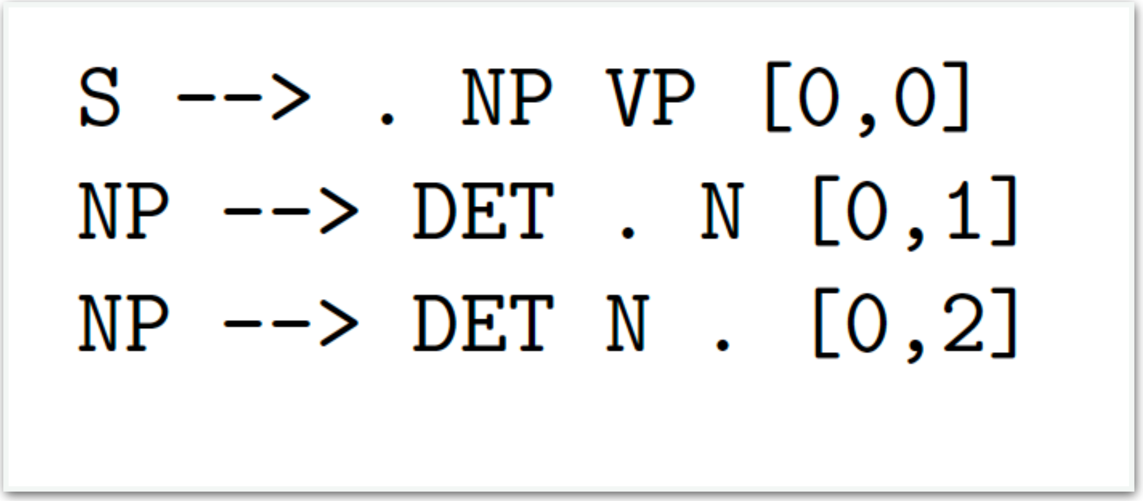
# Chart Parsing - Strategy

- Record all partial parses
- Build up subtrees and keep them in a table (chart)
- Keep only one instance of each chart entry
- Chart entries are never deleted
- No backtracking
- End of the sentence: chart contains all possible parses
- Example algorithms: *Earley* algorithm (top-down); *CKY* algorithm (bottom-up)

# Earley Algorithm - Overview

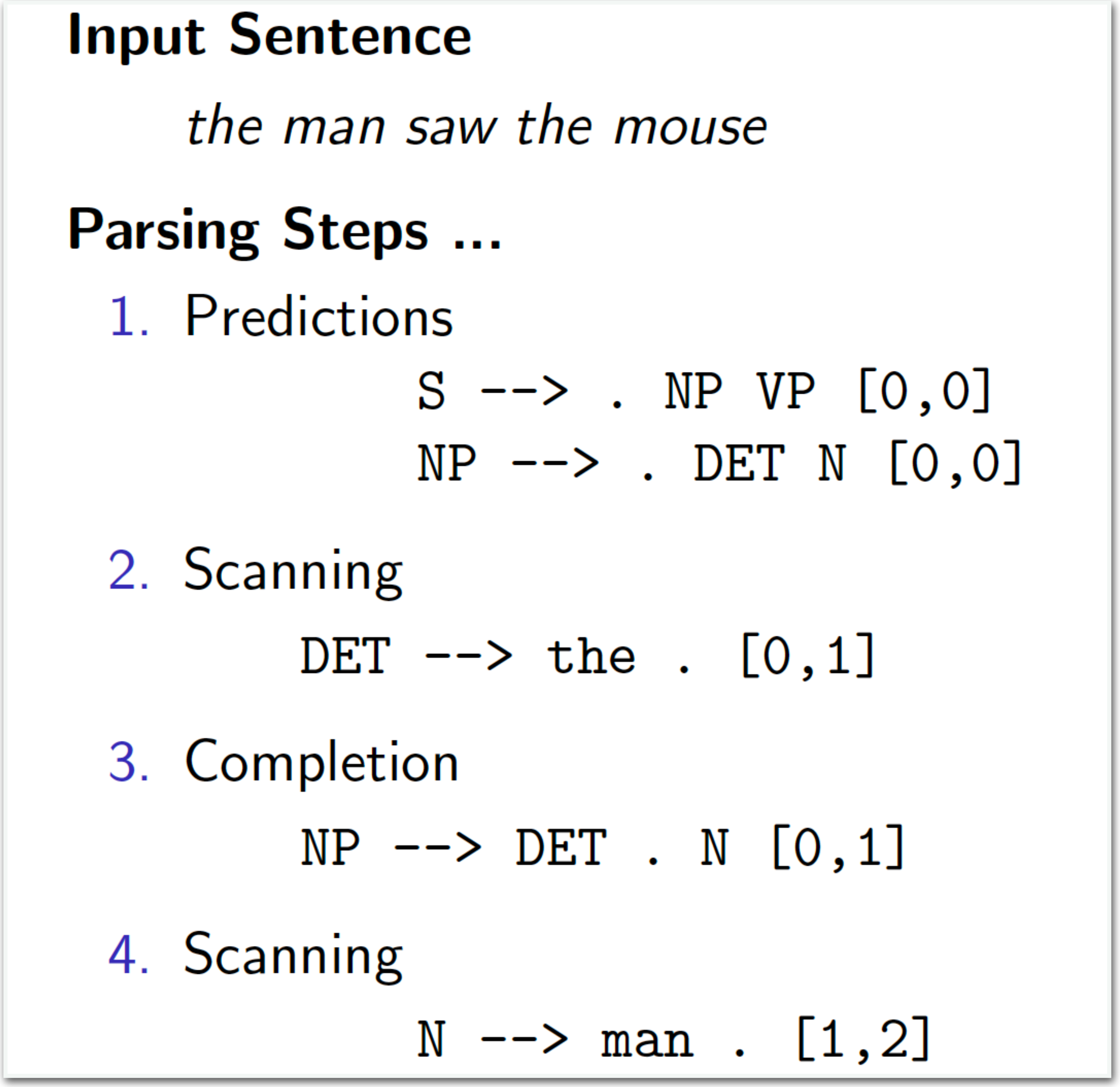
- Left-to-right top-down parsing
- Chart entries (dotted rules) consist of:
    - Subtree corresponding to a grammar rule
    - Information about how much of this rule has been found
    - Position of subtree in respect to input
- Three operators:
    - Predictor
    - Scanner
    - Completer
- See detailed examples in the book

# Earley Algorithm - Examples (Chart Entries)



```
S --> . NP VP [0,0]
NP --> DET . N [0,1]
NP --> DET N . [0,2]
```

# Earley Algorithm - Example



**Input Sentence**

*the man saw the mouse*

**Parsing Steps ...**

1. Predictions

```
S --> . NP VP [0,0]
NP --> . DET N [0,0]
```

2. Scanning

```
DET --> the . [0,1]
```

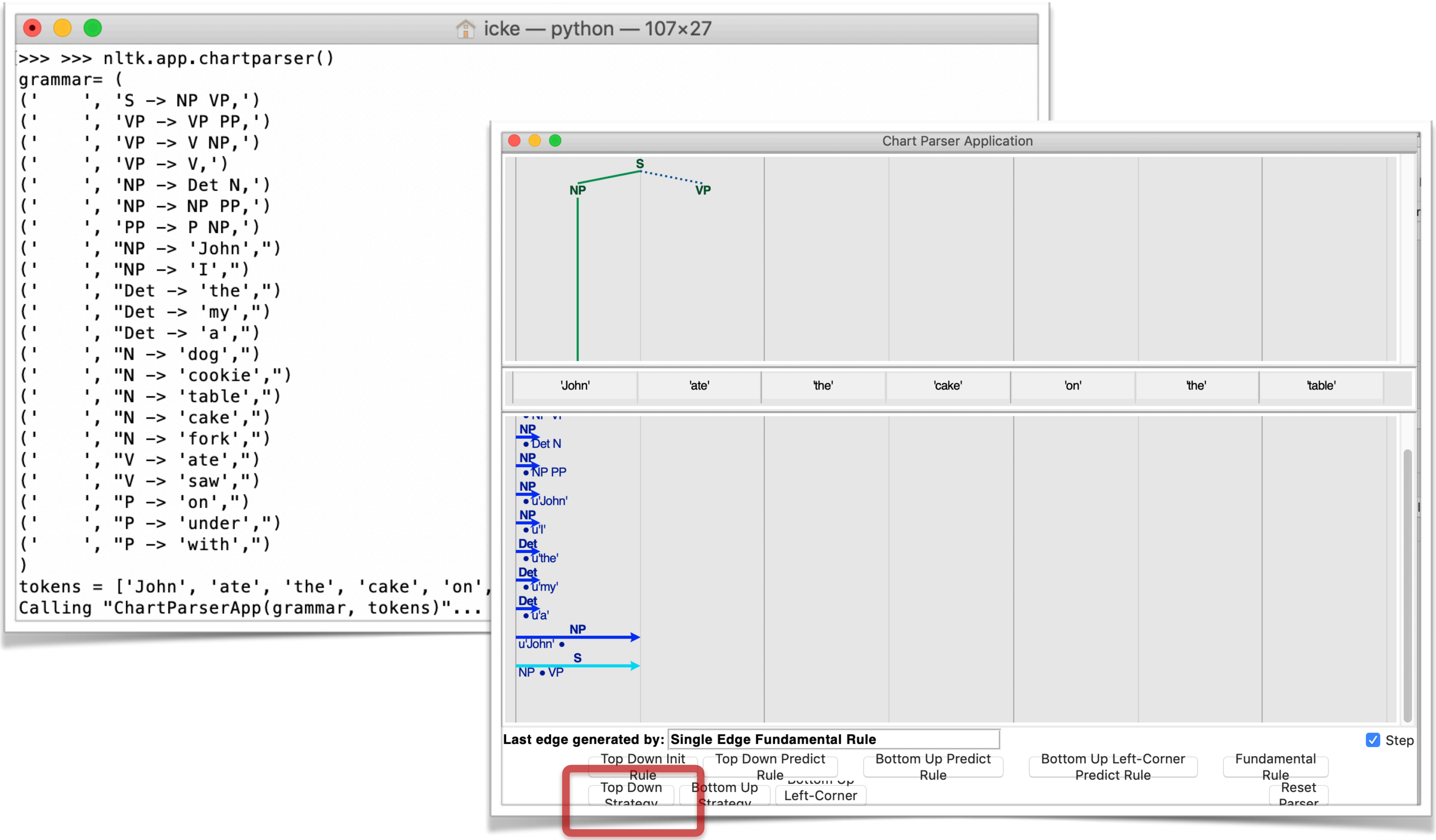
3. Completion

```
NP --> DET . N [0,1]
```

