

Natural Language Processing

Lecture 1: Course Overview and Introduction.

1/17/2017

NLP

COMS W4705
Daniel Bauer

The 4705 Team

- **Instructor:** Daniel Bauer <bauer@cs.columbia.edu>
Office Hours: Mon 3:30pm-5:00pm
704 Shapiro CEPSR

- **Assistants:**



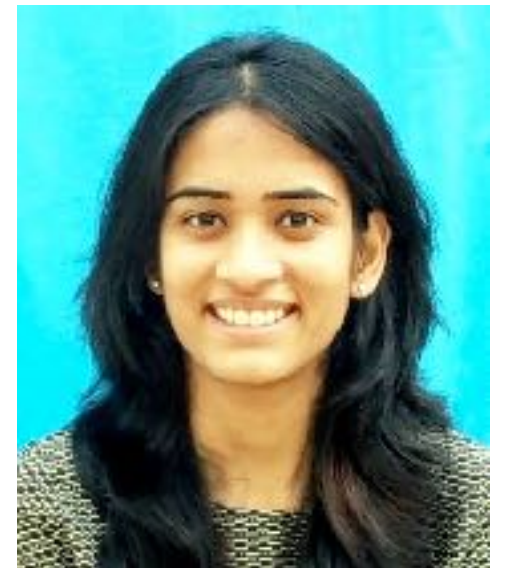
Bryan Li
<bl2557>



Sidhi Adkoli
<sa3505>



Mukund Raghuprasad
<my2541>



Surabhi Bhargava
<sb4019>

- **IA office hours / recitations start next week.
Time/Location TBA by email.**

Lectures & Recitation Sessions

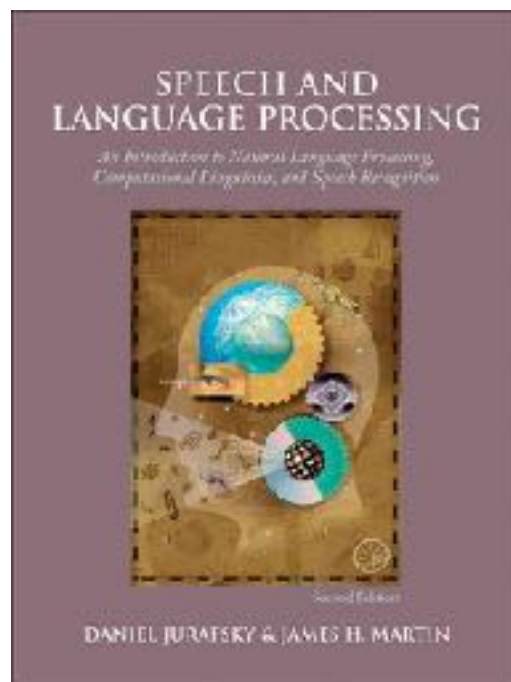
- **Lectures:** Mon & Wed 10:10am-11:24am
209 Havemeyer Hall
- **Recitation Sessions:**
 - Optional recitation sessions, led by the IAs
(schedule TBA)

Course Resources

- **Courseworks 2 (a.k.a Canvas):**
 - All course materials: Lecture notes, code, announcements, assignments, reading materials
 - Homework submission, grade book.
- **Piazza** used for Q & A.
Do not email the instructor or IAs with questions about the course content.

Textbook / Reading

- There is **NO official textbook** for this course.
- Recommended textbook (somewhat outdated, we won't stick to this too closely):



Dan Jurafsky & James Martin
Speech and Language Processing
2nd Ed. Prentice Hall (2009).

- Instead we will read a number of research papers.

Prerequisites

- Data Structures (COMS W3134 or COMS W3137)
- Discrete Math (COMS W3202, strongly recommended)
- Some previous or concurrent exposure to AI and machine learning is beneficial, but not required.
- Some experience with Python is helpful.

Grading

- Midterm 20%
- Final 30%
- 5 Homework assignments, each contains an analytical and a programming part, 10% each

Homework

- Homework uploaded through Courseworks. Do not email!
- Analytical part: Must be a plain txt or pdf documents (give LaTeX a shot).
- Programming part: We will use Python.
 - Testing code will be provided to make sure your programs are functionally correct (function signatures, data types...)
 - Passing these tests does not guarantee full score!

Homework Late Policy

- Written homework and programming problems may be submitted up to four days late for a 20 point penalty.
- No homework will be accepted more than four days after the deadline.
- Other extensions will only be granted in exceptional circumstances.

Academic Honesty

- Submit your own answers and code.
- Review academic honesty policy on the syllabus (Courseworks).
- When in doubt, ask.
- When in trouble, ask for help (and early).

NLP in the Movies



I am fluent in over
six million forms
of communication

Open the pod bay
doors HAL!



I'm sorry Dave, I'm
afraid I can't do
that!

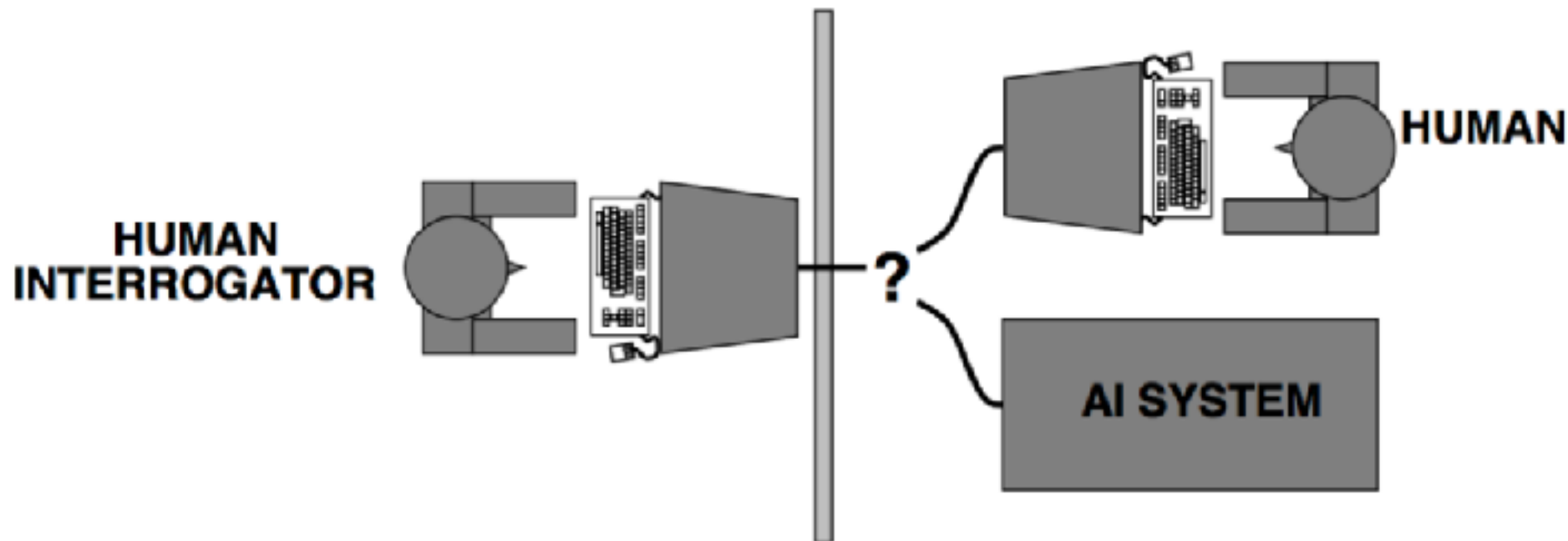
Natural Language Processing

- Important and active research area within AI.
- Timely: Most of our activities online are text based (web-pages, email, social media, blogs, news, product descriptions and reviews, medical reports, course content, ...)
- NLP leverages more and more available training data and modern Machine Learning techniques.
- Communicating with computers is the “holy grail” of AI.

Turing Test

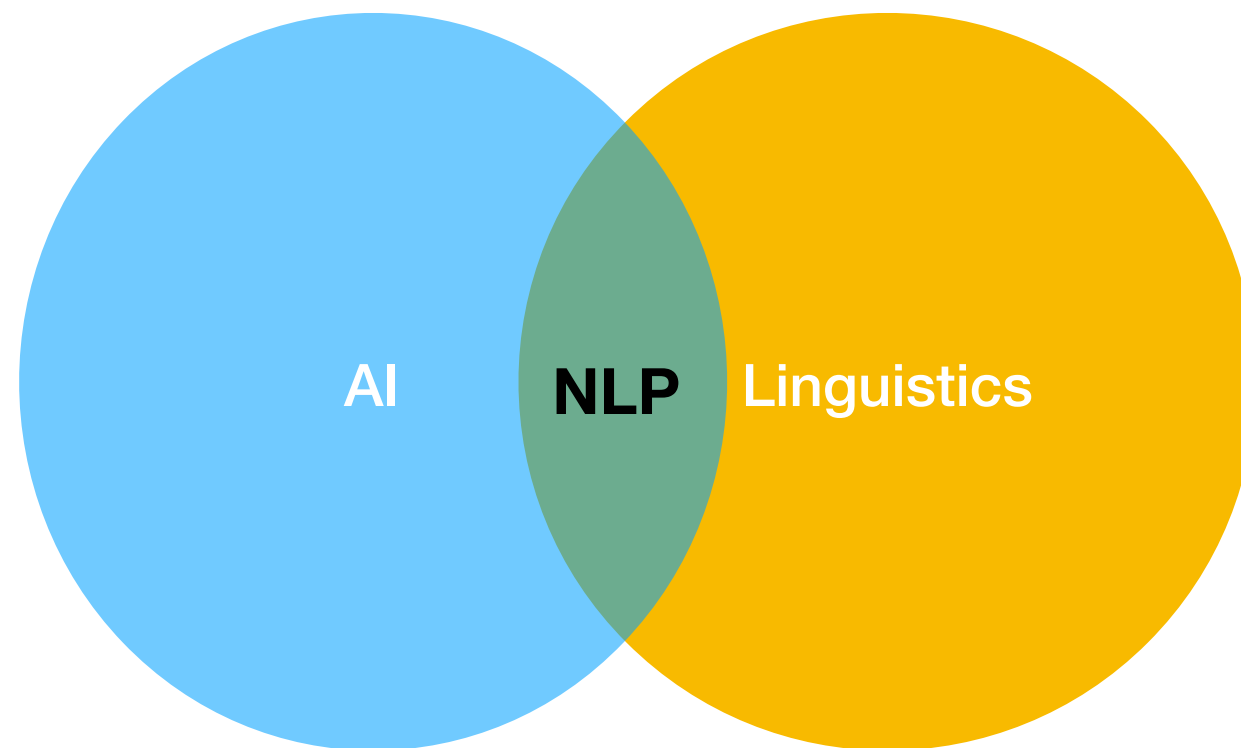
(Alan Turing, 1950)

- A computer passes the test of intelligence if it can fool a human interrogator into believing it is human.



- What skills are needed to build such a system?
 - **Language processing**, knowledge representation, reasoning, learning.

Natural Language Processing

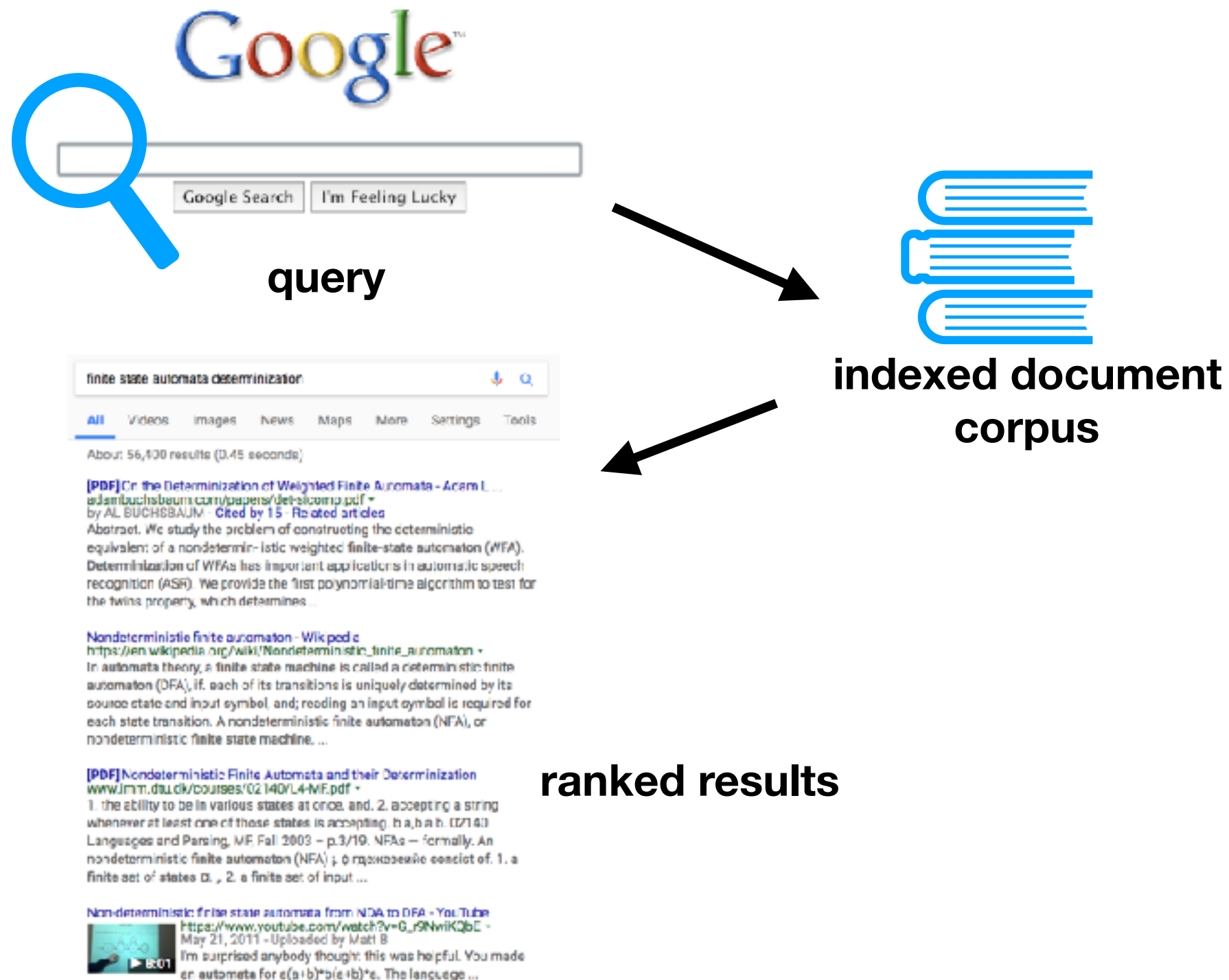


“Every time I fire a linguist, my performance goes up” (Fred Jelinek)

Natural Language Processing vs. Computational Linguistics

- **NLP:** Build systems that can understand and generate natural language. Focus on applications.
- **Computational Linguistics:** Study human language using computational approaches.
- Many overlapping techniques.

Applications: Information Retrieval



Applications: Text Classification

- Spam filtering.
- Detecting topics / genre.
- Sentiment analysis, author recognition, forensic linguistics, ...

Applications: Sentiment Analysis

Fantastic... truly a wonderful family movie



I have a mixed feeling about this movie.



Well it is fun for sure but definitely not appropriate for kids 10 and below



My kids loved it!!



The movie is very funny and entertaining. Big A+



I got so boooored...



Disappointed. They showed all fun details in the trailer



Cute but not for adults



Applications: News Summarization

Columbia Newsblaster

Summarizing all the news on the Web

Articles

Search for:

Offline summarization ▾

Go

U.S.
World
Finance
Sci/Tech
Entertainment
Sports

[View Today's Images](#)

[View Archive](#)

[About Newsblaster](#)

[About today's run](#)

[Newsblaster in Press](#)

[Academic Papers](#)

Article Sources:
[abcnews.go.com](#)
(71 articles)

Elon Musk unveils Dragon V2 reusable manned spacecraft

Summary from multiple countries, from articles in English
[UPDATED] (see summary with new information since yesterday)

In space there are currently two American astronauts on where the International Space Station living and working alongside three Russian cosmonauts tells more about the relationship. ([article 4](#)) A company that has flown unmanned capsules to the space station unveiled a spacecraft Thursday designed to ferry up to seven astronauts to low-Earth orbit that SpaceX CEO Elon Musk says will revolutionize access to space. ([article 3](#)) SpaceX unveiled its Dragon V2 spacecraft Thursday night, promising it will be able to carry seven astronauts to the International Space Station and back to Earth again, landing with the precision of a helicopter. ([article 5](#)) Lifting the vehicle's hatch, Musk settled into a reclined gold-and-black pilot's seat and pulled down a sleek, rounded glass control panel. ([article 2](#)) The cabin, designed to fly a crew of seven, looked more like a Star Trek movie set than the flight deck of NASA's now-retired space shuttle. ([article 2](#)) Dragon, which launches on a SpaceX Falcon 9 rocket, is one of three privately owned space taxis vying for NASA development funds and launch contracts. ([article 2](#)) The U.S. space agency turned over space station cargo runs and crew ferry flights after retiring its fleet of shuttles in 2011 and SpaceX already has a 1.6 billion contract for 12 station resupply missions ([article 2](#))

Other summaries about this story:

- [Summary from United States, from articles in English](#) (4 articles) [[compare](#)]
- [Summary from Canada, from articles in English](#) (1 articles) [[compare](#)]

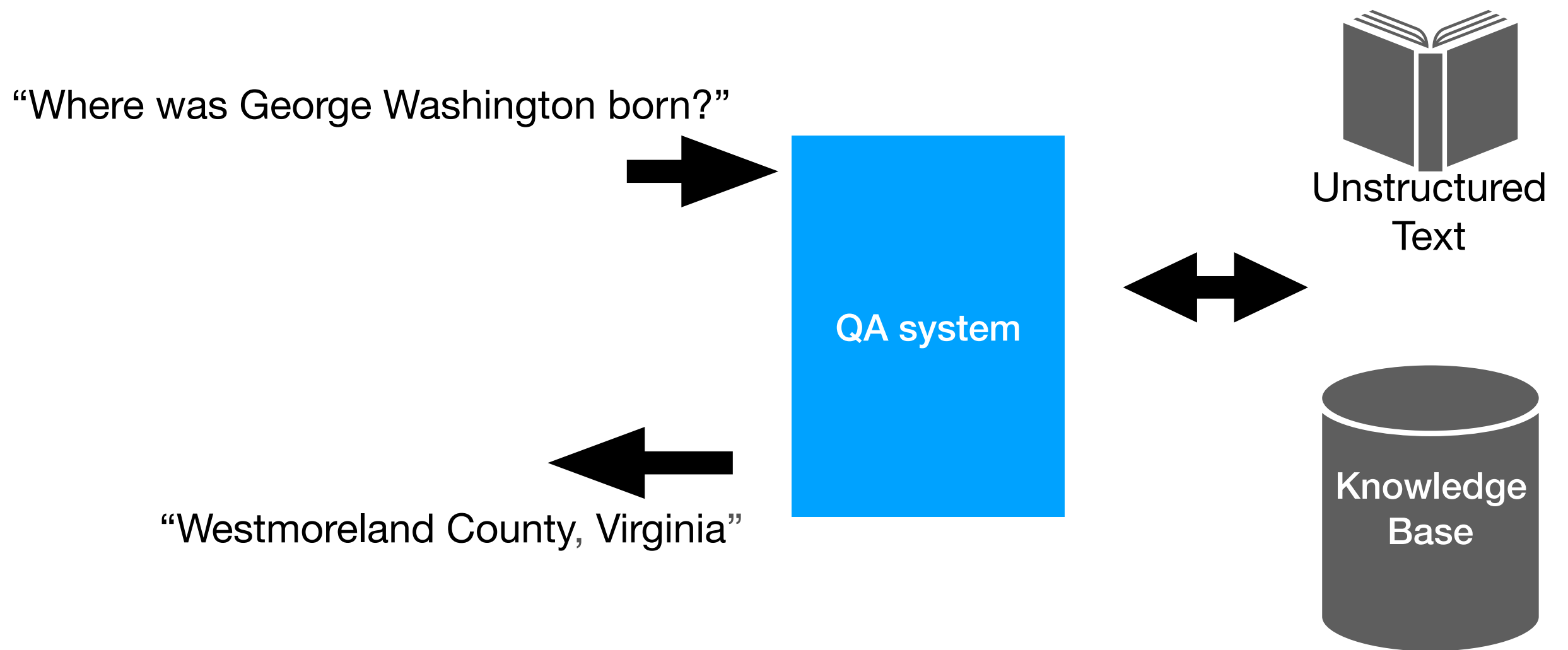
Event tracking:

- [Track this story's development in time](#)

Story keywords

Space, spacecraft, astronauts, Musk, SpaceX

Application: Question Answering



Applications: Playing Jeopardy!

IBM Watson [2011]



*William Wilkinson's "An Account of the Principalities of Wallachia and Moldavia" inspired **this author's** most famous novel.*

Who is Stoker?
(FOR ONE WELCOME OUR
NEW COMPUTER OVERLORDS)

Who is Bram
Stoker?
\$ 17,973

Combines information extraction & natural language understanding.

Applications: Machine Translation



Google

Translate From: English To: Spanish

Encontrar su camino en el centro comercial o en el aeropuerto, con un mapa Celular



Laura Pedrick para The New York Times

FastMall ofrece un plan de piso y puede buscar tiendas y trazar un camino allí. Agitar el teléfono mostrará el baño más cercano.

Por VERNE G. Kopytoff
Publicado: 10 de octubre 2010

SAN FRANCISCO - Mapas de telefonía móvil han guiado la gente por las calles y callejones de todo el mundo. Pero cuando esas personas dentro de un edificio en expansión, se pueden perder.

ENTRA PARA E-MAIL
IMPRESIÓN

Suscríbete a la tecnología

- Noticias de Tecnología
 - Internet
 - Start-Ups
 - Computing
 - Empresas
 - Negocios

MÁS POPULARES - TECNOLOGÍA

ENVÍE UN CORREO ELECTRÓNICO

1. Bits: Malcolm Gladwell 'So en un elemento de negocia
2. Gadgetwise: Breaking Up cables
3. Bits: cuál es el trato Beats Tastemakers
4. Estado del Arte: La Revolución Pleasurable
5. Bits: ¿Por qué es Amazon lo que realmente necesita
6. Bits: Las Fallas de iMessage
7. Bits: Google toma medidas olvido' Fallo
8. Bits: Cuantificación de ubicación de datos puede s
9. Q & A: Mantener en Wind
10. Bits: Dots, un juego altame sucesor

Machine Translation

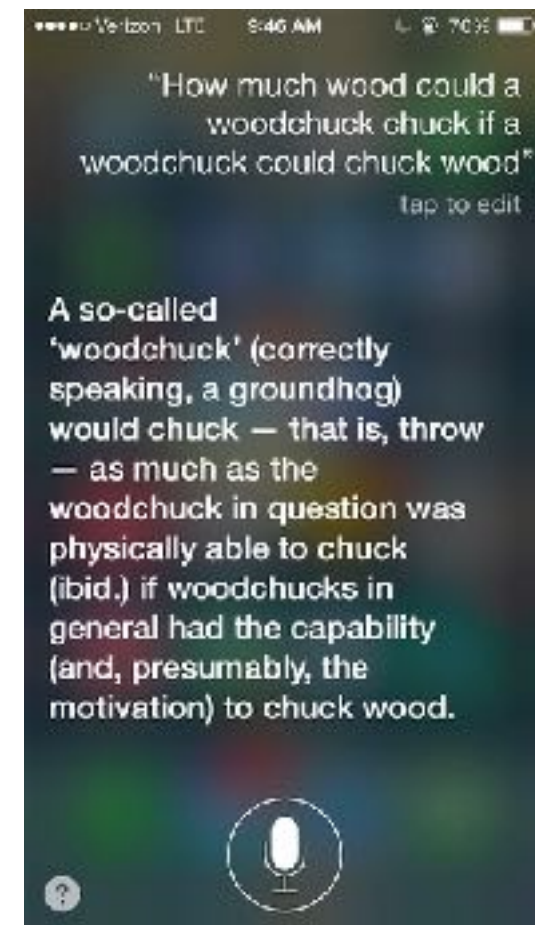
- One of the main research areas in NLP, and one of the oldest. Historical motivation: Translate Russian to English.
- MT is really difficult:
 - “Out of sight, out of mind” → “Invisible, imbecile”
 - “The spirit is willing, but the flesh is weak”
English → Russian → English
“The vodka is good, but the meat is rotten”
- Challenges: Word order, multiple translations for a word (need context), want to preserve meaning.

Machine Translation

- Until recently phrase-based translation was the predominant framework.
- Today neural network sequence-to-sequence models are used.
- Google Translate supports > 100 languages.

Applications: Virtual Assistants

- Siri (Apple), Google Now, Cortana (Microsoft), Alexa (Amazon).
- Subtasks: Speech recognition, language understanding (in context?), speech generation, ...

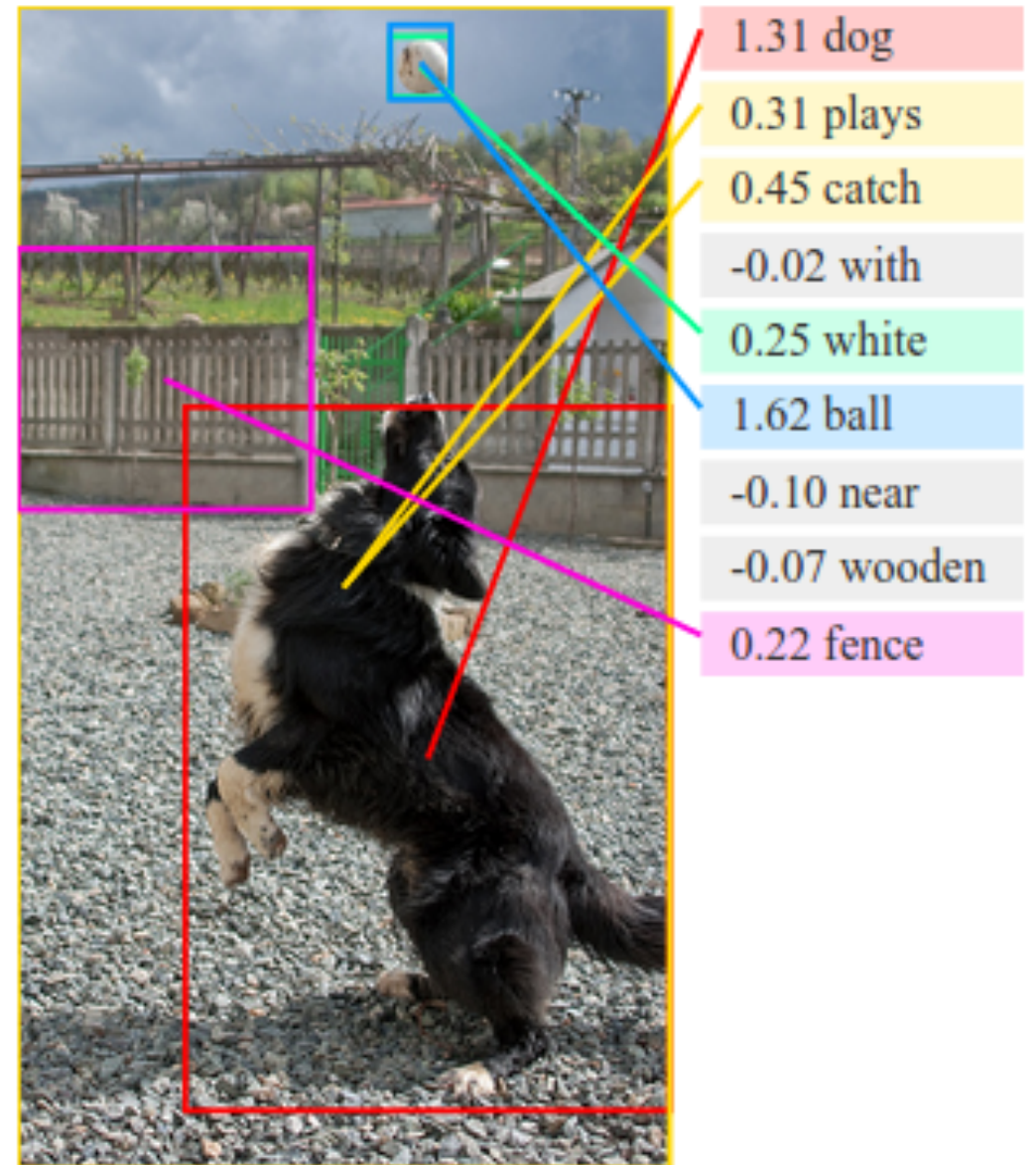


Applications: Image Captioning



Man in black t-shirt is playing guitar.

- Uses both recurrent neural nets and CNNs.
- “Multi-modal” embeddings.
- Microsoft COCO data set.



What You Will Learn In This Course

- How can machines **understand** and **generate** natural language?
 - Theories about language (linguistics).
 - Algorithms.
 - Statistical / Machine Learning Methods.
 - Applications.

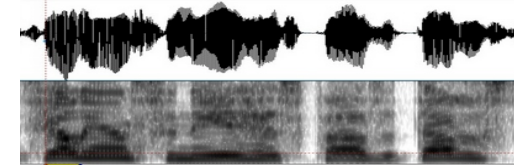
Course Overview

- Part I: Core NLP techniques.
 - Language modeling, part-of-speech tagging, syntactic parsing, word-sense disambiguation, semantic parsing, text similarity.
- Part II: Applications.
 - Information retrieval, question answering, text generation, summarization, machine translation, image captioning, dialog systems.

Levels of Linguistic Representation

phonetics
phonology

sounds and sound
patterns of language



/boɪ/

morphology

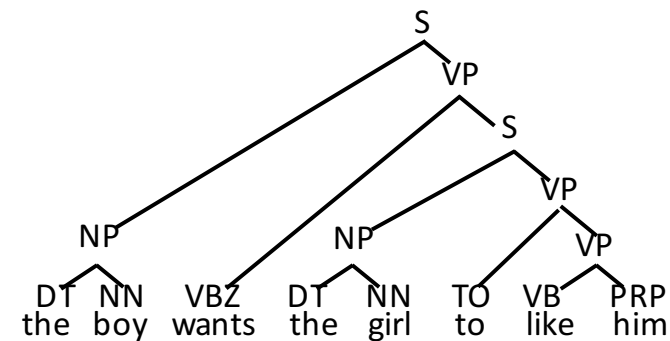
formation of words

in- + validate + -ed

DT	NN	VBZ	DT	NN	TO	VB	PRP	.
the	boy	want+s	the	girl	to	like	him	.

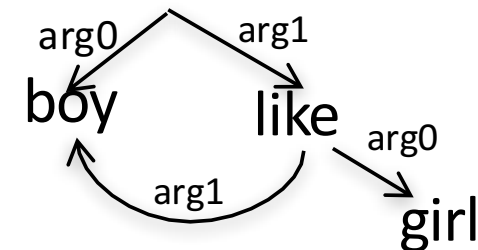
syntax

word order



semantics

word and sentence
meaning



pragmatics

influence of context and
situation

Natural Language Processing as Translation

- Most NLP techniques can be understood as translation tasks from one structure into another.
- For each translation step:
 - Construct search space of possible translations.
 - Find best paths through this space (decoding) according to some performance measure.
- Modern NLP relies on Machine Learning to figure out these translation steps.

NLP is hard: Ambiguity

- Unlike artificial languages, natural language is full of ambiguity.
- This can happen **on all levels of representation**.
 - *“Wreck a Nice Beach”, “Recognize Speech”*
 - *“flammable” = in + -flammable*
 - *“Enraged cow injures farmer with ax”*
 - *“Stolen Painting Found by Tree”*
 - *“Red Tape Holds Up New Bridges”*
 - *“Mouse”*



More Real Headlines

- *Ban on nude dancing on Governor's desk*
- *Eye drops off shelf*
- *Kids Make Nutritious Snacks*
- *Drunk gets nine months in violin case*
- *Government head seeks arms*
- *Patient at death's door – doctors pull him through*
- *In America a woman has a baby every 15 minutes*

Syntactic Structure

- What is the **part-of-speech** of each word? (noun, verb, adjective, adverb, determiner, ...)
- What are the **constituents**:
 - Noun phrase: *“Enraged cow”, “The cat with the hat”, “Columbia University”*
- What are the **subjects and objects**:
 - *“Dog bites man”* vs. *“Man bites dog”*
- **Modification**:
 - *“John saw the man in the park with a telescope”*

Structural Ambiguity

- Interplay between constituent structure and modification.
- Prepositional Phrase (PP) attachment:

Enraged cow injures farmer with axe.

[Enraged cow] injures [farmer with axe]
NP NP



[Enraged cow] injures farmer [with axe]
NP NP PP



Representing Modification with Brackets

[Enraged cow] [injures [farmer [with axe]]]
NP NP PP

[Enraged cow] injures [farmer] [with axe]
NP NP PP

More PP attachment

Ban on nude dancing on governor's desk

More PP attachment

[Ban] on [nude dancing] on [governor's desk]

NP

NP

NP

- What are the possible modifications? Which one is correct?

More PP attachment

[Ban] on [nude dancing] on [governor's desk]

NP

NP

NP

- What are the possible modifications? Which one is correct?

