ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Syntactic Parsing

2022–23

Ion Androutsopoulos

http://www.aueb.gr/users/ion/

These slides are partly based on material from the book *Speech and Language Processing* by D. Jurafsky and J.H. Martin, 2^{η} edition, Pearson Education, 2009 and 3^{rd} edition (in preparation).

Contents

- **Context-free grammars (CFGs)**.
- **Phrase-structure trees** and **dependency trees**.
- **Chomsky Normal Form** and **CKY parsing**.
- **Transition-based dependency parsing** with **neural models**.

Extra optional slides:

- **Graph-based dependency parsing** with **neural models**.
- **Chomsky's hierarchy** and corresponding **automata**.
- **Parsing as search**.
- **Augmented CFGs**.
- **Probabilistic CFGs, probabilistic CKY**.

Context-Free Grammars (CFGs)

NP → Det Nominal

Nominal → N | Adj Nominal — **Disjunction**. In effect, two rules.

Det → ο | η | το | …

Adj → πράσινο | μεγάλο | βαρύ | …

N → βιβλίο | αυτοκίνητο | …

Lexicon: in practice, possibly information from morphological analysis.

- **Terminal** symbols, e.g., "βιβλίο" (book), "το" (the, neuter).
- **Non terminal** symbols, e.g., "Nominal", "Adj" (adjective).
- **Rules** $\alpha \rightarrow \beta$:
  - In **CFGs**, ***α* must be a single non-terminal**, *β* can be a sequence of any terminals and/or non-terminals (even an empty sequence).
- **Initial** symbol: one of the non-terminals (here "NP").
- **Language** of the grammar: the **sequences of terminal symbols** that can be produced from the initial symbol.

Grammar-based parsing algorithms

- **Inputs**:
  - A **grammar** of the type supported by the algorithm (e.g., CFG).
  - A **sequence of symbols** $\boldsymbol{\sigma}$.
- **Outputs**:
  - **Is** $\boldsymbol{\sigma}$ **part of the language** defined by the grammar?
  - What is the **parse tree of** $\boldsymbol{\sigma}$?
  - The parse tree is a **proof** that σ complies with the grammar. It also provides information about the **syntactic structure** of σ.

Slightly larger CFG example

- NP → Det PN | Pron | Det Nominal
- Nominal → N | Adj Nominal | Nominal PP
- PP → Prep NP

την Αθήνα, εγώ, μια πτήση, μια πρωινή πτήση, μια πρωινή πτήση προς την Αθήνα, …

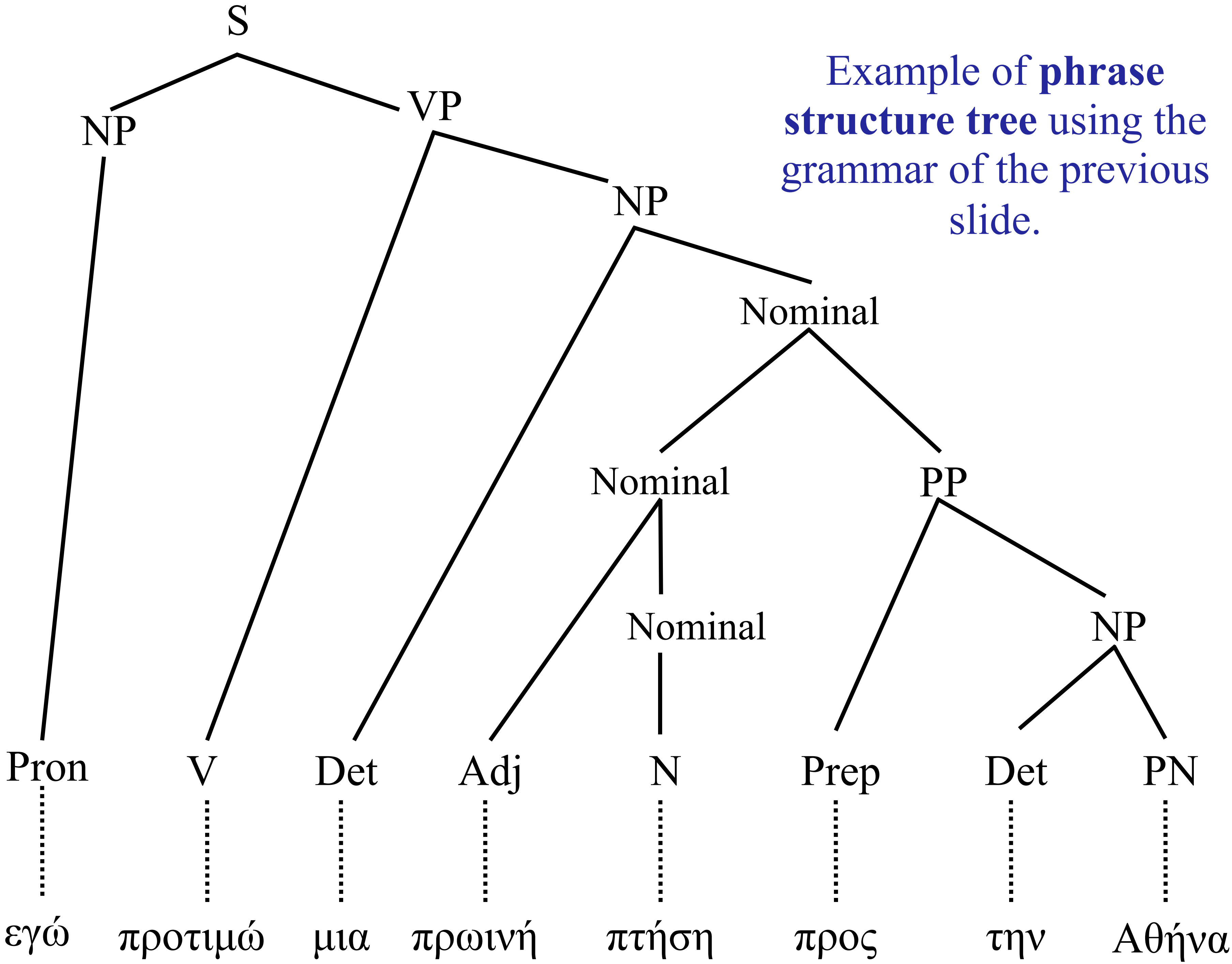
- <u>S</u> → NP VP | VP
- VP → V | V NP

εγώ θέλω μια πρωινή πτήση, …

- Pron → εγώ
- Det → ο | η | έναν | μια | τον | την
- PN → Θεσσαλονίκη | Αθήνα
- N → πτήση | πελάτης | πελάτη
- Adj → πρωινή | απογευματινή
- V → θέλω | θέλει | προτιμώ | συμφωνώ
- Prep → προς | από

Acting as a lexicon.

Example of **phrase structure tree** using the grammar of the previous slide.



Syntactically ambiguous sentences

- "We saw the **scientist with the telescope**."
  - We saw [$_{NP}$ the [$_{Nominal}$ scientist [$_{PP}$ with the telescope]]].
  - As in "the flight from Thessaloniki".
- "**We saw** the scientist **with the telescope**."
  - We saw [$_{NP}$ the scientist] [$_{PP}$ with the telescope].
  - We would also have a rule: VP → V NP PP.
- "We saw the scientist with the telescope from Paris."
  - We saw [the scientist] [with the telescope] [from Paris].
  - We saw [the scientist with the telescope] [from Paris].
  - We saw [the scientist] [with the [telescope from Paris]].
  - We saw [the [scientist with the [telescope from Paris]]].

Syntactically ambiguous sentences

- "We saw the scientist **with the white coat**."
  - We need **semantic constraints** to **rule out** the possibility that the **coat** might be the **observation instrument**.
- From a **purely syntactic point of view**, most sentences are **very ambiguous**.
  - Large number of parse trees (often **exponential increase** as the number of phrases that can be combined increases).
  - **Time-consuming** to discover and return all trees **separately**.
  - **Problem** for simplistic parsers that use **generic search algorithms** (e.g., depth-first search – see optional slides).

Chomsky Normal Form

- **Context Free Grammars** (CFG) in **Chomsky Normal Form** (CNF):
    - Only rules of the form ***A* → *B C*** and ***A* → *w***, were ***A*, *B*, *C* non-terminals** and ***w* terminal**. For example:

S → V NP		
V → θέλω	V → επιθυμώ	
NP → Det Nominal	Nominal → Adj Nominal	~~Nominal → N~~
Det → μια	N → πτήση	Nominal → πτήση
Adj → πρωινή	Adj → απογευματινή	

- **Every CFG** can be **converted** to **CNF** (see J&M).
    - But the new grammar may not produce the same parse trees.
- The **CKY algorithm** (next slides) is for **CFGs in CNF**.
    - Yet another dynamic programming algorithm.
    - Other algorithms (e.g., **Earley**) can handle **CFGs not in CNF**.

CKY algorithm

(0) θέλω (1) μία (2) πρωινή (3) πτήση (4)

	0	1	2	3	4
0		V (0,1)			
1			Det (1,2)		
2				Adj (2,3)	
3					Nominal N (3,4)

CKY algorithm

0 θέλω 1 μία 2 πρωινή 3 πτήση 4

	0	1	2	3	4
0		V (0,1)	X (0,2)		
1			Det (1,2)		
2				Adj (2,3)	
3					Nominal N (3,4)

There is no grammar rule to combine V and Det.

CKY algorithm

(0) θέλω (1) μία (2) πρωινή (3) πτήση (4)

	0	1	2	3	4
0		V (0,1)			
1			Det (1,2)	X (1,3)	
2				Adj (2,3)	
3					Nominal N (3,4)

There is no grammar rule to combine Det and Adj.

CKY algorithm

0 θέλω 1 μία 2 πρωινή 3 πτήση 4

	0	1	2	3	4
0		V (0,1)		X (0,3)	
1			Det (1,2)	(1,3)	
2				Adj (2,3)	
3					Nominal N (3,4)

Cell (1,3) is empty.

CKY algorithm

(0) θέλω (1) μία (2) πρωινή (3) πτήση (4)

	0	1	2	3	4
0		V (0,1)	(0,2)	X (0,3)	
1			Det (1,2)	(1,3)	
2				Adj (2,3)	
3					Nominal N (3,4)

Cell (0,2) is empty.

CKY algorithm

0 θέλω 1 μία 2 πρωινή 3 πτήση 4

	0	1	2	3	4
0		V (0,1)	(0,2)		
1			Det (1,2)	(1,3)	
2				Adj (2,3)	**Nominal** (2,4)
3					Nominal N (3,4)

CKY algorithm

(0) θέλω (1) μία (2) πρωινή (3) πτήση (4)

	0	1	2	3	4
0		V (0,1)	(0,2)		
1			Det (1,2)	(1,3)	X **NP** (1,4)
2				Adj (2,3)	Nominal (2,4)
3					Nominal N (3,4)

CKY algorithm

0 θέλω 1 μία 2 πρωινή 3 πτήση 4

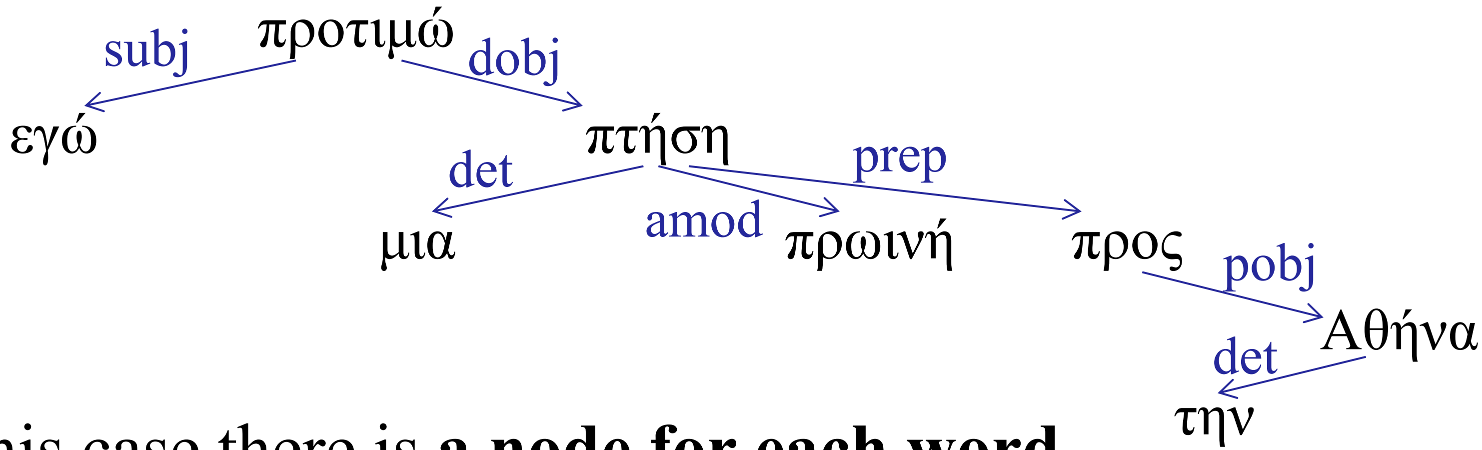
	0	1	2	3	4
0		V (0,1)	(0,2)	(0,3)	S X X (0,4)
1			Det (1,2)	(1,3)	NP (1,4)
2				Adj (2,3)	Nominal (2,4)
3					Nominal N (3,4)

Try also: http://lxmls.it.pt/2015/cky.html

Extracting trees from CKY's table

- We can **store** in each cell the **rules** that **produced** the corresponding **non-terminals**.
  - This allows **extracting the parse tree** from the table.
  - For **syntactically ambiguous** sentences, **multiple parse trees** will be extracted.
  - But **extracting** the parse tree makes the worst case **time complexity** of the algorithm **exponential**, because there are exponentially many parse trees in the worst case.
  - **Without parse tree** extraction, the **time complexity** is $O(n^3)$, where n is the sentence length in words.