4. Scanning

```
N --> man . [1,2]
```

# Earley Algorithm - Example (Step by Step)



```
>>> >>> nltk.app.chartparser()
grammar= (
('    ', 'S -> NP VP,')
('    ', 'VP -> VP PP,')
('    ', 'VP -> V NP,')
('    ', 'VP -> V,')
('    ', 'NP -> Det N,')
('    ', 'NP -> NP PP,')
('    ', 'PP -> P NP,')
('    ', "NP -> 'John',")
('    ', "NP -> 'I',")
('    ', "Det -> 'the',")
('    ', "Det -> 'my',")
('    ', "Det -> 'a',")
('    ', "N -> 'dog',")
('    ', "N -> 'cookie',")
('    ', "N -> 'table',")
('    ', "N -> 'cake',")
('    ', "N -> 'fork',")
('    ', "V -> 'ate',")
('    ', "V -> 'saw',")
('    ', "P -> 'on',")
('    ', "P -> 'under',")
('    ', "P -> 'with',")
)
tokens = ['John', 'ate', 'the', 'cake', 'on',
Calling "ChartParserApp(grammar, tokens)"...
```

Last edge generated by: Single Edge Fundamental Rule

☑ Step

# Earley Algorithm - Example (Step by Step)

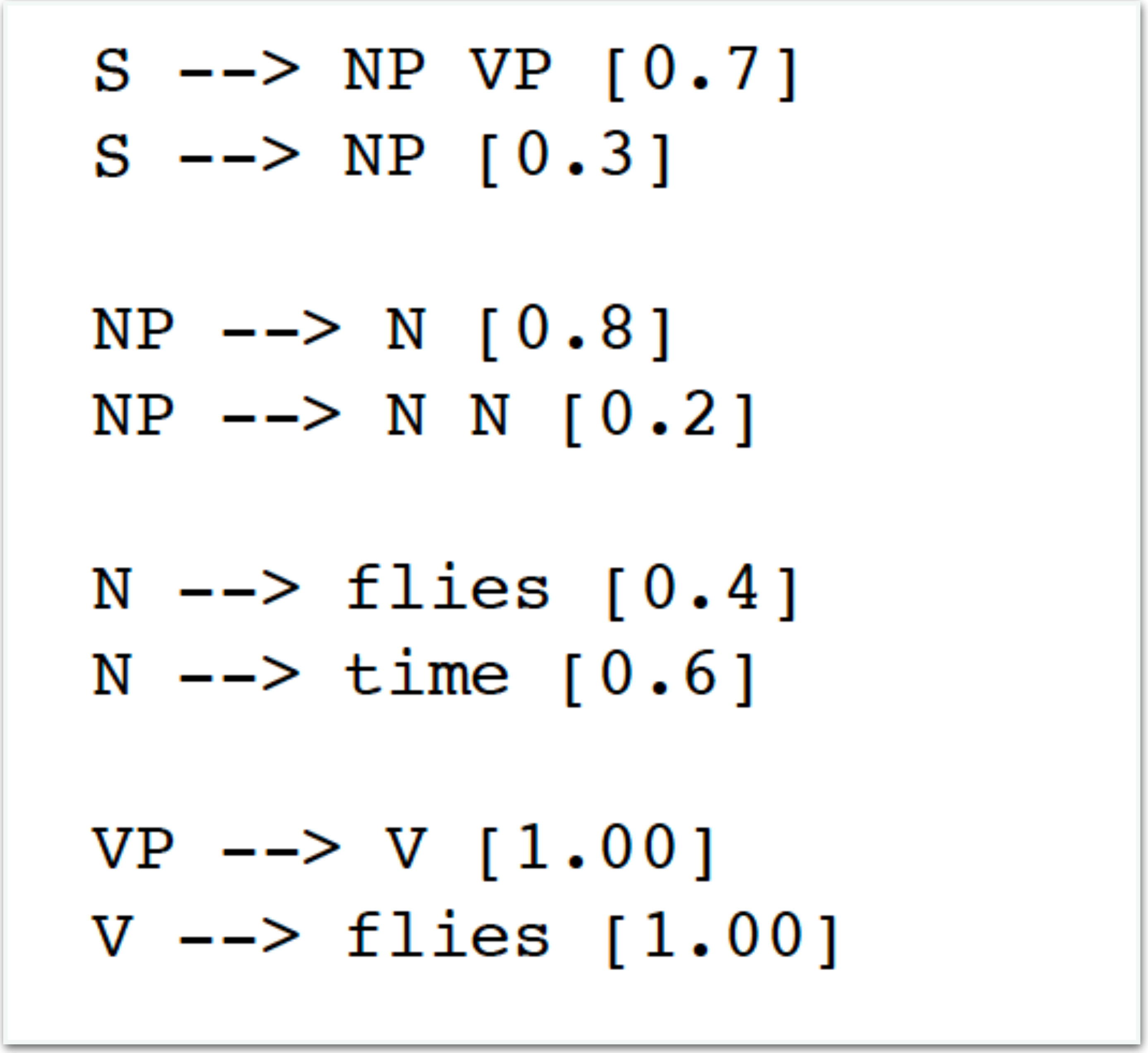
- Try it out yourself! (via NLTK)
- Also try the bottom-up chart parser (CKY)

# Adding Probabilities to our Grammar

# Probabilistic Parsing: Motivation

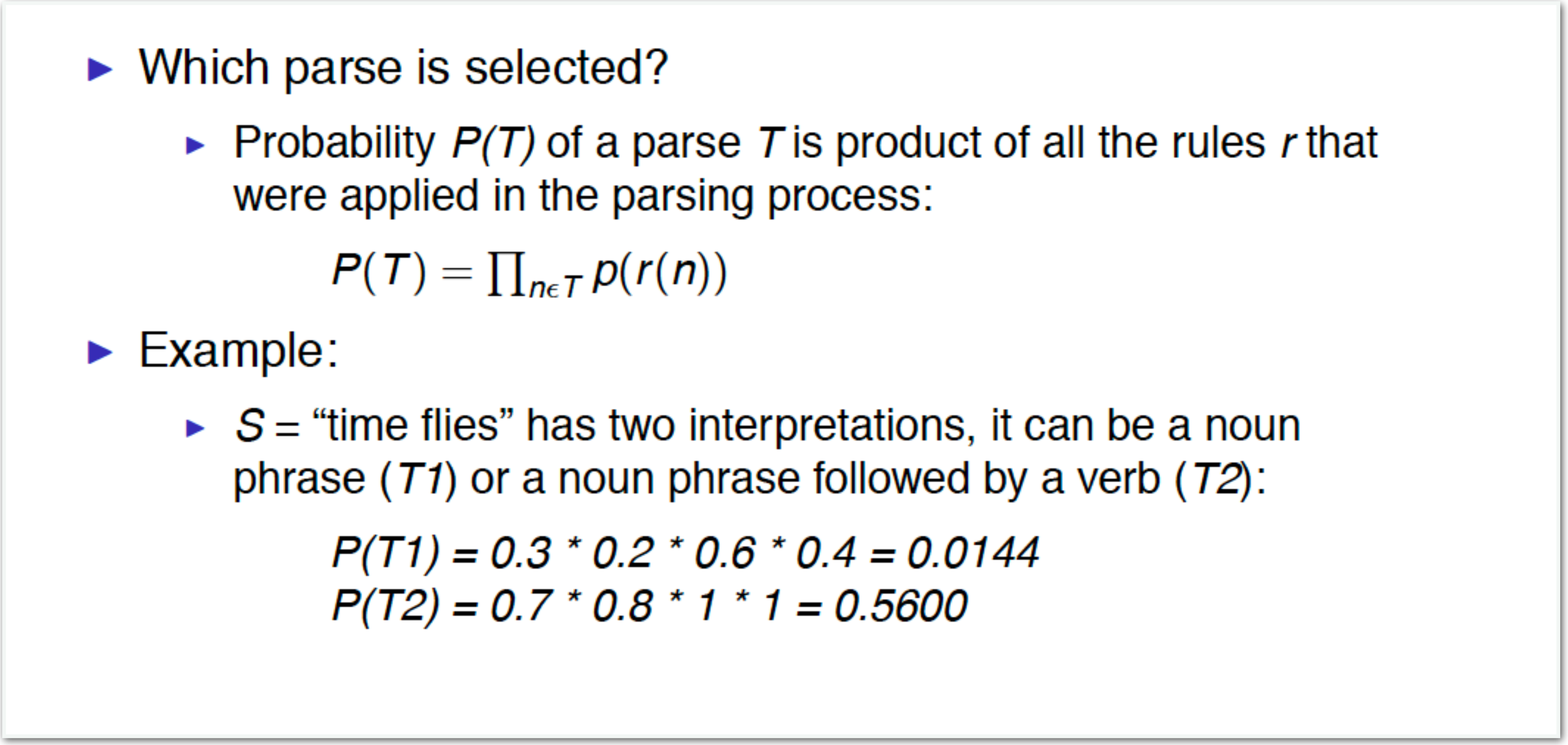
- Ambiguity, but some parses are more likely than others
- Augment context-free grammars with additional knowledge (probabilities for each rule)
- Where do we get these probabilities from?
- Find the most likely parse