[[Ban] on [nude dancing]] [on governor's desk]

NP

PP

[Ban] on [[nude dancing] [on governor's desk]]

NP

PP

NP

Noun-Noun Modification

- Compound nouns also have internal structure:

country song platinum album

1. *[country [song [platinum album]]]*

2. *[country [[song platinum] album]]*

3. *[[country song] [platinum album]]*

4. *[[country [song platinum]] album]*

5. *[[[country song] platinum] album]*

Noun-Noun Modification

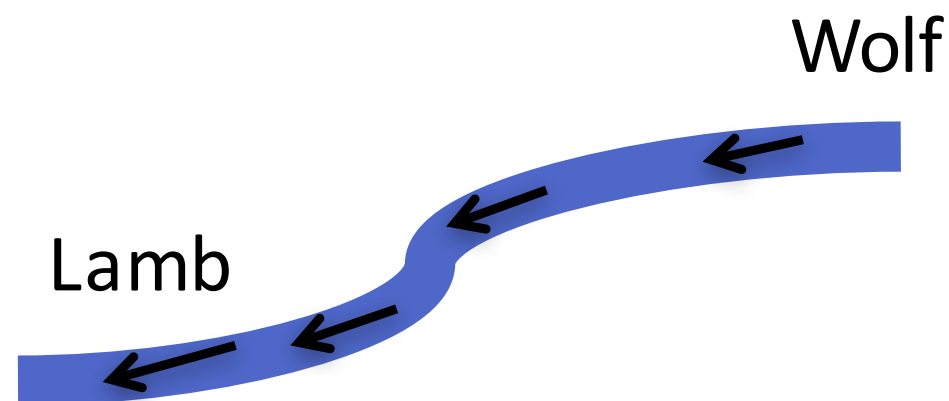
- What is the *semantic* relationship between nouns in a noun compound?
 - *Water fountain:* A fountain that **supplies** water.
 - *Water ballet:* A ballet that **takes place** in water.
 - *Water meter:* A device that **measures** water.
 - *Water barometer:* A barometer that **uses** water (instead of mercury) to measure air pressure.
 - *Water glass:* A glass that is meant to **hold** water.

Other tricky phenomena


- Need for semantic representation.

There was once a Wolf who saw a Lamb drinking at a river and wanted an excuse to eat it.

*For that purpose, **even though** he himself was **upstream**, he accused the Lamb of stirring up the water and keeping him from drinking. . .*

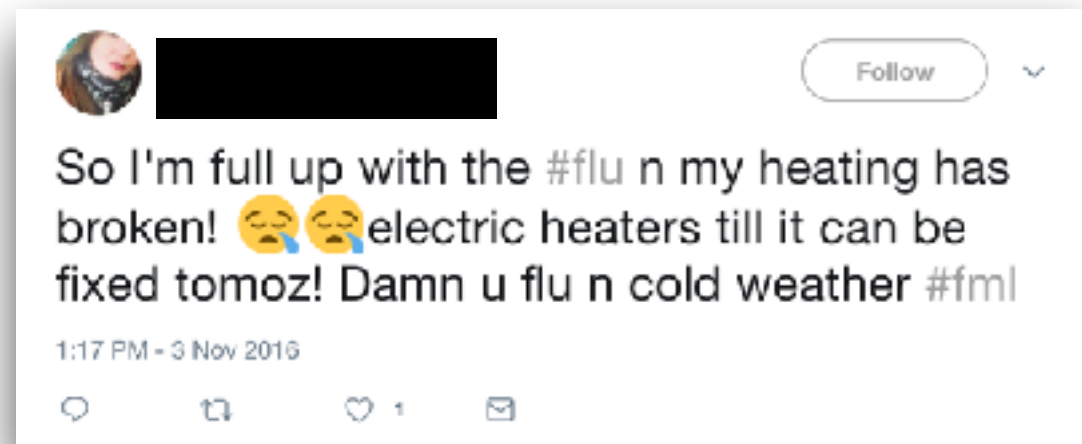


Other tricky issues: Language Variety

- Problem: Most NLP techniques were developed on English (specifically financial news written in American English in the 1980s), or other languages with many resources.
 - Languages use different mechanisms to express meaning (morphology vs. word-order).
- 



Other tricky issues: Domains and Language Change



- Non-standard English
- Idioms: *throw in the towel*, *get cold feet*, *kick the bucket*
- Neologisms (fixed lexicon doesn't work)
 - *noob*, *crowdsource*, *unfriend*, *retweet*, *bromance*, ...

Morphology

- Structure and formation of words.
- **Derivational** morphology: Create new words from old words (can also change the part-of-speech).

anti- + dis- + **establish** + -ment + -arian + -ism
- **Inflectional** morphology:
 - Convey information about number, person, tense, aspect, mood, voice, and the role a word plays in the sentence (case).
 - English has few morphological categories, but many languages are morphologically rich.

Morphology

- Morphological categories in English
 - Number (“*dog*”, “*dog +s*”)
 - Person (“*I run*”, “*She runs*”)
 - Tense (“*He waited*”)
 - Voice (“*The issue was decided*”)
- Other examples from other languages?

Acknowledgments

- Some slides and examples from:
 - Kathy McKeown, Dan Jurafsky, Dragomir Radev

Natural Language Processing

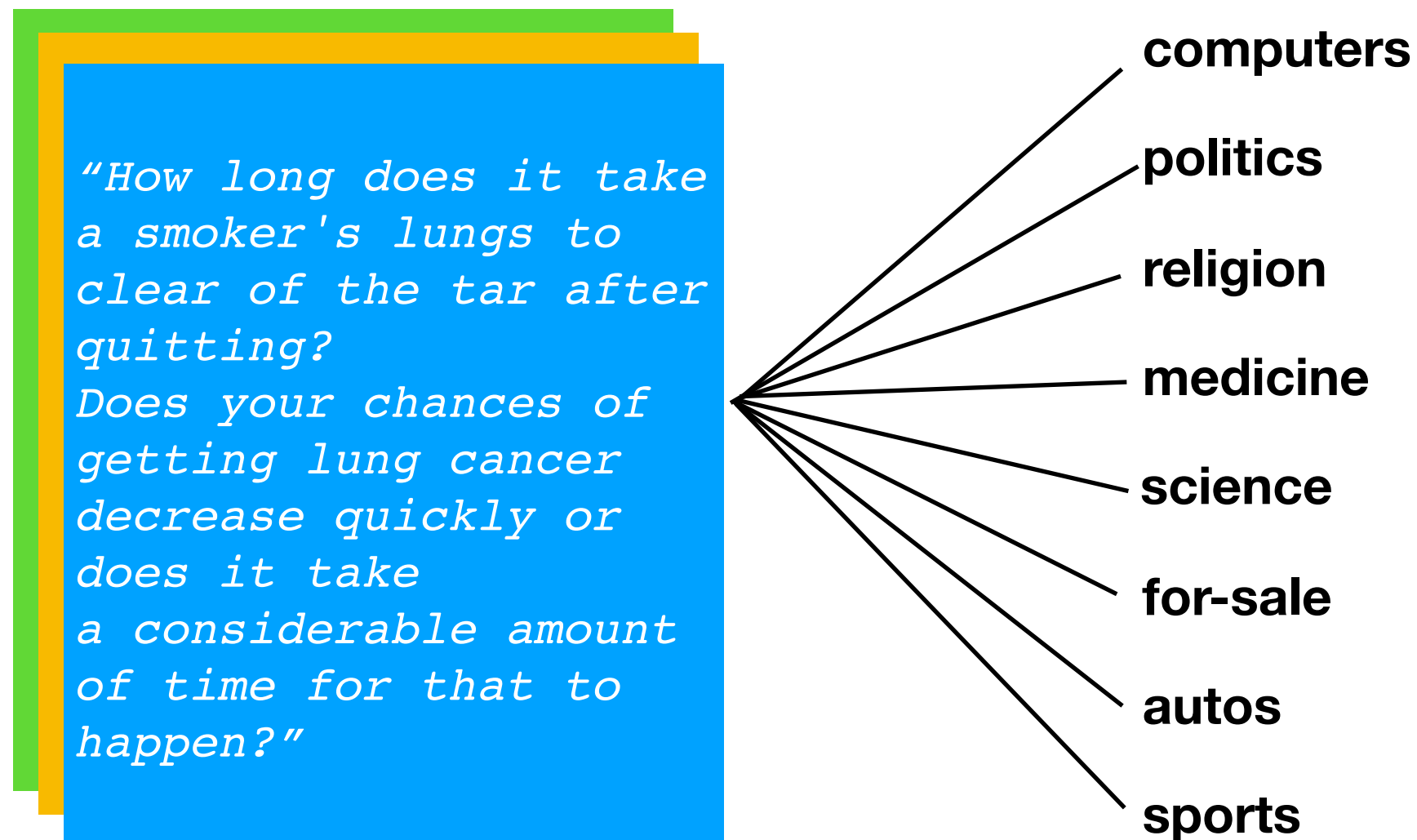
Lecture 2: Language Classification. Probability Review.
Machine Learning Background. Naive Bayes' Classifier.

1/19/2017

COMS W4705
Daniel Bauer

Text Classification

- Given a representation of some document d , identify which class $c \in C$ the document belongs to.



From the 20-Newsgroups data set:

<http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html>

Text Classification

- Applications:
 - Spam detection.
 - Mood / Sentiment detection.
 - Automatic category assignment.
 - Author identification.
 - Identifying political affiliation.
 - Word Sense Disambiguation.
 - ...

Text Classification

- This is a machine learning problem.
- How do we represent each document?
(feature representation).
- Can use different ML techniques.
 - **Supervised ML:** Fixed set of classes C .
Train a classifier from a set of labeled <document,class> pairs.
 - Discriminative vs. Generative models.
 - Unsupervised ML: Unknown set of classes C .
Topic modeling.

Types of Feedback

- **Supervised learning:** Given a set of input-output pairs, learn a function that maps inputs to outputs.
- **Unsupervised learning:** Learn patterns in the input without any explicit feedback.
One typical approach: clustering, identify clusters of input examples.
- **Semi-supervised learning:** Start with a few labeled input/output pairs, then use a lot of unlabeled data to improve.
- **Reinforcement learning:** Start with a *policy* determining the agent's actions. Feedback in the form of reward or punishment.

Supervised Learning

- Given: Training data consisting of training examples $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$, where \mathbf{x}_i is an input example (a d -dimensional vector of attribute values) and y_i is the label.

example					label
1	x_{11}	x_{12}	...	x_{1d}	y_1
...
i	x_{i1}	x_{i2}	...	x_{id}	y_i
...
n	x_{n1}	x_{n2}	...	x_{nd}	y_n

- Goal: learn a hypothesis function $h(x)$ that approximates the true relationship between x and y . This functions should
 - 1) ideally be consistent with the training data.
 - 2) generalize to unseen examples.
- In NLP y_i typically form a finite, discrete set.

What Americans Have Heard or Read About Hillary Clinton

What specifically do you recall reading, hearing or seeing about Hillary Clinton in the last day or two?



GALLUP DAILY TRACKING
JULY 17-SEPT 18, 2016

What Americans Have Heard or Read About Donald Trump

What specifically do you recall reading, hearing or seeing about Donald Trump in the last day or two?



GALLUP DAILY TRACKING
JULY 17-SEPT 18, 2016

Representing Documents

to be, or not to be

- Set-of-words representation.
- Bag-of-words representation (Multi-set).
- Vector-space model: Each word corresponds to one dimension in vector space. Entries are either:
 - Binary (Word appears / does not appear)
 - Raw or normalized frequency counts.
 - Weighted frequency counts
 - Probabilities.

*to or
be not*

*be to or not
be to*

be	2
⋮	⋮
not	1
⋮	⋮
or	1
⋮	⋮
to	2

What is a Word?

- e.g., are “*Cat*”, “*cat*” and “*cats*” the same word?
- “*September*” and “*Sept*”?
- “*zero*” and “*oh*”?
- Is “*_*” a word? “*.*”? “***”? “*(*”?
- How many words are there in “*don’t*” ? “*Gonna*” ? “*I.B.M.*”?
- In Japanese and Chinese text -- how do we identify a word?
- ...

Text Normalization

- Every NLP task needs to do some text normalization.
 - Segmenting / tokenizing words in running text.
 - Normalizing word forms (lemmatization or stemming, possibly replacing named-entities).
 - Sentence splitting.

Linguistic Terminology

- **Sentence:** Unit of written language.
- **Utterance:** Unit of spoken language.
- Word **Form:** the inflected form as it actually appears in the corpus. *“produced”*
- Word **Stem:** The part of the word that never changes between morphological variations. *“produc”*
- **Lemma:** an abstract base form, shared by word forms, having the same **stem**, part of speech, and word sense – stands for the **class** of words with **stem**.
“produce”
- **Type:** number of distinct words in a corpus (vocabulary size).
- **Token:** Total number of word occurrences.

Tokenization

- Tokenization: The process of segmenting text (a sequence of characters) into a sequence of tokens (words).

"Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing."

mr.	o'neill	thinks	that	the	boys'	stories	about	Chile's	capital	are	n't
amusing	.										

- Simple (but weak) approach: Separate off punctuation. Then split on whitespaces.
- Typical implementations use regular expressions (finite state automata).

Tokenization Issues

- Dealing with punctuation (some may be part of a word)
“Ph.D.”, “O’Reilly”, “pick-me-up”
- Which tokens to include (punctuation might be useful for parsing, but not for text classification)?
- Language dependent: Some languages don’t separate words with whitespaces.

de: “*Lebensversicherungsgesellschaftsangestellter*”

zh: 日文章鱼怎么说? - *Japanese Octopus* how say?

日文章鱼怎么说? - *Sun article fish* how say?

Lemmatization

- Converting Lemmas into their base form.

“Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.”

mr.	o'neill	think	that	the	boy	story	about	chile	's	capital	are	n't
amusing		.										

PER	PER	think	that	the	boy	story	about	LOC	's	capital	are	n't
amusing		.										

Probabilities in NLP

- Ambiguity is everywhere in NLP. There is often *uncertainty* about the “correct” interpretation. Which is more likely:
 - Speech recognition: “*recognize speech*” vs. “*wreck a nice beach*”
 - Machine translation: “*l’avocat general*”: “*the attorney general*” vs. “*the general avocado*”
 - Text classification: is a document that contains the word “*rice*” more likely to be about politics or about agriculture?
What if it also includes several occurrences of the word “*stir*”?
- Probabilities make it possible to combine evidence from multiple sources systematically to (using Bayesian statistics)

Bayesian Statistics

- Typically, we observe some evidence (for example, words in a document) and the goal is to infer the “correct” interpretation (for example, the topic of a text).
- Probabilities express the degree of belief we have in the possible interpretations.
- **Prior probabilities:** Probability of an interpretation prior to seeing any evidence.
- **Conditional (Posterior) probability:** Probability of an interpretation after taking evidence into account.

Probability Basics

- Begin with a **sample space** Ω
 - Each $\omega \in \Omega$ is a possible basic outcome / “possible worlds” (e.g. the 6 possible rolls of a die).
- A **probability distribution** assigns a probability to each basic outcome.

$$\sum P(\omega) \leq 1.0 \text{ for every } \omega \in \Omega$$

$$\sum_{\omega \in \Omega} P(\omega) = 1.0$$

- E.g: six-sided die

$$P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1.0$$

Events

- An *event* A is any subset of Ω .

$$P(A) = \sum_{\omega \in A} P(\omega)$$

- Example:

$$P(\text{die roll} < 4) = P(1) + P(2) + P(3) = 1/6 + 1/6 + 1/6 = 1/2$$

Random Variables

- A random variable is a function from basic outcomes to some range, e.g. real numbers or booleans.

$$\textit{Odd}(1) = \textit{true}$$

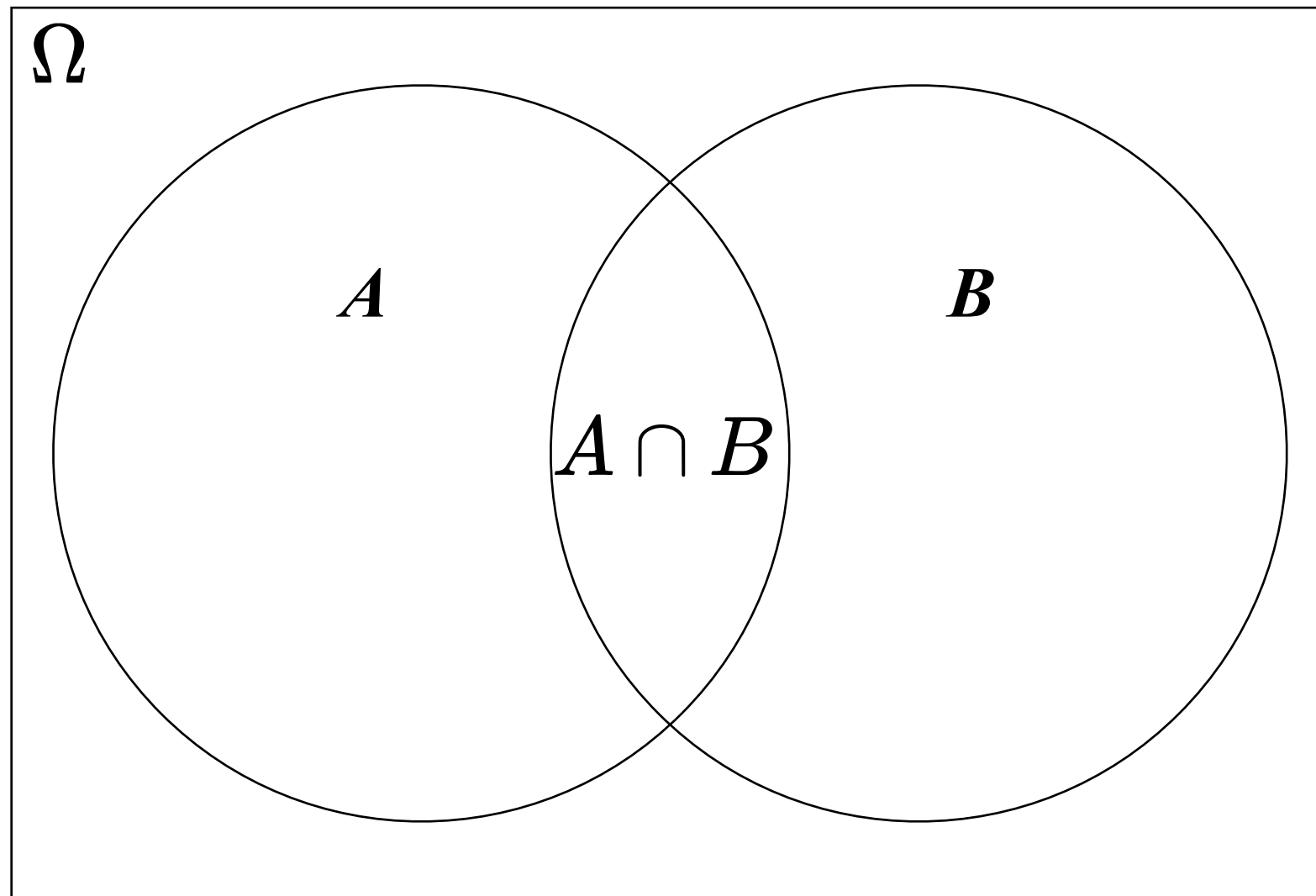
- A distribution P induces a probability distribution for any random variable.

$$P(X = x_i) = \sum_{\{\omega: X(\omega)=x_i\}} P(\omega)$$

- E.g $P(\textit{Odd} = \textit{true}) = P(1) + P(3) + P(5) = 1/2$

Joint and Conditional Probability

Joint probability: $P(A \cap B)$ also written as $P(A, B)$



Conditional probability: $P(A|B) = \frac{P(A, B)}{P(B)}$

Conditional Probability

- Product rule: $P(A, B) = P(B) \cdot P(A|B) = P(A) \cdot P(B|A)$
- Chain rule (generalization of product rule):

$$P(A_n, \dots, A_1) = P(A_n | A_{n-1}, \dots, A_1) \cdot P(A_{n-1}, \dots, A_1)$$

- **Bayes' Rule:**

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Independence

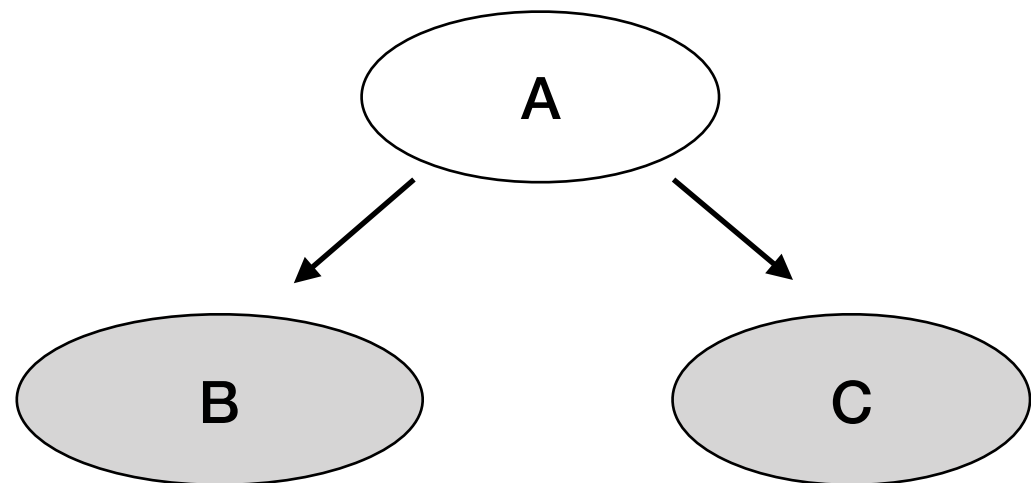
- Two events are independent if $P(A) = P(A|B)$
or equivalently $P(A, B) = P(A) \cdot P(B)$ (if $P(B) > 0$)

- Two events are **conditionally independent** if:

$$P(B, C|A) = P(B|A)P(C|A)$$

or equivalently

$$P(B|A, C) = P(B|A) \text{ and } P(C|A, B) = P(C|A)$$



Probabilities and Supervised Learning

- Given: Training data consisting of training examples
data = $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$,
Goal: Learn a mapping h from x to y .
- We would like to learn this mapping using $P(y|x)$.
- Two approaches:
 - Discriminative algorithms learn $P(y|x)$ directly.
 - Generative algorithms use Bayes rule

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}$$

Discriminative Algorithms

- Model conditional distribution of the label given the data $P(y|x)$
- Learns decision boundaries that separate instances of the different classes.
- To predict a new example, check on which side of the decision boundary it falls.
- Examples:
log-linear model, support vector machine (SVM), decision trees, random forests, neural networks.

Generative Algorithms

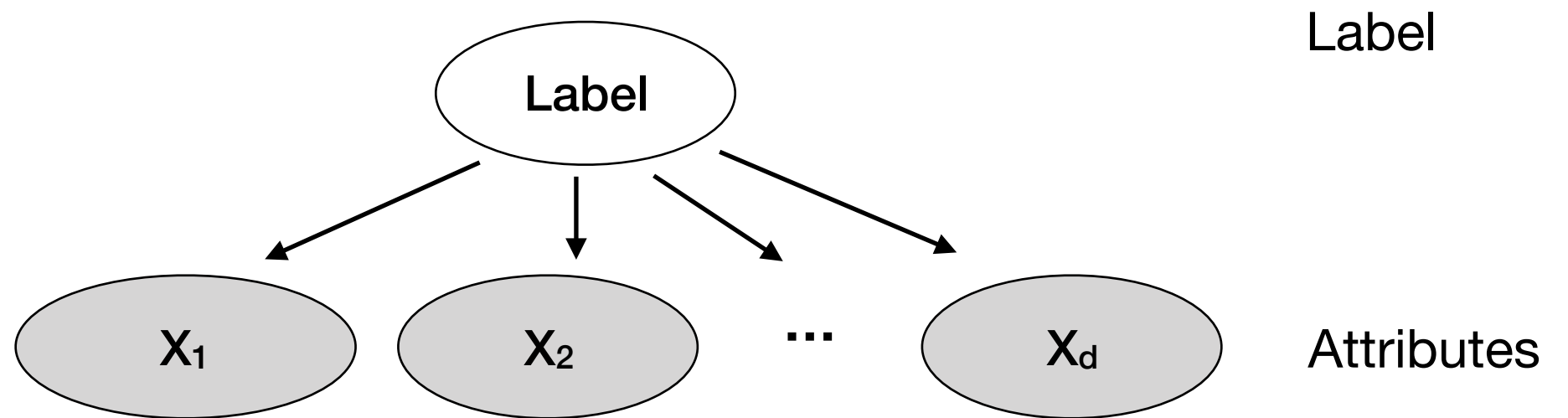
- Assume the observed data is being “generated” by a “hidden” class label.
- Build a different model for each class.
- To predict a new example, check it under each of the models and see which one matches best.

- Model $P(x|y)$ and $P(y)$. Then use bases rule

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}$$

- Examples:
Naive Bayes, Hidden Markov Models, Gaussian Mixture Models, PCFGs, ...

Naive Bayes

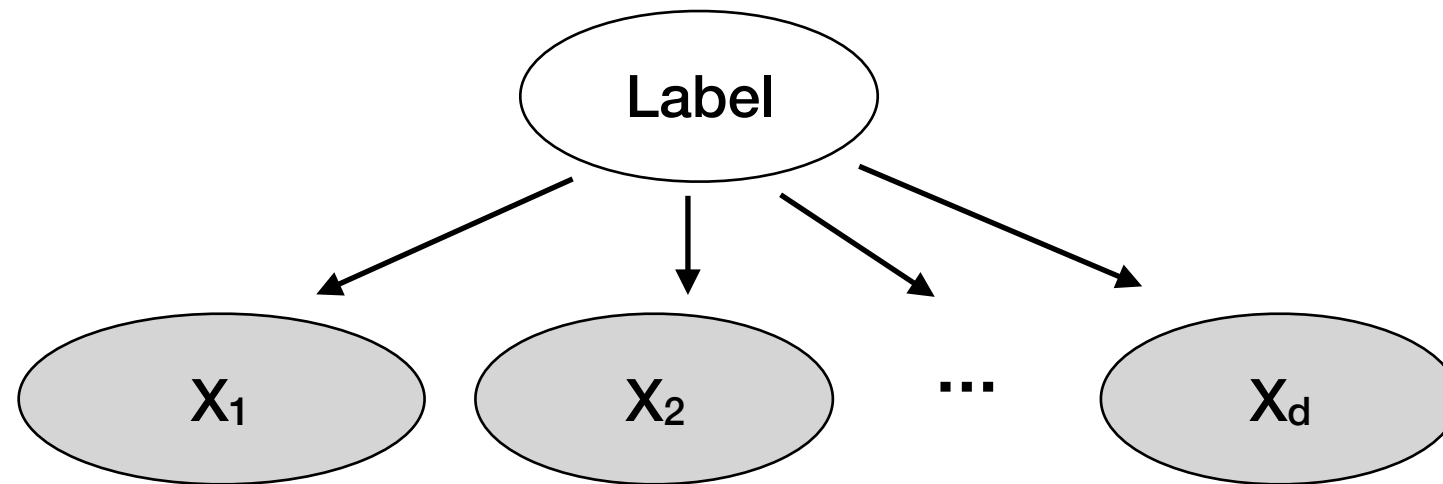


$$\mathbf{P}(Label, X_1, \dots, X_d) = \mathbf{P}(Label) \prod_i P(X_i | Label)$$

$$\mathbf{P}(Label | X_1, \dots, X_d) = \frac{\mathbf{P}(Label) \prod_i P(X_i | Label)}{\prod_i P(X_i)}$$

$$= \alpha [\mathbf{P}(Label) \prod_i P(X_i | Label)]$$

Naive Bayes Classifier



$$\mathbf{P}(Label|X_1, \dots, X_d) = \alpha[\mathbf{P}(Label) \prod_i P(X_i|Label)]$$

$$y^* = \arg \max_y P(y) \prod_i P(x_i|y)$$

Note that the normalizer α does no longer matter for the argmax because α is independent of the class label.

Training the Naive Bayes' Classifier

- Goal: Use the training data to estimate $P(\text{Label})$ and $P(X_i|\text{Label})$ from training data.
- Estimate the prior and posterior probabilities using **Maximum Likelihood Estimates (MLE)**:

$$P(y) = \frac{\text{Count}(y)}{\sum_{y' \in Y} \text{Count}(y')}$$

$$P(x_i|y) = \frac{\text{Count}(x_i, y)}{\sum_{x'} \text{Count}(x', y)} = \frac{\text{Count}(x_i, y)}{\text{Count}(y)}$$

- I.e. we just count how often each token in the document appears together with each class label.

Why the Independence Assumption Matters

- Without the independence assumption we would have to estimate $\mathbf{P}(X_1, \dots, X_d | Label)$
- There would be many combinations of x_1, \dots, x_d that are never seen (sparse data).
- The independence assumption allows us to estimate each $\mathbf{P}(X_1 | label)$ independently.

Is this a safe assumption for documents?
Are the words really independent of each other?

Training the Naive Bayes' Classifier

- Ways to improve this model?
- Some issues to consider...
 - What if there are words that do not appear in the training set? What if it appears only once?
 - What if the plural of a word never appears in the training set?
 - How are extremely common words (e.g., “the”, “a”) handled?

Probability of a Sentence

- What is the probability that the Naive Bayes' model actually computes?
- On Wednesday, we will explore different approaches to computing sentence probability.

“But it must be recognized that the notion of ‘probability of a sentence’ is an entirely useless one, under any known interpretation of this term.”

Noam Chomsky (1969)

Natural Language Processing

Lecture 4: n-gram language models

01/29/2018

COMS W4705
Daniel Bauer

Probability of a Sentence

- What is the probability that the Naive Bayes' model actually computes?

“But it must be recognized that the notion of ‘probability of a sentence’ is an entirely useless one, under any known interpretation of this term.”

Noam Chomsky (1969)

Language Modeling

- Task: predict the next word given the context.
- Used in speech recognition, handwritten character recognition, spelling correction, text entry UI, machine translation,...

Language Modeling

- Stocks plunged this ...
- Let's meet in Times ...
- I took the subway to ...

From a NYT story

- *Stocks plunged this*
- *Stocks plunged this morning, despite a cut interest rates by the ...*
- *Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall ...*
- *Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began*

Human Word Prediction

- Clearly at least some of us have the ability to predict the future.
- How does this work?
 - Domain knowledge
 - Syntactic knowledge (guess correct part of speech)
 - Lexical knowledge

Probability of the Next Word

- Idea: We do not need to model domain, syntactic, and lexical knowledge perfectly.
- Instead, we can rely on the notion of **probability of a sequence** (letters, words...).

$$P(w_n | w_1, w_2, w_3, \dots, w_{n-1})$$

Applications

- Speech recognition: $P(\textit{“recognize speech”}) > P(\textit{“wreck a nice beach”})$
- Text generation: $P(\textit{“three houses”}) > P(\textit{“three house”})$
- Spelling correction $P(\textit{“my cat eats fish”}) > P(\textit{“my xat eats fish”})$
- Machine Translation $P(\textit{“the blue house”}) > P(\textit{“the house blue”})$
- Other uses
 - OCR
 - Summarization
 - Document classification
 - Essay scoring

Language Models

- This model can also be used to describe the probability of an entire sentence, not just the last word.
- Use the chain rule:

$$P(w_1, \dots, w_n) =$$

$$P(w_1 | w_2, w_3, \dots, w_n) P(w_2, w_3, \dots, w_n) =$$

$$P(w_1 | w_2, w_3, \dots, w_n) P(w_2 | w_3, \dots, w_n) P(w_3, \dots, w_n) =$$

...

Markov Assumption

- $P(w_n | w_1, w_2, w_3, \dots, w_{n-1})$ is difficult to estimate.
- The longer the sequence becomes, the less likely $w_1 w_2 w_3 \dots w_{n-1}$ will appear in training data.
- Instead, we make the following simple independence assumption (Markov assumption):
 - The probability to see w_n depends only on the previous $k-1$ words.

$$\begin{aligned} P(w_n | w_1, w_2, w_3, \dots, w_{n-1}) \\ \approx P(w_n | w_{n-k+1}, \dots, w_{n-1}) \end{aligned}$$

bi-gram language model

- Using the Markov assumption and the chain rule:

$$P(w_1, \dots, w_n) \approx$$

$$P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdots P(w_n|w_{n-1})$$

- More consistent to use only bigrams:

$$P(w_1|start) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdots P(w_n|w_{n-1})$$

n-grams

- The sequence w_n is a unigram.
- The sequence w_{n-1}, w_n is a bigram.
- The sequence w_{n-2}, w_{n-1}, w_n is a trigram....
- The sequence w_{n-2}, w_{n-1}, w_n is a quadrigram...

Variable-Length Language Models

- We typically don't know what the length of the sentence is.
- Instead, we use a special marker STOP that indicates the end of a sentence.
- We typically just augment the sentence with START and STOP markers to provide the appropriate context.