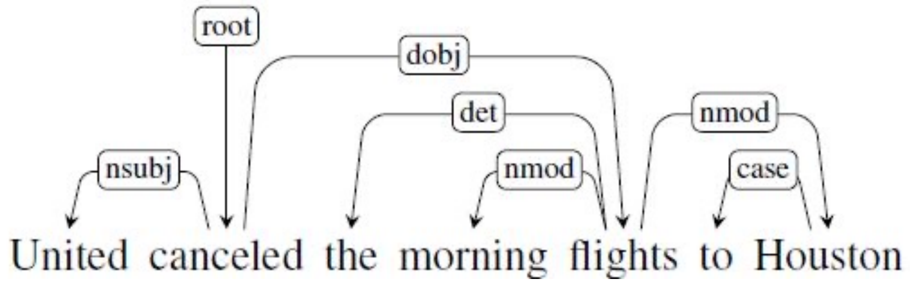
Dependency trees



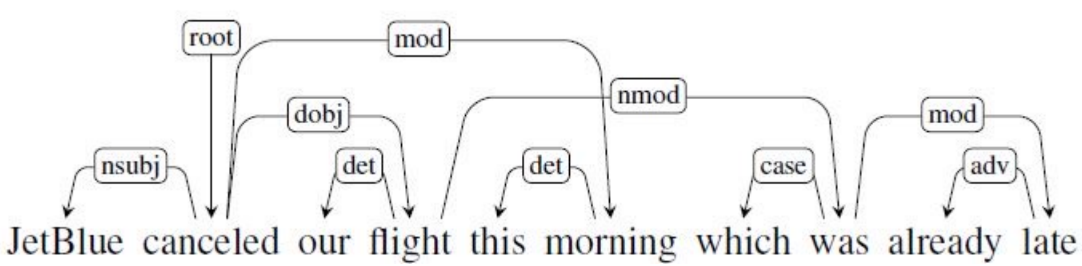
- In this case there is **a node for each word**.
  - The arcs denote **dependencies** between words.
  - **Same trees** for different **word orders** in free word order languages.
  - **Closer** to **graph-based semantic representations**.
- **Obtaining dependency trees**:
  - We can **automatically** produce **dependency trees from phrase structure trees** (with some additional effort – see optional slides).
  - This allows **reusing treebanks of phrase structure trees** to train **dependency parsers**. And **using parsers** that produce **phrase structure trees** to obtain dependency trees.
  - But there are also **parsers** that **produce directly dependency trees**.

Projective vs. non-projective dependency trees

- **Projective** dependency **tree**: **all its arcs** are **projective**.
  - **Projective arc**: There is a **path** from the **head** to **every word between the head and the dependent (modifier)**.

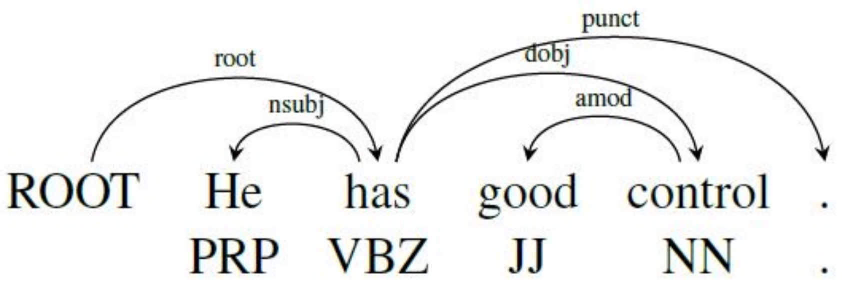


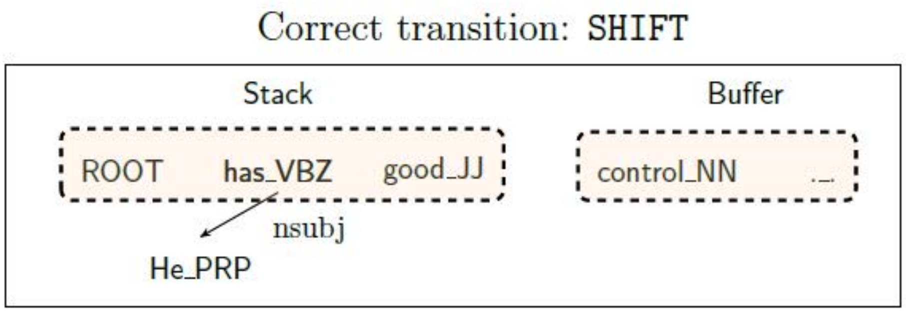
- **Non-projective** dependency **tree**:
  - Contains **at least one non-projective arc**. Less common in English, **more common** in more **free-word order languages**.
  - **Some parsing algorithms** can produce **only projective** trees.



Images from "Speech and Language Processing" by D. Jurafsky and D.H. Martin, 3rd Edition (draft Jan. 2017), http://web.stanford.edu/~jurafsky/slp3/

Transition-based dependency parsing





Transition	Stack	Buffer	A
	[ROOT]	[He has good control .]	∅
`SHIFT`	[ROOT He]	[has good control .]	
`SHIFT`	[ROOT He has]	[good control .]	
`LEFT-ARC(nsubj)`	[ROOT has]	[good control .]	$A\cup$ nsubj(has,He)
`SHIFT`	[ROOT has good]	[control .]	
`SHIFT`	[ROOT has good control]	[.]	
`LEFT-ARC(amod)`	[ROOT has control]	[.]	$A\cup$amod(control,good)
`RIGHT-ARC(dobj)`	[ROOT has]	[.]	$A\cup$ dobj(has,control)
...	...	...	...
`RIGHT-ARC(root)`	[ROOT]	[]	$A\cup$ root(ROOT,has)

Figure 1: An example of transition-based dependency parsing. Above left: a desired dependency tree, above right: an intermediate configuration, bottom: a transition sequence of the arc-standard system.

From the paper of D. Chen and C. Manning "A Fast and Accurate Dependency Parser using Neural Networks", EMNLP 2014.
http://aclweb.org/anthology/D/D14/D14-1082.pdf

Transition-based dependency parsing

- **Initially** all **words** in the **buffer**, **stack** contains only **ROOT**.
- **Possible actions** at each step (‘**arc-standard**’ model):
  - **Shift** the **first word** of the **buffer** to the **stack**.
  - **Connect** the **top two words** of the **stack** with a **left arc** and particular **label** (e.g., NSUBJ), leaving only the right word in the stack.
  - **Connect** the **top two words** of the **stack** with a **right arc** and a particular **label**, leaving only the left word in the stack.
- **Final state**: only **ROOT** in the **stack**, **no words** in the **buffer**.
- A **classifier** selects the **action** to take at each point.
  - The classifier **may select the wrong action**.
  - **Greedy** search, no going back once an action is selected.
  - But **people** seem to **backtrack** (e.g., “**garden path**” sentences).
- **Linear complexity** in sentence length.

Garden path sentences

- The horse raced past the barn fell. (The horse that was raced past the barn fell.)
- The old man the boat. (The old operate the boat.)
- While the man hunted the deer ran into the woods. (While the man hunted, the deer ran into the woods.)
- While Anna dressed the baby played in the crib. (While Anna dressed, the baby played in the crib.)
- I convinced her children are noisy. (I convinced her that children are noisy)
- The coach smiled at the player tossed the Frisbee. (The coach smiled at the player who was tossed the Frisbee.)
- The cotton clothes are made up of grows in Mississippi. (The cotton that clothes are made up of grows in Mississippi.)

Examples from https://www.washingtonpost.com/news/wonk/wp/2016/05/18/googles-new-artificial-intelligence-cant-understand-these-sentences-can-you/

An MLP oracle in transition parsing

Probabilities of the **possible actions**: SHIFT, LEFT-ARC(amod), RIGHT-ARC(dobj), etc.

Cube activation function.

Softmax layer:

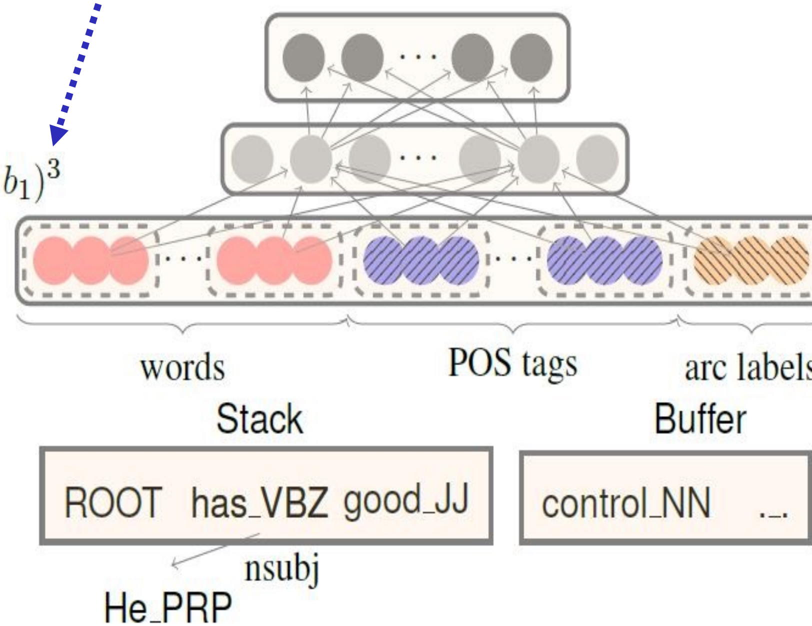
$$p = \texttt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

Input layer: $[x^w, x^t, x^l]$

Configuration



Word embeddings, **POS tag embeddings**, **label embeddings** (e.g., of top 3 words of the stack, of top 3 words of the buffer, of the leftmost and rightmost children of the top 2 words of the stack, …).

From the paper of D. Chen and C. Manning "A Fast and Accurate Dependency Parser Using Neural Networks", EMNLP 2014.
http://aclweb.org/anthology/D/D14/D14-1082.pdf

Adding context-aware word embeddings

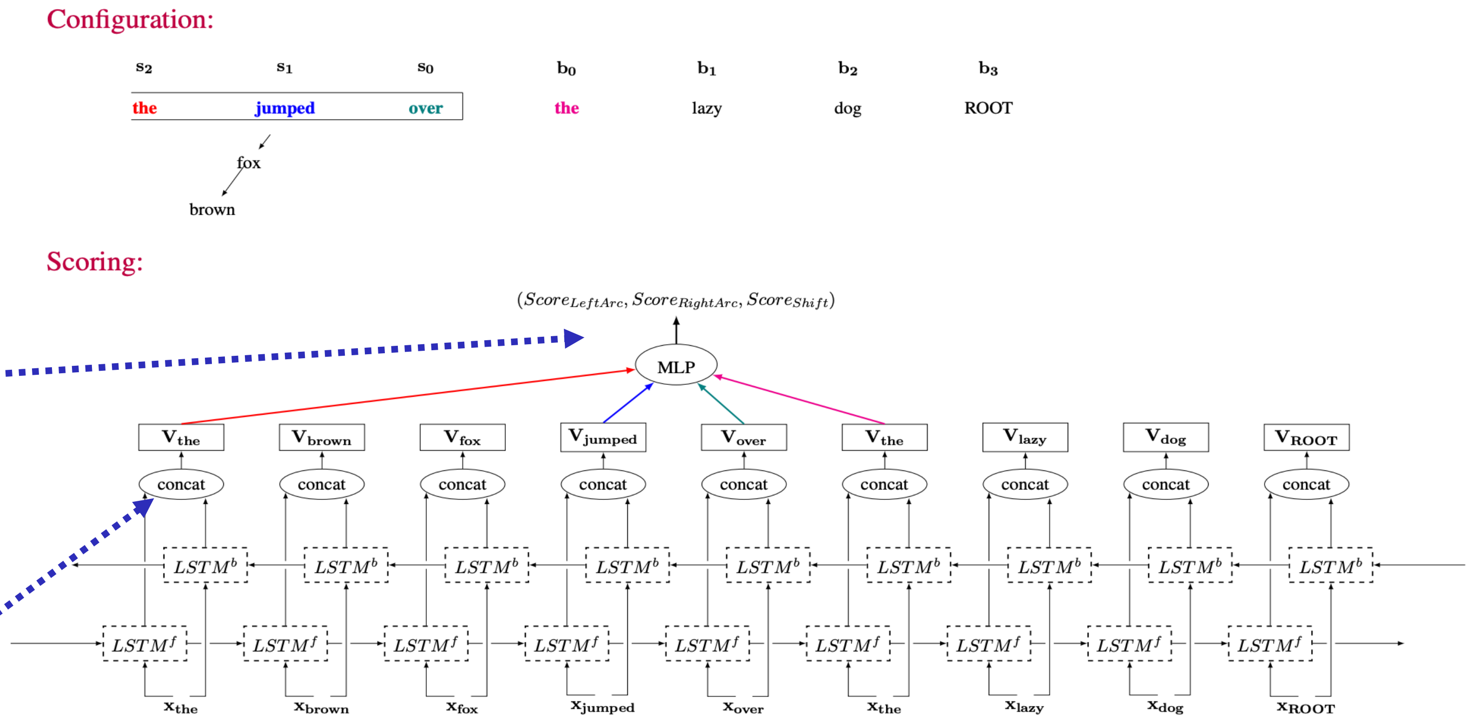


Figure 1: Illustration of the neural model scheme of the transition-based parser when calculating the scores of the possible transitions in a given configuration. The configuration (stack and buffer) is depicted on the top. Each transition is scored using an MLP that is fed the BiLSTM encodings of the first word in the buffer and the three words at the top of the stack (the colors of the words correspond to colors of the MLP inputs above), and a transition is picked greedily. Each x_i is a concatenation of a word and a POS vector, and possibly an additional external embedding vector for the word. The figure depicts a single-layer BiLSTM, while in practice we use two layers. When parsing a sentence, we iteratively compute scores for all possible transitions and apply the best scoring action until the final configuration is reached.