# Probabilistic Parsing: Example



```
S --> NP VP [0.7]
S --> NP [0.3]

NP --> N [0.8]
NP --> N N [0.2]

N --> flies [0.4]
N --> time [0.6]

VP --> V [1.00]
V --> flies [1.00]
```

# Probabilistic Parsing



- Which parse is selected?
  - Probability *P(T)* of a parse *T* is product of all the rules *r* that were applied in the parsing process:

$$P(T) = \prod_{n \epsilon T} p(r(n))$$

- Example:
  - *S* = “time flies” has two interpretations, it can be a noun phrase (*T1*) or a noun phrase followed by a verb (*T2*):

$$P(T1) = 0.3 * 0.2 * 0.6 * 0.4 = 0.0144$$
$$P(T2) = 0.7 * 0.8 * 1 * 1 = 0.5600$$

# Probabilistic Parsing

- Obtaining Probabilities
  - Analyze annotated corpus (treebank) or
  - Create statistics by parsing sample corpus
- Parsing of Probabilistic CFG
  - Same principles as with any CFG
  - Calculate probabilities during parsing
  - Optimization (e.g. pruning of unlikely parses)

# Probabilistic Parsing

- Problems
  - Usual problems with statistical approaches
  - Independence assumption
  - Structural dependencies
  - Lexical dependencies
- Solutions
  - Incorporate additional knowledge
  - Probabilistic lexicalized CFG
  - Chart parsers can easily be adjusted (see textbook)

# Probabilistic Parsing: Summary

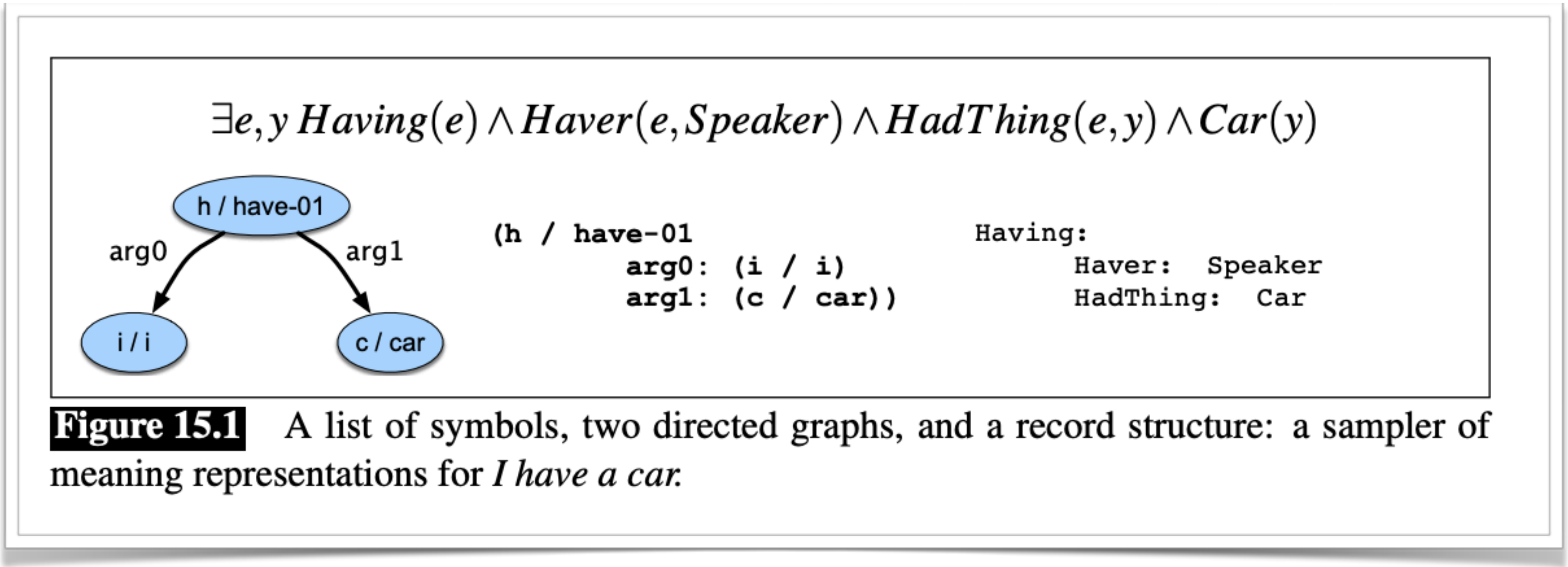
- Probabilities can help reducing the ambiguity problem
- Combination of symbolic and stochastic ideas
- Chart parsers can easily be adjusted (see textbook)
- There is a lot more to probabilistic parsing and we have only touched the surface

# Adding Semantics

# Meaning Representation

- So far concerned with syntax (structure)
- How do we capture semantics (meaning)?

# Meaning Representation (Example)



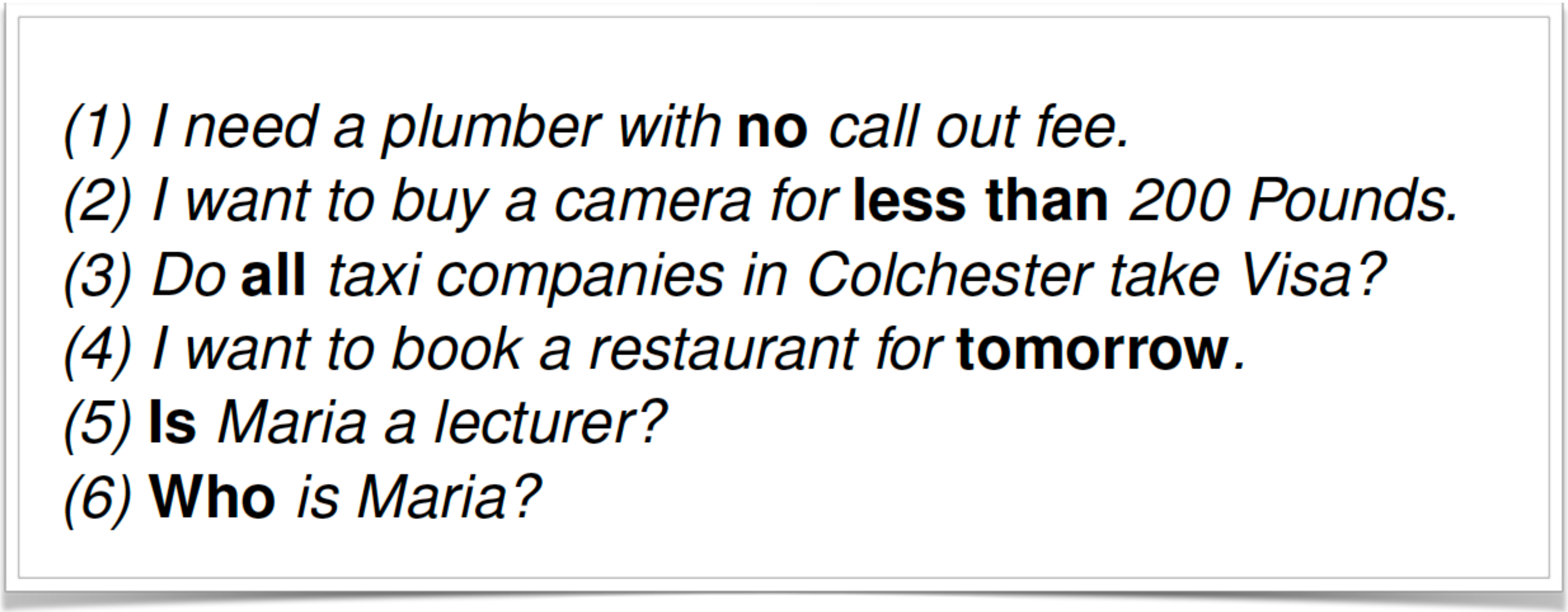
$$\exists e,y\ Having(e) \wedge Haver(e,Speaker) \wedge HadThing(e,y) \wedge Car(y)$$

```
(h / have-01
      arg0: (i / i)
      arg1: (c / car))
```

```
Having:
      Haver:   Speaker
      HadThing:   Car
```

**Figure 15.1** A list of symbols, two directed graphs, and a record structure: a sampler of meaning representations for *I have a car.*

# Meaning Representation (Examples)



*(1) I need a plumber with* **no** *call out fee.*  
*(2) I want to buy a camera for* **less than** *200 Pounds.*  
*(3) Do* **all** *taxi companies in Colchester take Visa?*  
*(4) I want to book a restaurant for* **tomorrow***.*  
*(5)* **Is** *Maria a lecturer?*  
*(6)* **Who** *is Maria?*

# Semantics: What do we need?

- Represent meaning of natural language (semantics)
- Meaning of words and their relations (lexical semantics)
- Meaning of phrases, sentences, questions (compositional semantics)
- Logical form as a result of semantic interpretation

# Semantics: What do we need it for?

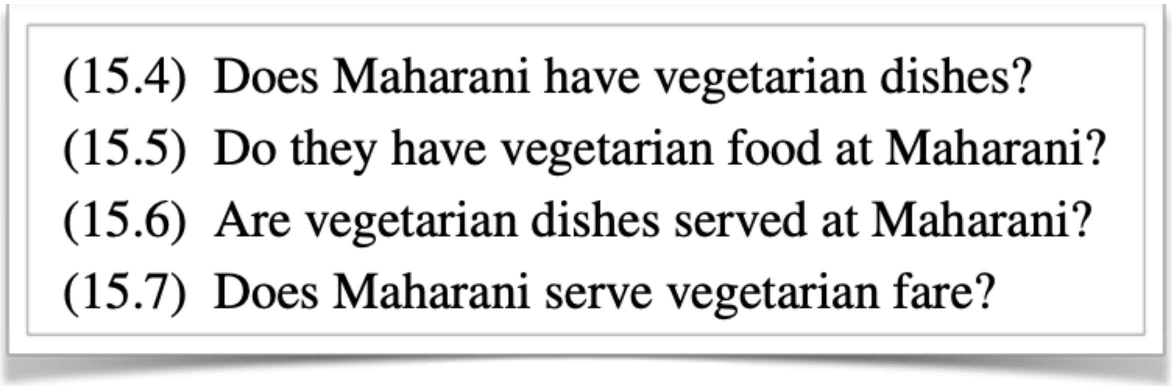
- Question answering (QA) systems (recall *MIT START*)
- Query databases (knowledge bases)
- Precise data representation
- Dialogue understanding
- Intelligent coffee machine?
- …

# Semantics: Requirements

- Verifiability, e.g.:

(15.1) Does Maharani serve vegetarian food?