START i want to eat Chinese food END

$P(i|START) \cdot P(want|i) \cdot P(to|want) \cdot P(eat|to) \cdot P(Chinese|eat) \cdot P(food|Chinese) \cdot P(END|food)$

trigram example

$$P(i|START, START) \cdot P(want|START, i) \cdot P(to|i, want) \cdot P(eat|want, to) \cdot \\ P(Chinese|to, eat) \cdot P(food|eat, Chinese) \cdot P(END|Chinese, food)$$

Bigram example from the Berkeley Restaurant Project (BeRP)

Eat on	0.16	Eat Thai	0.03
Eat some	0.06	Eat breakfast	0.03
Eat lunch	0.06	Eat in	0.02
Eat dinner	0.05	Eat Chinese	0.02
Eat at	0.04	Eat Mexican	0.02
Eat a	0.04	Eat tomorrow	0.01
Eat Indian	0.04	Eat dessert	0.007
Eat today	0.03	Eat British	0.001

Bigram example from the Berkeley Restaurant Project (BeRP)

START I	0.25	Want some	0.04
START I'd	0.06	Want Thai	0.01
START Tell	0.04	To eat	0.26
START I'm	0.02	To have	0.14
I want	0.32	To spend	0.09
I would	0.29	To be	0.02
I don't	0.08	British food	0.60
I have	0.04	British restaurant	0.15
Want to	0.65	British cuisine	0.01
Want a	0.05	British lunch	0.01

Bigram example from the Berkeley Restaurant Project (BeRP)

- Assume $P(\text{END} \mid \text{food}) = 0.2$

$$\begin{aligned} &P(\text{I want to eat British food}) = \\ &P(\text{I} \mid \text{START}) \cdot P(\text{want} \mid \text{I}) \cdot P(\text{to} \mid \text{want}) \cdot P(\text{eat} \mid \text{to}) \cdot \\ &P(\text{British} \mid \text{eat}) \cdot P(\text{food} \mid \text{British}) \cdot P(\text{END} \mid \text{food}) = \\ &.25 \cdot .32 \cdot .65 \cdot .26 \cdot .001 \cdot .60 \cdot .2 = .0000016 \end{aligned}$$

$$\begin{aligned} &P(\text{I want to eat Chinese food}) = \\ &P(\text{I} \mid \text{START}) \cdot P(\text{want} \mid \text{I}) \cdot P(\text{to} \mid \text{want}) \cdot P(\text{eat} \mid \text{to}) \cdot \\ &\mathbf{P(\text{Chinese} \mid \text{eat})} \cdot \mathbf{P(\text{food} \mid \text{Chinese})} \cdot P(\text{END} \mid \text{food}) = \\ &.25 \cdot .32 \cdot .65 \cdot .26 \cdot \mathbf{.02} \cdot \mathbf{.60} \cdot .2 = .000032 \end{aligned}$$

log probabilities

- Probabilities can become very small (a few orders of magnitude per token).
- We often work with log probabilities in practice.

$$p(w_1 \dots w_n) = \prod_{i=1}^n p(w_i | w_{i-1})$$

$$\log p(w_1 \dots w_n) = \sum_{i=1}^n \log p(w_i | w_{i-1})$$

$w_0 = \textit{START}$

What do ngrams capture?

- Probabilities seem to capture *syntactic facts* and *world knowledge*.
- *eat* is often followed by a NP.
- *British* food is not too popular, but *Chinese* is.

Estimating n-gram probabilities

- We can estimate n-gram probabilities using maximum likelihood estimates.

$$p(w|u) = \frac{\textit{count}(u, w)}{\textit{count}(u)}$$

- Or for trigrams:

$$p(w|u, v) = \frac{\textit{count}(u, v, w)}{\textit{count}(u, v)}$$

Bigram Counts from BeRP

	I	Want	To	Eat	Chinese	Food	lunch
I	8	1087	0	13	0	0	0
Want	3	0	786	0	6	8	6
To	3	0	10	860	3	0	12
Eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
Food	19	0	17	0	0	0	0
Lunch	4	0	0	0	0	1	0

Counts to Probabilities

	I	Want	To	Eat	Chinese	Food	lunch
I	8	1087	0	13	0	0	0
Want	3	0	786	0	6	8	6
To	3	0	10	860	3	0	12
Eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
Food	19	0	17	0	0	0	0
Lunch	4	0	0	0	0	1	0

- Unigram counts:

I	Want	To	Eat	Chinese	Food	Lunch
3437	1215	3256	938	213	1506	459

$$P(want|I) = \frac{count(I, want)}{count(I)} = 1087/3437 = 0.32$$

Corpora

- Large digital collections of text or speech. Different languages, domains, modalities. Annotated or un-annotated.
- English:
 - Brown Corpus
 - BNC, ANC
 - Wall Street Journal
 - AP newswire
 - DARPA/NIST text/speech corpora
(Call Home, ATIS, switchboard, Broadcast News,...)
 - MT: Hansards, Europarl

Google Web 1T 5-gram Corpus

File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens:	1,024,908,267,229
Number of sentences:	95,119,665,584
Number of unigrams:	13,588,391
Number of bigrams:	314,843,401
Number of trigrams:	977,069,902
Number of fourgrams:	1,313,818,354
Number of fivegrams:	1,176,470,663

Google Web 1T 5-gram Corpus

- 3-gram examples:

ceramics collectables collectibles 55

ceramics collectables fine 130

ceramics collected by 52

ceramics collectible pottery 50

ceramics collectibles cooking 45

ceramics collection , 144

ceramics collection . 247

ceramics collection </S> 120

ceramics collection and 43

ceramics collection at 52

ceramics collection is 68

ceramics collection of 76

Google Web 1T 5-gram Corpus

- 4-gram examples:

serve as the incoming 92
serve as the incubator 99
serve as the independent 794
serve as the index 223
serve as the indication 72
serve as the indicator 120
serve as the indicators 45
serve as the indispensable 111
serve as the indispensable 40
serve as the individual 234
serve as the industrial 52
serve as the industry 607
serve as the info 42
serve as the informal 102

Data sparsity in n-gram models

- Sparsity is a problem all over NLP: Test data contains language phenomena not encountered during training.
- For n-gram models there are two issues:
 - We may not have seen all tokens.
 - We may not have seen all ngrams (even though the individual tokens are known).
 - Token has not been encountered in this context before.

$$P(\text{lunch} \mid I) = 0.0$$

Unseen Tokens

- Typical approach to unseen tokens:
 - Start with a specific lexicon of known tokens.
 - Replace all tokens in the training and testing corpus that are not in the lexicon with an *UNK* token.
- Practical approach:
 - Lexicon contains all words that appear more than k times in the training corpus.
 - Replace all other tokens with UNK.

Unseen Contexts

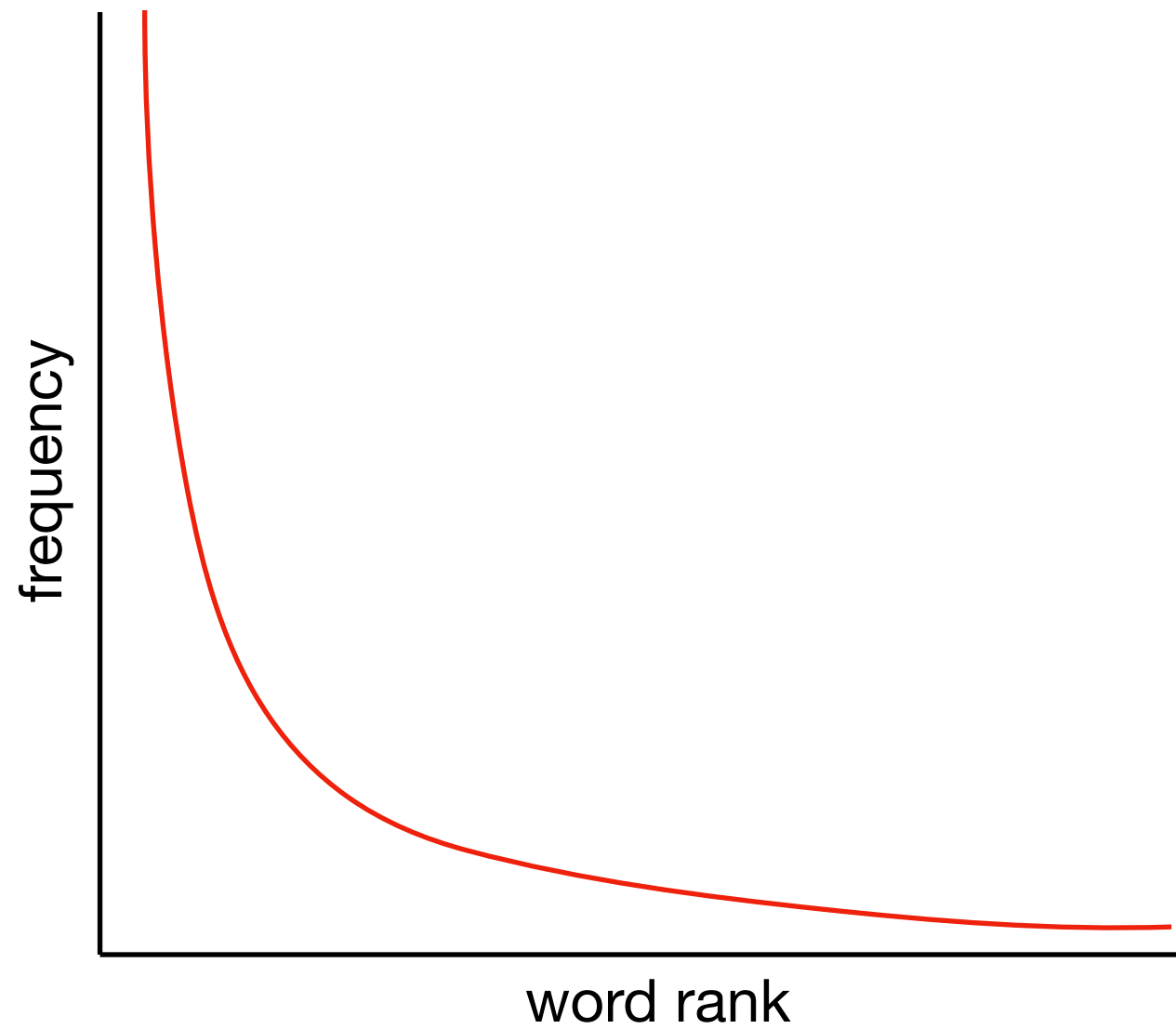
- Two basic approaches:
 - Smoothing / Discounting: Move some probability mass from seen trigrams to unseen trigrams.
 - Back-off: Use $n-1$ -, $n-2$ -... grams to compute n -gram probability.
- Other techniques:
 - Class-based backoff, use back-off probability for a specific word class / part-of-speech.

Zipf's Law

- Problem: n-grams (and most other linguistic phenomena) follow a *Zipfian* distribution.
- A few words occur very frequently.
- Most words occur very rarely. Many are seen only once.

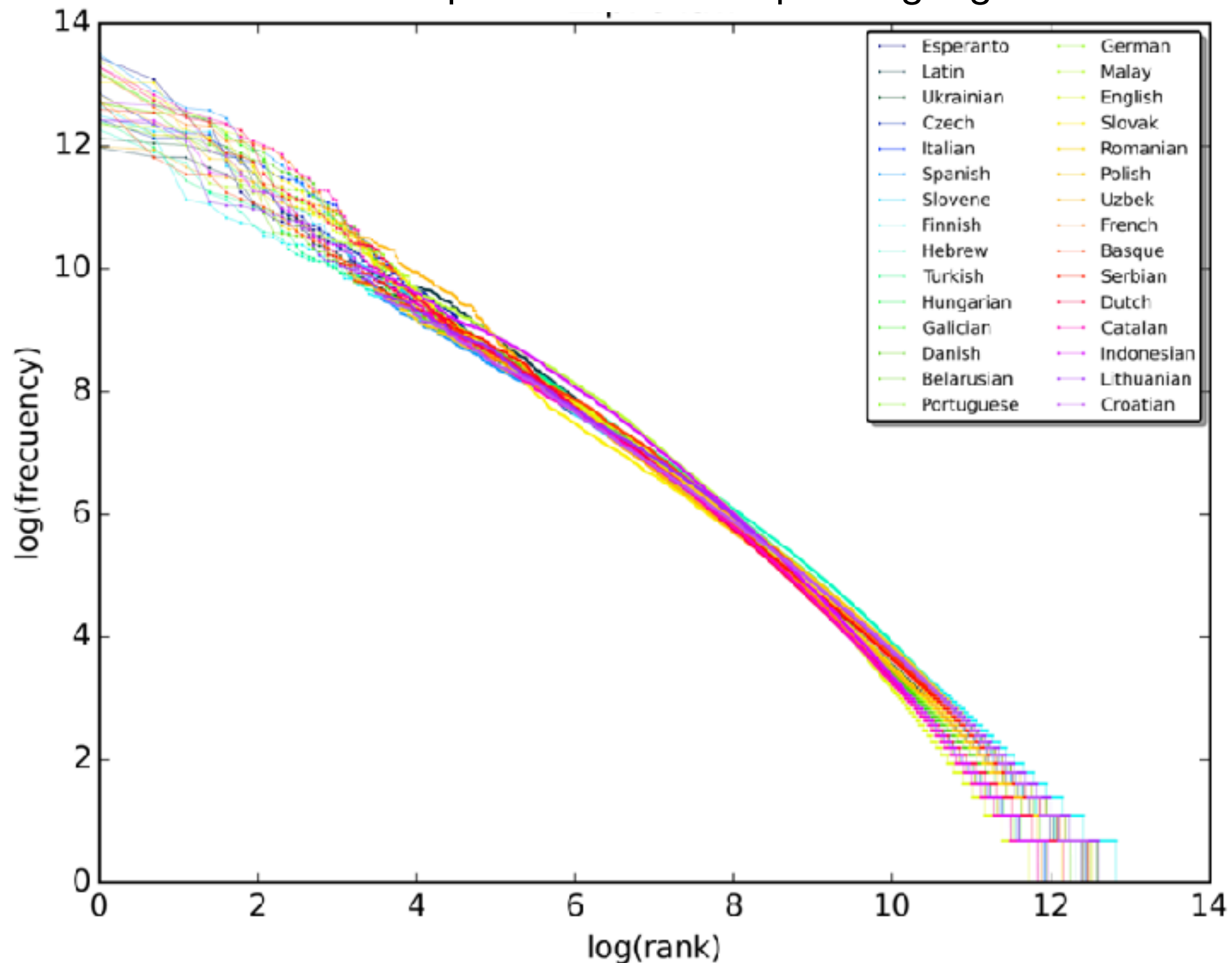
Zipf's law: a word's frequency is approximately inversely proportional to its rank in the word distribution list.

Zipf's Law



Zipf's Law

Wikipedia 10m words per language

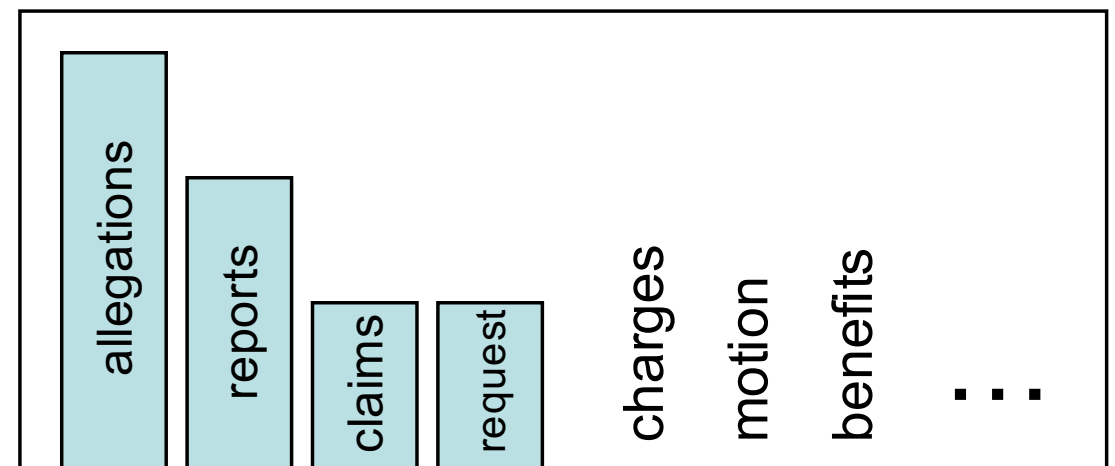


Smoothing

- Smoothing flattens spiky distributions.

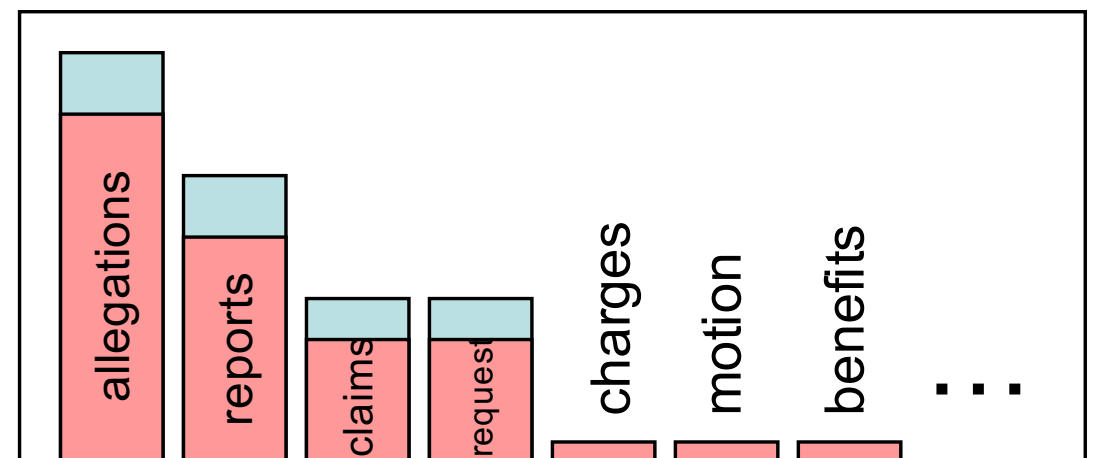
- before $P(w \mid \text{We denied the})$

3 allegations
2 reports
1 claims
1 request
7 total



- after $P(w \mid \text{We denied the})$

2.5 allegations
1.5 reports
0.5 claims
0.4 request
2 UNK
7 total



Smoothing is like Robin Hood: Steal from the rich, give to the poor.

Additive Smoothing

- Classic approach: Laplacian, a.k.a. additive smoothing.

$$P(w_i) = \frac{\text{count}(w_i) + 1}{N + V}$$

- N is the number of tokens, V is the number of types (i.e. size of the vocabulary)

$$P(w|u) = \frac{\text{count}(u, w) + 1}{\text{count}(u) + V}$$

- Inaccurate in practice.

Linear Interpolation

- Use denser distributions of shorter ngrams to “fill in” sparse ngram distributions.

$$p(w|u, v) = \lambda_1 \cdot p_{mle}(w|u, v) + \lambda_2 \cdot p_{mle}(w|u) + \lambda_3 \cdot p_{mle}(w)$$

- Where $\lambda_1, \lambda_2, \lambda_3 > 0$ and $\lambda_1 + \lambda_2 + \lambda_3 = 1$.
- Works well in practice (but not a lot of theoretical justification why).
- Parameters can be estimated on development data (for example, using EM — discussed later).

Discounting

- Idea: set aside some probability mass, then fill in the missing mass using back-off.
- $count^*(v, w) = count(v, w) - \beta$ where $0 < \beta < 1$.
- Then for all seen bigrams: $p(w|v) = \frac{count^*(v, w)}{count(v)}$
- For each context v the missing probability mass is

$$\alpha(v) = 1 - \sum_{w: count(v, w) > 0} \frac{count^*(v, w)}{count(v)}$$

- We can now divide this held-out mass between the unseen words (evenly or using back-off).

Katz' Backoff

- Divide the held-out probability mass proportionally to the unigram probability of the unseen words in context v .

$$p(w|v) = \begin{cases} \frac{\text{count}^*(v,w)}{\text{count}(v)} & \text{if } \text{count}(v, w) > 0 \\ \alpha \times \frac{p_{mle}(w)}{\sum_{w:\text{count}(v,w)=0} p_{ml}(w)} & \text{otherwise} \end{cases}$$

Katz' Backoff for Trigrams

- For trigrams: recursively compute backoff-probability for unseen bigrams. Then distribute the held-out probability mass proportionally to that bigram backoff-probability.

$$p(w|u, v) = \begin{cases} \frac{\text{count}^*(u, v, w)}{\text{count}(u, v)} & \text{if } \text{count}(u, v, w) > 0 \\ \alpha(u, v) \times \frac{p_{BO}(w|v)}{\sum_{w: \text{count}(u, v, w)=0} p_{BO}(w|v)} & \text{otherwise} \end{cases}$$

- where: $\alpha(u, v) = 1 - \sum_{w: \text{count}(u, v, w) > 0} \frac{\text{count}^*(u, v, w)}{\text{count}(u, v)}$
- Often combined with Good-Turing smoothing.

Evaluating n-gram models

- Extrinsic evaluation: Apply the model in an application (for example language classification). Evaluate the application.
- Intrinsic evaluation: measure how well the model approximates unseen language data.
 - Can compute the probability of each sentence according to the model. Higher probability -> better model.
 - Typically we compute *Perplexity instead*.

Perplexity

- Perplexity (per word) measures how well the ngram model predicts the sample.
- Perplexity is defined as 2^l , where $l = \frac{1}{M} \sum_{i=1}^m \log_2 p(s_i)$.
- Lower perplexity = better model. Intuition:
 - Assume we are predicting one word at a time.
 - With uniform distribution, all successor words are equally likely. Perplexity is equal to vocabulary size.
 - Perplexity can be thought of as “effective vocabulary size”.

Natural Language Processing

Lecture 5: Sequence Labeling with Hidden Markov Models. Part-of-Speech Tagging.

01/31/2018

COMS W4705
Daniel Bauer

Garden-Path Sentences

- *The horse raced past the barn.*
- *The horse raced past the barn fell.*
- *The old dog the footsteps of the young.*
- *The cotton clothing is made of grows in Mississippi.*

Garden-Path Sentences

- Why does this happen?

past tense verb
VBD
*The horse **raced** past the barn **fell*** ???

- **raced** can be a past tense verb or a a past participle (indicating passive voice).
- The verb interpretation is more likely before *fell* is read.

Garden-Path Sentences

- Why does this happen?

past participle
VBN VBD
*[The horse **raced** past the barn] fell*
NP

- **raced** can be a past tense verb or a a past participle (indicating passive voice).
- Once *fell* is read, the verb interpretation is impossible.

Garden-Path Sentences

- Why does this happen?

adjective
JJ NN
[The old dog] *[the footsteps of the young]*
NP NP

- **dog** can be a noun or a verb (plural, present tense)

Garden-Path Sentences

- Why does this happen?

 NN VB
[The old] dog [the footsteps of the young]
 NP NP

- **dog** can be a noun or a verb (plural, present tense)

Parts-of-Speech

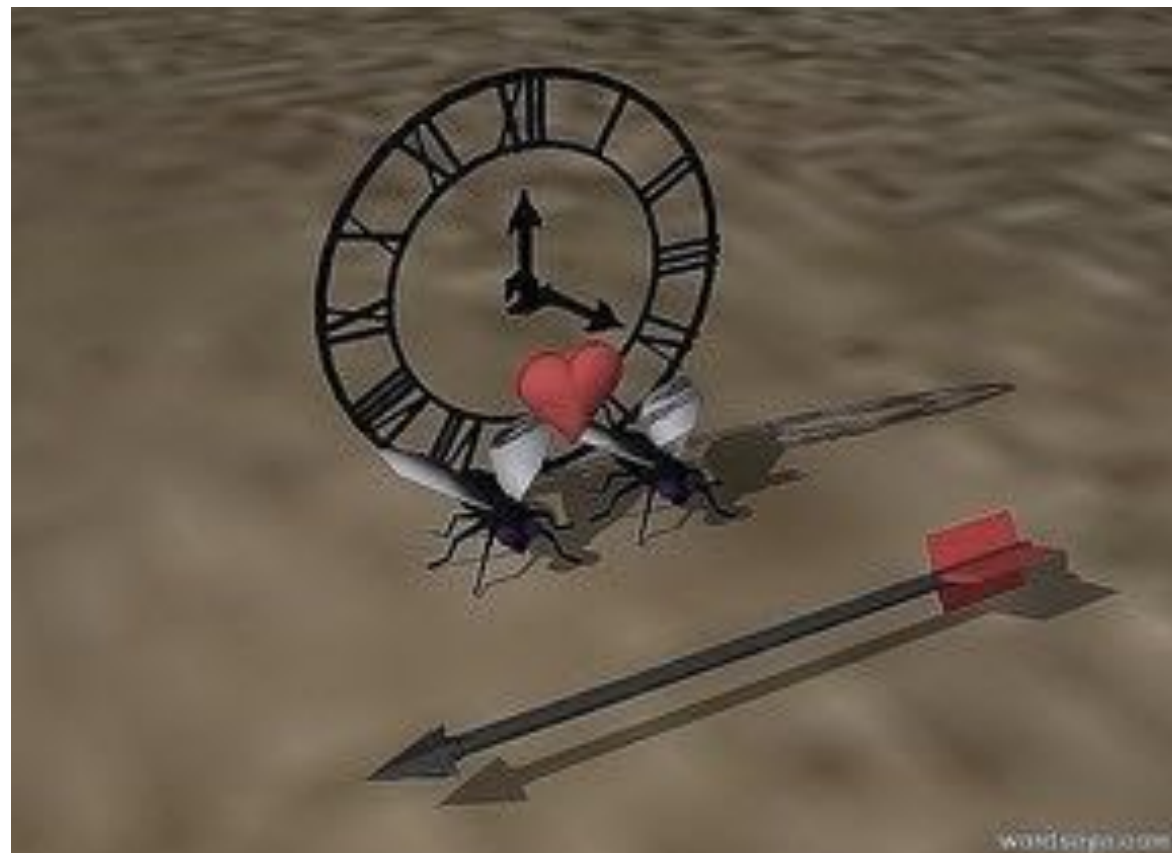
- Classes of words that behave alike:
 - Appear in similar contexts.
 - Perform a similar grammatical function in the sentence.
 - Undergo similar morphological transformations.
 - Have similar meaning.
- ~9 traditional parts-of-speech:
 - noun, pronoun, determiner, adjective, verb, adverb, preposition, conjunction, interjection

Syntactic Ambiguities and Parts-of-Speech

- | N / V? | N / V? | V / Preposition | |
|-------------|--------------|-----------------|------------------|
| <i>Time</i> | <i>flies</i> | <i>like</i> | <i>an arrow.</i> |

Syntactic Ambiguities and Parts-of-Speech

- N N V
 [Time flies] like an arrow.
 NP



Why do we need P.O.S.?

- Interacts with most levels of linguistic representation.
- Speech processing:
 - *lead (V) vs. lead (N).*
 - ***insult**, insult*
 - ***object**, object*
 - ***content**, content*
- Syntactic parsing
- ...
- P.O.S. tag-set should contain morphological and maybe syntactic information.

Penn Treebank Tagset

CC	Coordinating conjunction	PRP\$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential <i>there</i>	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	<i>to</i>
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3rd person singular present
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WP	Wh-pronoun
PDT	Predeterminer	WP\$	Possessive wh-pronoun
POS	Possessive ending	WRB	Wh-adverb
PRP	Personal pronoun	plus punctuation symbols	

P.O.S. Tagsets

- Tagset is language specific.
- Some language capture more morphological information which should be reflected in the tag set.
- “Universal Part Of Speech Tags?”
 - Petrov et al. 2011: Mapping of 25 language specific tag-sets to a common set of 12 universal tags

Part-of-Speech Tagging

- Goal: Assign a part-of-speech label to each word in a sentence.

<i>DT</i>	<i>NN</i>	<i>VBD</i>	<i>DT</i>	<i>NNS</i>	<i>IN</i>	<i>DT</i>	<i>NN</i>	<i>.</i>
<i>the</i>	<i>koala</i>	<i>put</i>	<i>the</i>	<i>keys</i>	<i>on</i>	<i>the</i>	<i>table</i>	<i>.</i>

- This is an example of a **sequence labeling** task.
- Think of this as a translation task from a sequence of words $(w_1, w_2, \dots, w_n) \in V^*$, to a sequence of tags $(t_1, t_2, \dots, t_n) \in T^*$.

Determining Part-of-Speech

- *A blue seat / A child seat:* noun or adj?
- Syntactic tests:
 - *A very **blue** seat*
 - **A very **child** seat*
 - *This seat is **blue***
 - **This seat is **child***
- Morphological Tests
 - *bluer*
 - **childer*

Determining Part-of-Speech

- Preposition or Particle?

- *He threw **out** the garbage.*

- He threw the garbage **out**.*

out is a particle

- *He threw the garbage **out** the door.*

- *He threw the garbage the door **out***

out is a preposition

Determining Part-of-Speech

- How about these:
 - the **back** door
 - on my **back**
 - promised **back** the bill
- Correct P.O.S. depends on context.
- What information do we need?

Part-of-Speech Tagging

- Goal: Translate from a sequence of words $(w_1, w_2, \dots, w_n) \in V^*$, to a sequence of tags $(t_1, t_2, \dots, t_n) \in T^*$.
- NLP is full of translation problems from one structure to another. Basic solution:
 - For each translation step:
 1. Construct search space of possible translations.
 2. Find best paths through this space (decoding) according to some performance measure.

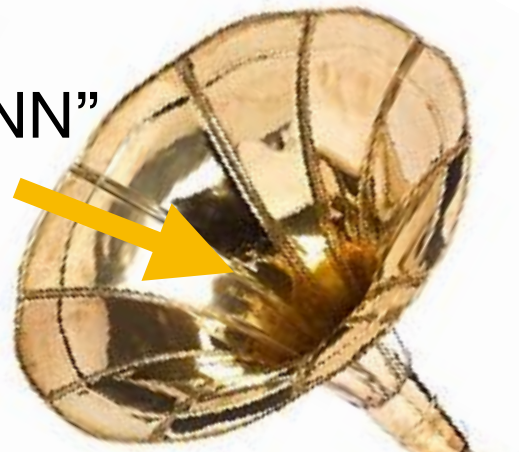
Bayesian Inference for Sequence Labeling

- Recall Bayesian Inference (Generative Models): Given some observation, infer the value of some *hidden variable*. (see Naive Bayes')
- We can apply this approach to sequence labeling:
 - Assume each word w_i in the observed sequence $(w_1, w_2, \dots, w_n) \in V^*$ was generated by some hidden variable t_i .
 - Infer the most likely sequence of hidden variables given the sequence of observed words.

Noisy Channel Model

“NN VBZ IN DT NN”

$P(\text{tags})$



$P(\text{words} | \text{tags})$

“time flies like an arrow”

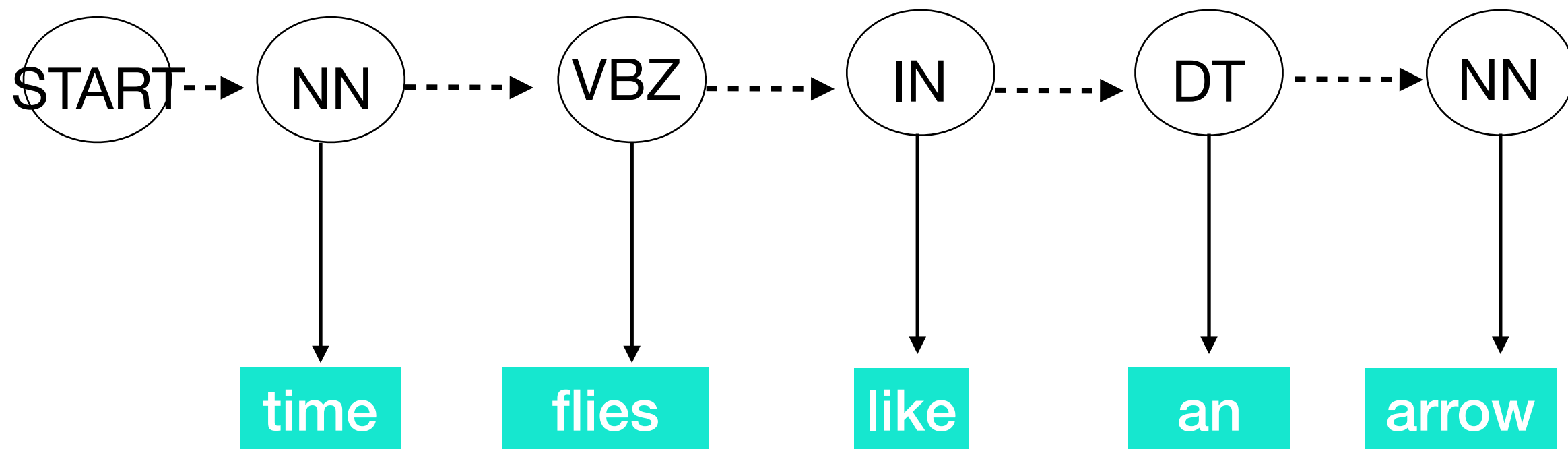
- Goal: figure out what the original input to the the channel was. Use Bayes' rule:

$$\arg \max_{\text{tags}} P(\text{tags} | \text{words}) = \arg \max_{\text{tags}} \frac{P(\text{tags}) \cdot P(\text{word} | \text{tags})}{P(\text{words})}$$

- This model is used widely (speech recognition, MT)

Hidden Markov Models (HMMs)

- Generative (Bayesian) probability model.
Observations: sequences of words.
Hidden states: sequence of part-of-speech labels.