An **MLP** decides the **next action**, using the **embeddings** of the **top 3 stack words** and the **1st buffer word**.

A **biLSTM** produces **context aware** word embeddings.

From the paper of E. Kiperwasser and Y. Goldberg “Simple and Accurate Dependency Parser Using Bidirectional LSTM Feature Representations”, *Transactions of ACL*, vol. 4, pp. 313 – 327, 2016. https://aclweb.org/anthology/Q16-1023

Training the oracle

- **For each training sentence**, we have the **correct tree**.
  - We use it to **figure out** the **correct action** that the oracle should take **at each point**, in order to **train** the oracle.
- At each point, the **correct decision** is:
  - **LEFT-ARC** if the **resulting dependency** is **in the correct tree**.
  - **RIGHT-ARC** if (1) the **resulting dependency** is **in the correct tree** and (2) **all the modifiers** (in the correct tree) **of the top token of the stack** have **already been linked** (as modifiers) to the top token of the stack.
    - E.g., if we link "book" to "flight" with a RIGHT-ARC, we won't be able to link "flight" to "through", because "flight" will no longer be in the stack.

Stack	Word buffer	Relations
[root, book, flight]	[through, Houston]	(the ← flight)

  - Otherwise, the correct decision is **SHIFT**.

Example image from "Speech and Language Processing" by D. Jurafsky and D.H. Martin, 3rd Edition (draft Jan. 2017), http://web.stanford.edu/~jurafsky/slp3/

Evaluating dependency parsers

- **Unlabeled Attachment Score (UAS):** How many **words** (viewed as modifiers) were linked to their **correct head**.
  - **Ignoring** the **labels** of the dependencies.
- **Labeled Attachment Score (LAS)**: How many **words** (viewed as modifiers) were linked to their **correct head** with the **correct dependency label**.
  - We can also measure how well we do **per dependency label**.

Extra <u>optional</u> slides follow.

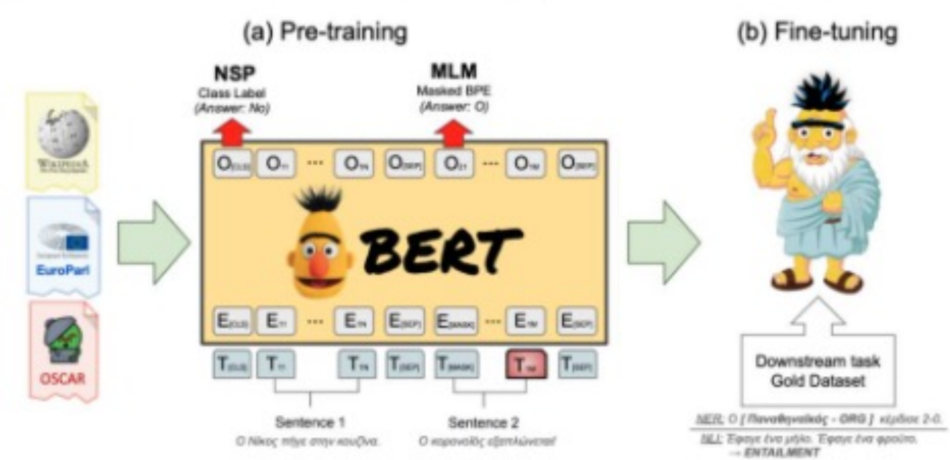
Ion Androutsopoulos
October 24, 2021 ·

We are happy to announce gr-nlp-toolkit, a natural language processing toolkit for (modern) Greek, built on top of Greek-BERT during the BSc theses of C. Dikonimaki and N. Smyrnioudis. The toolkit currently supports Greek named entity recognition, part-of-speech (POS) tagging, morphological tagging, and dependency parsing.

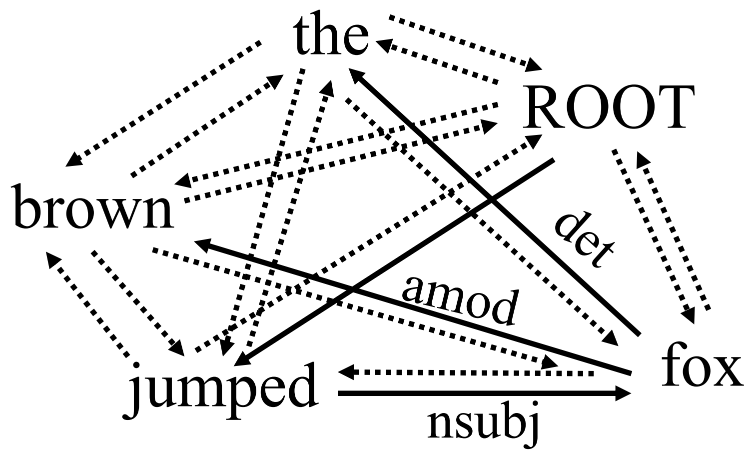
Toolkit code and installation instructions:
https://github.com/nlpaueb/gr-nlp-toolkit

For POS tagging, morphological parsing, and dependency parsing, we ... **See more**

Fig. 1.2: An illustration of GreekBERT pretraining. Figure taken from [Kou+20].



Graph-based dependency parsing



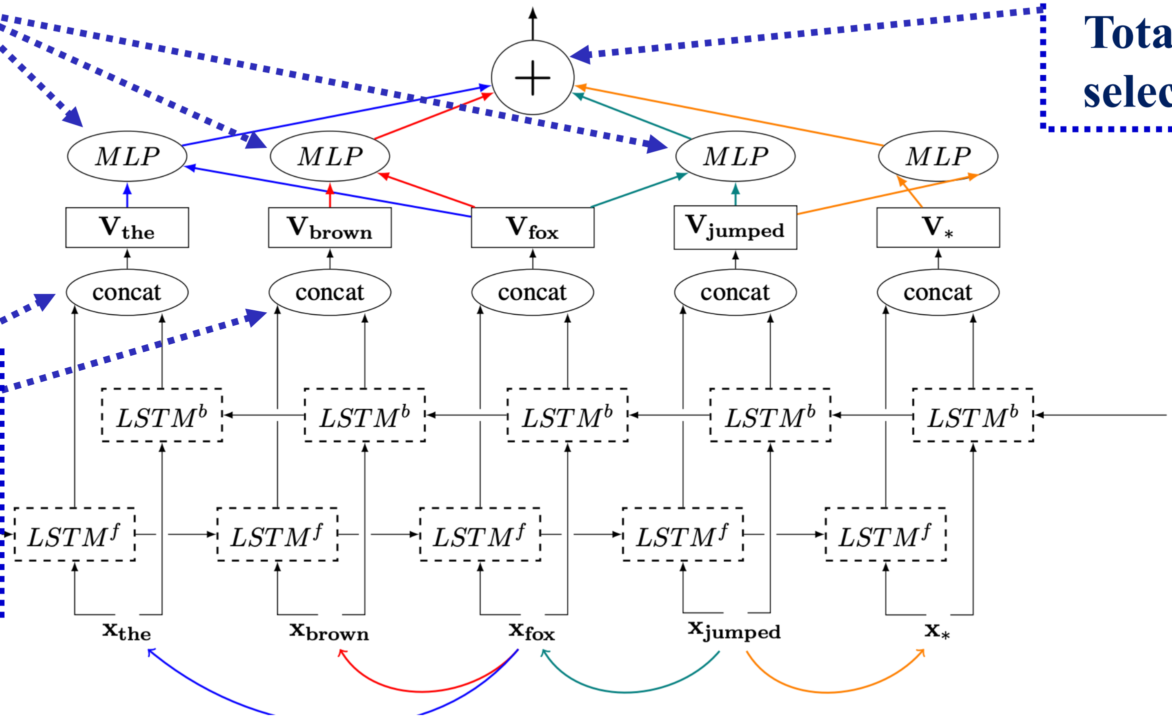
- **Which arcs** to keep and with **what labels?**
  - The selected arcs **must form a tree** (e.g., no circles).
  - And it must be the **correct tree**.
- **Arc-factored** graph-based dependency parsers:
  - **Score each candidate arc** (and candidate label) separately.
  - **Greedily** assign **to each word (modifier)** the **head** of the **arc** with the **best score**, even if the result is **not a tree**.
  - Or use a **decoder** to select the **tree** with the **best total score**.

Graph-based dependency parsing

An **MLP** computes the **score** of each candidate **arc** **(word pair)**.

Total score of the **selected** arcs/tree.

A **biLSTM** produces **context aware** word embeddings.



A **decoder** selects the **“best” tree** (**max total score of arcs**, arcs **forming a tree**). Alternatively, we can **greedily** link **each modifier** to its **most probable head**, even if the selected arcs **may not form a tree**. We may use the greedy approach always, or only during training, and use a decoder for testing.

From the paper of E. Kiperwasser and Y. Goldberg “Simple and Accurate Dependency Parser Using Bidirectional LSTM Feature Representations”, *Transactions of ACL*, vol. 4, pp. 313 – 327, 2016. https://aclweb.org/anthology/Q16-1023

Graph-based dependency parsing

- **Hinge loss** (L) between the **correct tree *y*** and the **most highly scored incorrect tree *y'*** (e.g., from a sample):

$$L = \max\left(0, m - \sum_{(h,m)\in y} \mathrm{MLP}([v_h; v_m]) + \max_{y'\neq y} \sum_{(h,m)\in y'} \mathrm{MLP}([v_h; v_m])\right)$$

  - If the **score of the correct (gold)** tree y **exceeds** that of the **top-scored incorrect** tree y' by a **margin** m, then $\boldsymbol{L = 0}$.
  - Otherwise, **loss** $\boldsymbol{\neq 0}$ and we **back-propagate** to update the weights of the MLP(s), biLSTM(s) etc.
  - See the paper of K&G for further improvements.
- **The hinge loss** can also be used in **other applications**.
  - It **does not** require **probability** scores. It only cares to **distinguish** the scores of **good** and **bad** instances by a **margin**.
  - E.g., it **does not** try to make the **probability** of a **gold tree** become **1**. It **suffices** if its score already **exceeds** the scores of (the best) **other candidate** trees by a **margin** m.

Chomsky's hierarchy of grammars

- **Type 3** (**regular grammars**, **right linear** or **left linear**):
  - Rules of the form ***A*** → ***x*** and ***A*** → ***x B*** (for right linear).
  - Rules of the form ***A*** → ***x*** και ***A*** → ***B x*** (for left linear).
  - ***x***: (possibly empty) **sequence of terminal symbols**.
  - ***A***, ***B*****:** single **non terminal symbols**.
  - Example of right linear regular grammar:
    NP → the Nominal NP → a Nominal
    Nominal→ large Nominal Nominal→ nice Nominal
    Nominal→ easy to drive Nominal
    Nominal → N N → person N → car
- **Type 2** (**context-free grammars**):
  - Rules of the form ***A*** → ***α***.
  - ***A*****:** single **non terminal symbol**.
  - ***α***: (possibly empty) **sequence of terminals** and **non terminals**.
  - E.g., we can now have the rule: NP → Det Nominal.

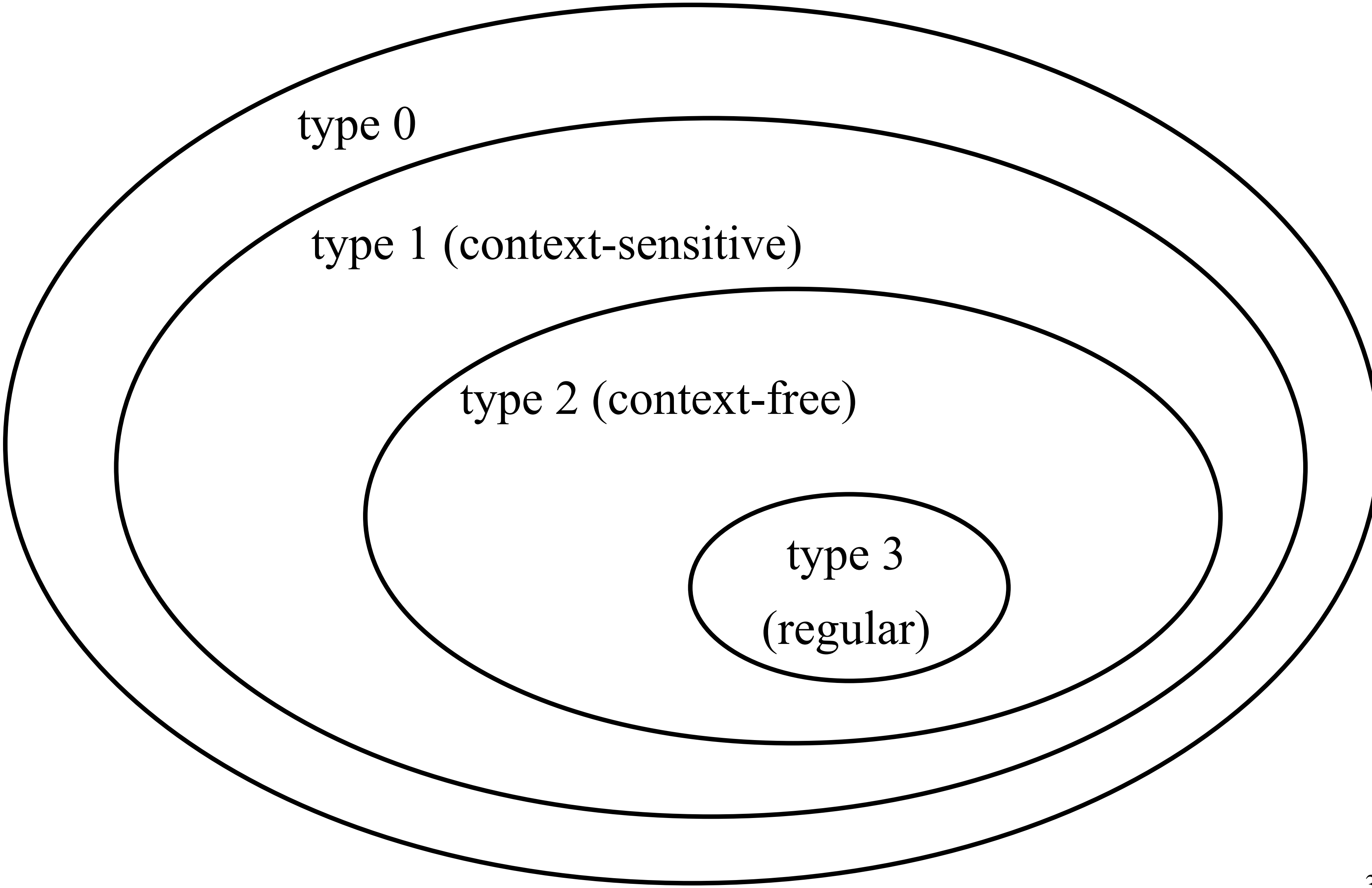
Chomsky's grammar hierarchy – cont.