- Unambiguous representation
- Canonical form, e.g.:



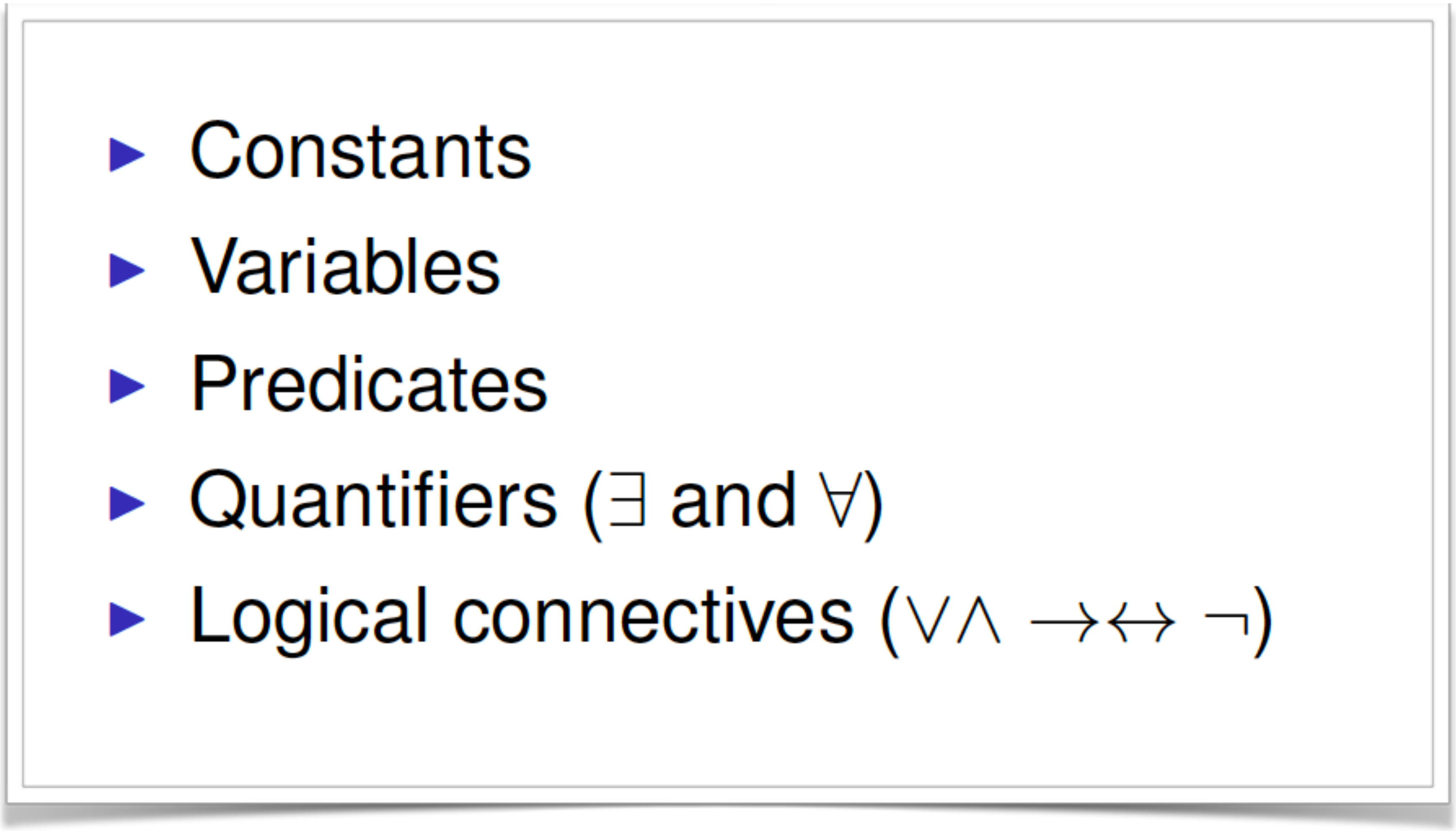
(15.4) Does Maharani have vegetarian dishes?
(15.5) Do they have vegetarian food at Maharani?
(15.6) Are vegetarian dishes served at Maharani?
(15.7) Does Maharani serve vegetarian fare?

- Inference and variables
- Expressiveness
- Combine syntax and semantics

# First Order Predicate Calculus (FOPC)

- Mathematical formalism to represent meaning
- Represent objects, properties of objects and relations among them (set of symbols and rules for combining them into terms)
- Inference rules
- Inference purely formal manipulation of symbols (no meaning or interpretation assigned to symbols)
- Meaning introduced by referencing to objects
- Set of terms (axioms) to represent some world model
- Terms in world model are true
- All formulae are either true or false (in respect to the model)

# FOPC: Elements



- Constants
- Variables
- Predicates
- Quantifiers ($\exists$ and $\forall$)
- Logical connectives ($\vee \wedge \rightarrow \leftrightarrow \neg$)

# FOPC: Examples

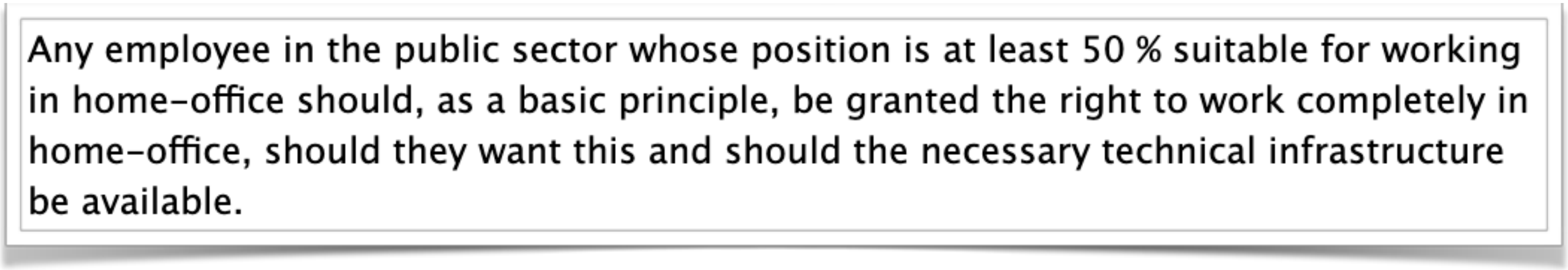
$lecturer(maria)$
$isa(maria, lecturer)$

$\forall x\ (taxi\_company(x) \wedge located(x, colchester)) \rightarrow accept(x, visa))$
$\forall x\ taxi\_company\_in\_colchester(x) \rightarrow accept(x, visa))$

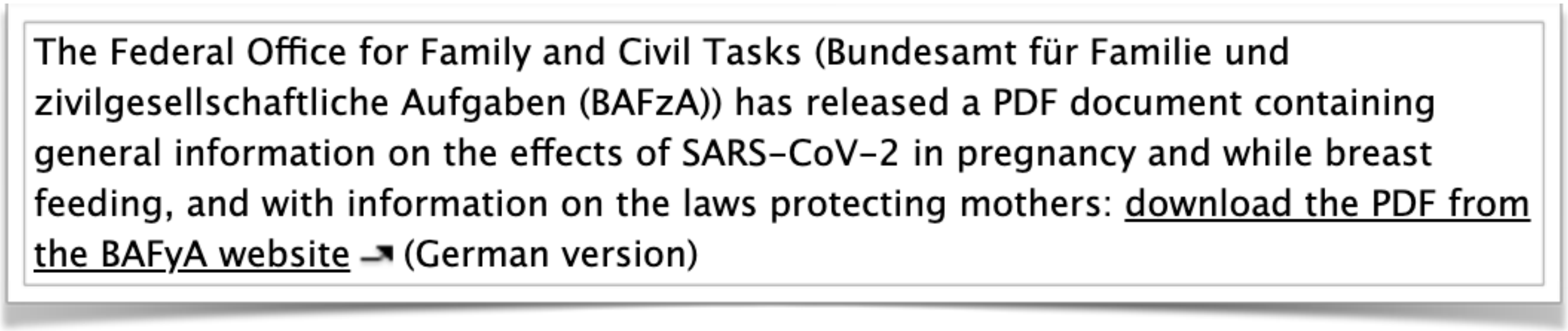
# Why is FOPC useful?