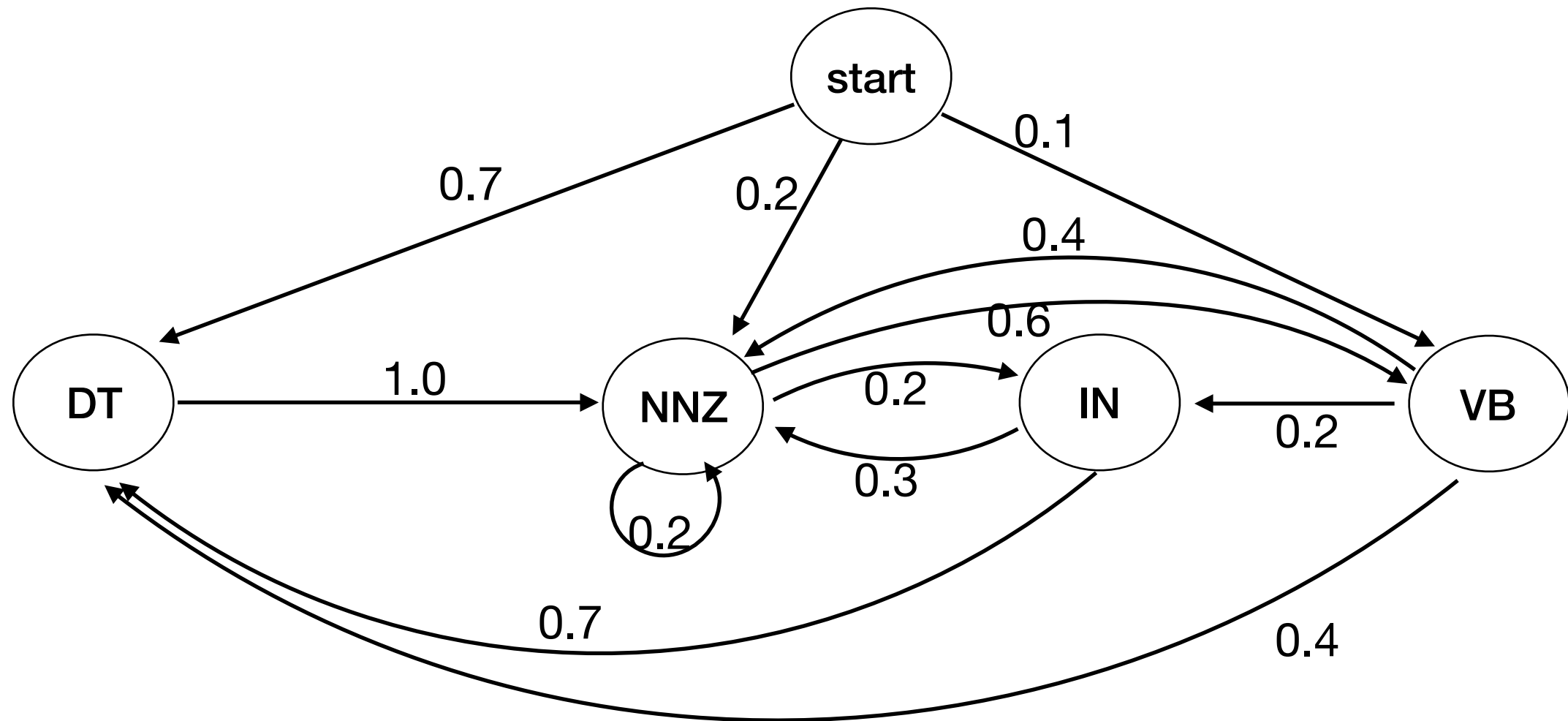


- Hidden sequence is generated by an n-gram language model.

$t_0 = START$

$$P(t_1, t_2, \dots, t_n) = \prod_{i=1}^n P(t_i | t_{i-1})$$

Markov Chains



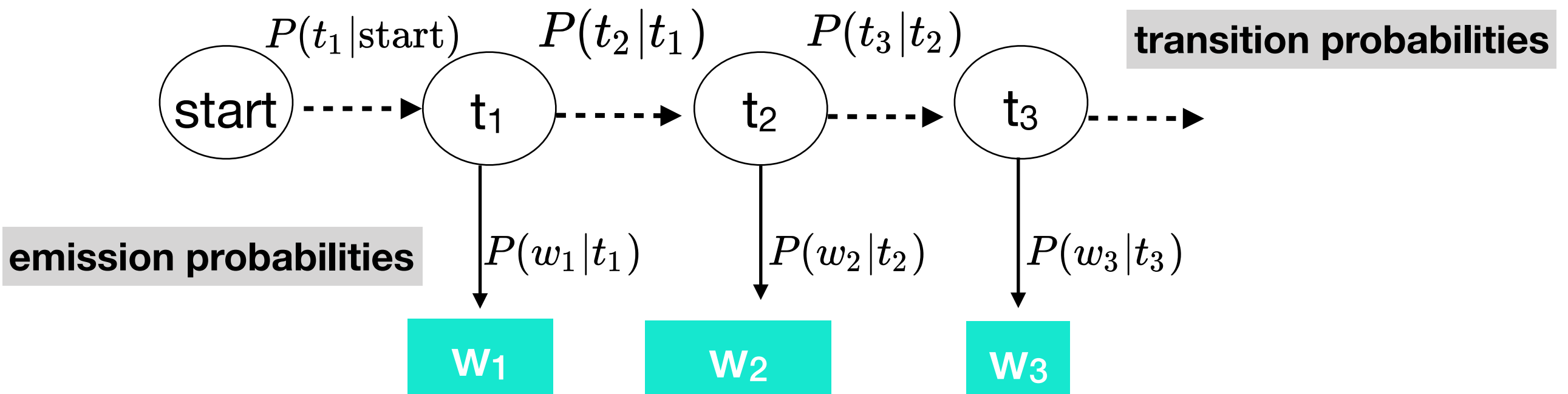
- A **Markov chain** is a sequence of random variables X_1, X_2, \dots
- The domain of these variables is a set of states.
- **Markov assumption** encodes independence assumption:

$$P(X_{n+1} | X_1, X_2, \dots, X_n) = P(X_{n+1} | X_n)$$

- This is a special case of a weighted finite state automaton (WFSA).

Hidden Markov Models (HMMs)

- There are two types of probabilities:
Transition probabilities and Emission Probabilities.



$$P(t_1, t_2, \dots, t_n, w_1, w_2, \dots, w_n) =$$

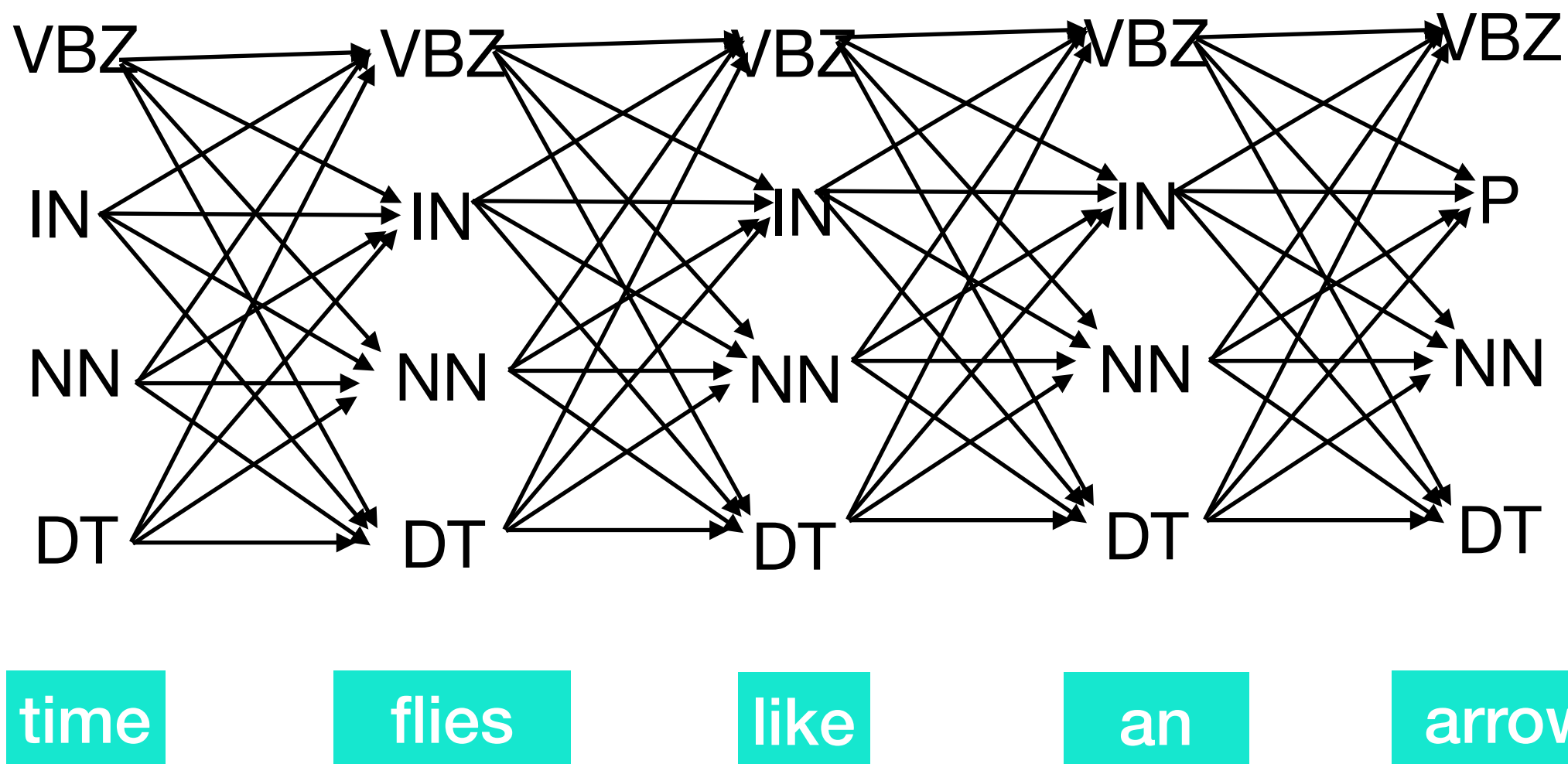
$$P(t_1 | \text{start}) P(w_1 | t_1) P(t_2 | t_1) P(w_2 | t_2) \cdots P(t_n | t_{n-1}) P(w_n | t_n)$$

$$= \prod_{i=1}^n P(t_i | t_{i-1}) P(w_i | t_i)$$

Important Tasks on HMMs

- **Decoding:** Given a sequence of words, find the most likely probability sequence.
(Bayesian inference using *Viterbi algorithm*).
- **Evaluation:** Given a sequence of words, find the total probability for this sequence given an HMM.
Note that we can view the HMM as another type of language model. (Forward algorithm)
- **Training:** Estimate emission and transition probabilities from training data. (MLE, Forward-Backward a.k.a Baum-Welch algorithm)

Decoding HMMs



Goal: Find the path with the highest total probability (given the words)

$$\arg \max_{t_1, \dots, t_n} \prod_{i=1}^n P(t_i | t_{i-1}) P(w_i | t_i)$$

There are d^n paths for n words and d tags.

Emission Probabilities

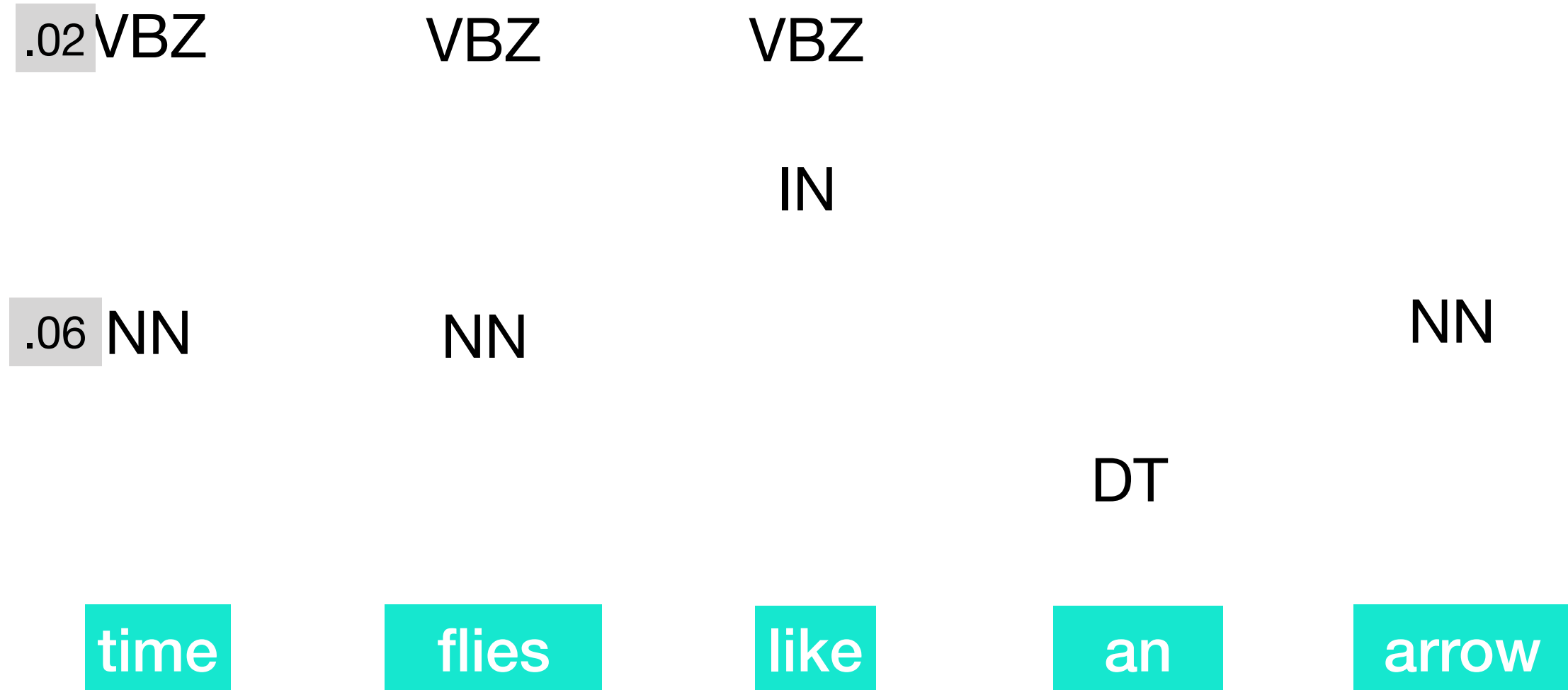
- $P(\text{time} \mid \text{VB}) = 0.2$
 $P(\text{flies} \mid \text{VB}) = 0.3$
 $P(\text{like} \mid \text{VB}) = 0.5$
- $P(\text{time} \mid \text{NN}) = 0.3$
 $P(\text{flies} \mid \text{NN}) = 0.2$
 $P(\text{arrow} \mid \text{NN}) = 0.5$
- $P(\text{like} \mid \text{P}) = 1.0$
- $P(\text{an} \mid \text{DT}) = 1.0$

Viterbi Algorithm

<div><div>.1 x .2 = .02</div><div>VBZ</div></div>	VBZ	VBZ	VBZ	VBZ
<div><div>0</div><div>IN</div></div>	IN	IN	IN	IN
<div><div>.2 x .3 = .06</div><div>NN</div></div>	NN	NN	NN	NN
<div><div>0</div><div>DT</div></div>	DT	DT	DT	DT
<div>time</div>	<div>flies</div>	<div>like</div>	<div>an</div>	<div>arrow</div>

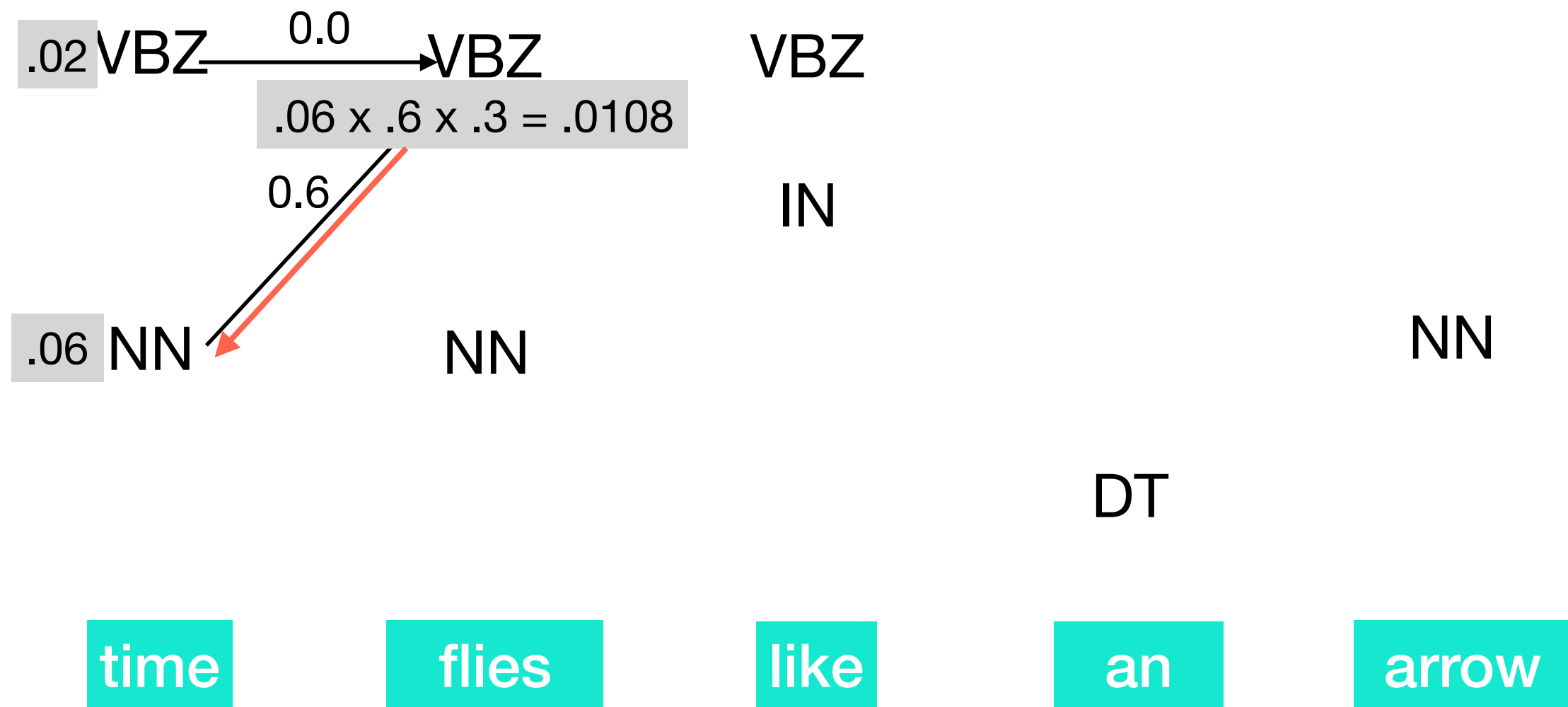
- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} .
This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



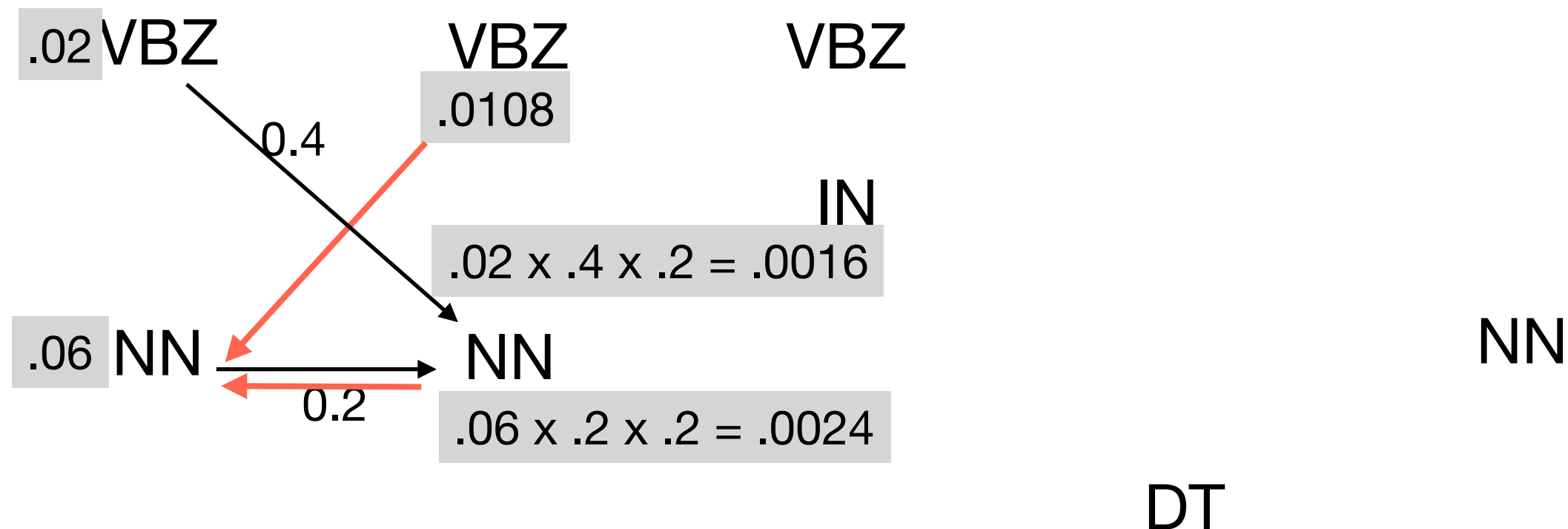
- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} .
This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} . This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



time

flies

like

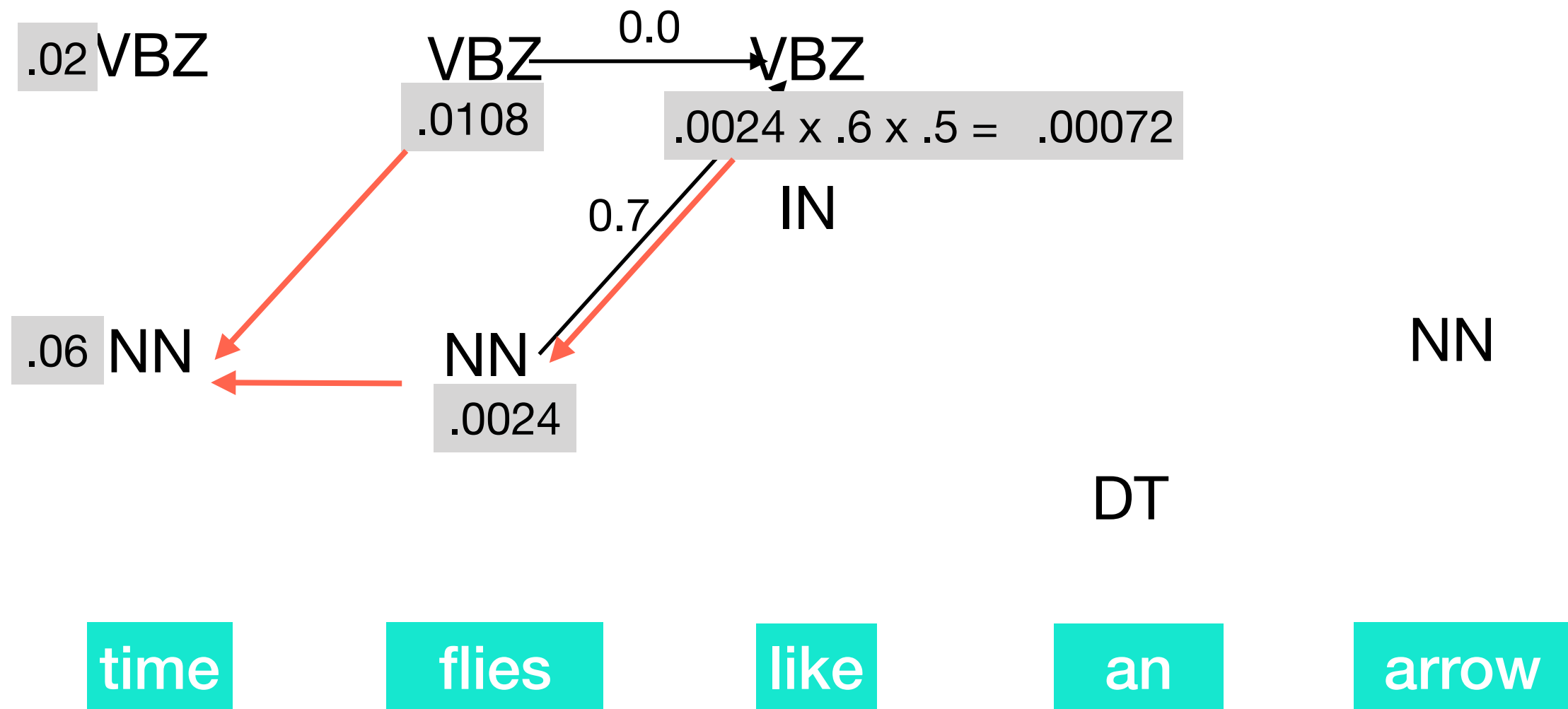
an

arrow

- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} .

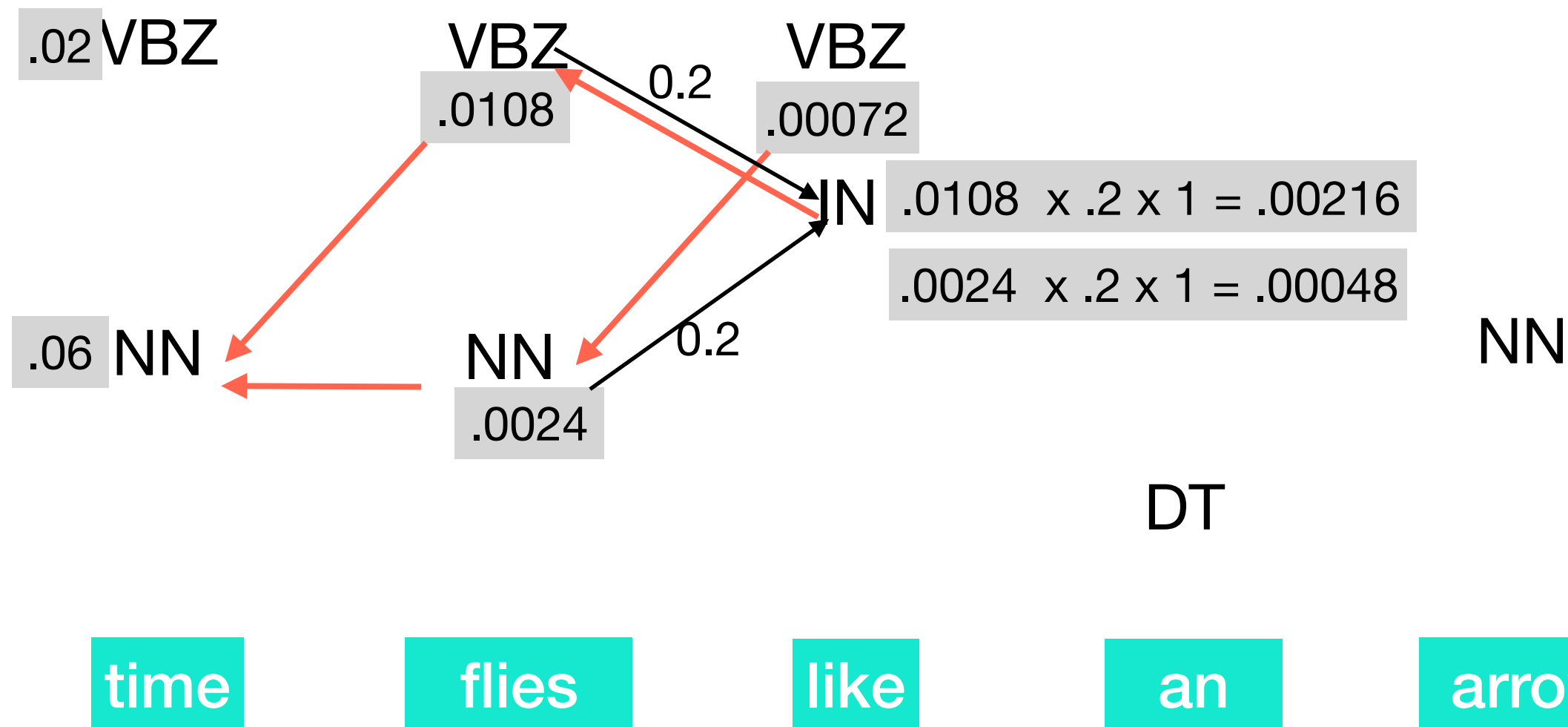
This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



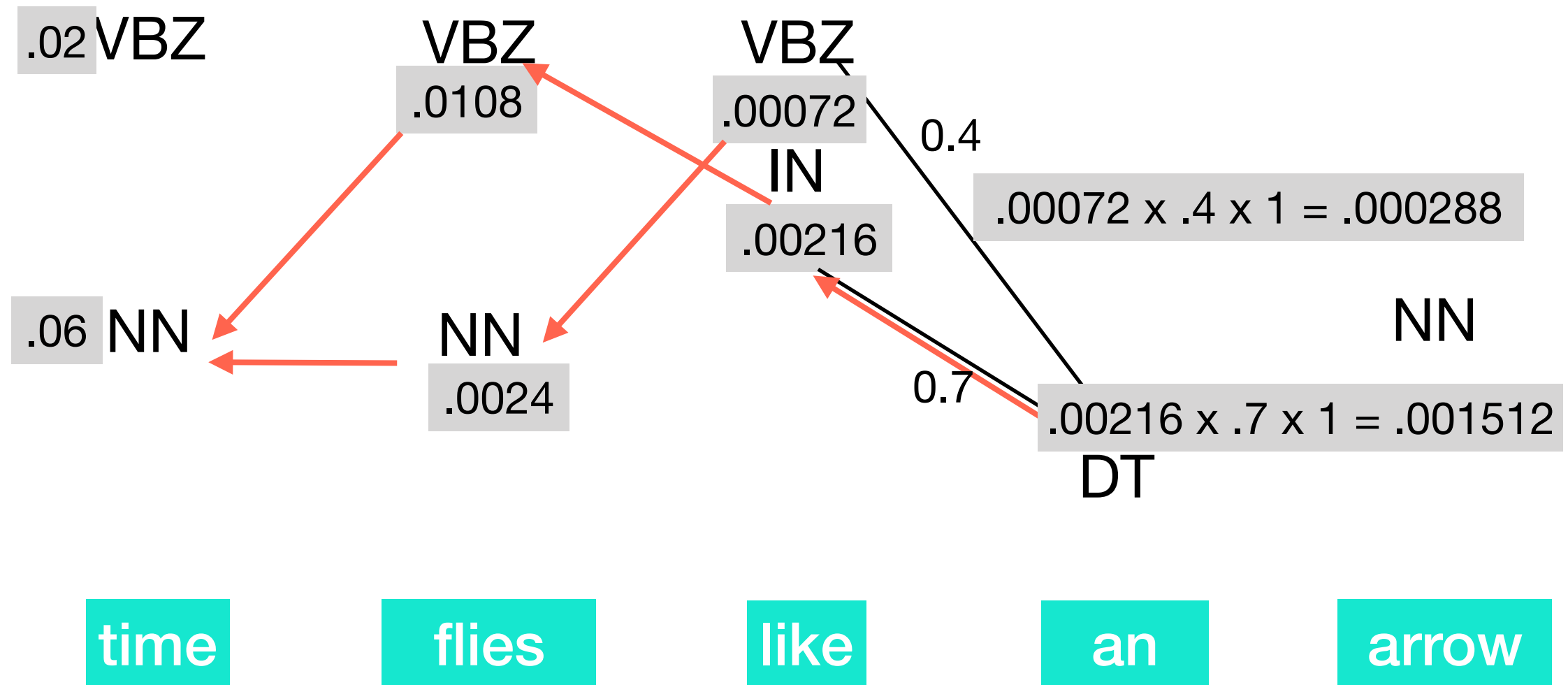
- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} . This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