- **Type 1 (context-sensitive grammars)**:
  - Rules of the form $\boldsymbol{\alpha A \beta} \rightarrow \boldsymbol{\alpha\gamma\beta}$.
  - $\boldsymbol{\alpha, \beta, \gamma}$: **sequences** of **terminals** and **non terminals** (γ must not be empty, whereas α, β can be empty).
  - E.g., (Date) → (Day / Month / Year)
  - The rule S → ε is also allowed, where S is the initial symbol and ε the empty string, provided that S is not used on the right-hand side of any rule.
  - **Alternative definition**: rules $\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}$, **with** $\mathbf{0 \leq |\alpha| \leq |\beta|}$.
  - The length of sequence α must me smaller or equal to that of sequence β. We can define the same languages, as with the first definition of Type 1 grammars, with the exception of languages that include ε.
- **Type 0 (recursively enumerable grammars)**:
  - Rules of the form $\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}$ (α not empty, $\boldsymbol{\beta}$ may be empty).
  - $\boldsymbol{\alpha, \beta}$: **sequences** of **terminals** and **non terminals**.

Generative power of grammars

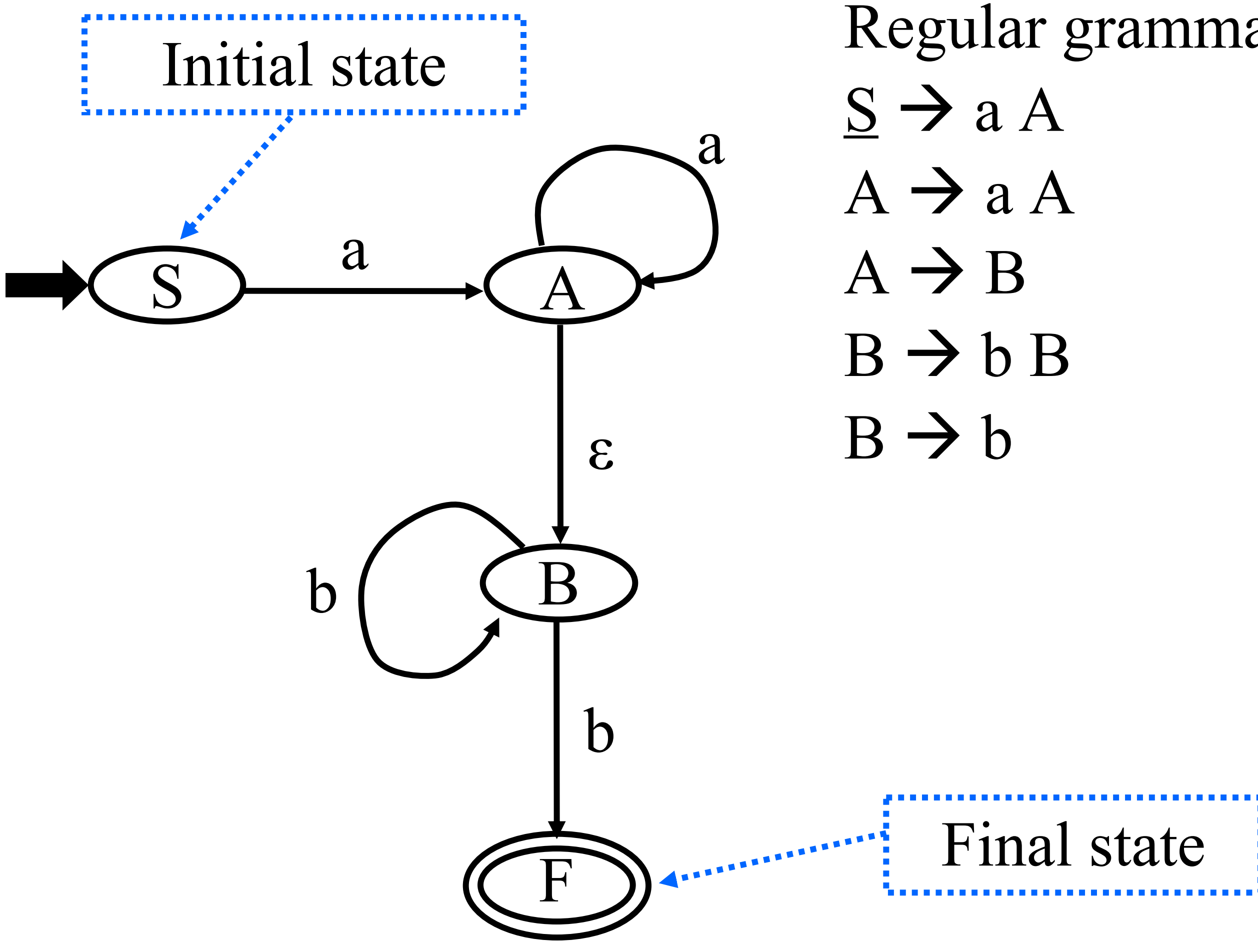
- **languages(*T*)**: The set of languages that can be defined with grammars of type *T*.
- **languages(type 3) ⊂ languages(type 2)**
  - E.g., **regular** grammars **cannot** define languages of the form $\mathbf{a}^n\mathbf{b}^n$ (language containing ab, aabb, aaabbb etc.),
  - whereas **context-free grammars can** (S → ab and S → aSb).
- **languages(type 2) ⊂ languages(type 1)**
  - E.g., **context-free** grammars **cannot** define languages of the form $a^n b^n c^n$,
  - whereas **context-sensitive** grammars **can** (S → abc, S → aSBc, cB → Bc, bB → bb, type 1 by the 2nd definition).
- **languages(type 1) ⊂ languages(type 0)**

Generative power of grammars



Finite state automata (FSAs)

For the language $a^m b^n$, with $m, n > 0$.



Regular grammar:

<u>S</u> → a A

A → a A

A → B

B → b B

B → b

Grammars and automata

- **Regular grammars** are equivalent to **finite state automata** (FSAs).
  - For each **regular grammar**, we can define an **FSA** to **produce or admit** the **same language** and vice versa.
- **To check** if a **sequence of terminals** belongs to the **language**, we **feed the sequence** to the **FSA**.
  - The FSA "reads" (consumes) the symbols of the sequence one by one, changing state (or remaining at the same state) after each symbol, if there is a corresponding (allowed) transition in its graph.
  - If there is no corresponding transition, the FSA gets stuck.
  - The **sequence of terminals** is **part of the language** if there is a **sequence of transitions** of the **FSA** that allows it to **consume the entire sequence of terminals**, leaving the FSA at a **final state**.

Grammars and automata

- **Context-free grammars** are equivalent to **non deterministic pushdown automata** (PDAs).
  - The automaton also has a **stack**, which has to be **empty** at the **final st**ates.
  - In $\mathbf{a}^n\mathbf{b}^n$ languages, we need the stack to know how many b symbols we need after the a symbols. Each time we encounter an a, we push a symbol to the stack. Each time we encounter a b, we pop a symbol from the stack.
  - **Non-deterministic**: the current state and the symbol being read (and the contents of the stack for PDAs) **do not functionally determine** the **next state**.
  - Every non-deterministic FSA (without a stack) can be converted to a deterministic FSA (with more states), but this does hold for PDAs.
- **Type 0 grammars** are equivalent to **Turing machines**.

What grammars for natural languages?

- **Almost all** the **syntactic phenomena** of natural languages can be captured using **regular grammars**.
  - Hence we can parse NLs using **FSAs**, for which there are very **efficient algorithms**.
  - But often we use **CFGs** because they are **shorter** (fewer rules).
  - And because their **parse trees** are **more useful in semantics**.
- There are **syntactic phenomena** that **seem to require CFGs**:
  - *The cat likes tuna fish.*
  - *The cat (that) the dog chased likes tuna fish.*
  - Similarities with $\mathbf{a}^n\mathbf{b}^n$ languages ($NP^n V^n$ tuna fish).
  - *The intersection (common sentences) of English with the regular language [$NP^n V^m$ tuna fish] is [$NP^n V^n$ tuna fish], which is non-regular. Hence, English is not a regular language, because the intersection of regular languages is a regular language.*
  - But **even people have trouble** for $n > 2$.
  - For **finite** values of n, **regular grammars** are enough.

What grammars for natural languages?

- There are syntactic phenomena in **some natural languages** that require **context-sensitive grammars**.
  - Swiss German and Bambara (Mali and neighbouring countries).
  - In Swiss German there are expressions of the form $wa^nb^mxc^nd^my$.
- But in most natural languages no phenomena of this kind have been found.

Gender agreement with CFGs

- S → NP VP | VP
- NP → Pron | DetFem PNFem | DetFem NominalFem | DetMasc PNMasc | DetMasc NominalMasc
- NominalFem → NFem | AdjFem NominalFem | NominalFem PP
- NominalMasc → NMasc | AdjMasc NominalMasc | NominalMasc PP
- VP → V | V NP
- PP → Prep NP
- Pron → εγώ
- DetFem → η | μια | την
- DetMasc → ο | έναν | τον
- PNFem → Θεσσαλονίκη | Αθήνα
- NFem → πτήση
- NMasc → πελάτης | πελάτη
- AdjFem → πρωινή | απογευματινή
- V → θέλω | θέλει| προτιμώ | προτιμά
- Prep → προς | από

Twice as many gender-sensitive rules. Even more rule variants for **number** and **case agreement**.

Gender agreement with augmented CFGs

- S → NP VP | VP
- NP → Pron | Det(**G**) PN(**G**) | Det(**G**) Nominal(**G**)
- Nominal(**G**) → N(**G**) | Adj(**G**) Nominal(**G**) | Nominal(**G**) PP
- VP → V | V NP
- PP → Prep NP
- Pron → εγώ
- Det(**masc**) → ο | έναν | τον
- Det(**fem**) → η | μια | την
- PN(**fem**) → Θεσσαλονίκη | Αθήνα
- N(**fem**) → πτήση
- N(**masc**) → πελάτης | πελάτη
- Adj(**fem**) → πρωινή | απογευματινή
- V → θέλω | θέλει| προτιμώ | προτιμά
- Prep → προς | από

Similar features for number, case:

Det(fem, **nom**, **sing**) → η

Not a CFG any more, but **can be converted to a CFG** with more rules, provided that the **possible feature values** are **finite**.

Parsing with Prolog

- **Prolog** supports **DCGs** out of the box.
  - **Definite Clause Grammars** are in effect **augmented CFGs**, as in the gender agreement slides.
  - It converts them to **First-Order Logic Horn clauses** and treats parsing as an inferencing problem.
  - In effect, it parses **top-down** with **depth-first search**.
  - We will use Prolog only to easily experiment with grammars.
- **More elaborate** parsing **algorithms** used in practice.
  - E.g., **CKY**, **Earley**, possibly **modified**, to support augmented CFGs.
  - They can also be implemented in Prolog (or other programming languages).

DCGs in Prolog

- **Augmented CFGs** written in the form:
  nominal(G) → adj(G), nominal(G).
  det(masc) → [έναν].
  - **Terminal symbols** written in **square brackets**.
  - Symbols starting with **capital let**ters are **variables**.
- **Limitation** due to the built-in DFS parsing:
  - We need to avoid rules with **left recursion**.
  - E.g., nominal → nominal, pp.
  - More generally rules allowing productions of the form:
    A → ... **→ A** ...

Example of DCG

```
s --> np, vp.
s --> vp.

np --> pron.
np --> det(G), pn(G).
np --> det(G), nominal(G).

nominal(G) --> n(G).
nominal(G) --> adj(G), nominal(G).
% Avoiding left recursion:
% nominal(G) --> nominal(G), pp.
nominal(G) --> n(G), manypp.
manypp --> pp.
manypp --> pp, manypp.
```

```
vp --> v.
vp --> v, np.

pp --> prep, np.

pron --> [εγώ].

det(masc) --> [ο].
det(masc) --> [έναν].

…
```

(Consult the course's documents for many more examples.)

Experimenting with DCGs

- You will need a **Prolog interpreter**.
  - Recommended: **SWI-Prolog** (http://www.swi-prolog.org/).
- **Loading** the grammar file (plain text):
  - **consult**(…) at the Prolog prompt.
  - For Windows: double-click on the .pl grammar file.
  - Many examples of DCGs in the course's documents.
- **Parsing**, once the grammar is loaded:
  - **phrase(s, [θέλω, μια, πτήση, από, την, αθήνα]).**
  - **phrase(nominal(masc), [πελάτης, από, την, αθήνα]).**
  - A **yes/no** response by Prolog means a parse tree (with the specified root) was found or not.

Experimenting with DCG – cont.