- Tractable and well-understood
- Flexible, easy to use
- Sufficient for many (simple) applications
- Inference (e.g.modus ponens)
- Structure of language can be mapped onto FOPC expressions, e.g. verbs + subcategorization

# But …



Any employee in the public sector whose position is at least 50 % suitable for working in home-office should, as a basic principle, be granted the right to work completely in home-office, should they want this and should the necessary technical infrastructure be available.



The Federal Office for Family and Civil Tasks (Bundesamt für Familie und zivilgesellschaftliche Aufgaben (BAFzA)) has released a PDF document containing general information on the effects of SARS-CoV-2 in pregnancy and while breast feeding, and with information on the laws protecting mothers: download the PDF from the BAFyA website ↗ (German version)

# FOPC: Problems

- Vague information
- Representation of belief
- Representation of events and time
- Discourse resolution

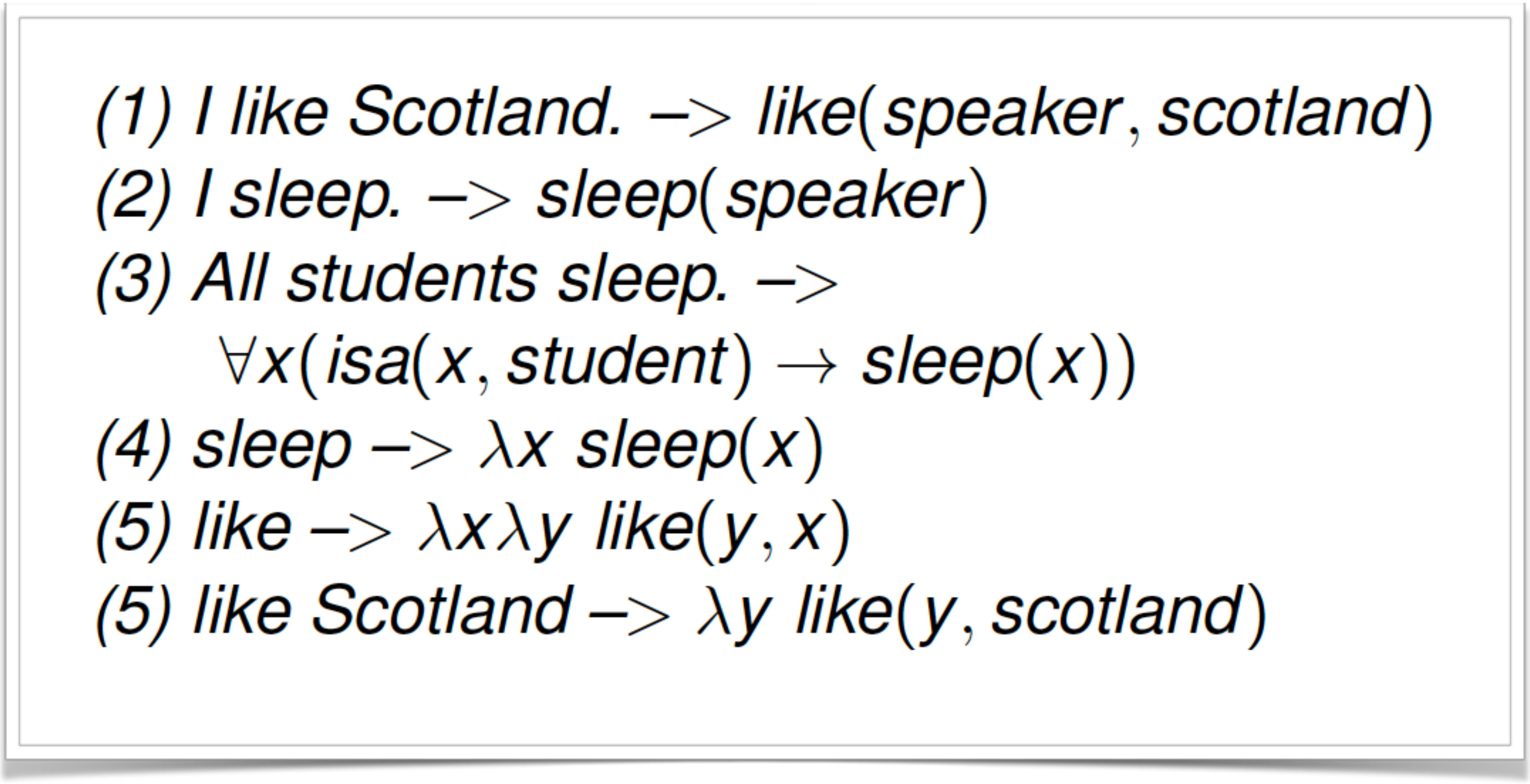
# Problems with FOPC: Ways Out

- Extensions to FOPC
- Higher-order logics
- Modal logics

# Combining Syntax and Semantics

- Syntax-driven semantic analysis
- Grammar rules combine syntax and semantics
- Semantics just an additional feature in feature-structure-based grammar
- Lexical items and rules are associated with logical forms
- Semantic information is passed from children to parents
- Need extension of FOPC to handle "incomplete" expressions: $\lambda$-calculus and complex terms to build quasi-logical forms

# Examples



(1) I like Scotland. –> $like(speaker, scotland)$

(2) I sleep. –> $sleep(speaker)$

(3) All students sleep. –>

$\forall x(isa(x, student) \rightarrow sleep(x))$

(4) sleep –> $\lambda x \; sleep(x)$

(5) like –> $\lambda x \lambda y \; like(y, x)$

(5) like Scotland –> $\lambda y \; like(y, scotland)$

# Simplified Example Grammar

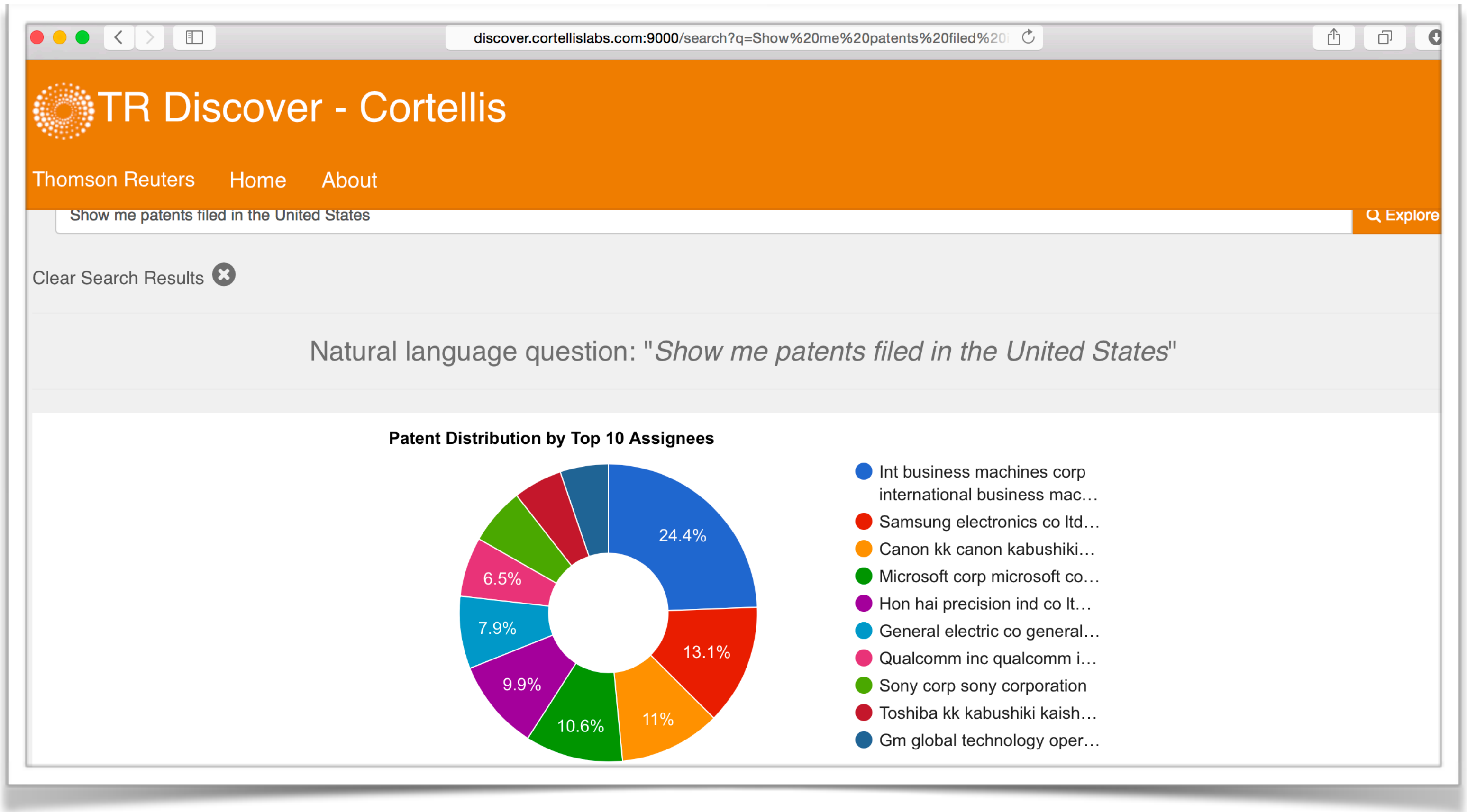
```
S --> NP VP
       {VP.sem(NP.sem)}
VP --> V
       {V.sem}
VP --> V NP
       {V.sem(NP.sem)}
NP --> ’i’
       {speaker}
NP --> ’scotland’
       {scotland}
V --> ’sleep’
       {λx sleep(x)}
V --> ’like’
       {λx,λy like(y,x)}
```

# Simplified Example Grammar

```
S --> NP VP
       {VP.sem(NP.sem)}
VP --> V
       {V.sem}
NP --> DET N
       {<DET.sem x isa(x,N.sem)>}
NP --> ’i’
       {speaker}
DET --> ’all’
       {∀}
N --> ’students’
       {student}
V --> ’sleep’
       {λ x sleep(x)}
```

# Real Example: TR Discover by Thomson Reuters

- Natural language questions over complex datasets
- Combination of syntax and semantics
- Based on context-free grammars
- Use of FOPC to encode semantics



Show me patents filed in the United States

Natural language question: "*Show me patents filed in the United States*"

# TR Discover: Sample Rules

```
N[TYPE=drug,NUM=pl,SEM=<λx.drug(x)>]
       --> ‘drugs’
V[TYPE=[org,drug],
       SEM=λXx.X(λy.develop_org_drug(x,y))>,
       TNS=prog, NUM=?n]
       --> ‘developing’
```

D. Song et al. “Natural Language Question Answering and Analytics for Diverse and Interlinked Datasets”. Proceedings of NAACL-HLT 2015.