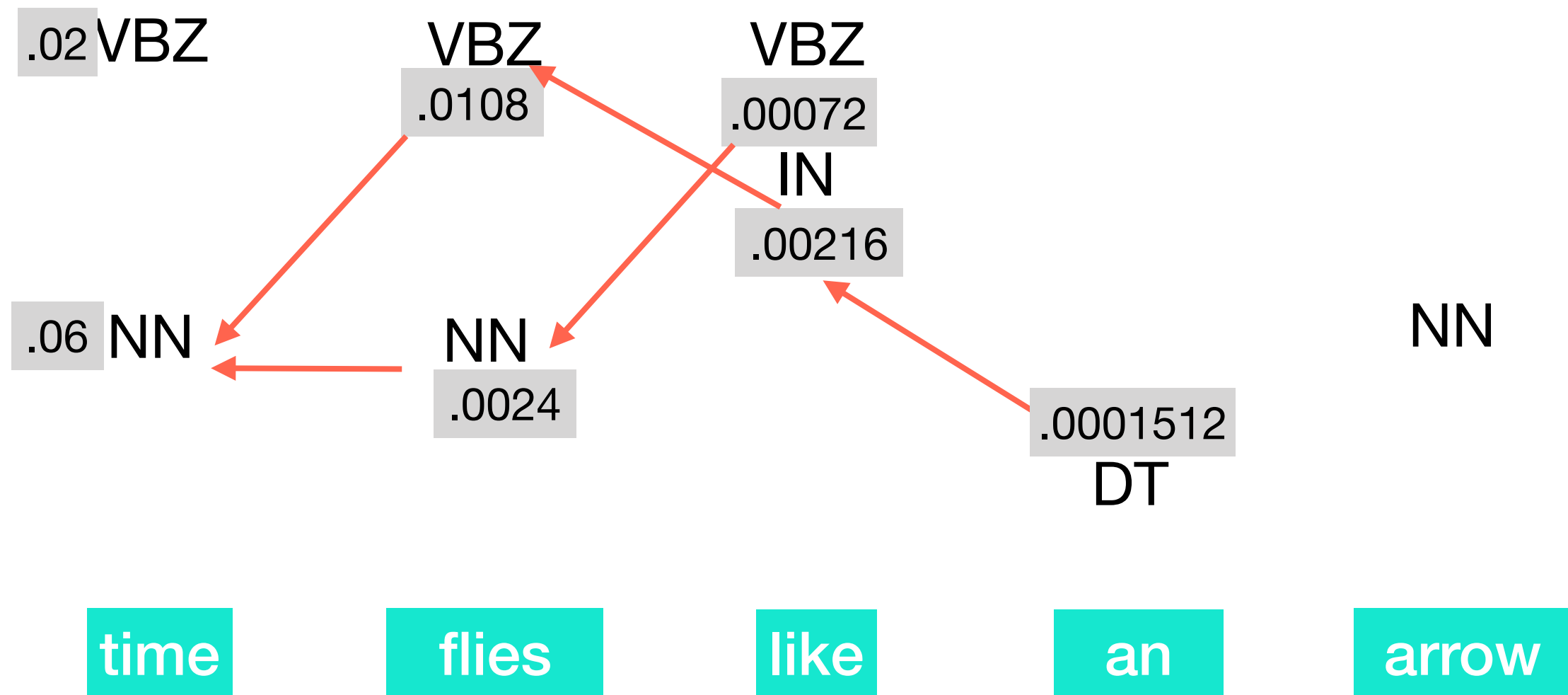
- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} . This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



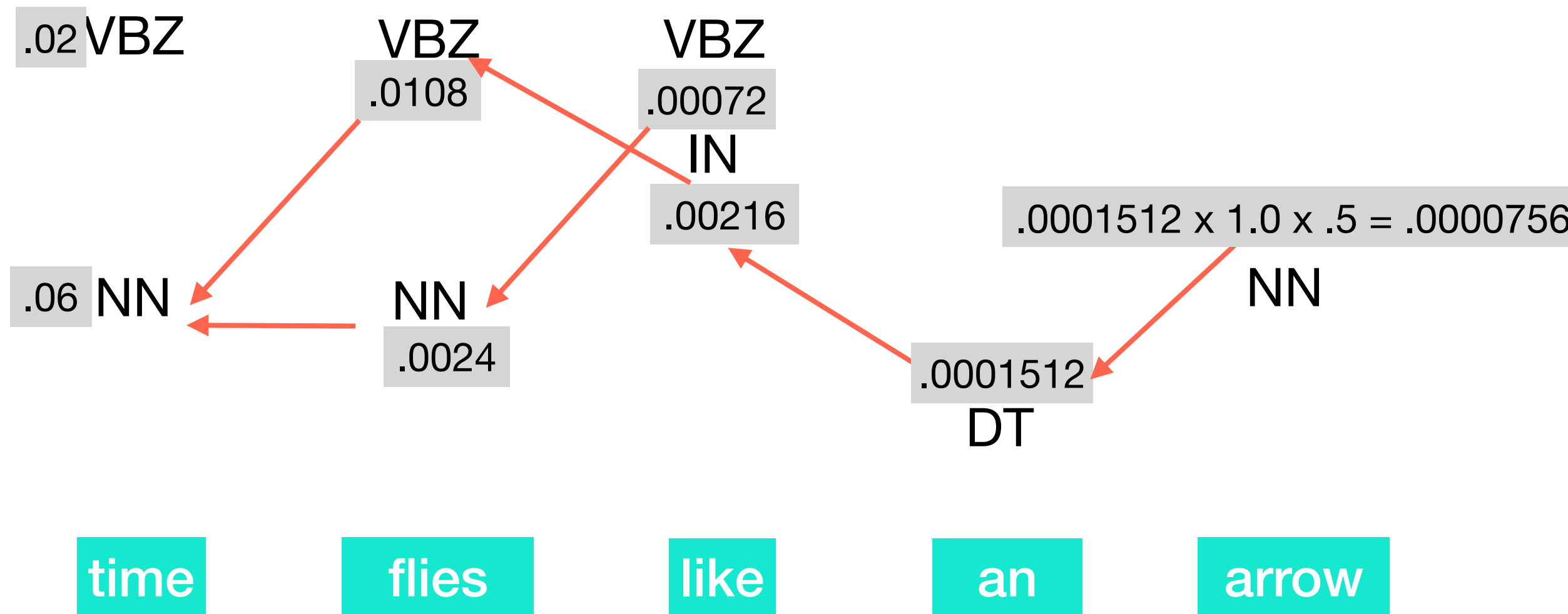
- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} . This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



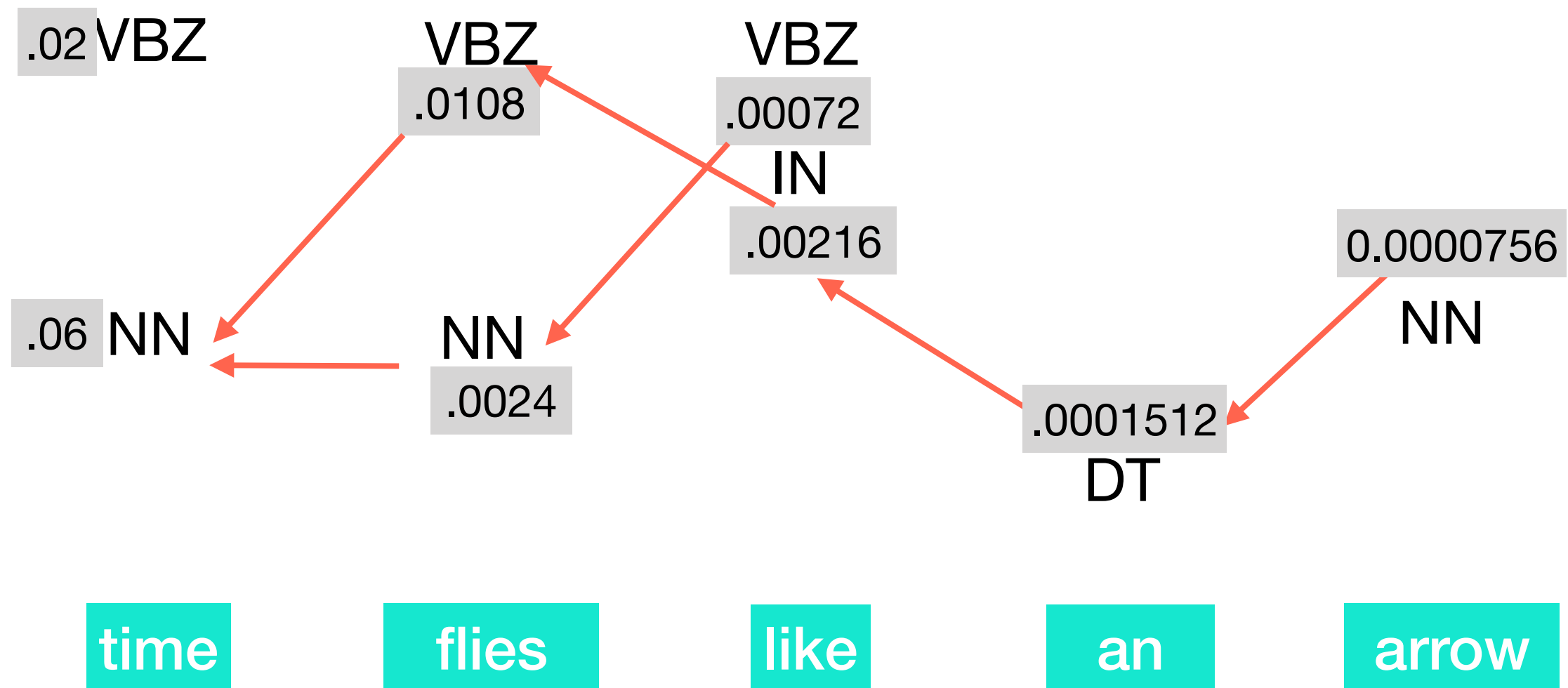
- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} . This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



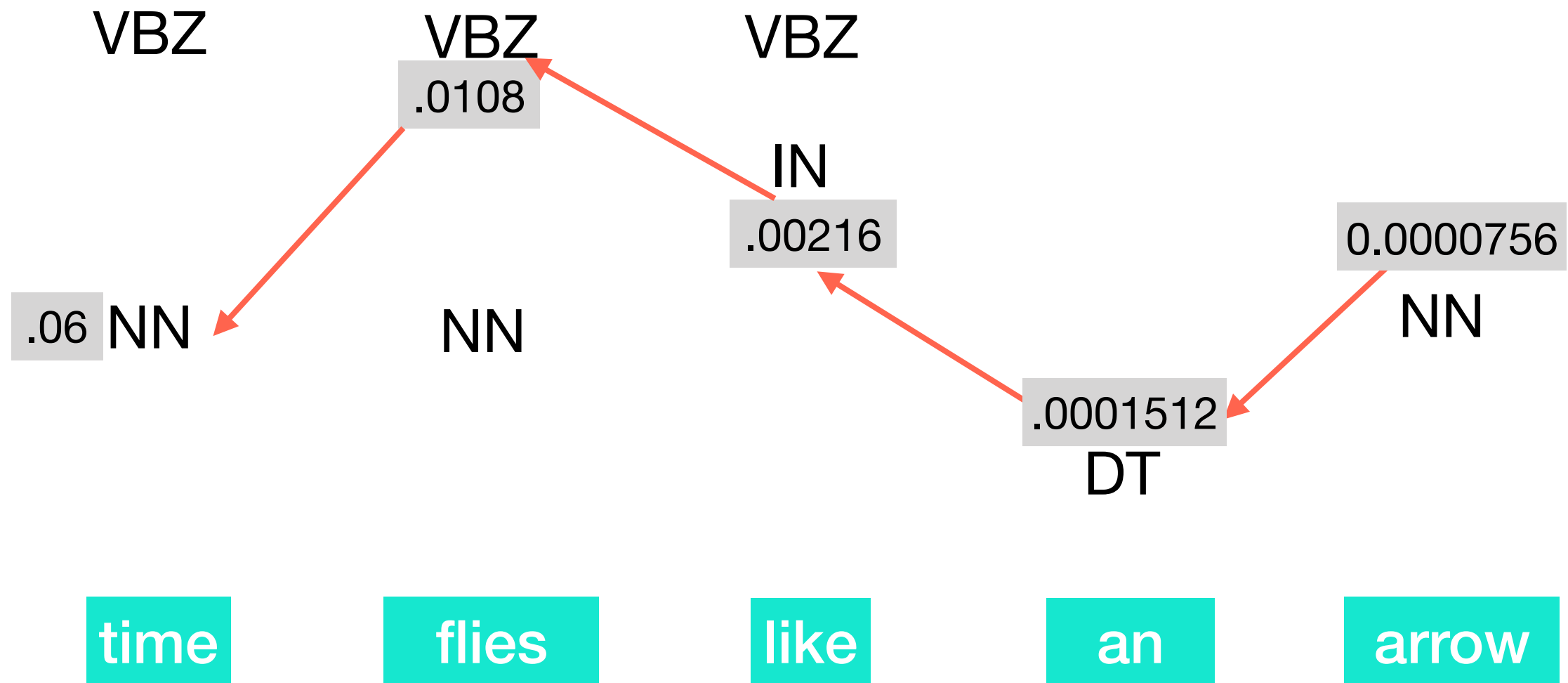
- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} . This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} . This suggests a **dynamic programming** algorithm.

Viterbi Algorithm



- Idea: Because of the Markov assumption, we only need the probabilities for X_n to compute the probabilities for X_{n+1} .
This suggests a **dynamic programming** algorithm.

Viterbi Algorithm

- **Input:** Sequence of observed words w_1, \dots, w_n
- Create a table π , such that each entry $\pi[k, t]$ contains the score of the highest-probability sequence ending in tag t at time k .
- initialize $\pi[0, \text{start}] = 1.0$ and $\pi[0, t] = 0.0$ for all tags $t \in T$.
- for $k = 1$ to n :
 - for $t \in T$:
 - $\pi[k, t] \leftarrow \max_s \pi[k - 1, s] \cdot P(s|t) \cdot P(w_k|s)$
 - transition probability
 - emission probability
- **return** $\max_s \pi[n, s]$

Trigram Language Model

- Instead of using a unigram context $P(t_i | t_{i-1})$, use a trigram context $P(t_i | t_{i-2} t_{i-1})$.
 - Think of this as having states that represent pairs of tags.
- So the HMM probability for a given tag and word sequence is:
$$\prod_{i=1}^n P(t_i | t_{i-2} t_{i-1}) P(w_i | t_i)$$
- Need to handle data sparseness when estimating transition probabilities (for example, linear interpolation).

More POS tagging tricks

- It is also often useful in practice to add an end-of-sentence marker (just like we did for n-gram language models).

$$P(t_1, \dots, t_n, w_1, \dots, w_n) = \left[\prod_{i=1}^n P(t_i | t_{i-2} t_{i-1}) P(w_i | t_i) \right] P(t_{n+1} | t_n)$$

where $t_{-1} = t_0 = \text{START}$ and $t_{n+1} = \text{STOP}$.

- Another useful trick is to replace words with “pseudo words” representing an entire class.
 - For example: replace {“01”, “85”, “90”, ...} with *twoDigitNumber*
replace {“1985”, “2018”, ...} with *fourDigitNumber*
replace {“1”, “1.0”, “234.3” ...} with *otherNum*
replace {“IBM”, “DNC”, ...} with *allCaps etc.*

Using a smoothed trigram HMM model with these tricks, we can build a tagger that is close to the state-of-the art (~97% accuracy on the Penn Treebank).

HMMs as Language Models

- We can also use an HMM as language models (language generation, MT, ...), i.e. **evaluate** $P(w_1, \dots, w_n)$ for a given sentence.

What is the advantage over a plain word n-gram model?

- Problem: There are many tag-sequences that could have generated w_1, \dots, w_n .

$$P(w_1, \dots, w_n, t_1, \dots, t_n) = \prod_{i=1}^n P(t_i | t_{i-1}) P(w_i | t_i)$$

- This is an example of **spurious ambiguity**.

- Need to compute:
$$P(w_1, \dots, w_n) = \sum_{t_1, \dots, t_n} P(w_1, \dots, w_n, t_1, \dots, t_n)$$
$$= \sum_{t_1, \dots, t_n} \left[\prod_{i=1}^n P(t_i | t_{i-1}) P(w_i | t_i) \right]$$

Forward Algorithm

- **Input:** Sequence of observed words w_1, \dots, w_n
- Create a table π , such that each entry $\pi[k, t]$ contains the score of the highest-probability sequence ending in tag t at time k .
- initialize $\pi[0, \text{start}] = 1.0$ and $\pi[0, t] = 0.0$ for all tags $t \in T$.
- for $k = 1$ to n :
 - for $t \in T$:
 - $\pi[k, t] \leftarrow \sum_s \pi[k - 1, s] \cdot P(s|t) \cdot P(w_k | s)$
- **return** $\sum_s \pi[n, s]$

Named Entity Recognition as Sequence Labeling

- Use 3 tags:
 - O - outside of named entity
 - I - inside named entity
 - B - first word (beginning) of named entity

... O O B I O ...
identification of tetronic acid in

- Other encodings are possible (for example, NE-type specific)
- This can also be used for phrase chunking.

Natural Language Processing

Lecture 6: Introduction to Syntax and
Formal Languages.

02/05/2018

COMS W4705
Daniel Bauer

Sentences:

the good, the bad, and the ugly

- Some good sentences:
 - *the boy likes a girl*
 - *the small girl likes a big girl*
 - *a very small nice boy sees a very nice boy*
- Some bad sentences:
 - *the boy the girl likes*
 - *small boy likes nice girl*
- Ugly word salad: *very like nice the girl boy*

More good and bad examples

- *I want to fly to New York.*
- * *I found to fly to New York.*
- *Maud expects there to be a riot.*
- * *Teri promised there to be a riot.*
- *Maud expects shit to hit the fan.*
- * *Teri promised shit to hit the fan.*

*The * means “ungrammatical sentence”*

Syntax

- Study of **structure** of language
 - How words are arranged in a sentence and the relationship between them.
 - Goal: relate surface form (perception) to semantics (meaning).

Syntax as an Interface

- Syntax can be seen as the interface between morphology (structure of words) and semantics.
- Why treat syntax separately from semantics?
- Can judge if a sentence is grammatical or not, even if it doesn't make sense semantically.

Colorless green ideas sleep furiously.

**Sleep ideas furiously colorless green.*

- Why treat syntax separately from morphology?

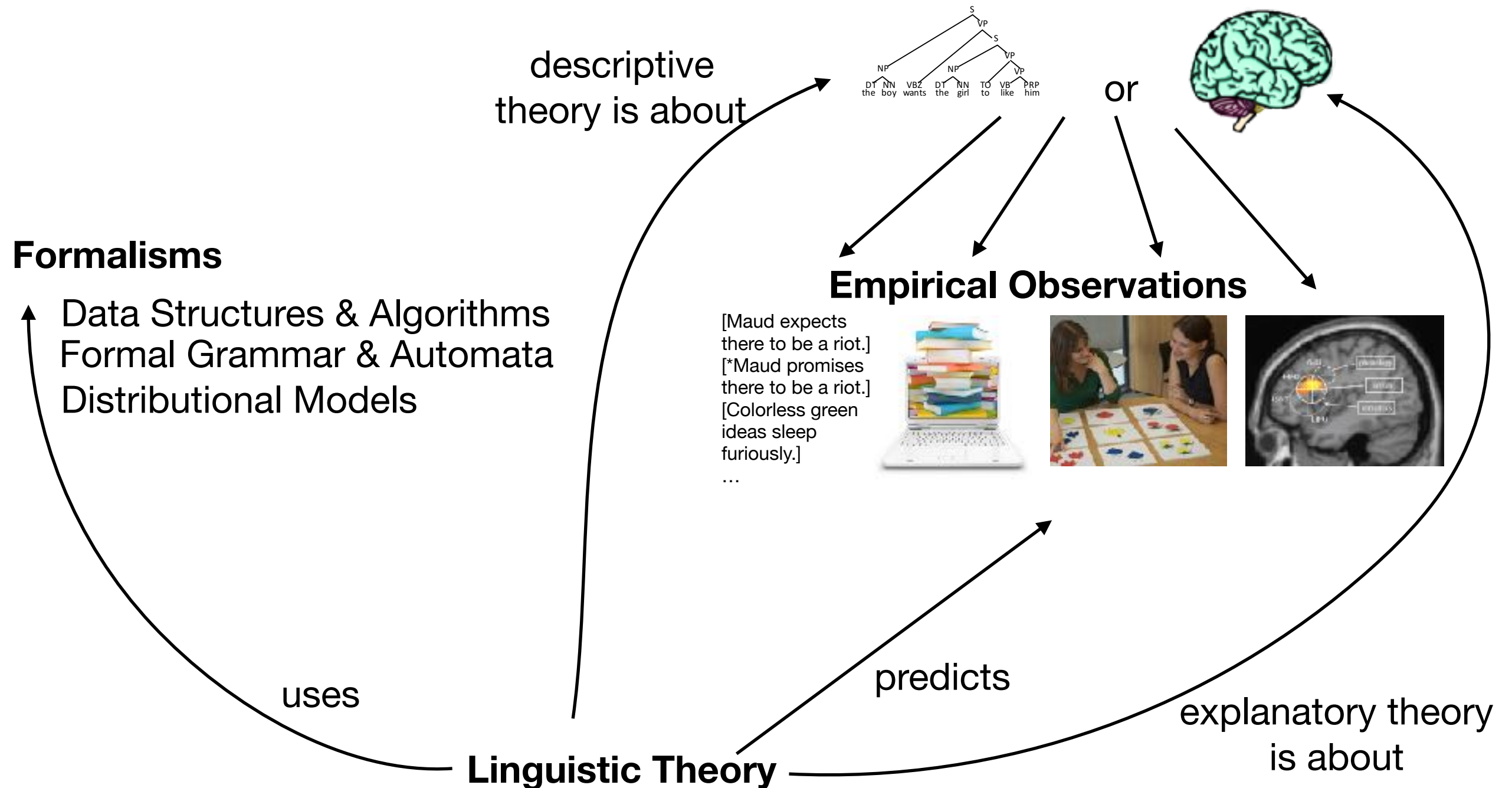
Types of Linguistic Theories

- **Prescriptive:** “This is how people ought to talk.”
 - (“prescriptive linguistics” is an oxymoron)
- **Descriptive:** provide a formal account of **how** people talk.
- **Explanatory:** explain **why** people talk a certain way (identify underlying cognitive, or neural mechanism)

NLP focuses on the descriptive part.

Computational linguistics is interested in finding explanatory theories, but often uses descriptive methods.

The Big Picture



Key Concepts of Syntax

- Constituency and Recursion.
- Dependency.
- Grammatical Relations.
- Subcategorization.
- Long-distance dependencies.

Constituents

- A constituent is a group of words that behave as a single unit (within a hierarchical structure).
- Noun-Phrase examples:
 - [they], [the woman], [three parties from Brooklyn],
[a high-class spot such as Mindy's], [the horse raced past the barn]
 - Noun phrases can appear before verbs (among other things) and they must be complete:
 - **from arrive...*
 - **the is*
 - **spot sat....*
 - **green sleep...*

Constituency Tests

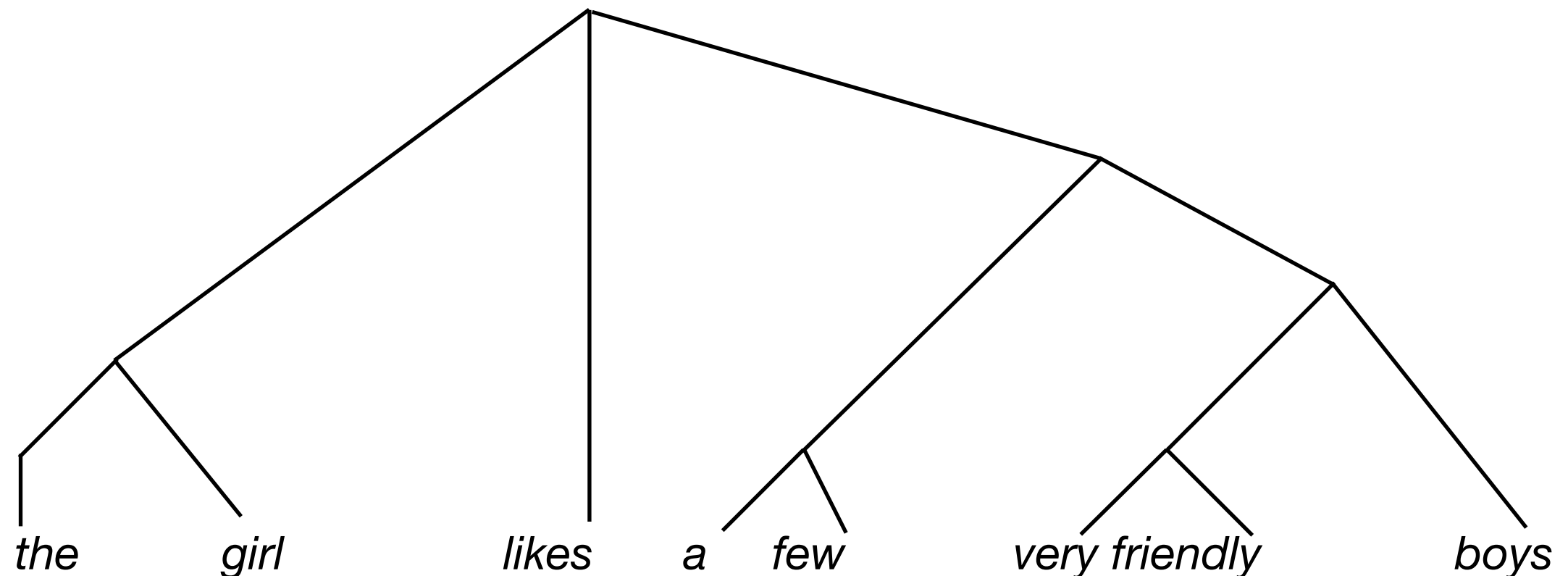
- ***On September seventeenth, I'd like to fly to New York.***
- ***I'd like to fly to New York on September seventeenth.***
- ***I'd like to fly on September seventeenth to New York.***
- ****On I'd like to fly to New York September seventeeth.***
- ****On September I'd like to fly seventeenth to New York.***

More Constituency Tests

- There is a great number of constituency tests. They typically involve moving constituents around or replacing them.
- Topicalization:
 - *I won't eat **that pizza** **That pizza**, I won't eat ***pizza** I won't eat that*
- Pro-form Substitution:
 - *I don't know **the man who sent flowers**. I don't know **him**.
*I don't know **him** flowers.*
- Question test.
 - ***Where** would you like to fly on September seventeenth?*
 - ***When** would you like to fly to New York?*

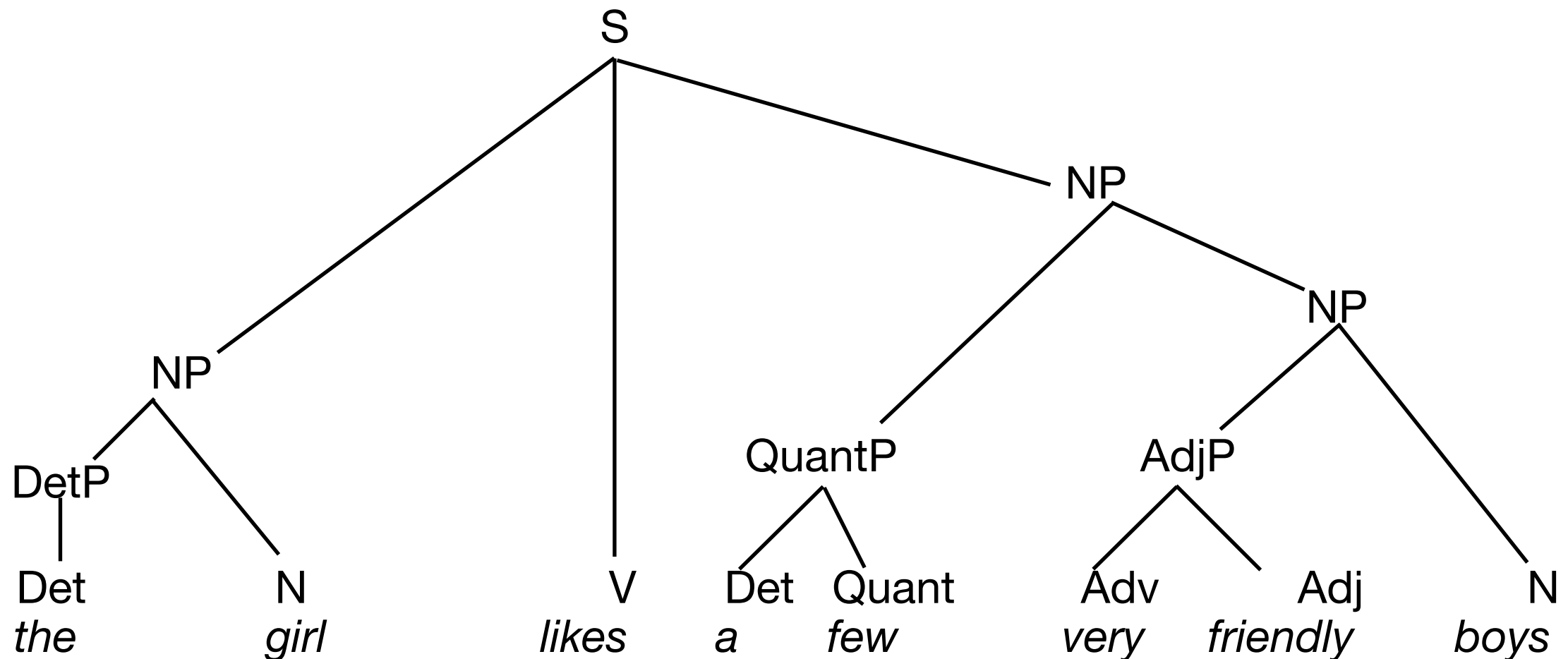
Sentence Structure as Trees

- [the tall girl likes a few very friendly boys]
- [[the tall girl] likes [a few very friendly boys]]
- [[the] tall girl] likes [[a few] [very friendly] boys]]



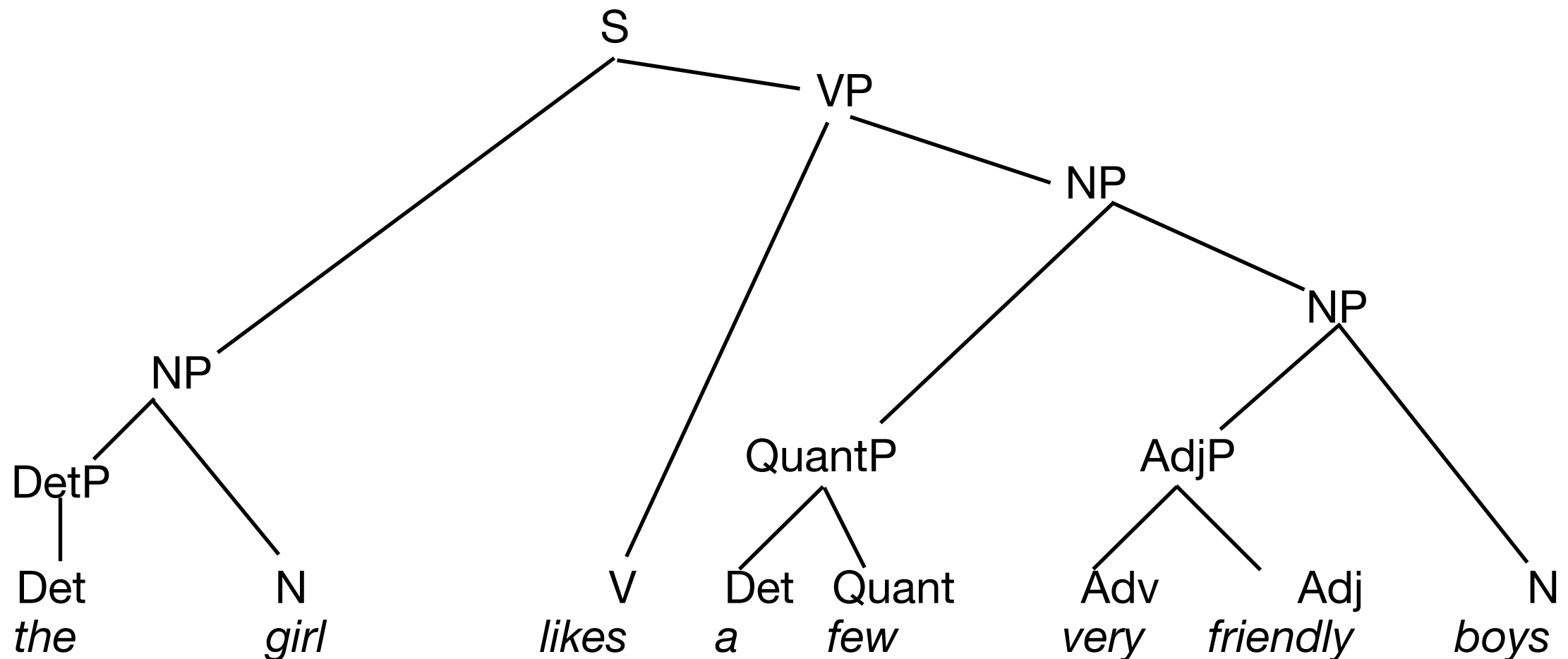
Constituent Labels

- Choose constituents so each one has one non-bracketed word: the **head**.
- Category of Constituent: XP, where X is the part-of-speech of the head
NP, VP, AdjP, AdvP, DetP