- **Queries** to the parser:
  - **phrase(nominal(G), [πελάτης, από, την, αθήνα]).**
  - Response: **G = masc**.
  - Typing «;» requests another solution (here there isn't).
- Returning the **parse tree**:
  - We can extend the grammars (see below), to make Prolog also report the parse tree:
  - **phrase(nominal(G, T), [πελάτης, από, την, αθήνα]).**
  - Response: **G = masc** and:
  - **T = nominal(n(πελάτης),**

manypp(pp(prep(από),

np(det(την),

pn(αθήνα)))))

Nodes with subtree representations

nominal (fem, nominal(adj(πρωινή), nominal(n(πτήση))))

At each internal node of the parse tree we now have an **extra feature** representing the **subtree** below that node.

nominal(fem, nominal(n(πτήση)))

adj(fem, adj(πρωινή))

n(fem, n(πτήση))

πρωινή

πτήση

New form of the DCG rules

adj(fem, adj(πρωινή)) --> [πρωινή].

n(fem, n(πτήση)) --> [πτήση].

n(masc, n(πελάτης)) --> [πελάτης].

nominal(G, nominal(T)) --> n(G, T).

nominal(G, nominal(T1, T2)) --> adj(G, T1), nominal(G, T2).

If you find and adjective of gender G and subtree T1, followed by a nominal of gender G and subtree T2, then you have found a (larger) nominal of gender G with parse tree nominal(T1, T2).

See file *tree_structure.pl* in the documents of the course.

Chunking

- **Sentence chunking** into **non-overlapping segments**.
  - Usually (non-recursive) NPs, VPs etc.

    [$_{NP}$The morning flight] [$_{PP}$ from Athens] [$_{VP}$ has landed].
- **Flat structure** produced instead of deeper trees.
- We can train **sequence labeling** algorithms (e.g., with sliding windows, RNNs, CNNs, Transformers).
  - **B-NP**: initial word of NP.

    I-NP: inside word of NP.

    B-VP: initial word of VP.

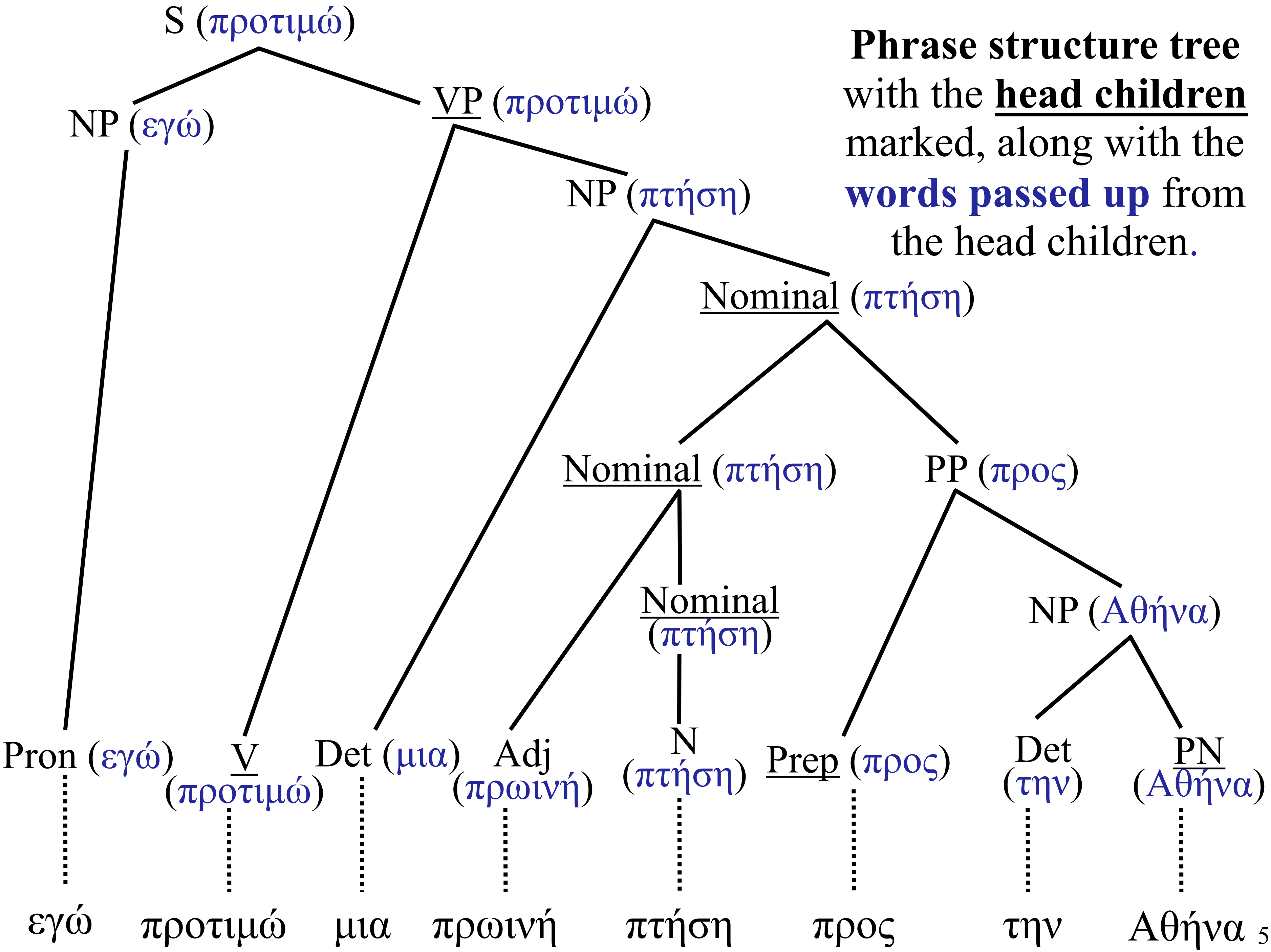
    ...

    O: word outside any other segment

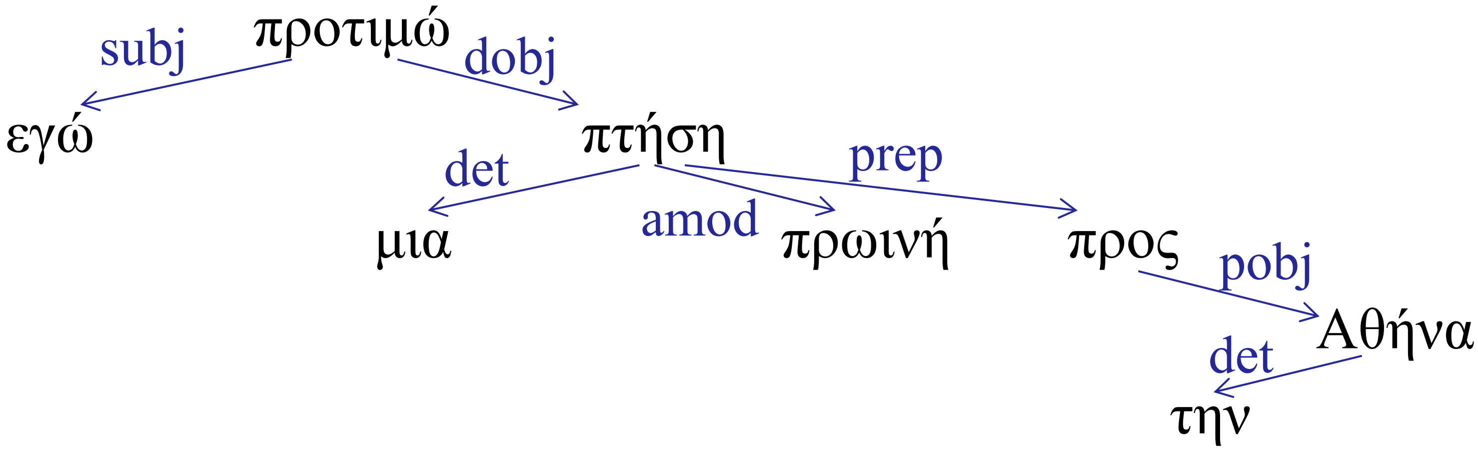
Head children

- In rules with **only one right-hand side symbol**, that symbol (child) is the **head child**.
  - E.g., Nominal → N
- In rules with **multiple right-hand side symbols**, we can **define which** symbol is the **head child**.
  - E.g., S→ NP VP and VP → V NP
  - Usually the (main) **verb** is considered the head child of a verb phrase, the **verb phrase** is considered the head child of a sentence, the (main) **noun** is considered the head child of a noun phrase etc.
  - Or we may have **separate rules** to **traverse** the **parse tree** and **mark the head child** of each non-terminal node.

Phrase structure tree with the **head children** marked, along with the **words passed up** from the head children.

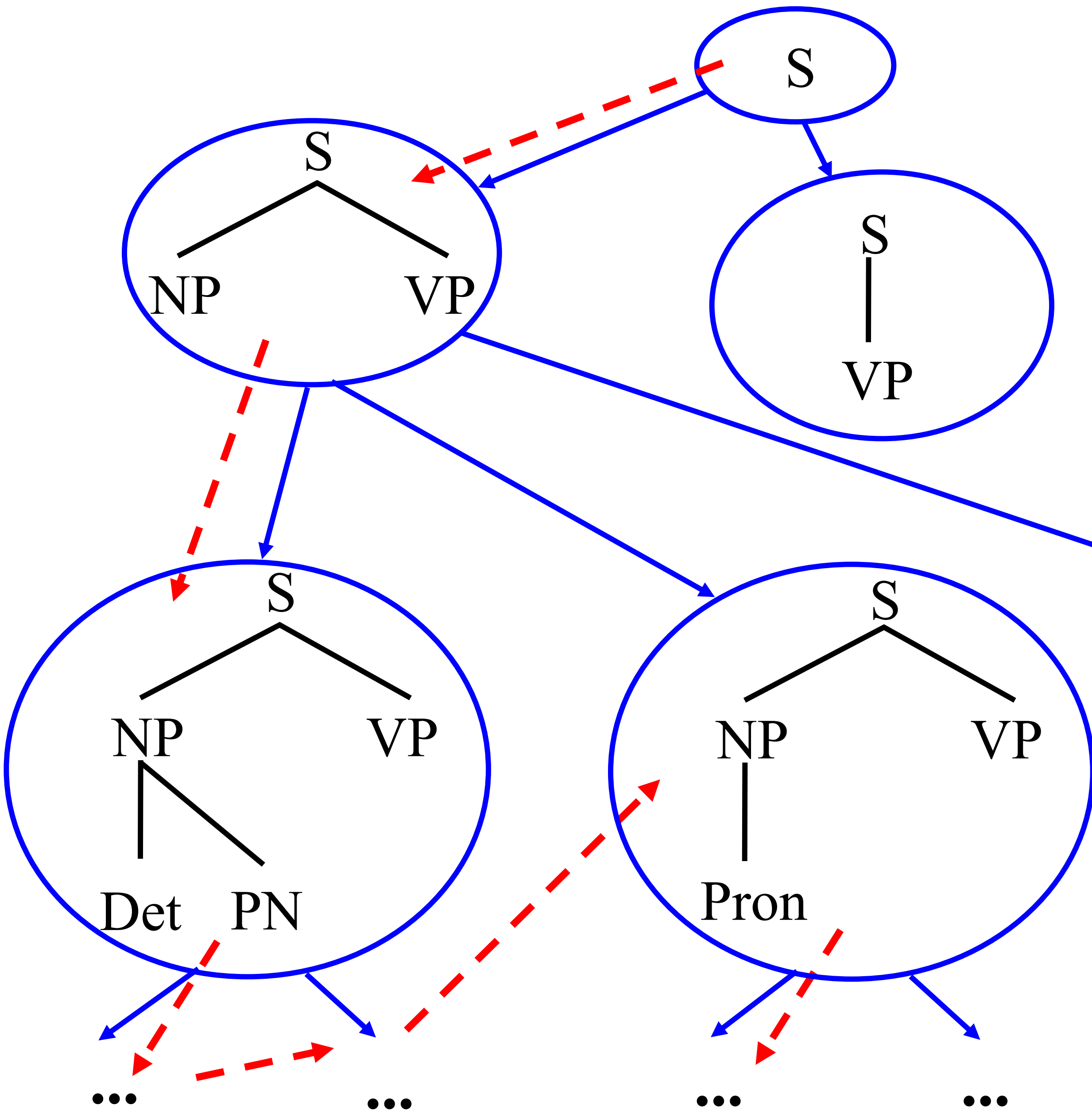


Phrase structure trees to dependency trees



- **Producing dependency trees** from **phrase structure trees**:
  - Create a **node** for each **word**.
  - **For each node** of the **phrase structure tree** that has **more than one children**, add **dependencies** from the word of the **head child** to the words of each one of the **other children**.
  - Usually **separate rules** produce the **labels** of the arcs.
- See also slides for **parsing algorithms** that produce **directly dependency trees**.