# Problems (Compositional Approach)

- Natural language is not mathematics
- Idioms
- Ambiguity, e.g. quantifier scoping
- Compositional approach does not tell us anything about individual meaning of words
- Usual problems with symbolic approaches

# Summary Meaning Representation

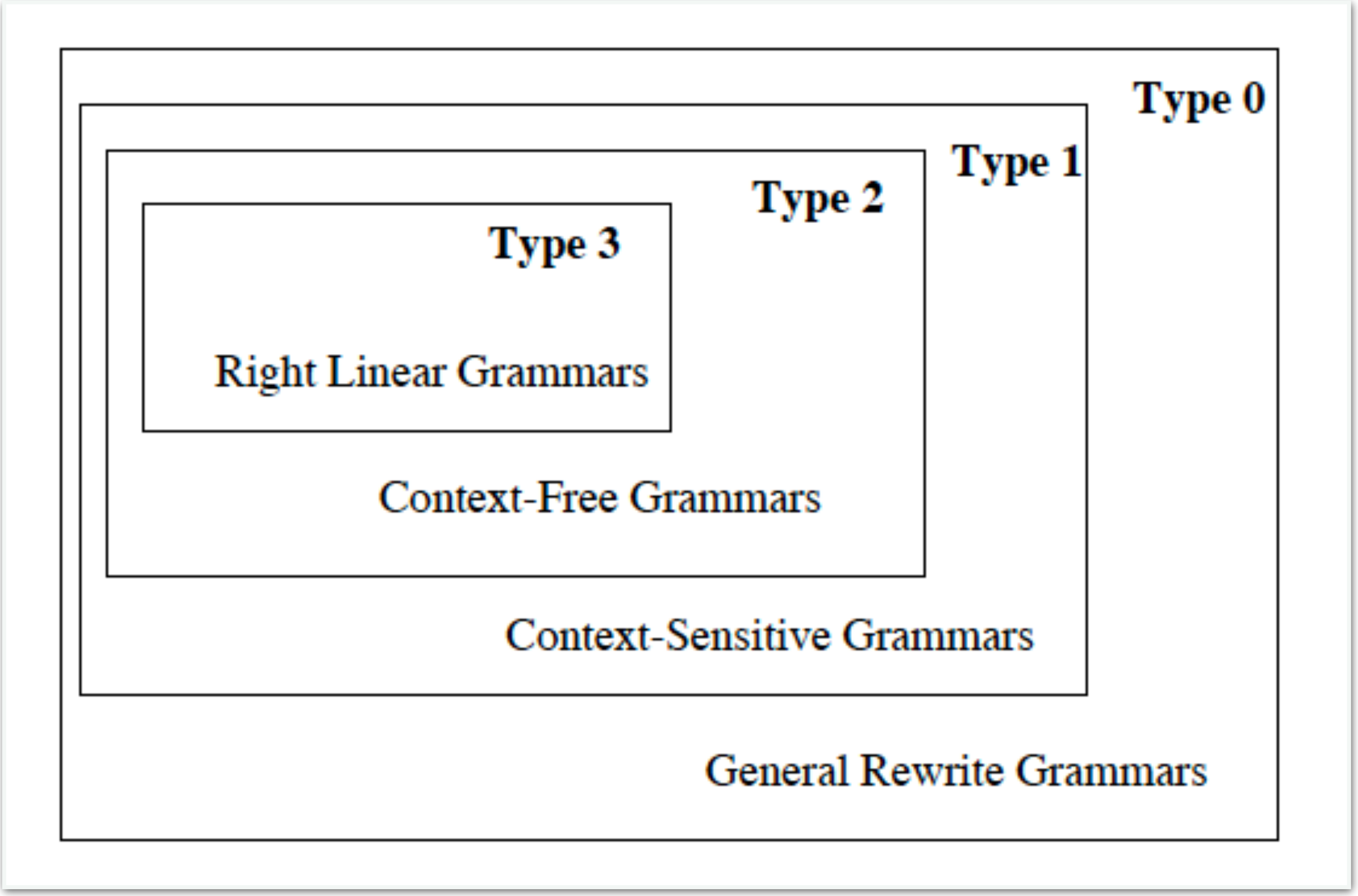
- Expressing meaning of natural language is very difficult
- FOPC can be a good approximation
- Other logics are needed to express phenomena like time,beliefs etc.
- $\lambda$ - calculus permits syntax-driven semantic analysis

# Language and Complexity

# Language and Complexity: Motivation

- Complexity is a major issue in computer science
- The complexity of languages can be defined by the type of grammar they require
- The Chomsky hierarchy defines four types of grammar (the higher the complexity the lower the number)
- Certain constructions in languages require certain types of grammar
- The types of grammar correspond to different types of automata
- This allows reasoning about mathematical complexity