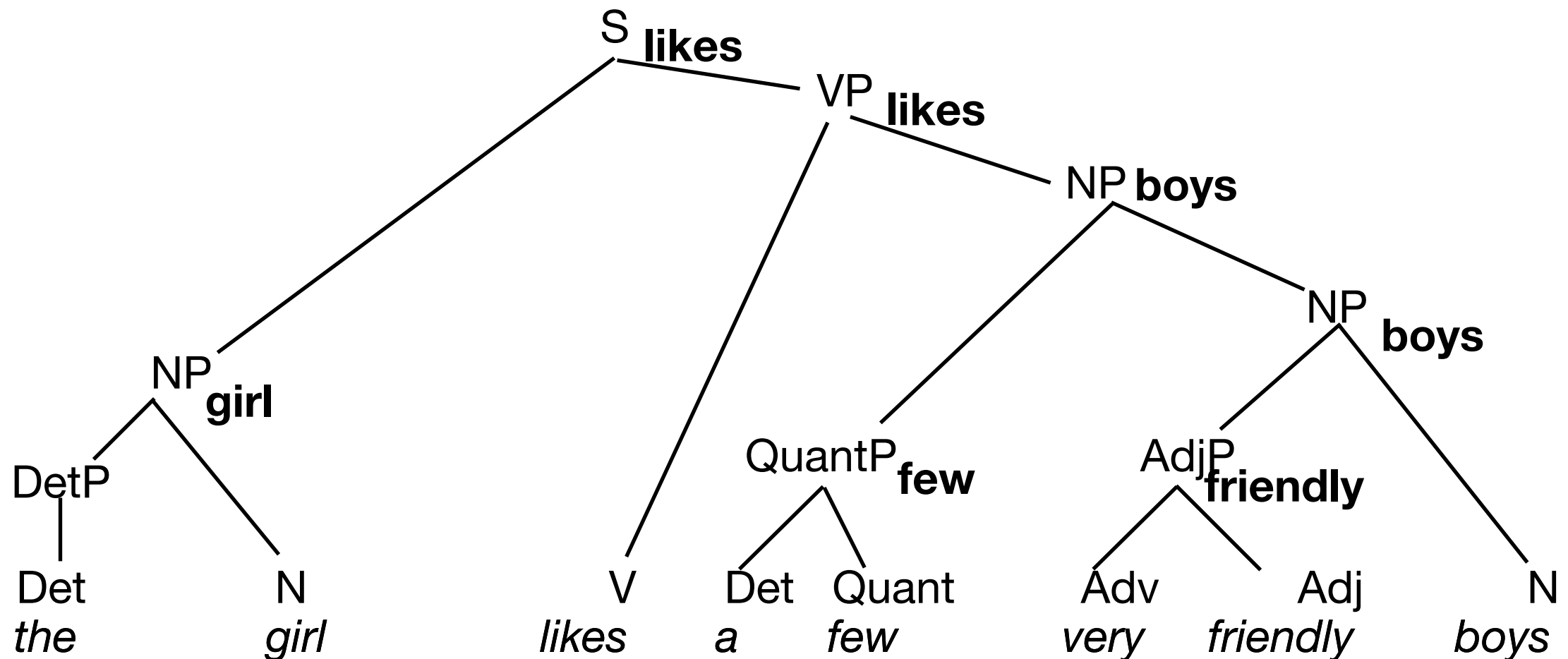
Constituent Labels

- Choose constituents so each one has one non-bracketed word: the **head**.
- Category of Constituent: XP, where X is the P.O.S. of the head
NP, VP, AdjP, AdvP, DetP



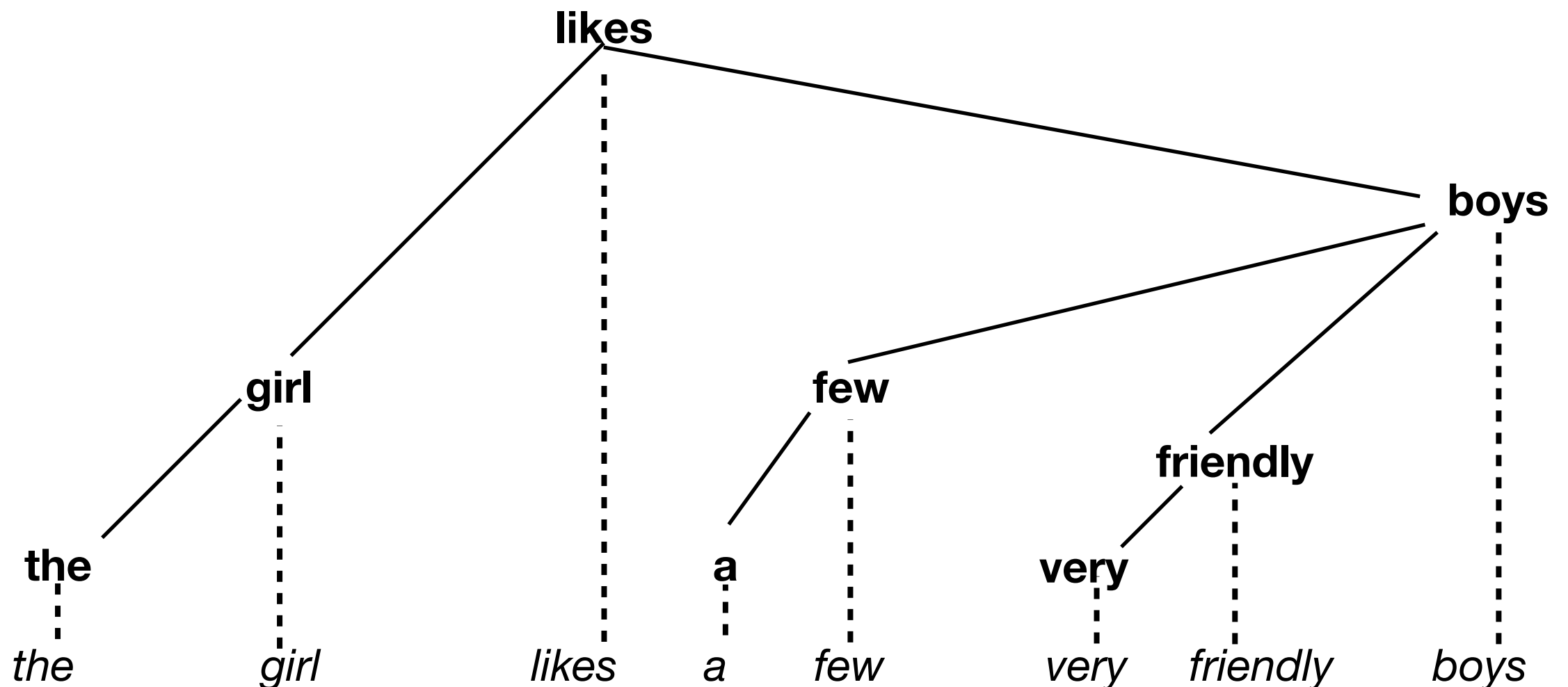
Phrase Structure and Dependency Structure

- We can obtain a dependency structure by projecting the heads up the tree.
- This is not always trivial. In some cases we need **head percolation tables** to know which head to percolate up.



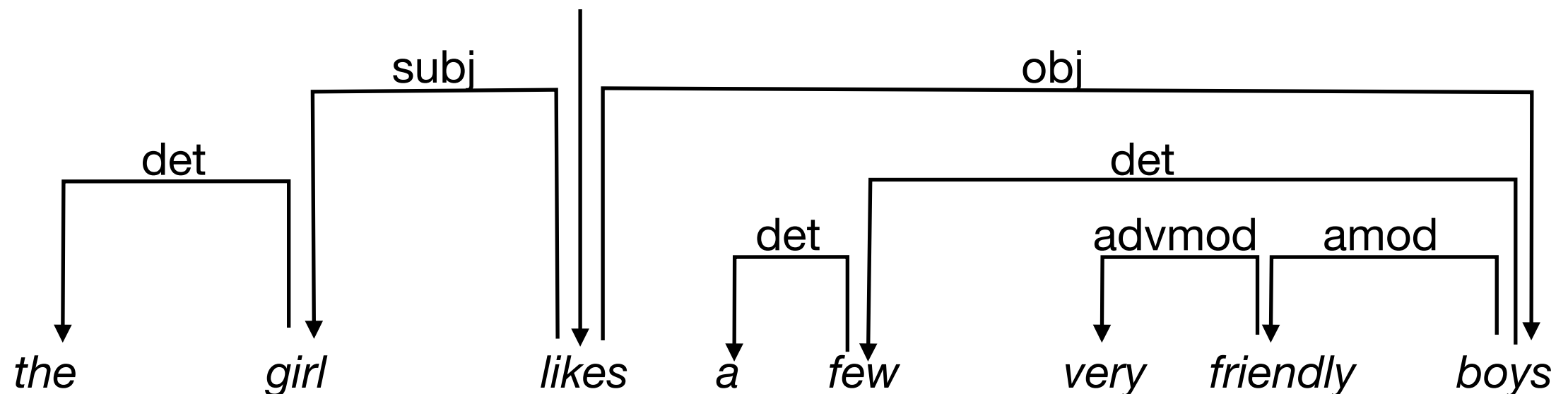
Phrase Structure and Dependency Structure

- We can obtain a dependency structure by projecting the heads up the tree.
- Do we lose any information? What about the VP?



Phrase Structure and Dependency Structure

- Dependency structures are often written like this:



- The edges can be labeled with **grammatical relations** between words (typed dependencies):
 - Arguments (Subject, Object, Indirect Object, Prepositional Object)
 - Adjunct (Temporal, Locative, Causal, Manner...) / Modifier
 - Function words

Natural Language Processing

Lecture 7: Parsing with Context Free Grammars

02/07/2018

COMS W4705
Daniel Bauer

Review: Constituency

The students easily completed the difficult NLP homework.

Which constituents can you identify? What tests could you use?

Recursion in Language

- One of the most important attributes of Natural Languages is that they are **recursive**.
 - *He made pie
[with apples [from the orchard [near the farm [in ...]]]]*
 - *[The mouse [the cat [the dog chased]] ate] died.*
- There are infinitely many sentences in a language, but in predictable structures.
- How do we model the set of sentences in a language and their structure?

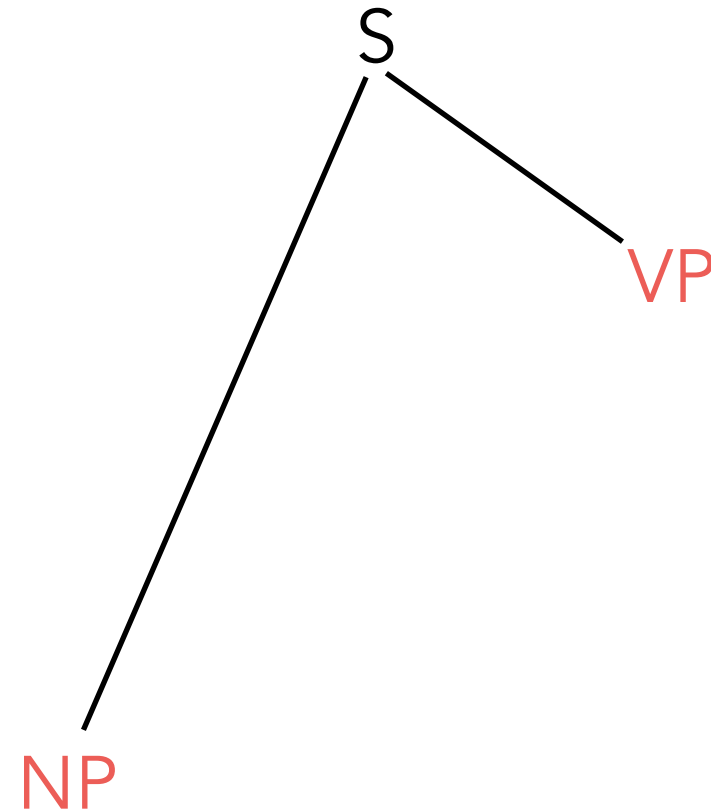
Context Free Grammars (CFG)

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

S

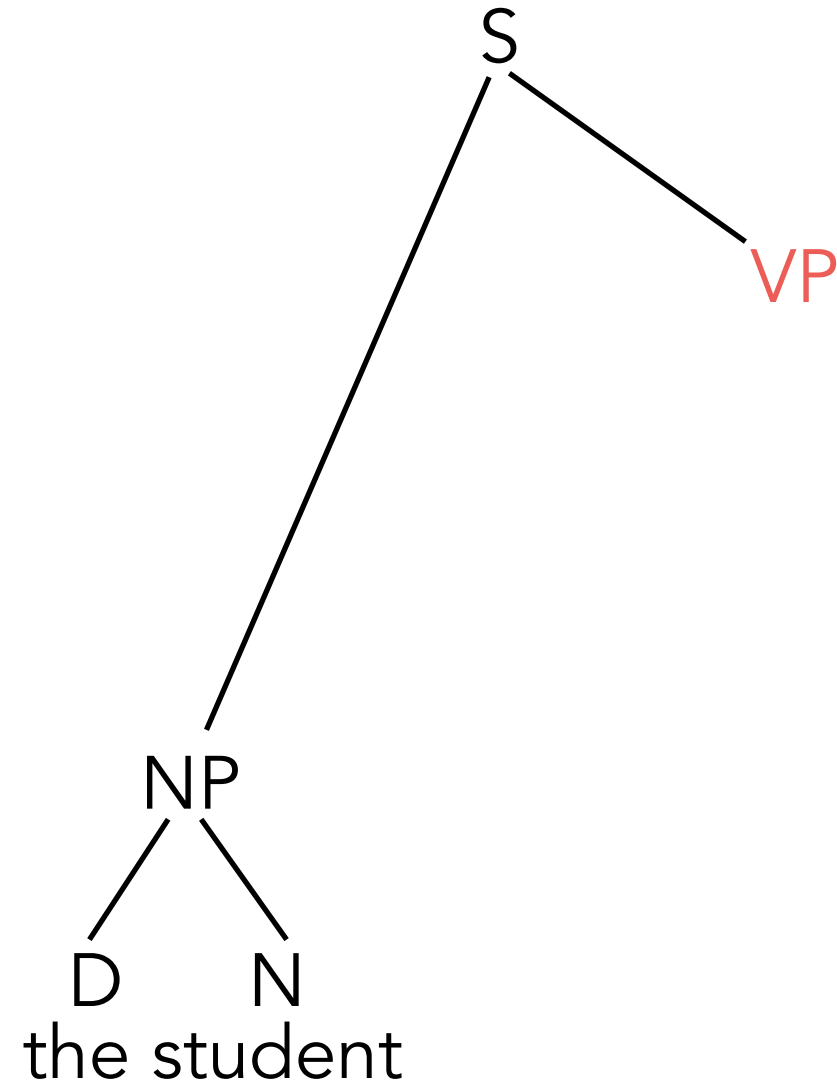
Context Free Grammars (CFG)

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$



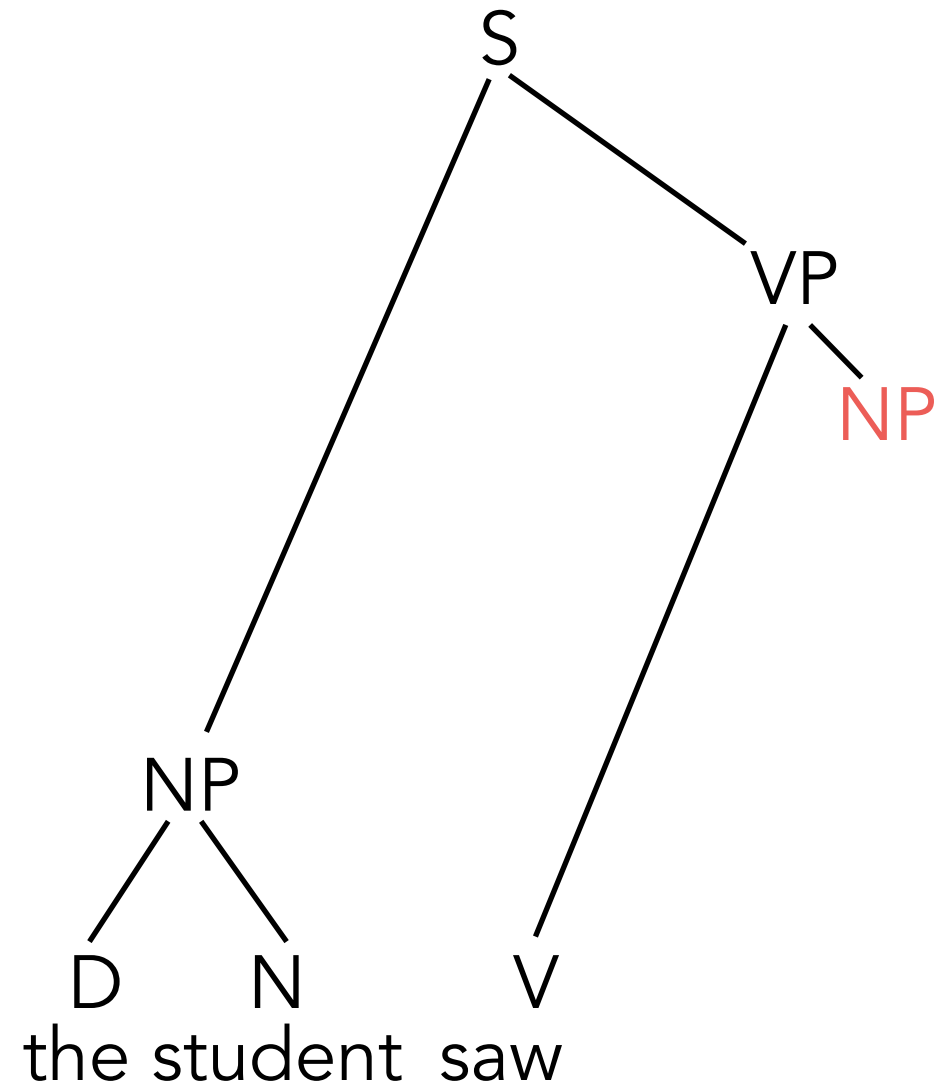
Context Free Grammars (CFG)

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student



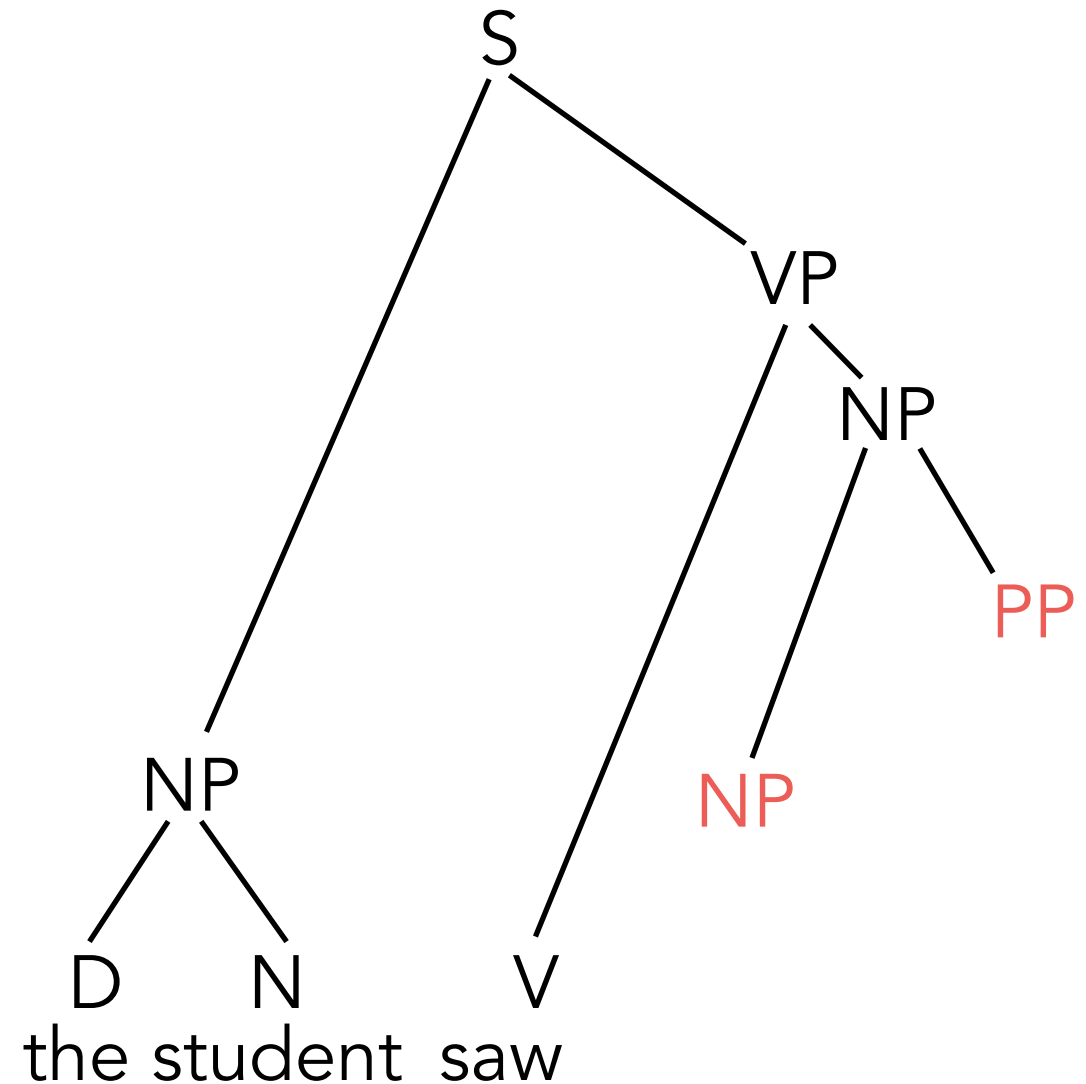
Context Free Grammars (CFG)

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student



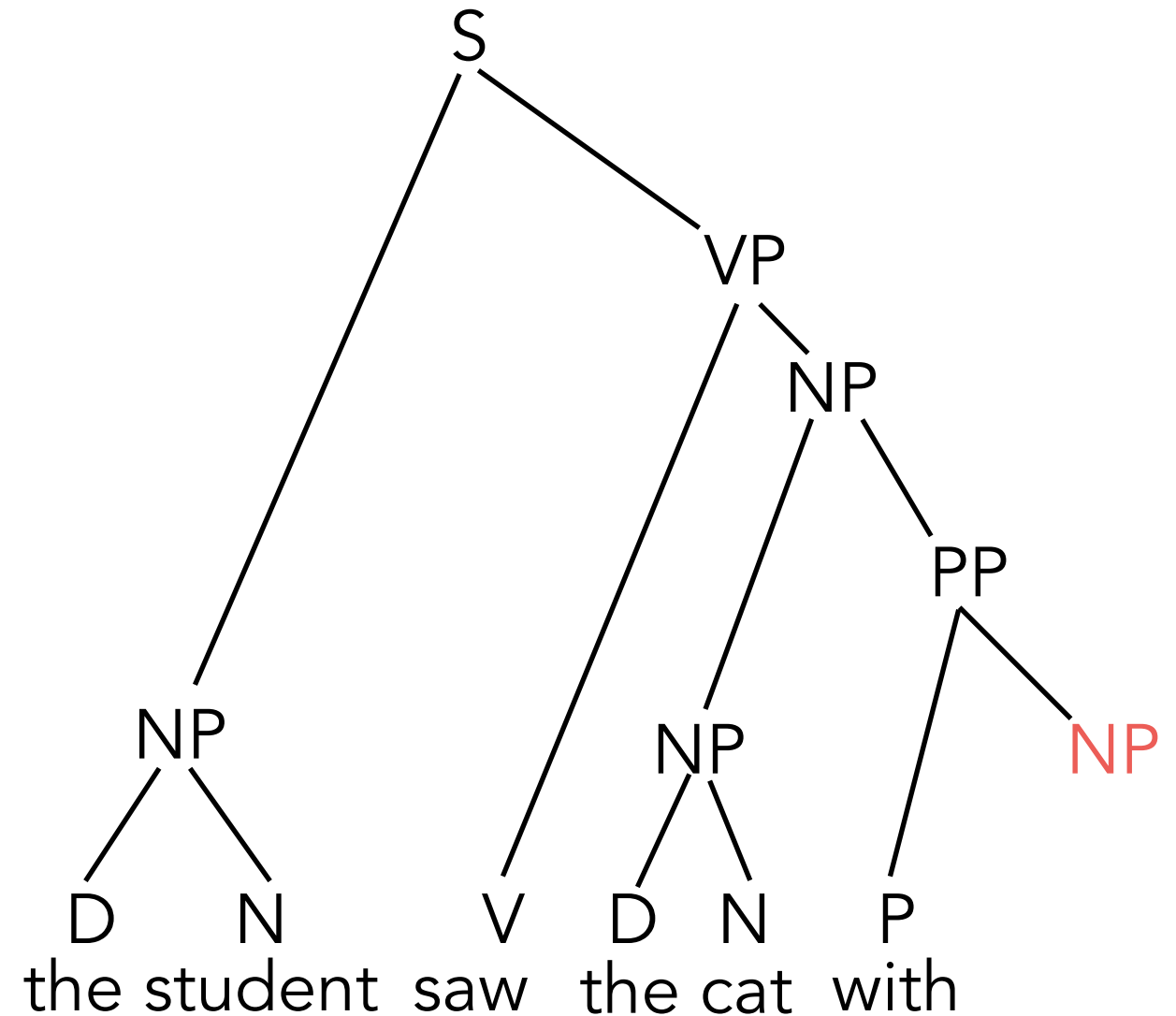
Context Free Grammars (CFG)

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student



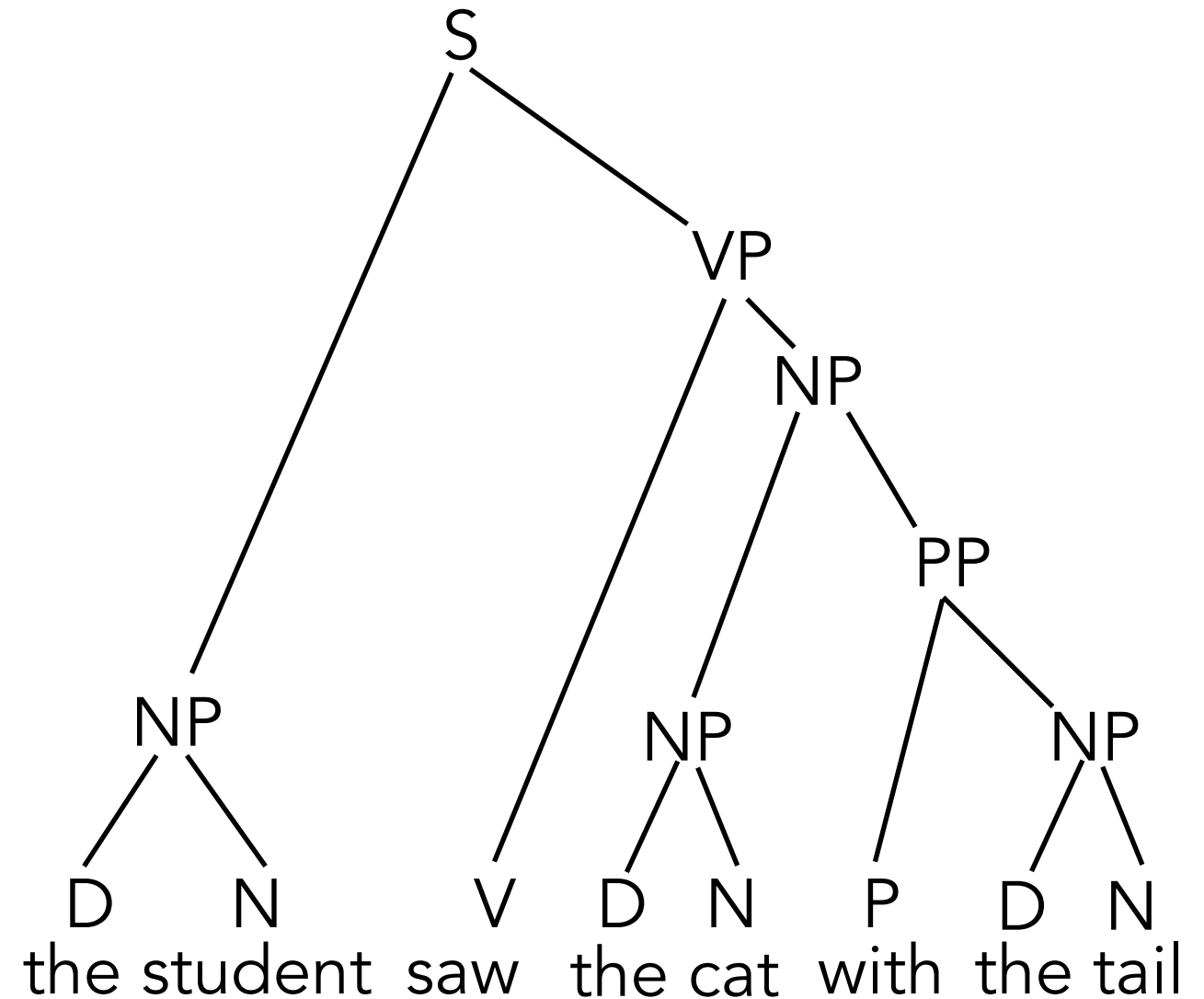
Context Free Grammars (CFG)

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student



Context Free Grammars (CFG)

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student



Context Free Grammars

- A context free grammar is defined by:
 - Set of **terminal symbols** Σ .
 - Set of **non-terminal symbols** N .
 - A **start symbol** $S \in N$.
 - Set R of **productions** of the form $A \rightarrow \beta$,
where $A \in N$ and $\beta \in (\Sigma \cup N)^*$, i.e. β is a string of
terminals and non-terminals.

Language of a CFG

- Given a CFG $G=(N, \Sigma, R, S)$:
 - Given a string $\alpha A \gamma$, where $A \in N$, we can derive $\alpha \beta \gamma$ if there is a production $A \rightarrow \beta \in R$.
 - $\alpha \Rightarrow \beta$ means that G can derive β from α in a single step.
 - $\alpha \Rightarrow^* \beta$ means that G can derive β from α in a finite number of steps.
 - The **language of G** is defined as the set of all terminal strings that can be derived from the start symbol.

$$L(G) = \{\beta \in T^*, \text{ s.t. } S \Rightarrow^* \beta\}$$

Derivations and Derived Strings

- CFG is a string rewriting formalism, so the **derived objects** are strings.
- A derivation is a sequence of rewriting steps.
- CFGs are **context free**: applicability of a rule depends only on the nonterminal symbol, not on its context.
 - Therefore, the order in which multiple non-terminals in a partially derived string are replaced does not matter.
We can represent identical derivations in a **derivation tree**.
- The derivation tree implies a parse tree.

Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Parse Tree:

NP

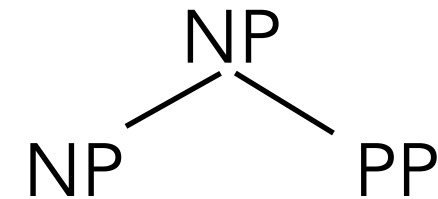
Derived String:

NP

Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Parse Tree:



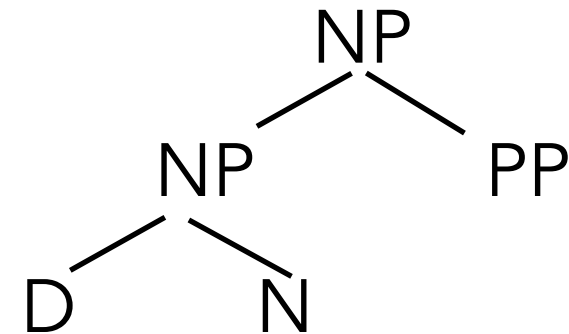
Derived String:

NP PP

Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Parse Tree:



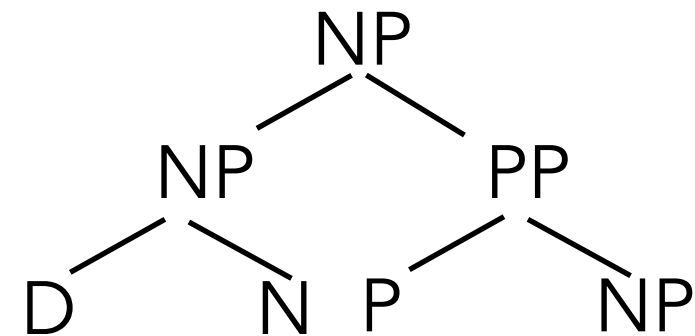
Derived String:

the student PP

Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Parse Tree:



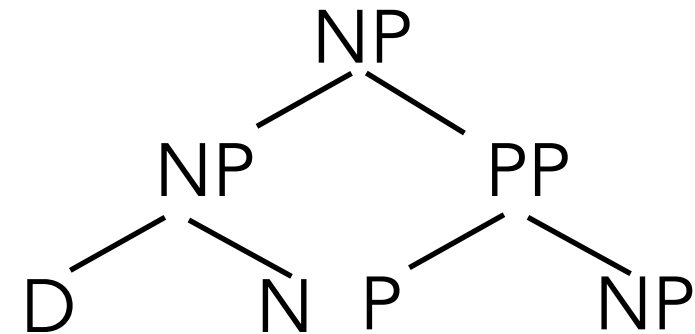
Derived String:

the student P NP

Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Parse Tree:



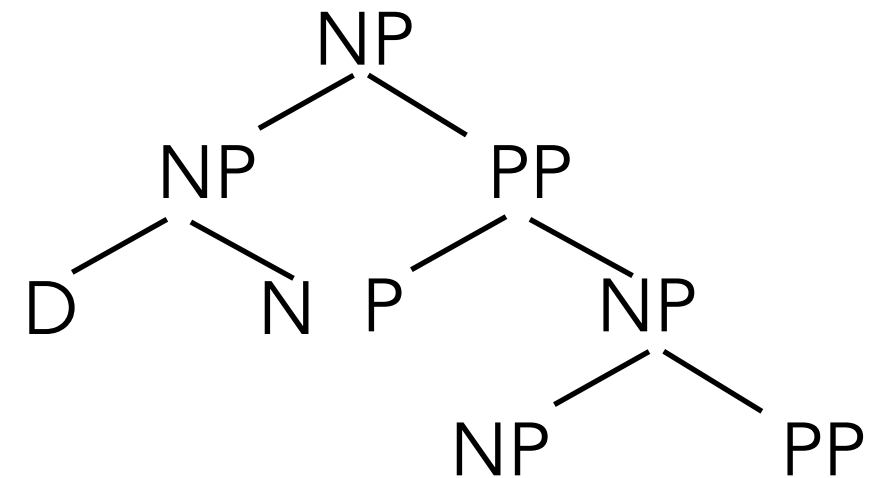
Derived String:

the student with NP

Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Parse Tree:



Derived String:

the student with NP PP

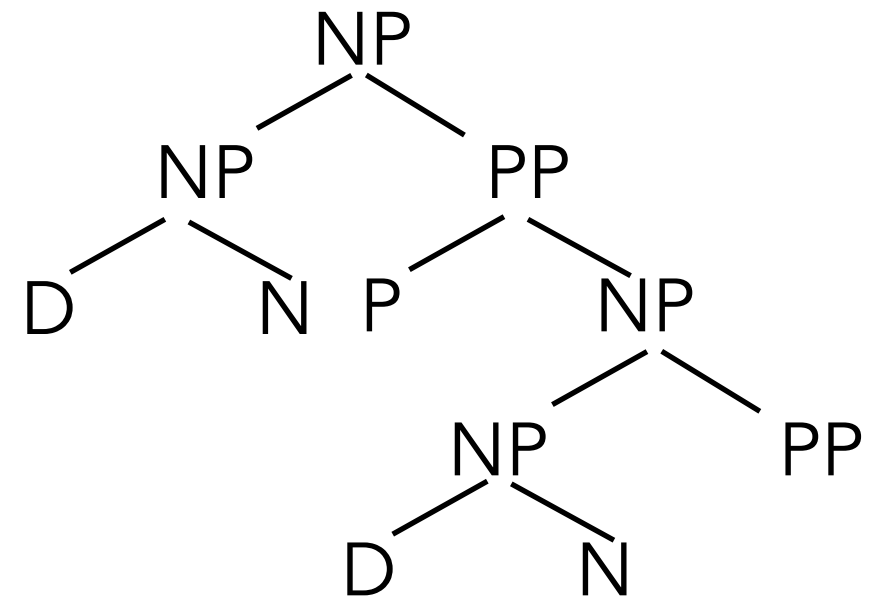
Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Derived String:

the student with the cat PP

Parse Tree:



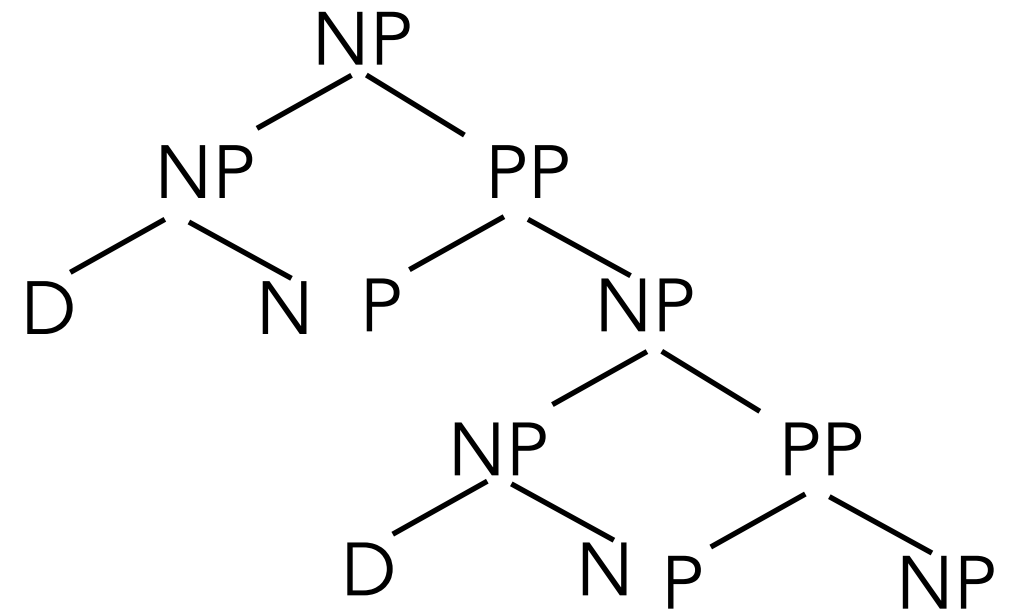
Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Derived String:

the student with the cat with NP

Parse Tree:



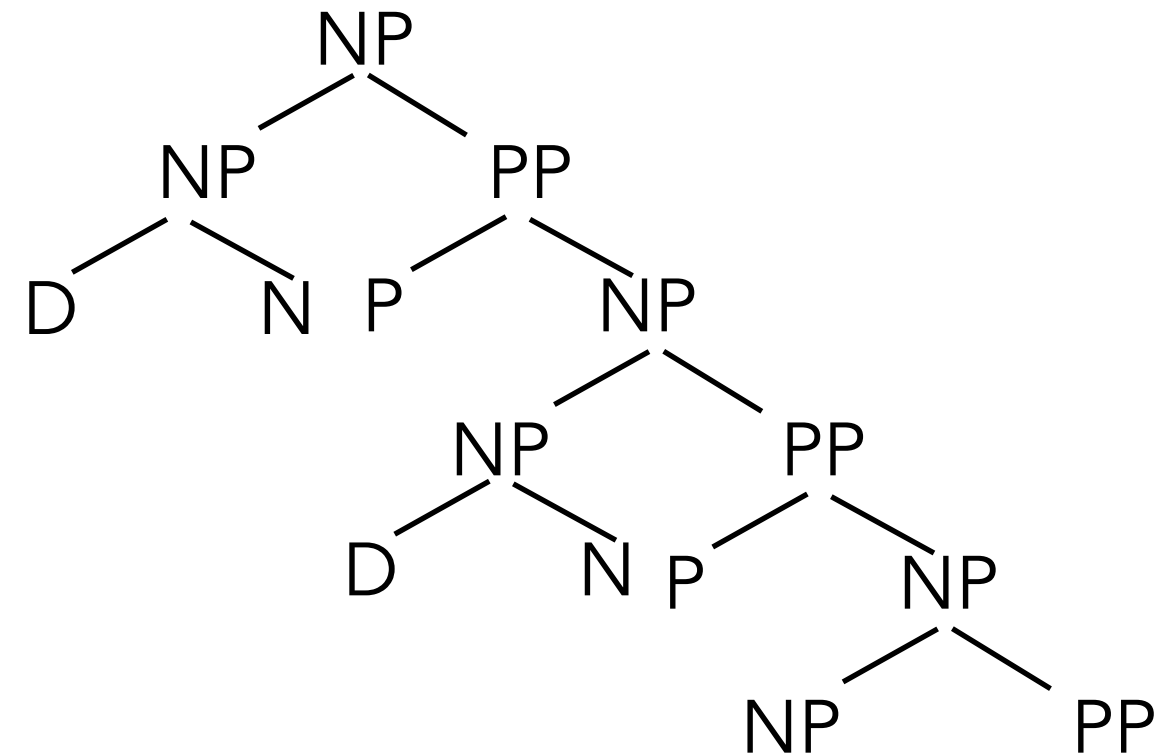
Recursion in CFGs

S	→	NP VP	V	→	saw
VP	→	V NP	P	→	with
VP	→	VP PP	D	→	the
PP	→	P NP	N	→	cat
NP	→	D N	N	→	tail
NP	→	NP PP	N	→	student

Derived String:

the student with the cat with NP PP

Parse Tree:



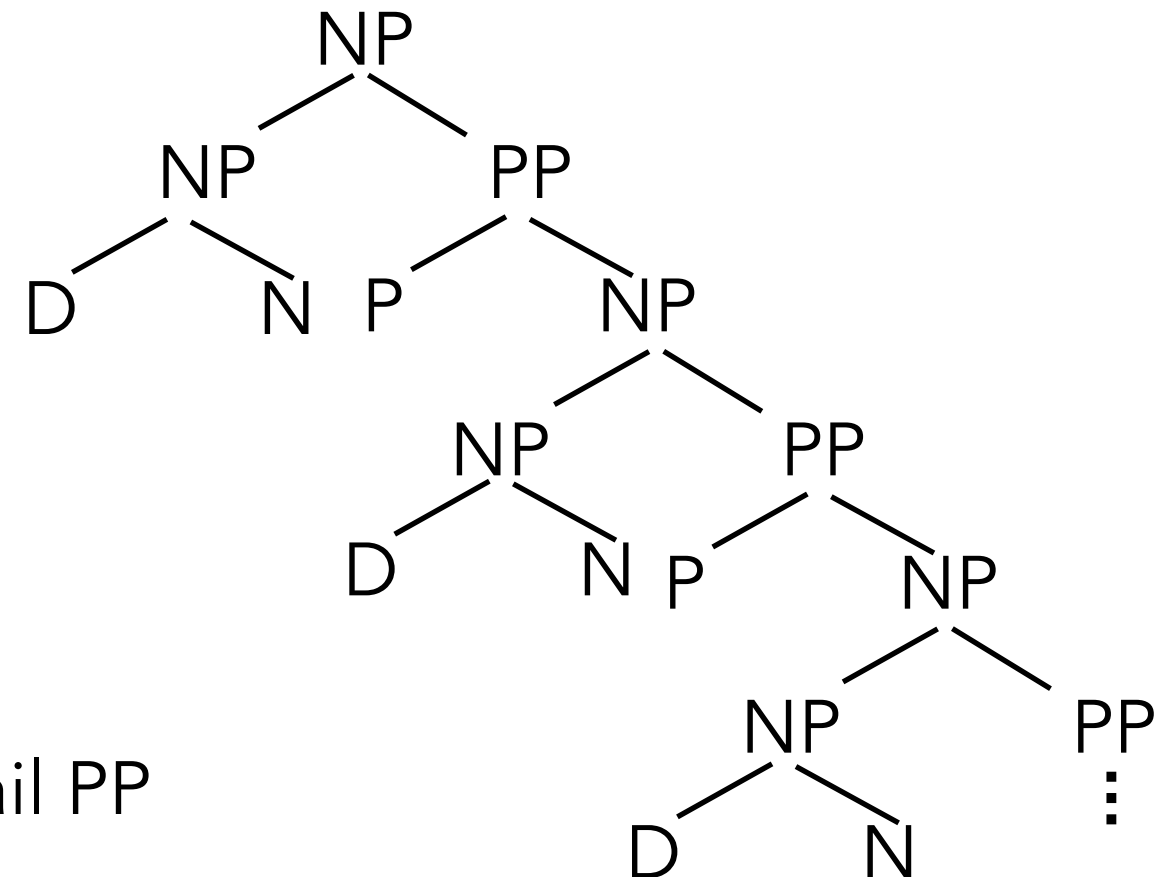
Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Derived String:

the student with the cat with the tail PP

Parse Tree:



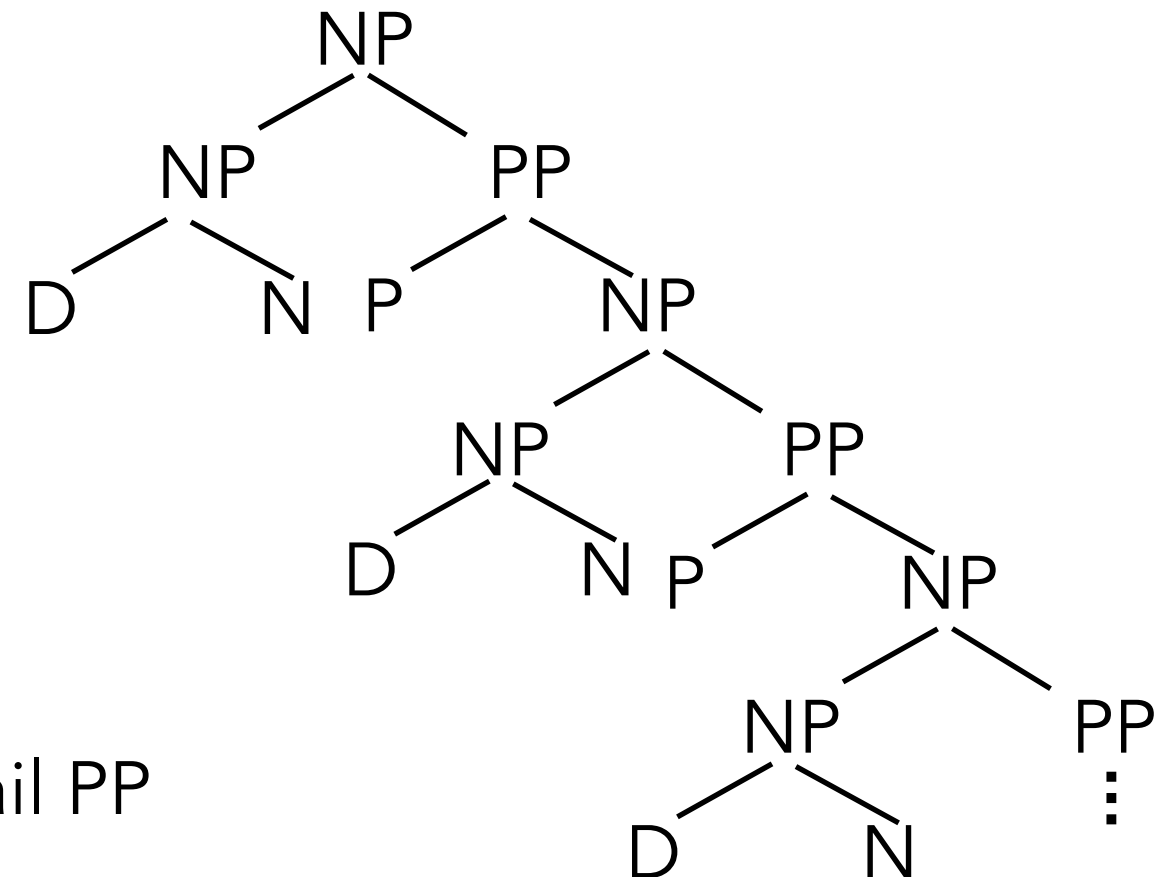
Recursion in CFGs

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

Derived String:

the student with the cat with the tail PP

Parse Tree:



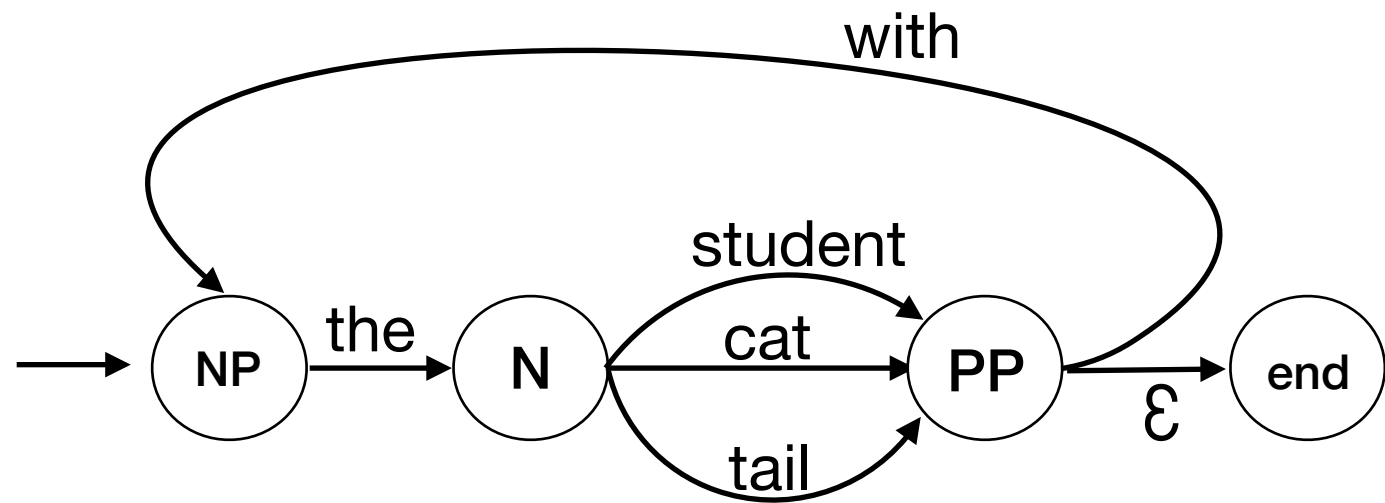
Regular Grammars

- A regular grammar is defined by:
 - Set of **terminal symbols** Σ .
 - Set of **non-terminal symbols** N .
 - A **start symbol** $S \in N$.
 - Set R of **productions** of the form $A \rightarrow aB$, or $A \rightarrow a$ where $A, B \in N$ and $a \in \Sigma$.

Finite State Automata

- Regular grammars can be implemented as finite state automata.

NP → the N
N → student PP
N → cat PP
N → tail PP
PP → with NP
PP → ϵ



- The set of all regular languages is strictly smaller than the set of context-free languages.

Regular Expressions

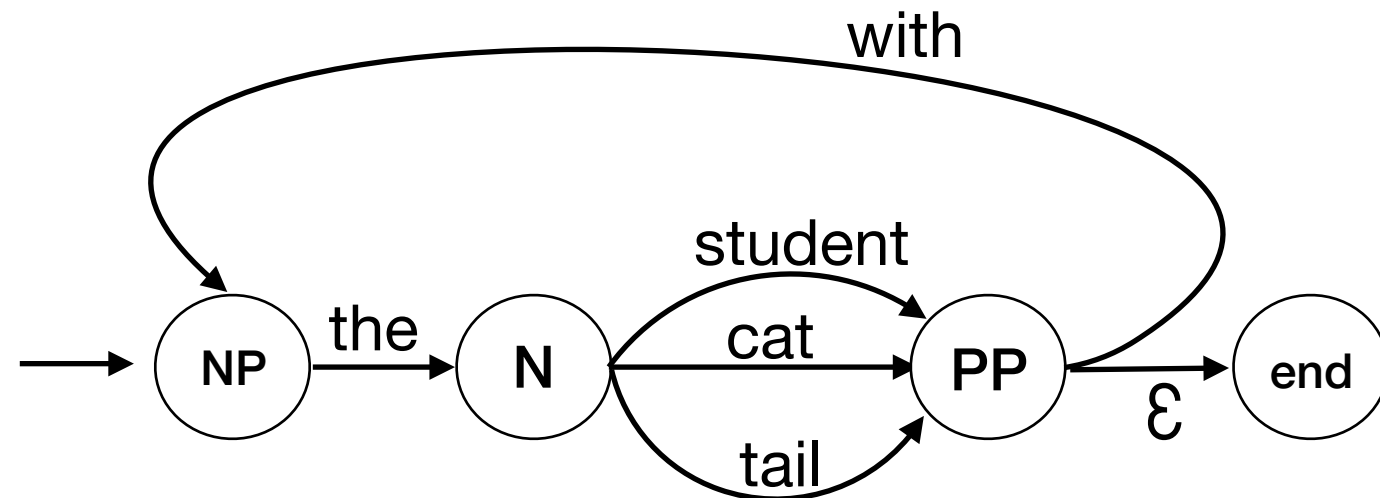
- Regular grammars can also be written as regular expressions. Recursive definition:
 - The empty string \mathcal{E} is a regular expression.
 - $a \in \Sigma$ is a regular expression.
 - if R is a regular expression R^* is a regular expression.
(Kleene star / repetition, “repeat R zero or more times”)
 - if R and S are regular expressions
 - RS is a regular expression. (concatenation, “ R followed by S ”)
 - $R|S$ is a regular expression. (altercation, “ R or S ”)

Used to define search expressions (in text editors etc.)

Also used in morphology: *invest(ed|s|or|ment)*

Regular Expression for Noun Phrases

NP \rightarrow the N
N \rightarrow student PP
N \rightarrow cat PP
N \rightarrow tail PP
PP \rightarrow with NP
PP \rightarrow ϵ

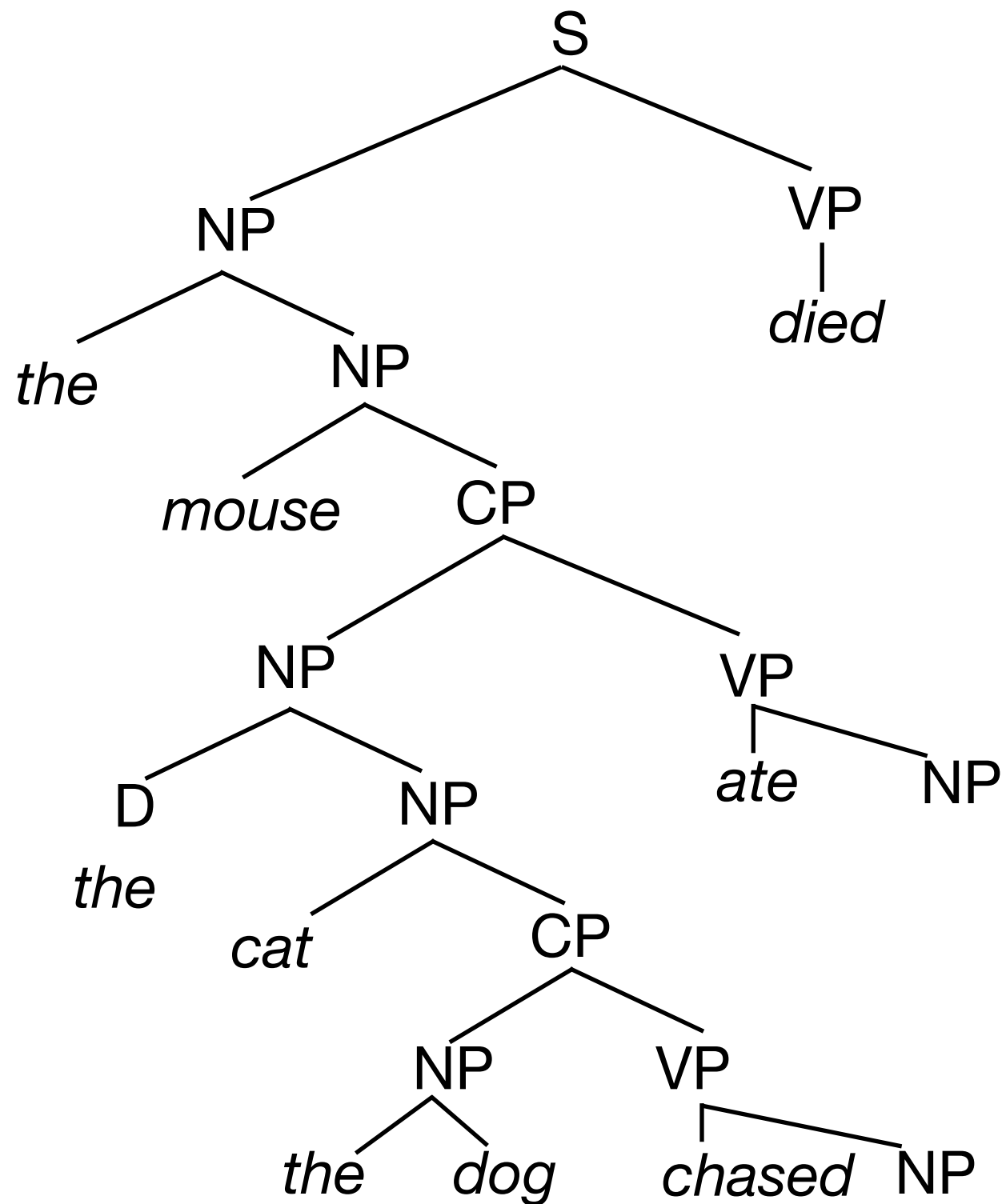


*the (student | cat | tail) (with the (student | cat | tail))**

Are natural languages (such as English, specifically) regular?
No! Example: Center embeddings.

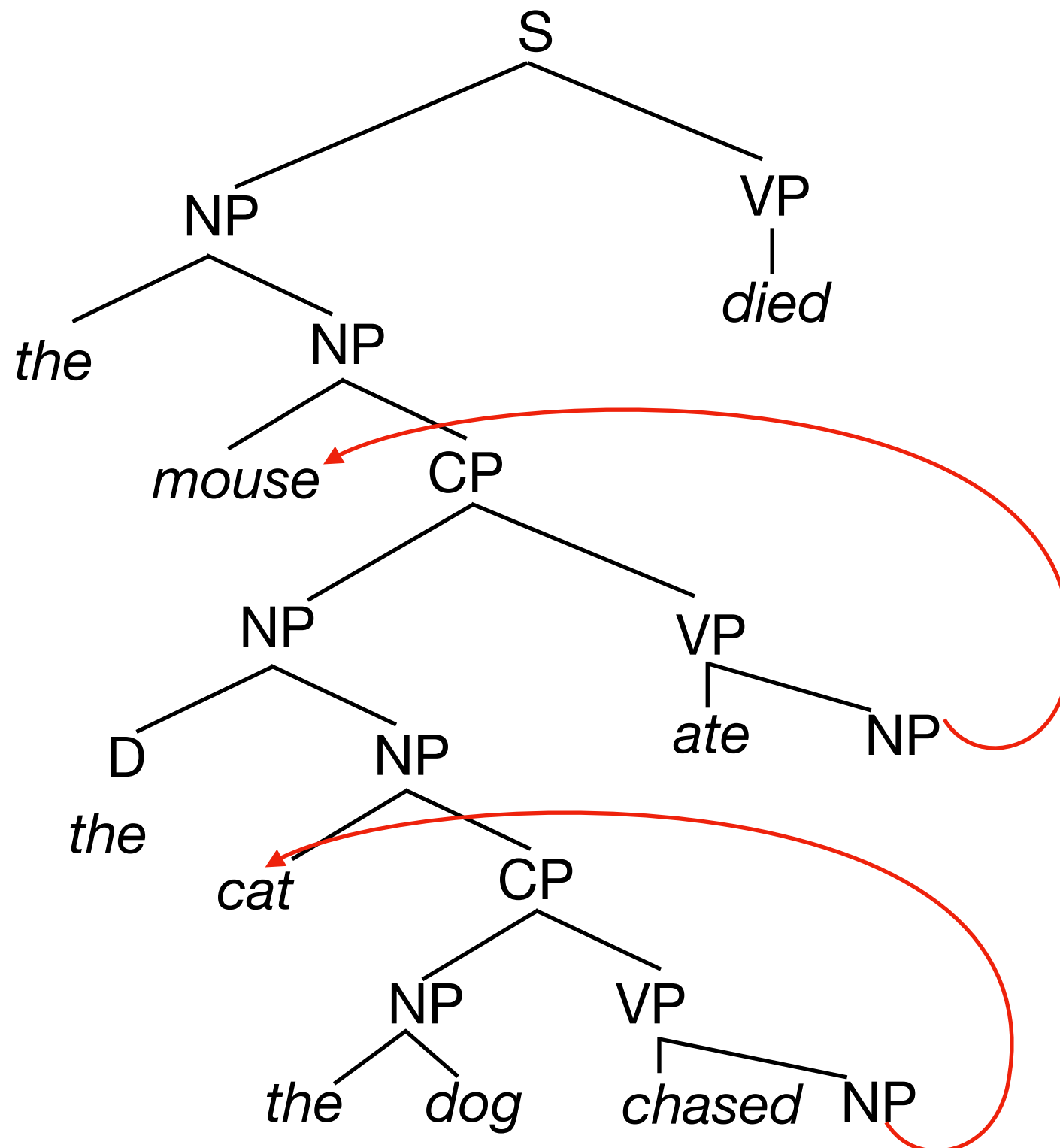
[The mouse [the cat [the dog chased]] ate] died.

Center Embeddings



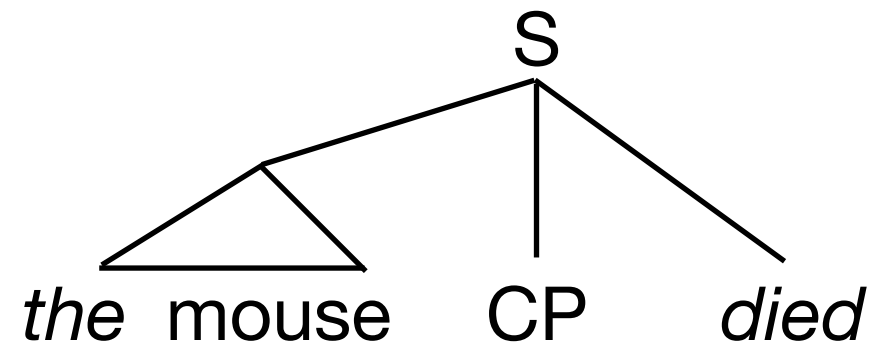
Linguistically, this is not a perfect analysis.

Center Embeddings



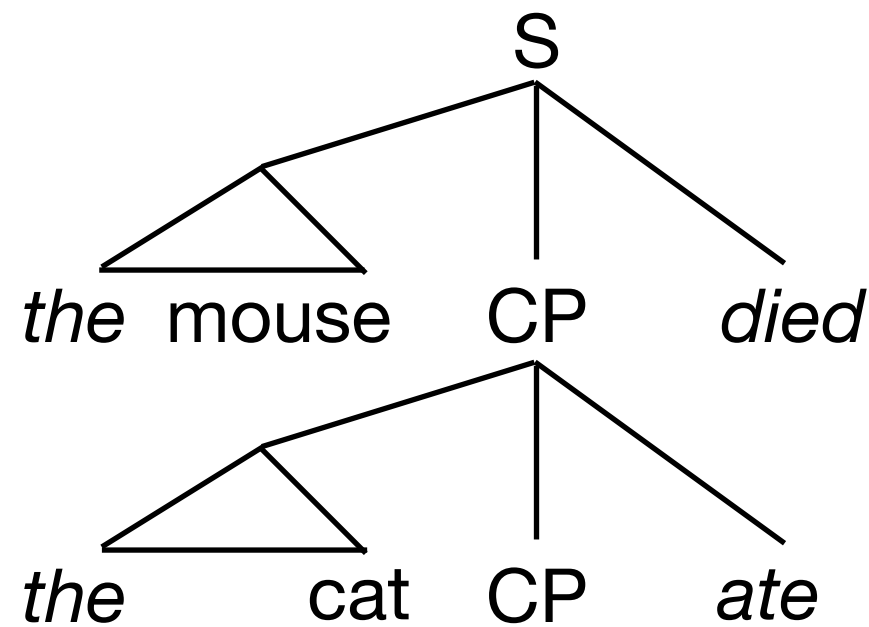
Linguistically, this is not a perfect analysis.

Center Embeddings



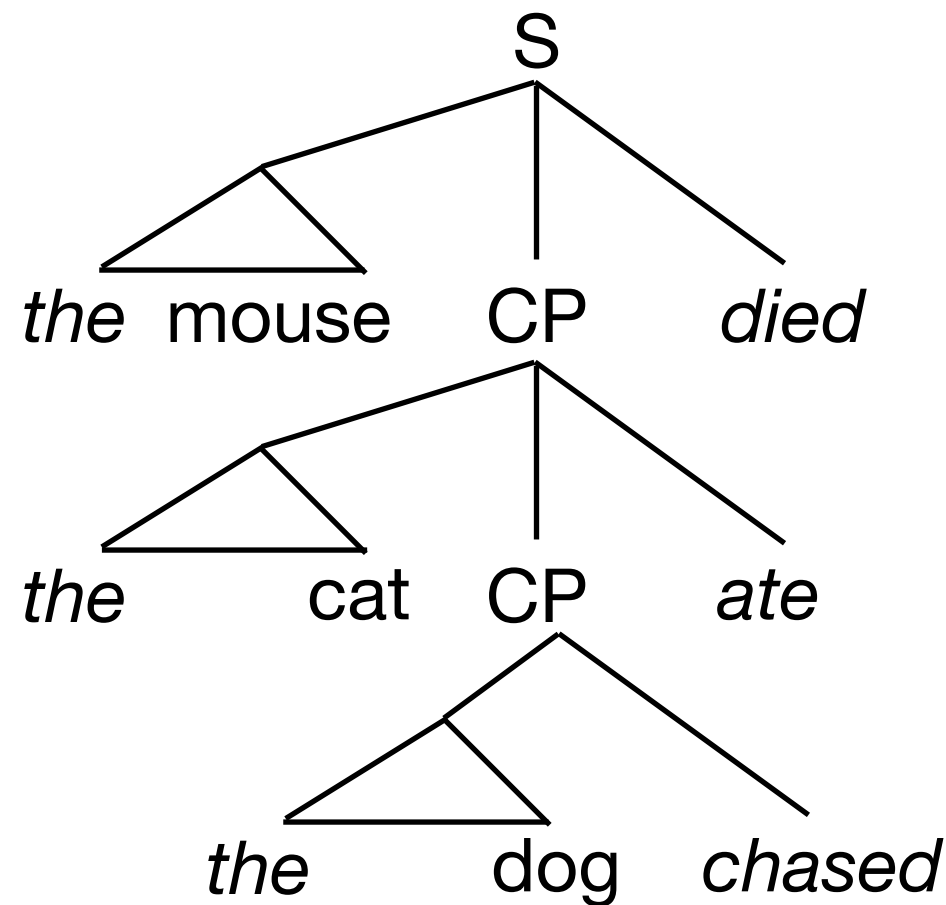
Linguistically, this is not a perfect analysis.

Center Embeddings



Linguistically, this is not a perfect analysis.