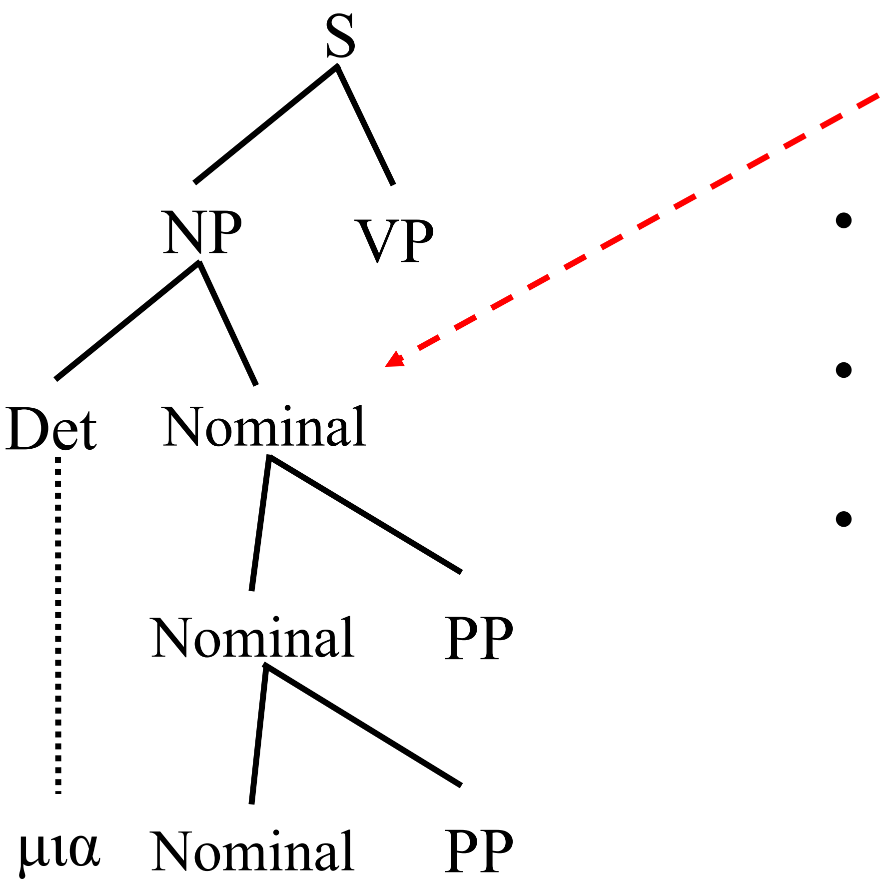
Finding phrase structure trees via depth-first search



We usually **extend** the **leftmost non terminal** we can, using the **topmost rule** of the grammar that applies.

Problems with left recursion

- We search top-down with DFS and backtracking.
- The input sequence does not agree with the grammar:
  - «μια από την Αθήνα»
- We have produced the tree:



- The first two rules for Nominal fail:

  Nominal → N

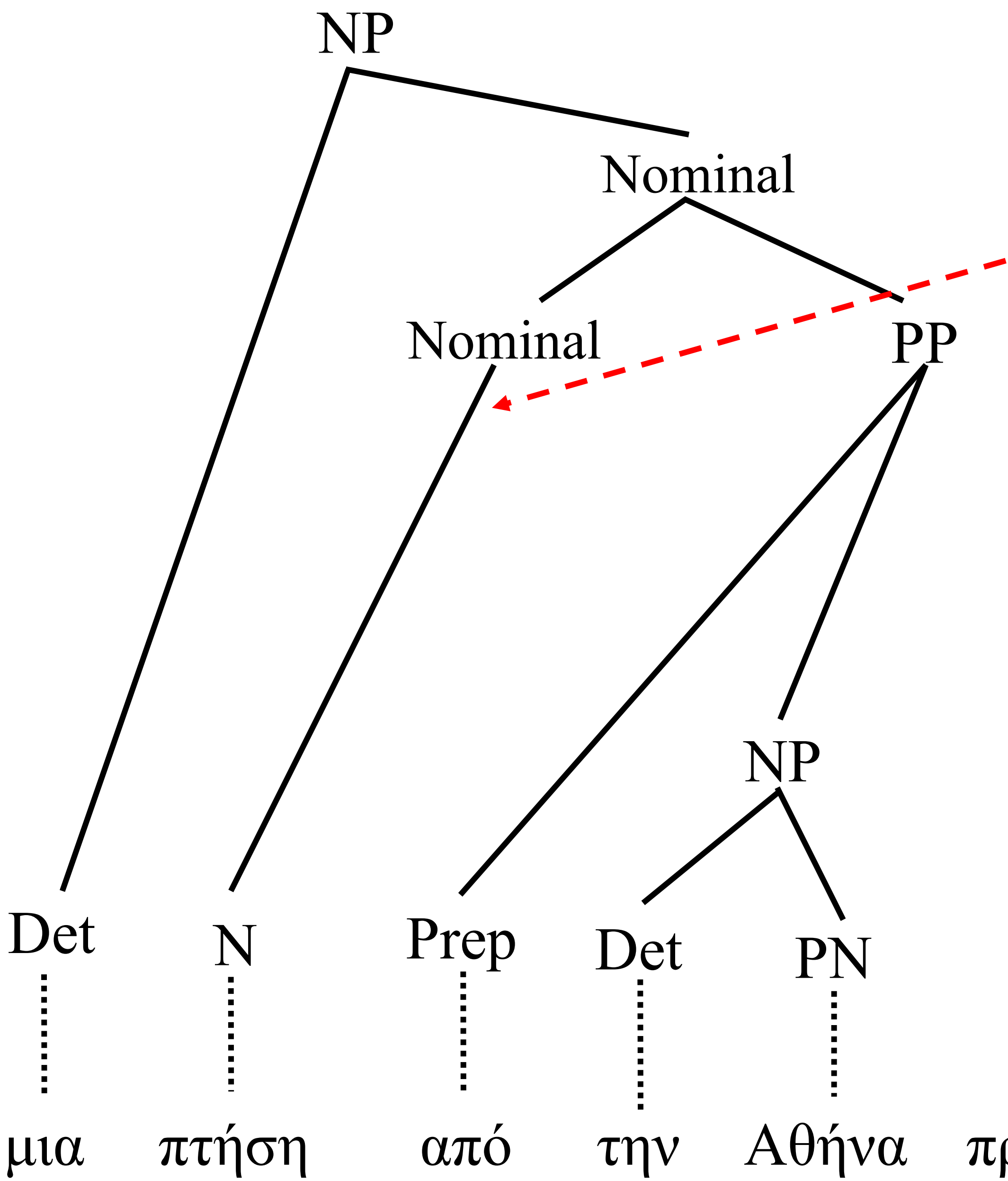
  Nominal → Adj Nominal
- We try the third rule:

  Nominal → Nominal PP
- Infinite loop without consuming words of the input.
- If the third rule is above the other two, we get an infinite loop even if the input agrees with the grammar.

Problems with left recursion

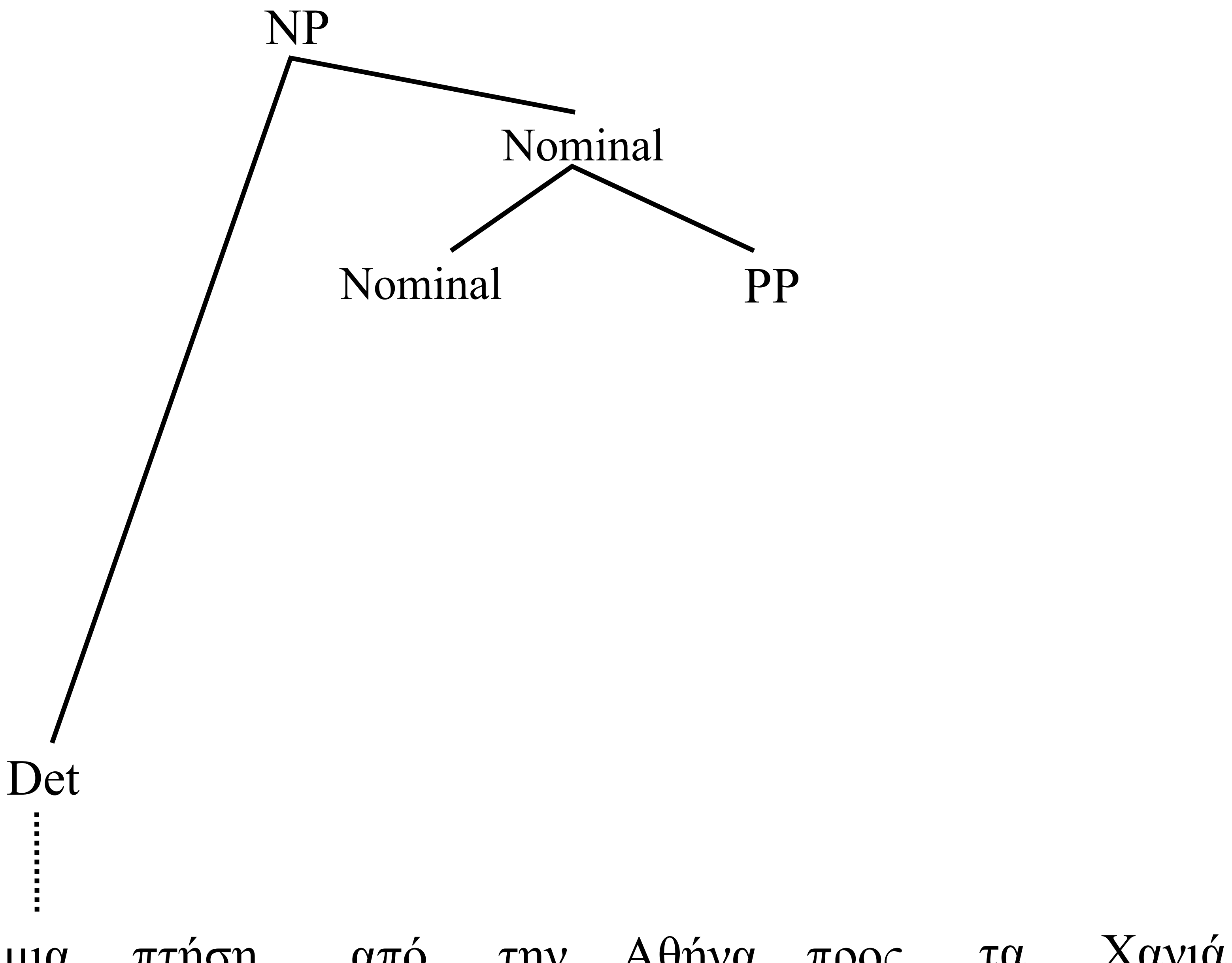
- The problem is caused by **rules of the form**:
  - $\mathbf{A_1} \rightarrow A_2\ \alpha_1$ (A$_i$ non-terminal, α_i sequences of terminals
  - $A_2 \rightarrow A_3\ \alpha_2$ and non-terminals)
  - ...
  - $A_n \rightarrow \mathbf{A_1}\ \alpha_n$
- The problem can often be **solved** by **modifying the grammar**, to avoid left recursion.
- Similar problems with other **generic search algorithms** when applied to parsing.
  - E.g., if there is **left recursion** in the grammar, **best-first search** finds the parse tree if there is one, but **never stops if there is no parse tree**, because the search space is infinite.

Reparsing the same subtrees

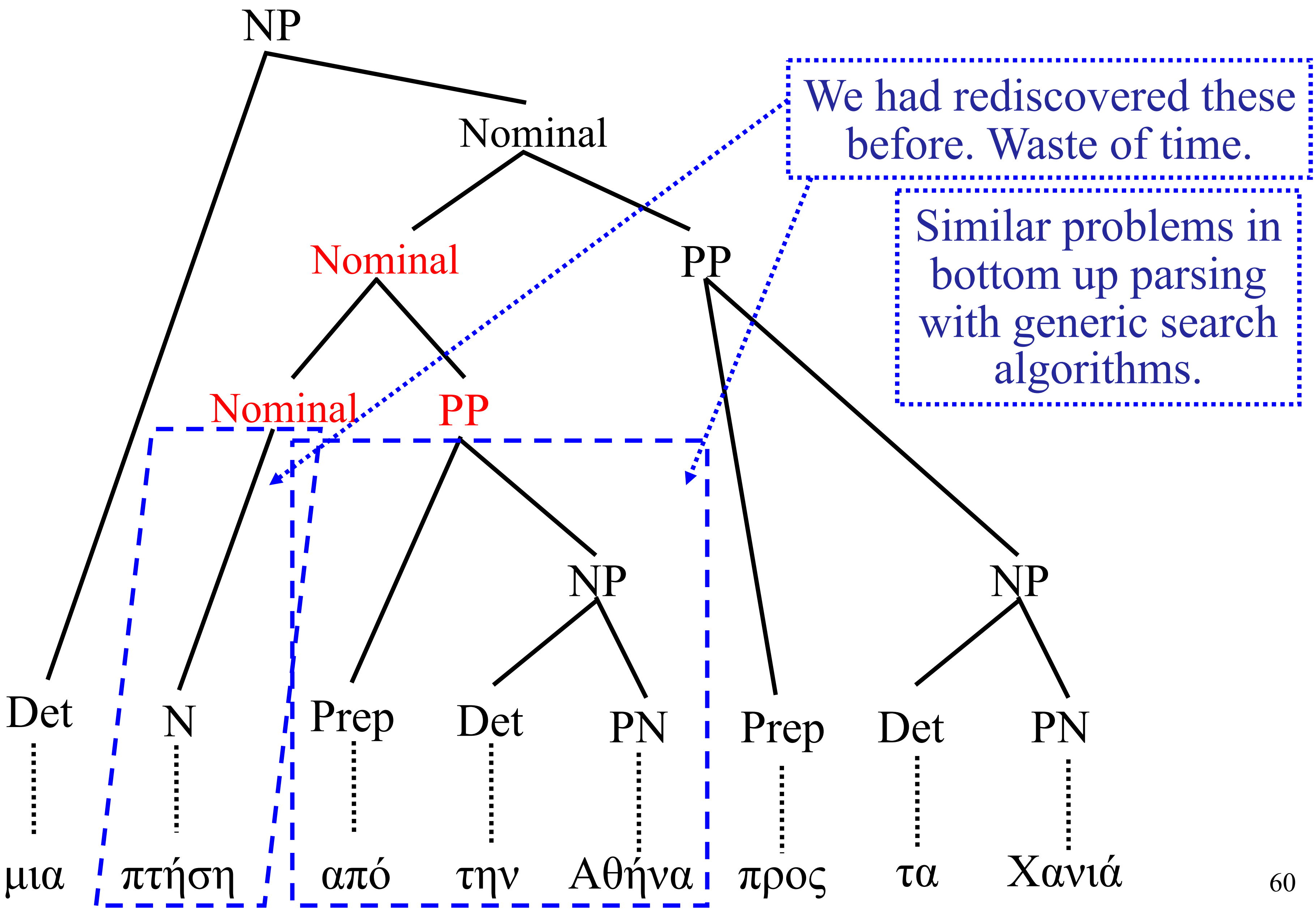


- Top-down with DFS and backtracking.
- Wrong rule selected.
- Uncovered words left.
- Forced to backtrack.