# Chomsky Hierarchy



**Type 0**

**Type 1**

**Type 2**

**Type 3**

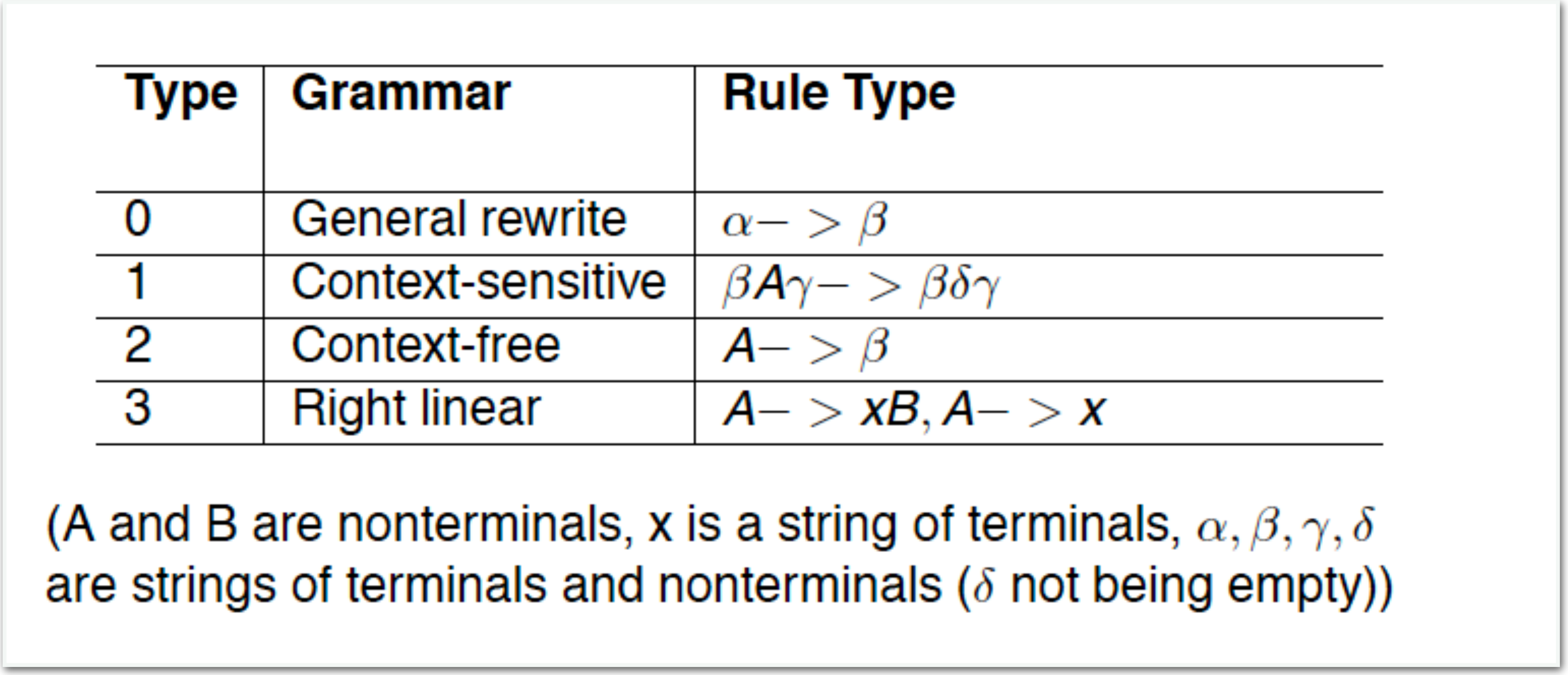
Right Linear Grammars

Context-Free Grammars

Context-Sensitive Grammars

General Rewrite Grammars

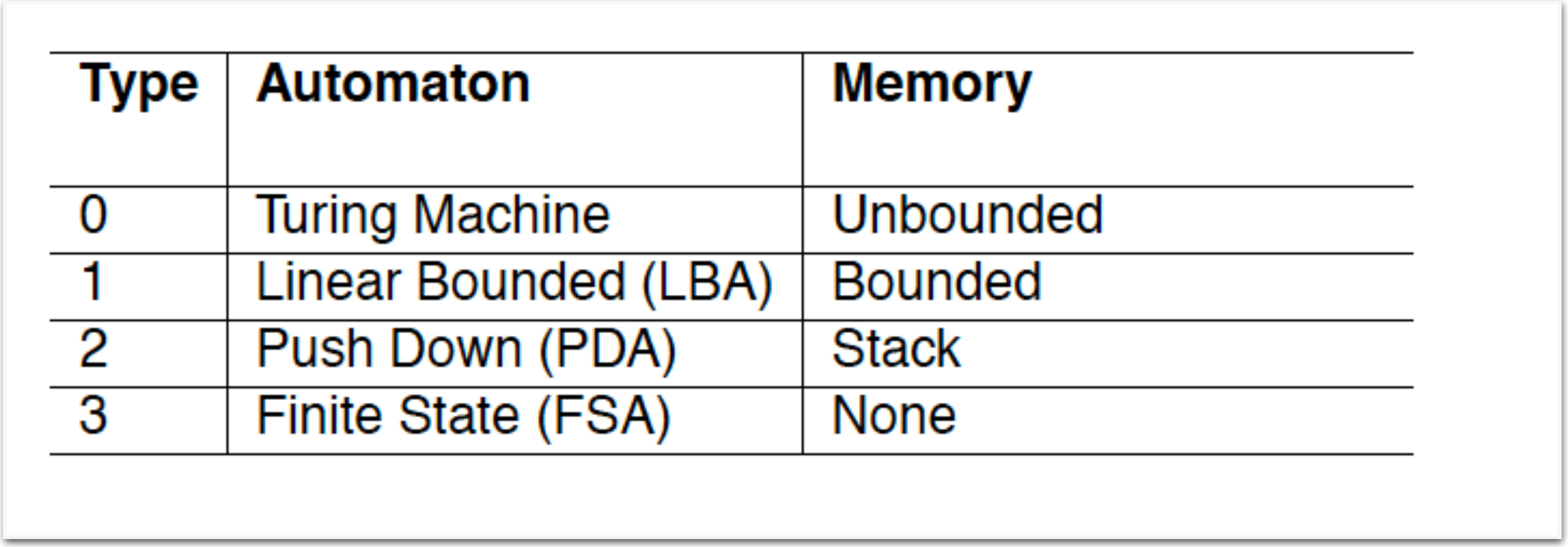
# Grammars vs. Automata



| Type | Grammar | Rule Type |
|---|---|---|
| 0 | General rewrite | $\alpha- > \beta$ |
| 1 | Context-sensitive | $\beta A\gamma- > \beta\delta\gamma$ |
| 2 | Context-free | $A- > \beta$ |
| 3 | Right linear | $A- > xB, A- > x$ |

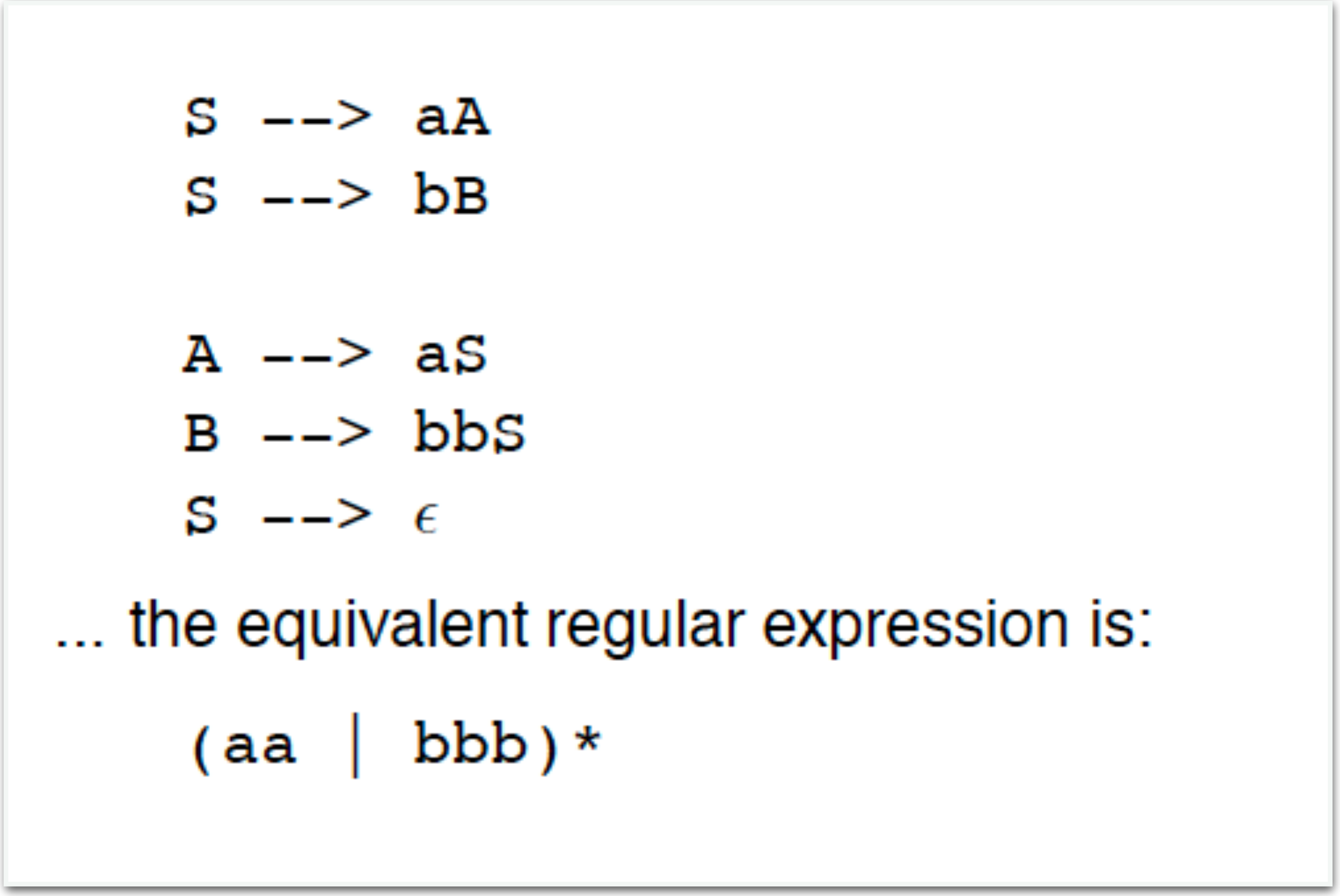
(A and B are nonterminals, x is a string of terminals, $\alpha, \beta, \gamma, \delta$ are strings of terminals and nonterminals ($\delta$ not being empty))

# Grammars vs. Automata II



| Type | Automaton | Memory |
|---|---|---|
| 0 | Turing Machine | Unbounded |
| 1 | Linear Bounded (LBA) | Bounded |
| 2 | Push Down (PDA) | Stack |
| 3 | Finite State (FSA) | None |

# Example 1: Right Linear Grammar

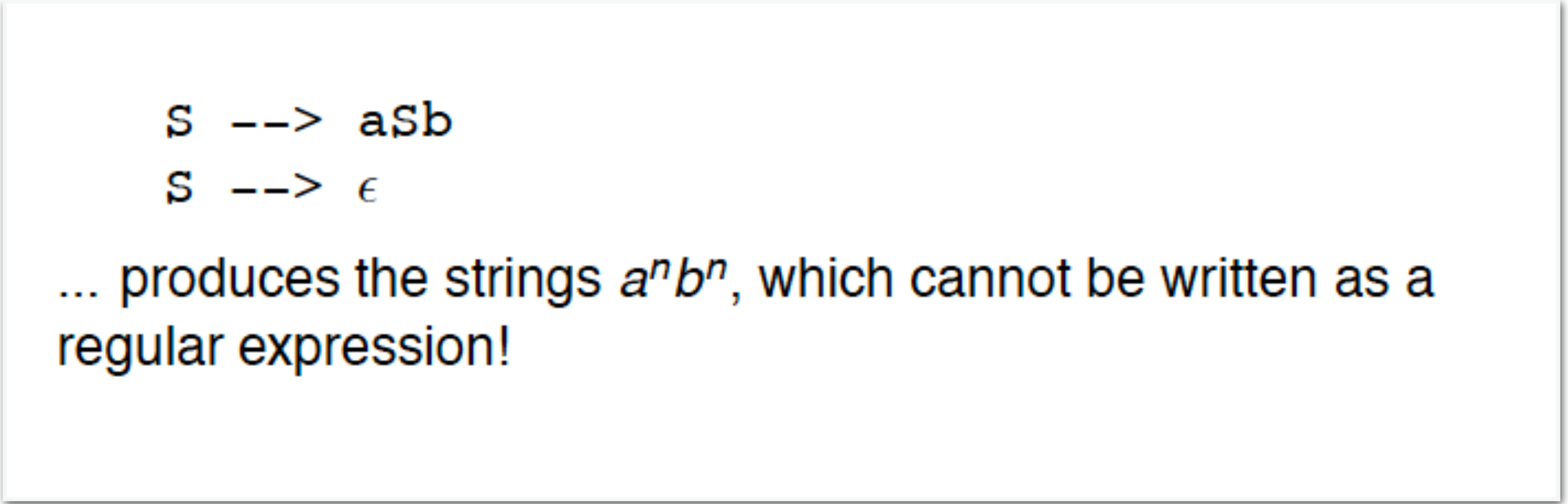


```
S --> aA
S --> bB

A --> aS
B --> bbS
S --> ϵ
```