Reparsing the same subtrees



Reparsing the same subtrees



Probabilistic CFGs (PCFGs)

- Like plain CFGs, but now **each rule has a probability**.

  S → NP VP [0.7]
  S → VP [0.3]
  NP → Det Nominal [0.6]
  NP→ Det PN [0.4]
  …
  V → θέλω [0.03]
  V → επιθυμώ [0.02]
  …

  The total probability of all the rules for S must be 1.

  The total probability of all the rules for NP must be 1.

  The total probability of all the rules for V must be 1.

- The probability of each rule shows **how likely** it is for the **left-hand side** non-terminal to have the **form** of the **right-hand side**.
  - The scores are **conditional probabilities**, like *P*(NP VP | S).

Probability of a parse tree

- We take the **probability** of each **parse tree** to be the **product** of the **probabilities** of the **rules** that were used to construct it.
  - Hence, we assume that **rule applications** are **independent**...

    S → NP VP [0.7]

    S → VP [0.3]

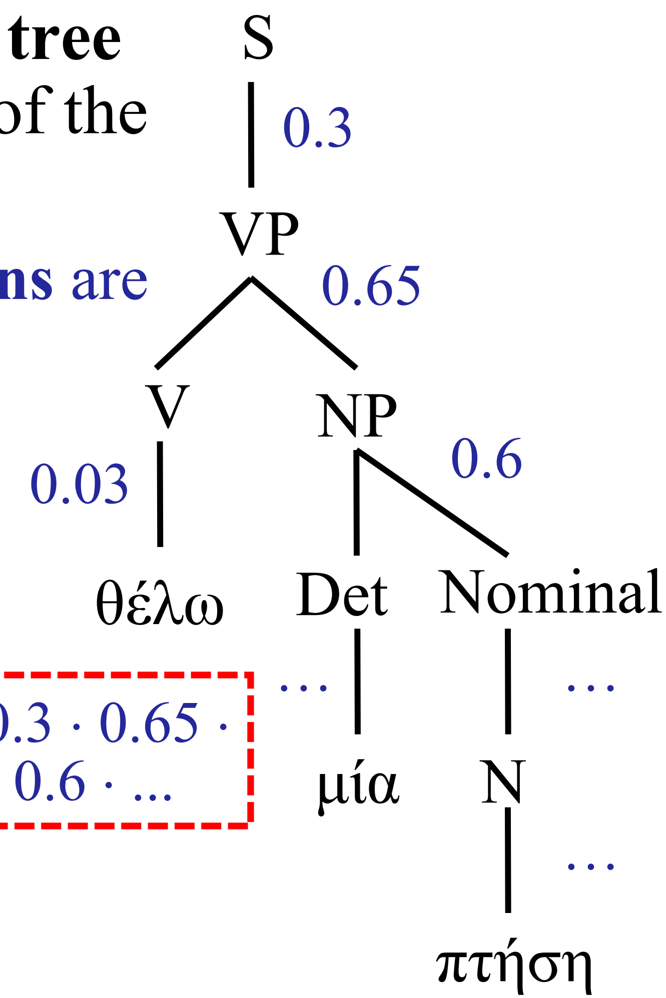
    …

    VP → V NP [0.65]

    NP → Det Nominal [0.6]

    …

    V → θέλω [0.03]



$P(T) = 0.3 \cdot 0.65 \cdot 0.03 \cdot 0.6 \cdot \ldots$

- If we get **multiple parse trees** for a sentence, we **prefer the most probable one**.

Probabilistic CKY

- For **probabilistic CFGs** in **Chomsky Normal Form** (CNF).
- Only rules of the form ***A* → *B C* [*p*]** and ***A* → *w* [*p*]**, where ***A*, *B*, *C* non-terminals**, ***w* terminal**, ***p* probability**.

S → V NP [0.7]

...

V → επιθυμώ [0.01]

V → θέλω [0.03]

...

Det → μια [0.2]

Adj → πρωινή [0.01]

NP → Det Nominal [0.8]

...

Nominal → Adj N [0.4]

Nominal → πτήση [0.01]

...

N → πτήση [0.02]

Adj → απογευματινή [...]

Probabilistic CKY

(0) θέλω (1) μία (2) πρωινή (3) πτήση (4)

	0	1	2	3	4
0		V [0.03] (0,1)			
1			Det [0.2] (1,2)		
2				Adj [0.01] (2,3)	
3					N [0.02] Nominal [0.01] (3,4)

Probabilistic CKY

0 θέλω 1 μία 2 πρωινή 3 πτήση 4

	0	1	2	3	4
0		V [0.03] (0,1)	(0,2)		
1			Det [0.2] (1,2)	(1,3)	
2				Adj [0.01] (2,3)	**Nominal [0.4 · 0.01 · 0.02]** (2,4)
3					N [0.02] Nominal [0.01] (3,4)

Probability of the rule (0.4).

How do we learn the rules and probabilities?

- The most common way is to use a **treebank**.
  - **Corpus** with **sentences annotated** (usually manually) with their **parse trees**.
  - The **rules** follow **from the parse trees**.
  - We usually **exclude** very **rare rules**.
  - **Probabilities** of the remaining rules:

$$P(\alpha \to \beta) = \frac{\text{count}(\alpha \to \beta)}{\text{count}(\alpha)}$$

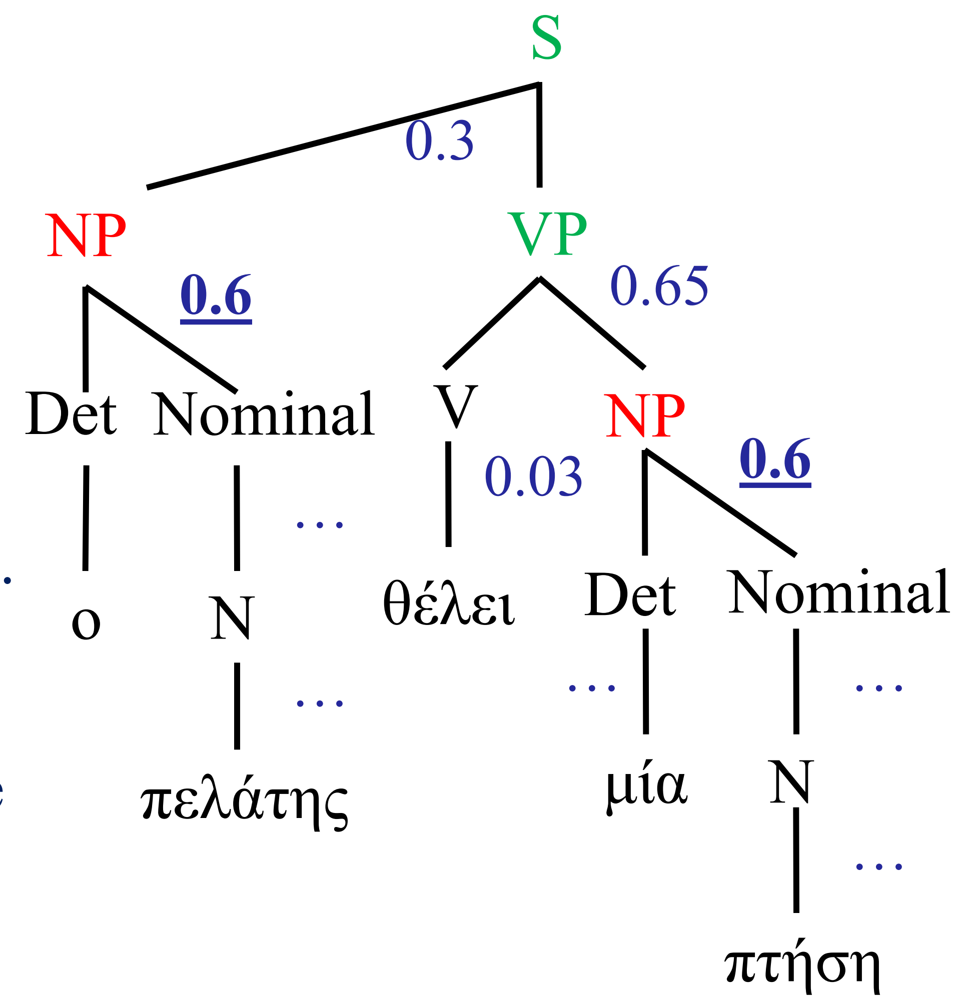
How frequently does α become β in the corpus?

How frequent is the non-terminal α in the corpus?

- If we only have **plain texts**, without manually annotated trees, we can use a form of **Expectation Maximization** (EM).
  - "Inside-outside" algorithm. See J&M.

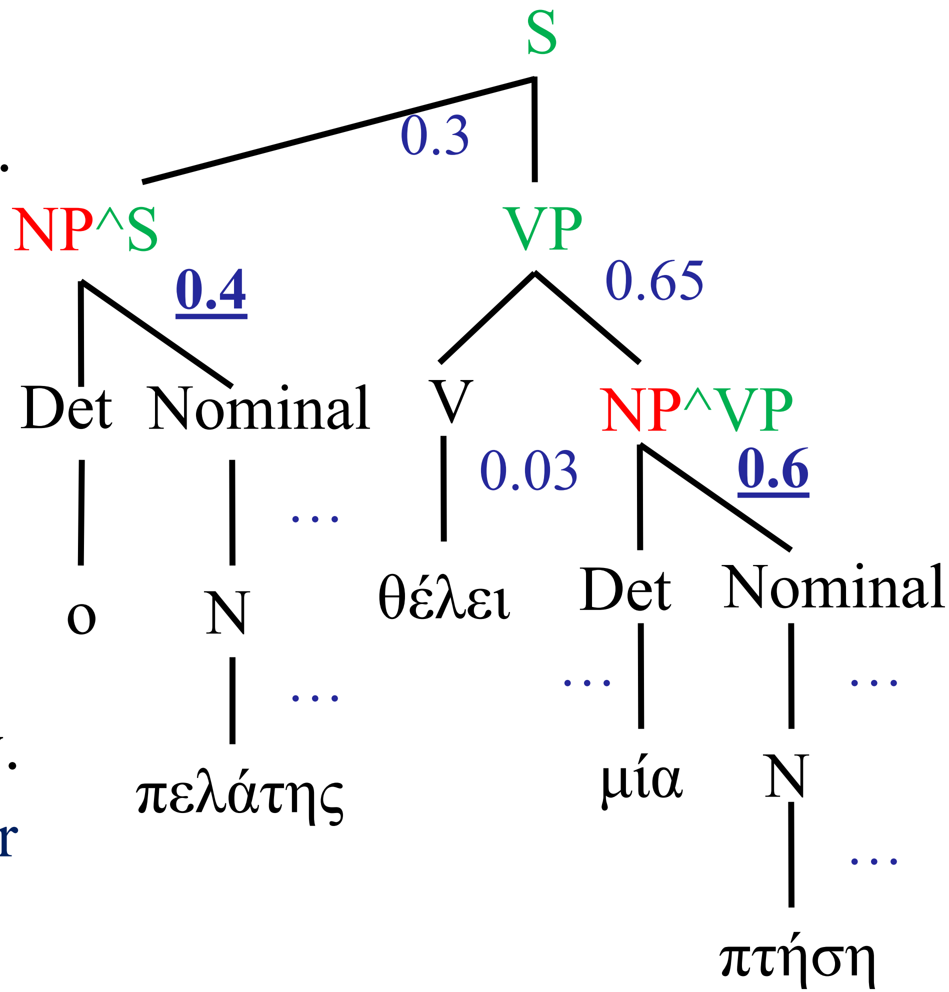
Problems with PCFGs

- They assume that **rule applications** are **independent**.
  - E.g., that applying the rule NP → Det Nominal is **equally probable** regardless of whether the **father** of the NP is S or VP.
  - But if the **father** of the NP is an S, the probability of NP → Det Nominal may be **lower**, perhaps because NP → Pron is more likely.
  - Perhaps **more likely** to encounter a **pronoun** as a **subject**, than as an **object**.