... the equivalent regular expression is:

```
(aa | bbb)*
```

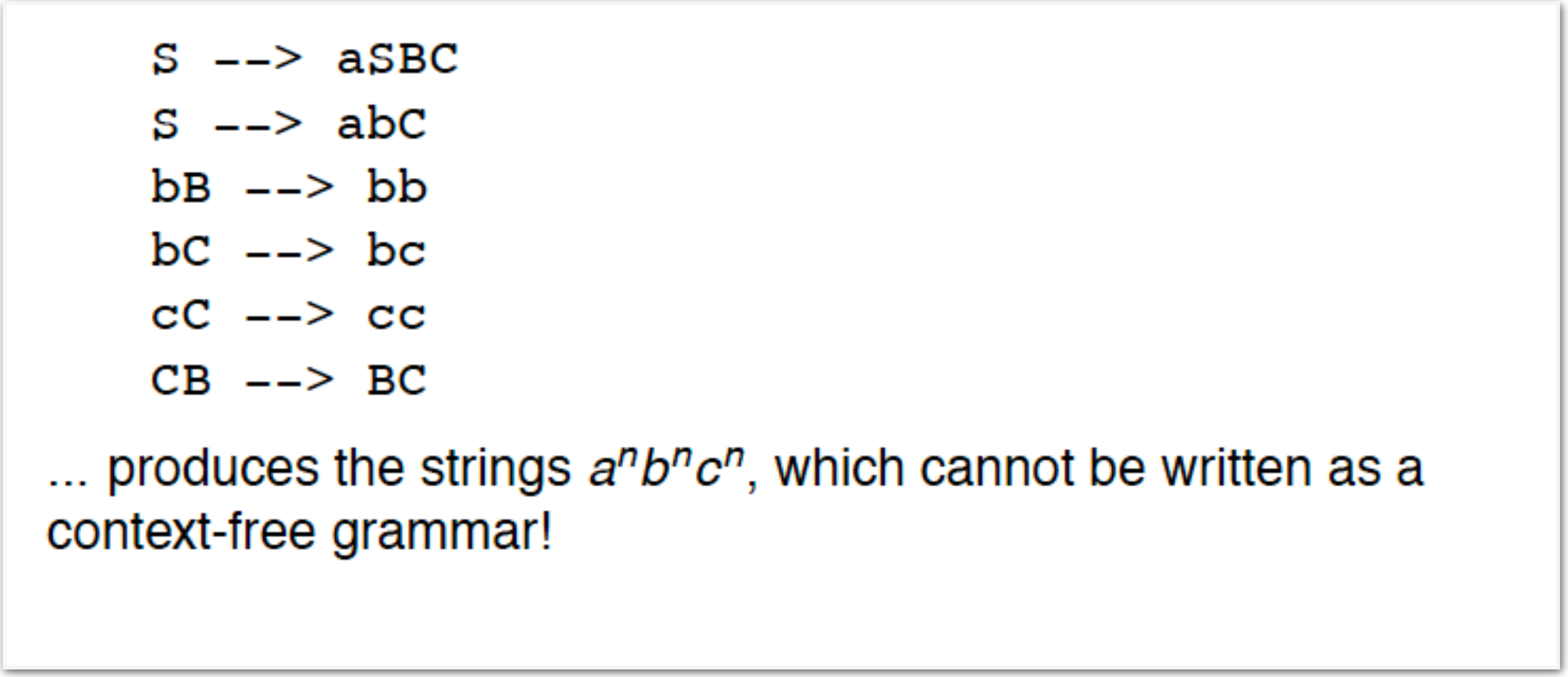
# Example 2: Context-Free Grammar



```
S --> aSb
S --> ϵ
```

... produces the strings $a^n b^n$, which cannot be written as a regular expression!

# Example 3: Context-Sensitive Grammar



```
S --> aSBC
S --> abC
bB --> bb
bC --> bc
cC --> cc
CB --> BC
```

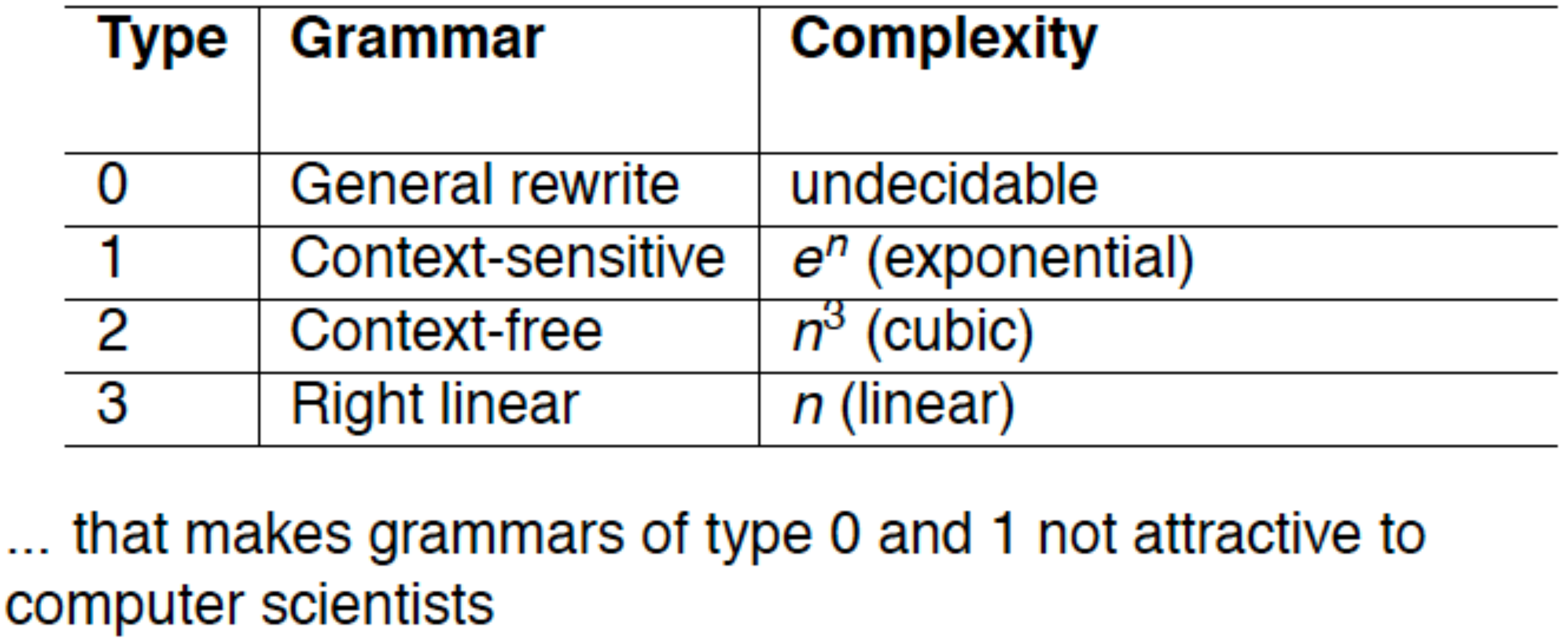
... produces the strings $a^n b^n c^n$, which cannot be written as a context-free grammar!

# Natural Languages in the Chomsky Hierarchy

- Are natural languages regular?
- Not really as they often come with patterns that correspond to languages like $a^n b^n$
- Centre-embedding, for example, causes languages to be non-regular:
  - The student likes the Meetup.
  - The student the student likes likes the Meetup.
  - The student the student the student likes likes likes the Meetup.
- Certain natural languages are not even context-free
- Fair enough, this is all a bit hypothetical …

# Complexity of Grammar Types

- Measure the amount of work to decide whether a string is in a given language or not
- Cost as a function of input length (n)
- *Worst case scenarios*

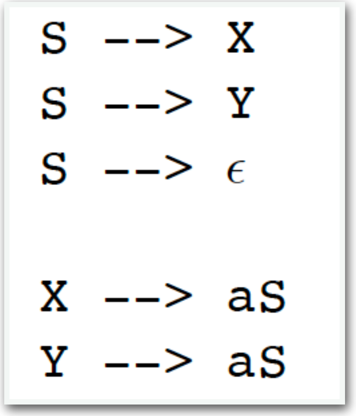


| Type | Grammar | Complexity |
|---|---|---|
| 0 | General rewrite | undecidable |
| 1 | Context-sensitive | $e^n$ (exponential) |
| 2 | Context-free | $n^3$ (cubic) |
| 3 | Right linear | $n$ (linear) |

... that makes grammars of type 0 and 1 not attractive to computer scientists

# Complexity of Parsing

- So far we looked at the decision problem
- Parsing is more complex
- Example:



```
S --> X
S --> Y
S --> ϵ

X --> aS
Y --> aS
```

- Number of possible parse trees for this grammar is exponential ($2^n$)
- Enumerating all possible parse trees therefore even for type 3 grammars exponential

# Summary

- Languages can be defined by means of grammars or automata
- Parsers return tree structure(s) of some input given a grammar
- First-order predicate calculus is a basis for simple meaning representation
- Knowledge about grammar allows to reason about mathematical complexity
- All this is a *knowledge-based* approach, i.e. at the other end of the spectrum of *embeddings*

# Reading

- Jurafsky and Martin (2020), chapters 13-15 and Appendix C
- The third edition focuses on the CKY parser only, the second edition has both Earley and CKY with running examples
- Jurafsky and Martin (second edition), chapter 16 discusses complexity
- D. Song, F. Schilder, C. Smiley and C. Brew. “Natural Language Question Answering and Analytics for Diverse and Interlinked Datasets”. Proceedings of NAACL-HLT 2015.
- See previous slide deck for links to parsers