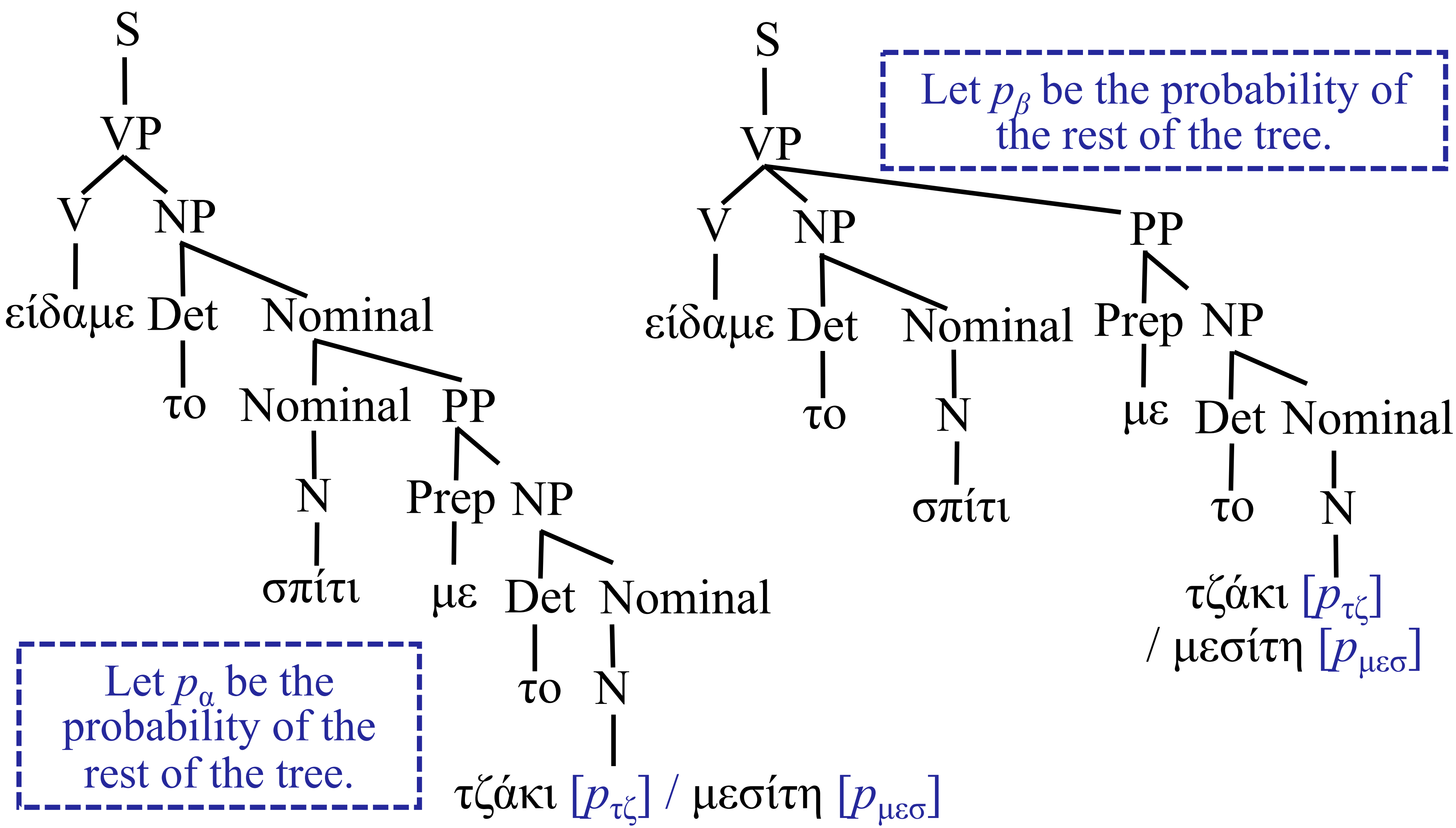


Splitting non-terminals

- We can distinguish NP^S (NP with S father) from NP^VP (NP with VP father).
  - S → NP^S VP [0.3]
  - NP^S → Det Nominal [**0.4**]
  - VP → V NP^VP [0.65]
  - NP^VP → Det Nominal [**0.6**]
- We now have **two variants** of NP → Det Nominal, each with a **different probability**.
  - One for **subject NP**, one for **object NP**.
- We can **split other non-terminals** too.



Problems with PCFGs



"We saw the house with the fireplace/broker."

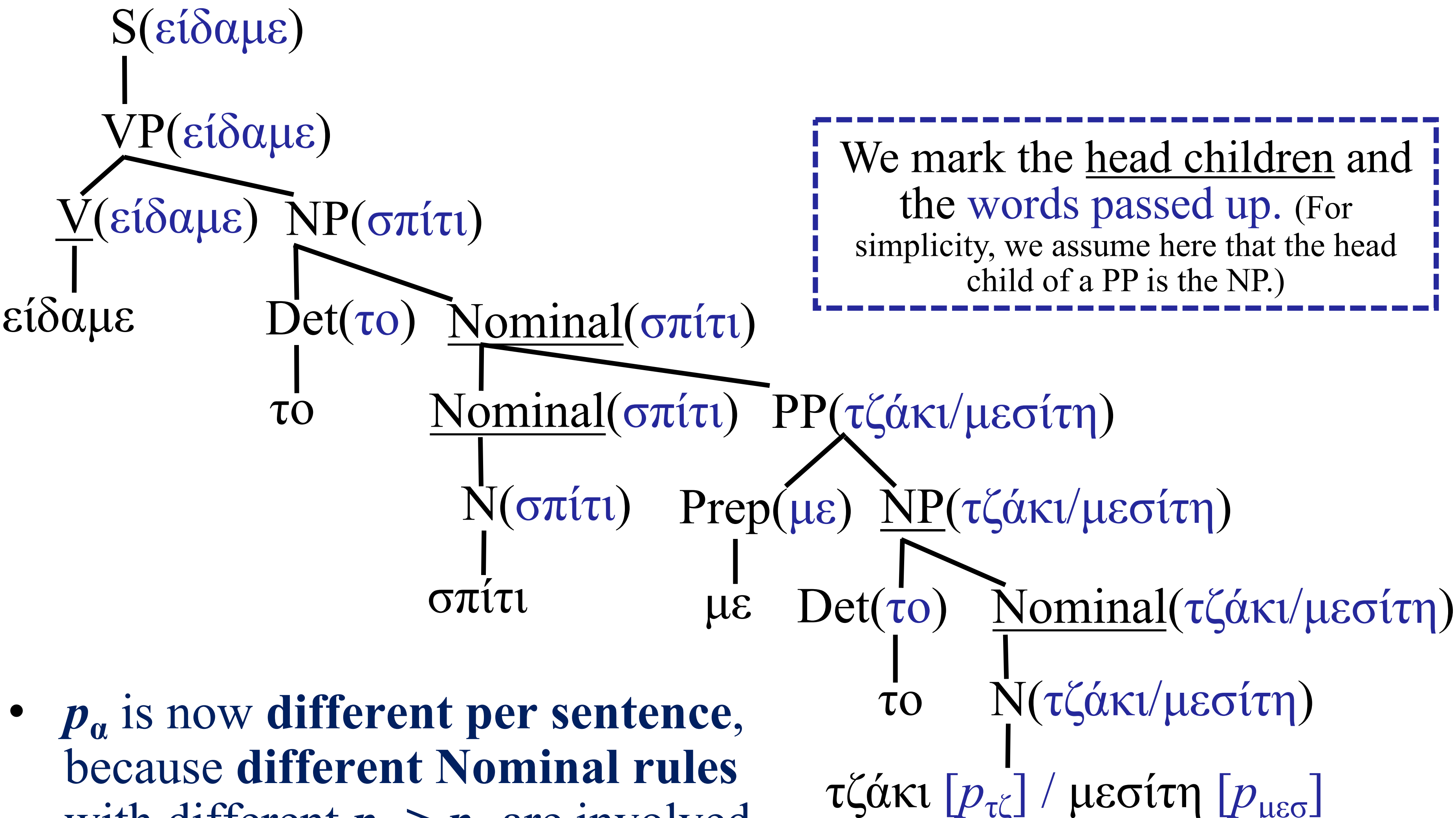
Problems with PCFGs

- Είδαμε το [σπίτι με το τζάκι]. $p_\alpha \cdot \cancel{p_{\tau\zeta}}$
- Είδαμε [το σπίτι] [με το τζάκι]. $p_\beta \cdot \cancel{p_{\tau\zeta}}$

- Είδαμε το [σπίτι με το μεσίτη]. $p_\alpha \cdot \cancel{p_{\mu\varepsilon\sigma}}$
- Είδαμε [το σπίτι] [με το μεσίτη]. $p_\beta \cdot \cancel{p_{\mu\varepsilon\sigma}}$

- If $\boldsymbol{p_\alpha > p_\beta}$, we prefer the **left tree** in **both sentences**.
- If $\boldsymbol{p_\alpha < p_\beta}$, we prefer the **right tree** in **both sentences**.
- We **want** to prefer the **left tree** in the **first sentence** (with τζάκι, fireplace) and the **right tree** in the second sentence (with μεσίτη, broker).

Lexicalized PCFGs



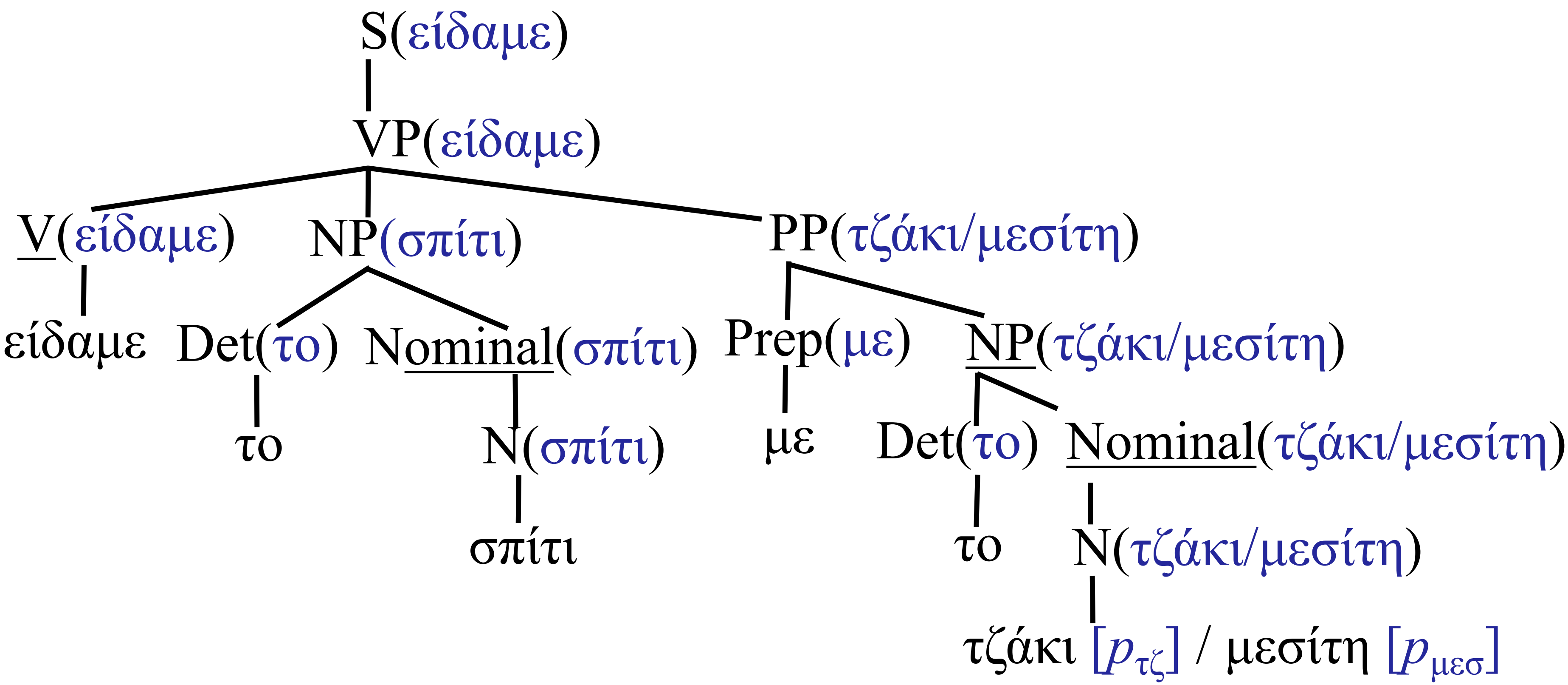
We mark the <u>head children</u> and the words passed up. (For simplicity, we assume here that the head child of a PP is the NP.)

- $p_α$ is now **different per sentence**, because **different Nominal rules** with different $p_1 > p_2$ are involved.

Nominal(σπίτι) → Nominal(σπίτι) PP(τζάκι) [p_1]

Nominal(σπίτι) → Nominal(σπίτι) PP(μεσίτη) [p_2]

Lexicalized PCFGs



- p_β is now also **different per sentence**, again because **different Nominal rules** with different $p_3 < p_4$ are involved.

VP(είδαμε) → V(είδαμε) NP(σπίτι) PP(τζάκι) [p_3]

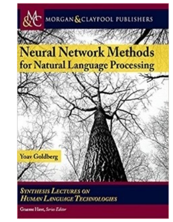
VP(είδαμε) → V(είδαμε) NP(σπίτι) PP(μεσίτη) [p_4]

Lexicalized PCFGs and CPCFGs

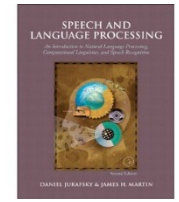
- **Improved results** compared to non-lexicalized PCFGs.
- **Much larger number of rules**, more difficult to estimate their probabilities.
  - **Many rules** will have been used **rarely** in the **treebank**.
  - Special probability **smoothing** techniques employed.
  - E.g., replacing the **words in brackets** by their **POS tags** (esp. if the tags also indicate gender, number, case etc.) or with **semantic classes** (e.g., person, location).
  - See J&M for more information.
- In **Conditional PCFGs (CPCFGs)**, **whenever a rule is applied**, it may have a **different probability**.
  - The **probability is generated by a model** (nowadays, possibly an MLP) that considers **features of the rules** and the **parts of the input text** its symbols correspond to.

Recommended reading

- Y. Goldberg, *Neural Network Models for Natural Language Processing*, Morgan & Claypool Publishers, 2017.
  - Mostly sections 7.7, 8.6, 16.2.3.

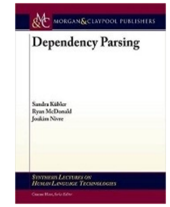


- Jurafsky & Martin (2nd ed.): chapters 12, 13, 14, 16.
  - Check also the 3rd edition (in preparation): http://web.stanford.edu/~jurafsky/slp3/ .



- For probabilistic parsing you may optionally want to consult chapters 11 and 12 of Manning & Schütze.

- For more background on dependency parsing, consult the book *Dependency Parsing* by S. Kubler, R. McDonald, and J. Nivre, Morgan & Claypool, 2009.



- The Universal Dependencies Project provides treebanks for many languages (including English, Greek).
  - http://universaldependencies.org/