Center Embeddings

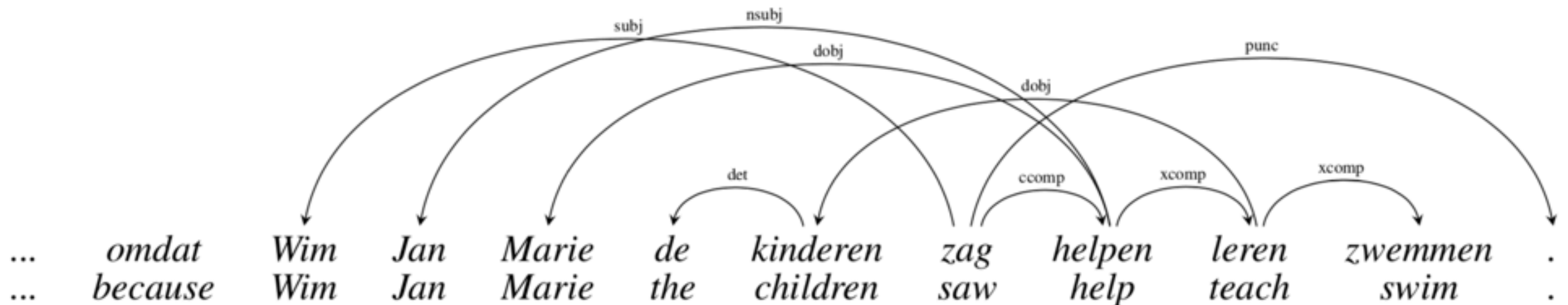


- Problem: Regular grammars cannot capture long-distance dependencies.
- This example follows the pattern **$a^n b^n$** .
Can show that is language is not context-free (using the “pumping lemma”).

Linguistically, this is not a perfect analysis.

Is Natural Language Context Free?

- Probably not. An example from Dutch:



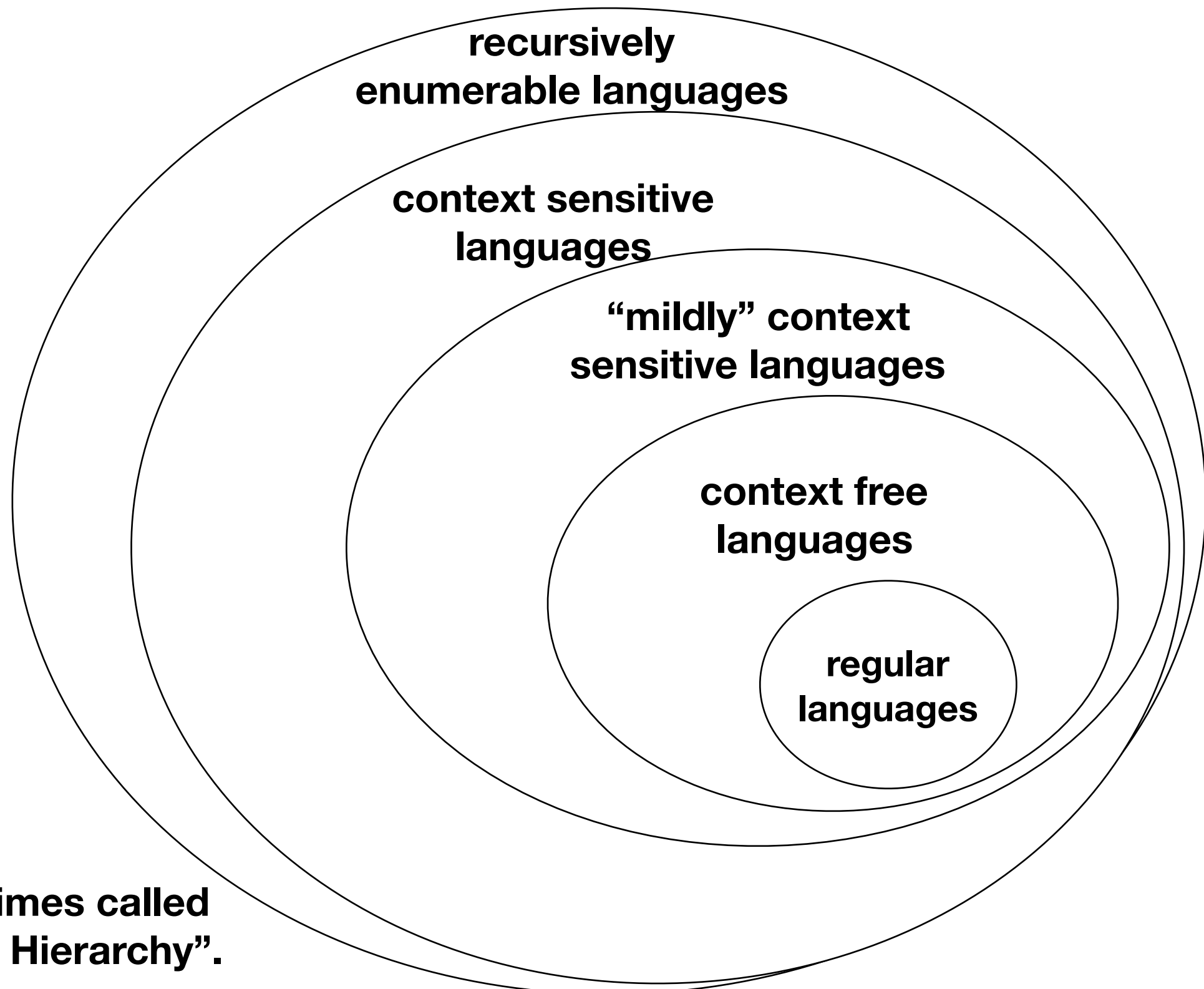
"...because Wim saw Jan help Marie teach the children to swim"

- Context Free Grammars cannot describe crossing dependencies. For example, it can be shown that

$a^n b^m c^n d^m$

is not a context free language.

Complexity Classes



**This is sometimes called
the “Chomsky Hierarchy”.**

Formal Grammar and Parsing

- Formal Grammars are used in linguistics, NLP, programming languages.
- We want to build a compact model that describes a complete language.
- Need efficient algorithms to determine if a sentence is in the language or not (**recognition problem**).
- We also want to recover the structure imposed by the grammar (**parsing problem**).

Two Approaches to Parsing

- Bottom-up: Start at the words (terminal symbols) and see which subtrees you can build. Then combine these subtrees into larger trees. (Driven by the input sentence.)
CKY algorithm - requires Grammars in Chomsky Normal Form.
- Top-down: Start at the start symbol (S), try to apply production rules that are compatible with the input. (Driven by the grammar - next week)
Earley algorithm
- Both approaches can be seen as a kind of search problem (next week).

Chomsky Normal Form

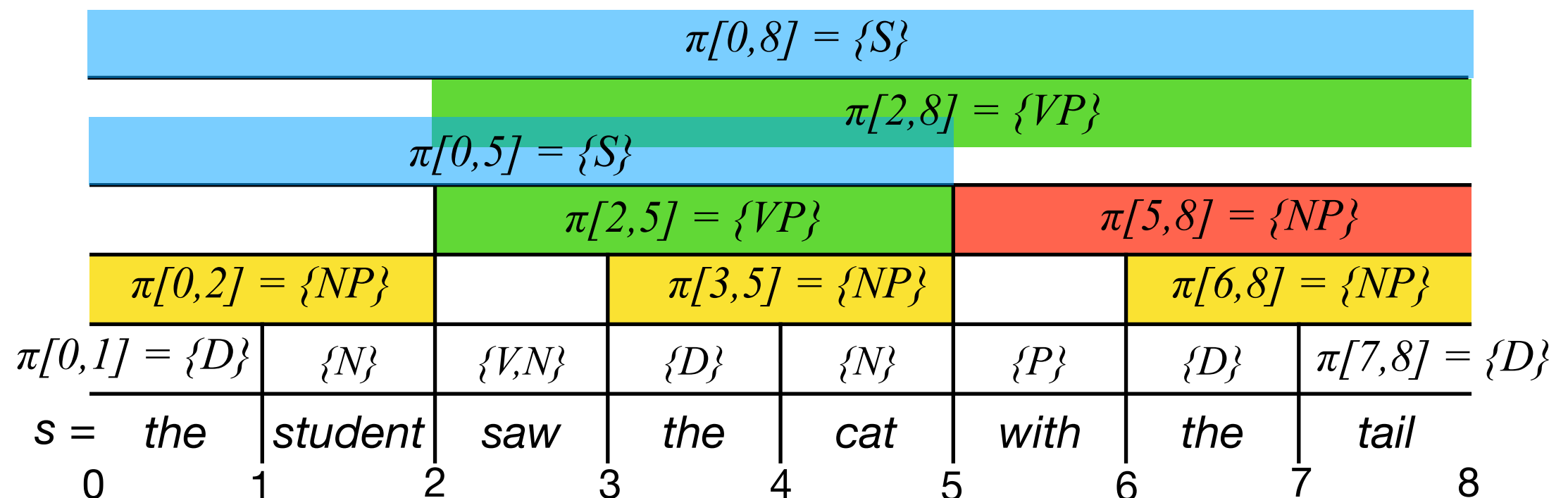
- A CFG $G=(N, \Sigma, R, S)$ is in Chomsky Normal Form (CNF) if the rules take one of the following forms:
 - $A \rightarrow B C$, where $A \in N, B \in N, C \in N$.
 - $A \rightarrow b$, where $A \in N, b \in \Sigma$.

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

Any CFG can be converted to an equivalent grammar in CNF that expresses the same language.

Cocke-Kasami-Younger (CKY) Algorithm - Motivation

- A nonterminal A covers a sub-span $[i,j]$ of the input string s if the rules in the grammar can derive $s[i,j]$ from A .
Let $\pi[i,j]$ be the set of nonterminals that cover $[i,j]$.
- The string is recognized by the grammar if $S \in \pi[i,j]$.
- Approach: Compute $\pi[i,j]$ for all sub-spans bottom-up, using dynamic-programming.



CKY Data Structure

- Use a 2-dimensional “parse table” to represent $\pi[i,j]$.

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

0 she 1 saw 2 the 3 cat 4 with 5 glasses

0		0,1	0,2	0,3	0,4	0,5	0,6
1			1,2	1,3	1,4	1,5	1,6
2				2,3	2,4	2,5	2,6
3					3,4	3,5	3,6
4						4,5	4,6
5							5,6
6							

CKY Initialization

- For $i=0 \dots \text{length}(s)-1$:
 $\pi[i, i+1] = \{A \mid A \rightarrow s[i:i+1] \in R\}$

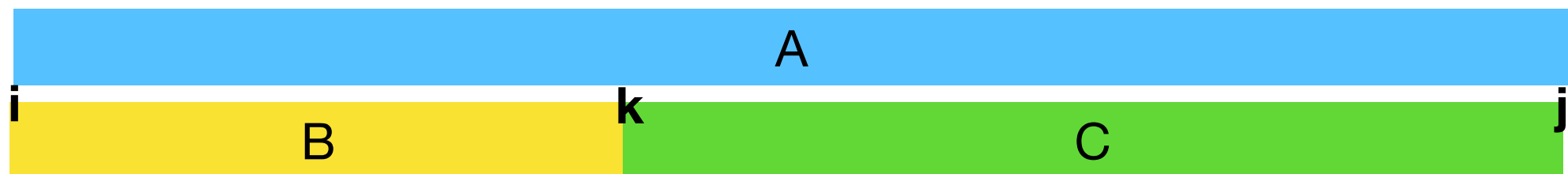
S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

0 she 1 saw 2 the 3 cat 4 with 5 glasses

0		NP	0,2	0,3	0,4	0,5	0,6
1			V	1,3	1,4	1,5	1,6
2				D	2,4	2,5	2,6
3					N	3,5	3,6
4						P	4,6
5							NP,N
6							

CKY - finding the split

- CKY requires grammar to be in CNF.
- Assume subspan $[i,j]$ is covered by nonterminal A .
- Then this nonterminal was recognized by some production of the form $A \rightarrow B C$, where $A \in N$, $B \in N$, $C \in N$ (grammar is in CNF).
- Span $[i,j]$ can be split into two parts:
 $[i,k]$, which is covered by B , and
 $[k,j]$ which is covered by C .



CKY - Recursive Definition

- To compute $\pi[i, j]$, try all possible split points k , such that $i < k < j$.
- For each k , check if the nonterminals in $\pi[i, k]$ and $\pi[k, j]$ match any of the rules in the grammar.
- Recursive definition for $\pi[i, j]$:

$$\pi[i, j] =$$

$$\bigcup_{k=i+1 \dots j-1} \{A \mid A \rightarrow B C \in R \text{ and } B \in \pi[i, k] \text{ and } C \in \pi[k, j]\}$$

CKY Full Algorithm

- **Input:** Grammar $G=(N, \Sigma, R, S)$, input string s of length n .
- for $i=0\dots n-1$: initialization
 $\pi[i, i+1] = \{A \mid A \rightarrow s[i] \}$
- for $length=2\dots n$: main loop
 for $i=0\dots(n-length)$:
 $j = i+length$
 for $k=i+1\dots j-1$:
 $M = \{A \mid A \rightarrow B C \in R \text{ and } B \in \pi[i, k] \text{ and } C \in \pi[k, j]\}$
 $\pi[i, j] = \pi[i, j] \cup M$
- if $S \in \pi[0, i+1]$ return True, otherwise False

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=2$

$i=0, k=1, j=2$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP		0,3	0,4	0,5	0,6
1			V	1,3	1,4	1,5	1,6
2				D	2,4	2,5	2,6
3					N	3,5	3,6
4						P	4,6
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=2$

$i=1, k=2, j=3$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP		0,3	0,4	0,5	0,6
1			V		1,4	1,5	1,6
2				D	2,4	2,5	2,6
3					N	3,5	3,6
4						P	4,6
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

$S \rightarrow NP VP$	$NP \rightarrow she$
$VP \rightarrow V NP$	$NP \rightarrow glasses$
$VP \rightarrow VP PP$	$D \rightarrow the$
$PP \rightarrow P NP$	$N \rightarrow cat$
$NP \rightarrow D N$	$N \rightarrow glasses$
$NP \rightarrow NP PP$	$V \rightarrow saw$
	$P \rightarrow with$

$length=2$

$i=2, k=3, j=4$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP		0,3	0,4	0,5	0,6
1			V		1,4	1,5	1,6
2				D	NP	2,5	2,6
3					N	3,5	3,6
4						P	4,6
5							NP,N
6							

CKY Algorithm

for $i=0 \dots (n-length)$:
 $j = i+length$
 for $k=i+1 \dots j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=2$

$i=3, k=4, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP		0,3	0,4	0,5	0,6
1			V		1,4	1,5	1,6
2				D	NP	2,5	2,6
3					N		3,6
4						P	4,6
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with

$length=2$

$i=4, k=5, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP		0,3	0,4	0,5	0,6
1			V		1,4	1,5	1,6
2				D	NP	2,5	2,6
3					N		3,6
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=3$

$i=0, k=1, j=3$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			0,4	0,5	0,6
1			V		1,4	1,5	1,6
2				D	NP	2,5	2,6
3					N		3,6
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0 \dots (n-length)$:
 $j = i+length$
 for $k=i+1 \dots j-1$:

....

$S \rightarrow NP VP$	$NP \rightarrow she$
$VP \rightarrow V NP$	$NP \rightarrow glasses$
$VP \rightarrow VP PP$	$D \rightarrow the$
$PP \rightarrow P NP$	$N \rightarrow cat$
$NP \rightarrow D N$	$N \rightarrow glasses$
$NP \rightarrow NP PP$	$V \rightarrow saw$
	$P \rightarrow with$

$length=3$

$i=0, k=2, j=3$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP	→		0,4	0,5	0,6
1			V		1,4	1,5	1,6
2				D	NP	2,5	2,6
3					N		3,6
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with

$length=3$

$i=1, k=2, j=4$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			0,4	0,5	0,6
1			V		VP	1,5	1,6
2				D	NP	2,5	2,6
3					N		3,6
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=3$

$i=1, k=3, j=4$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			0,4	0,5	0,6
1			V	→	VP	1,5	1,6
2					D	NP	2,5
3						N	3,6
4							P
5							PP
6							NP,N

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=3$

$i=2, k=3, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			0,4	0,5	0,6
1			V		VP	1,5	1,6
2				D	NP		2,6
3					N		3,6
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=3$

$i=2, k=4, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			0,4	0,5	0,6
1			V		VP	1,5	1,6
2				D	NP		2,6
3					N		3,6
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=3$

$i=3, k=4, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			0,4	0,5	0,6
1			V		VP	1,5	1,6
2				D	NP		2,6
3					N		
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=3$

$i=3, k=5, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			0,4	0,5	0,6
1			V		VP	1,5	1,6
2				D	NP		2,6
3					N		
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with

$length=4$

$i=0, k=1, j=4$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0	NP			S	0,5	0,6
1		V		VP	1,5	1,6
2			D	NP		2,6
3				N		
4					P	PP
5						NP,N
6						

CKY Algorithm

for $i=0 \dots (n-length)$:
 $j = i+length$
 for $k=i+1 \dots j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=4$

$i=0, k=2, j=4$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			S	0,5	0,6
1			V		VP	1,5	1,6
2				D	NP		2,6
3					N		
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with

$length=4$

$i=0, k=3, j=4$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0	NP		→	S	0,5	0,6
1		V		VP	1,5	1,6
2			D	NP		2,6
3				N		
4					P	PP
5						NP,N
6						

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=4$

$i=1, k=2, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0	NP			S	0,5	0,6
1		V		VP		1,6
2			D	NP		2,6
3				N		
4					P	PP
5						NP,N
6						

CKY Algorithm

for $i=0...(n-length)$:

$j = i+length$

for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=4$

$i=1, k=3, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0		NP			S	0,5	0,6
1			V		VP		1,6
2				D	NP		2,6
3					N		
4						P	PP
5							NP,N
6							

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

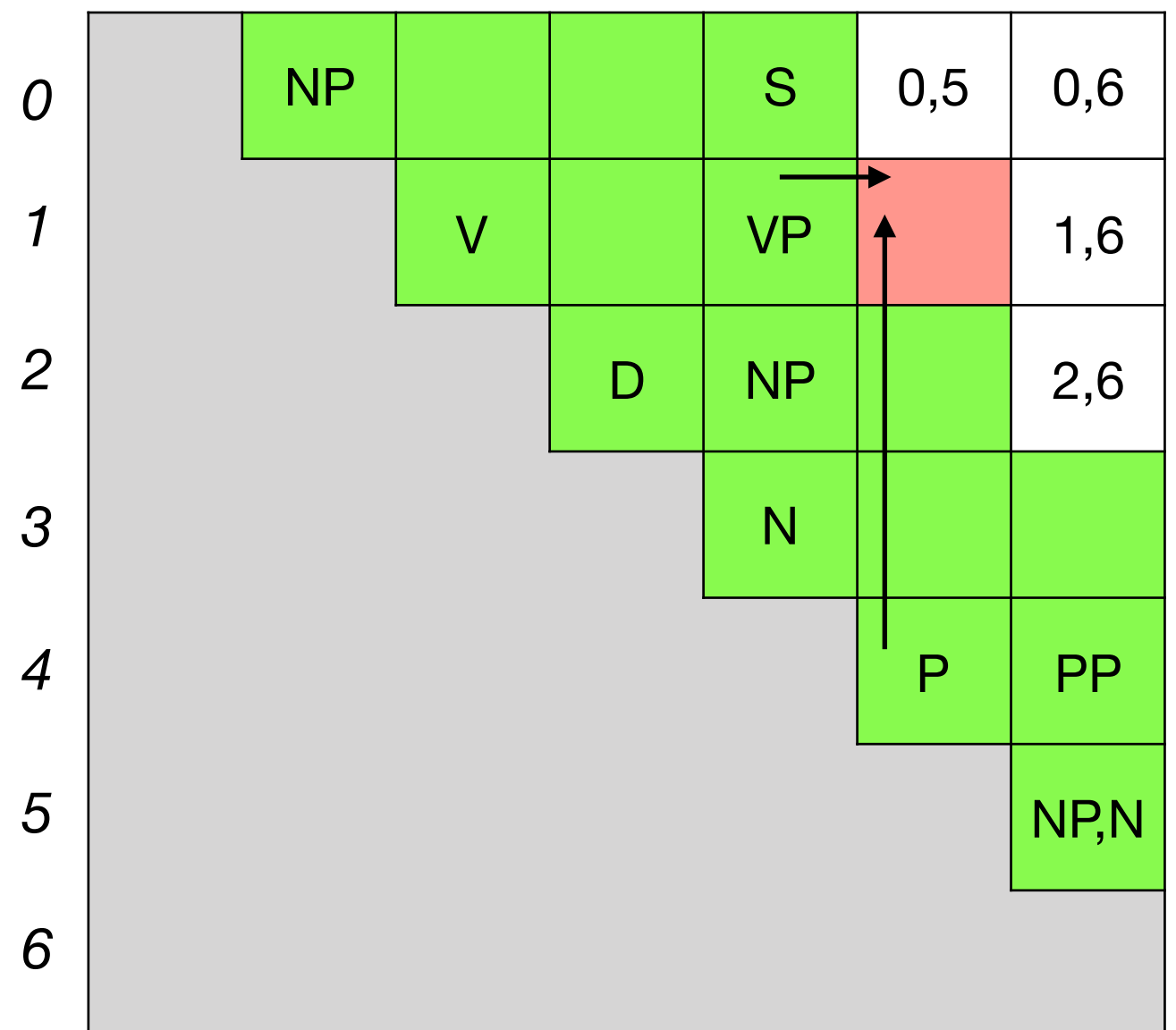
....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=4$

$i=1, k=4, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

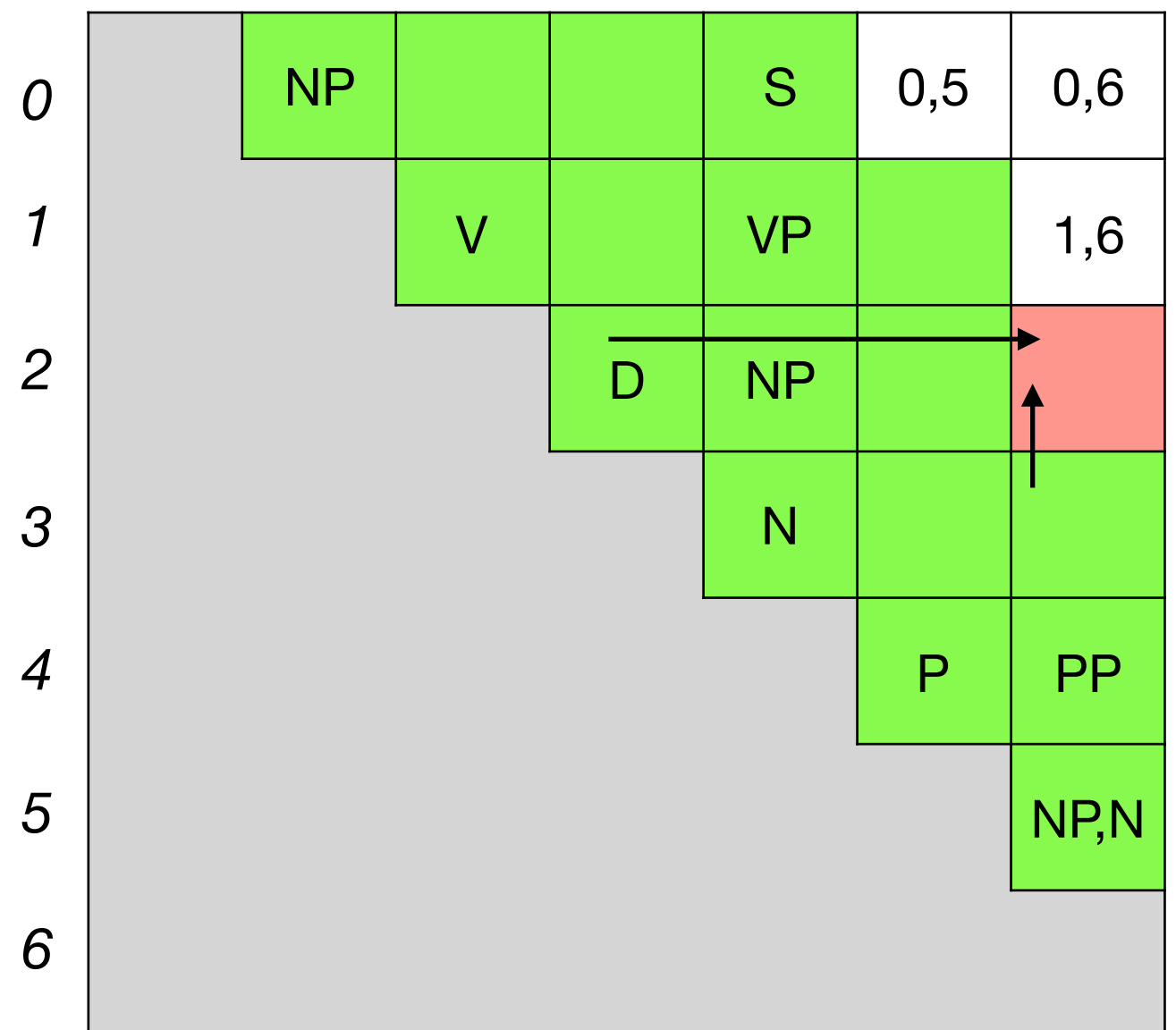
....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=4$

$i=2, k=3, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

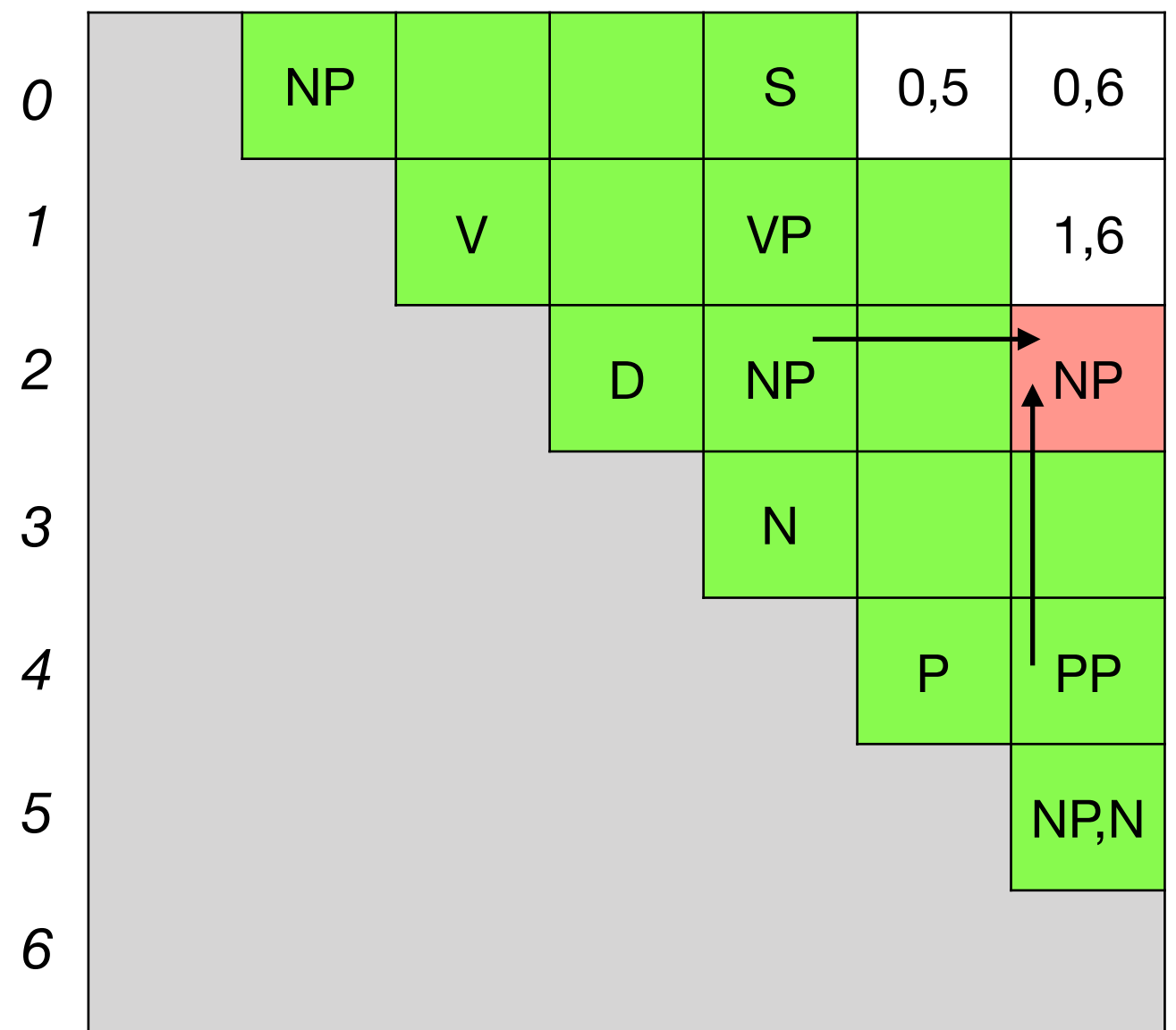
....

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with

$length=4$

$i=2, k=4, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

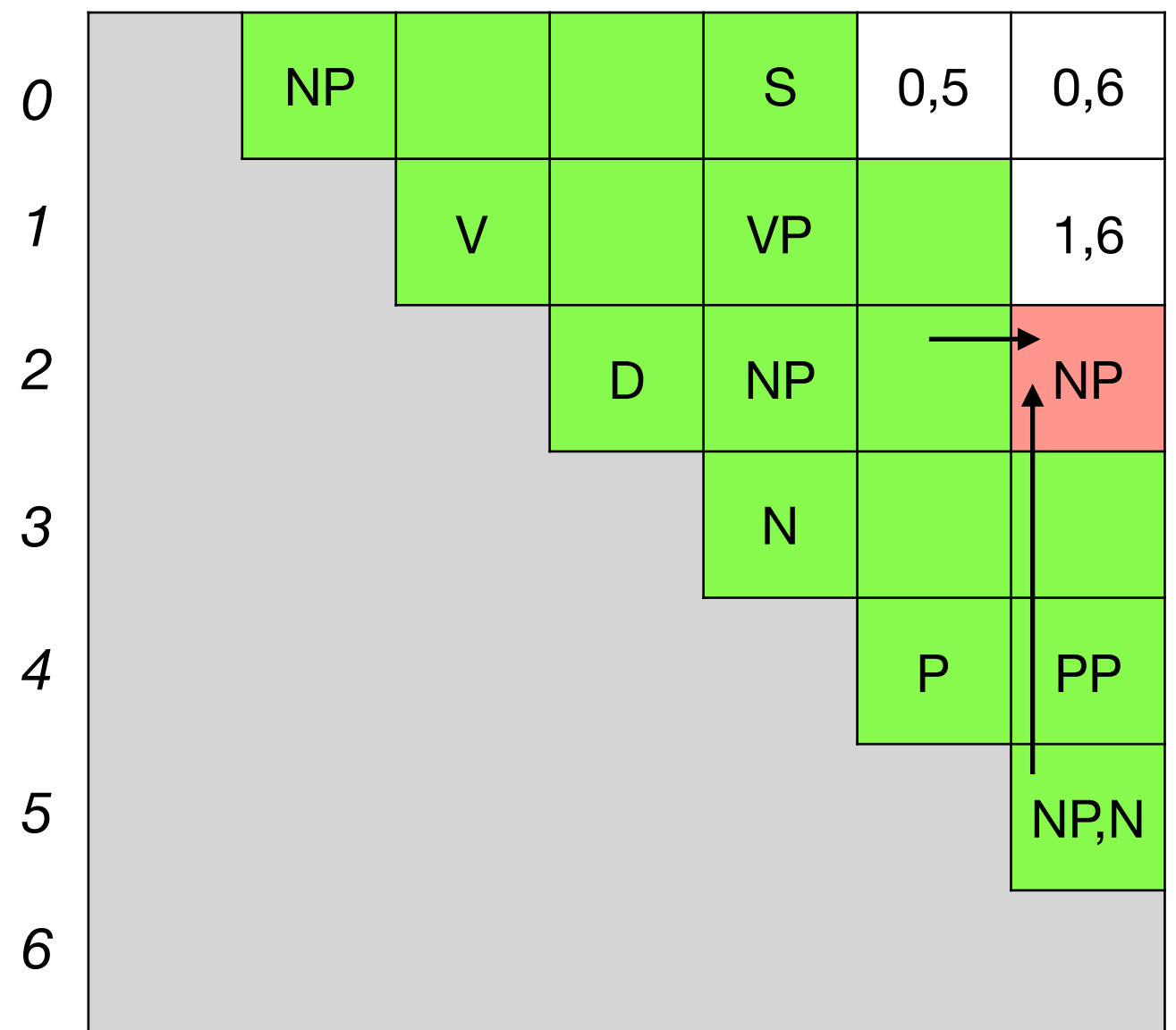
....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=4$

$i=2, k=5, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=5$

$i=0, k=1, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0	NP			S		0,6
1		V		VP		1,6
2			D	NP		NP
3				N		
4					P	PP
5						NP,N
6						

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=5$

$i=0, k=2, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

0	NP			S		0,6
1		V		VP		1,6
2			D	NP		NP
3				N		
4					P	PP
5						NP,N
6						

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

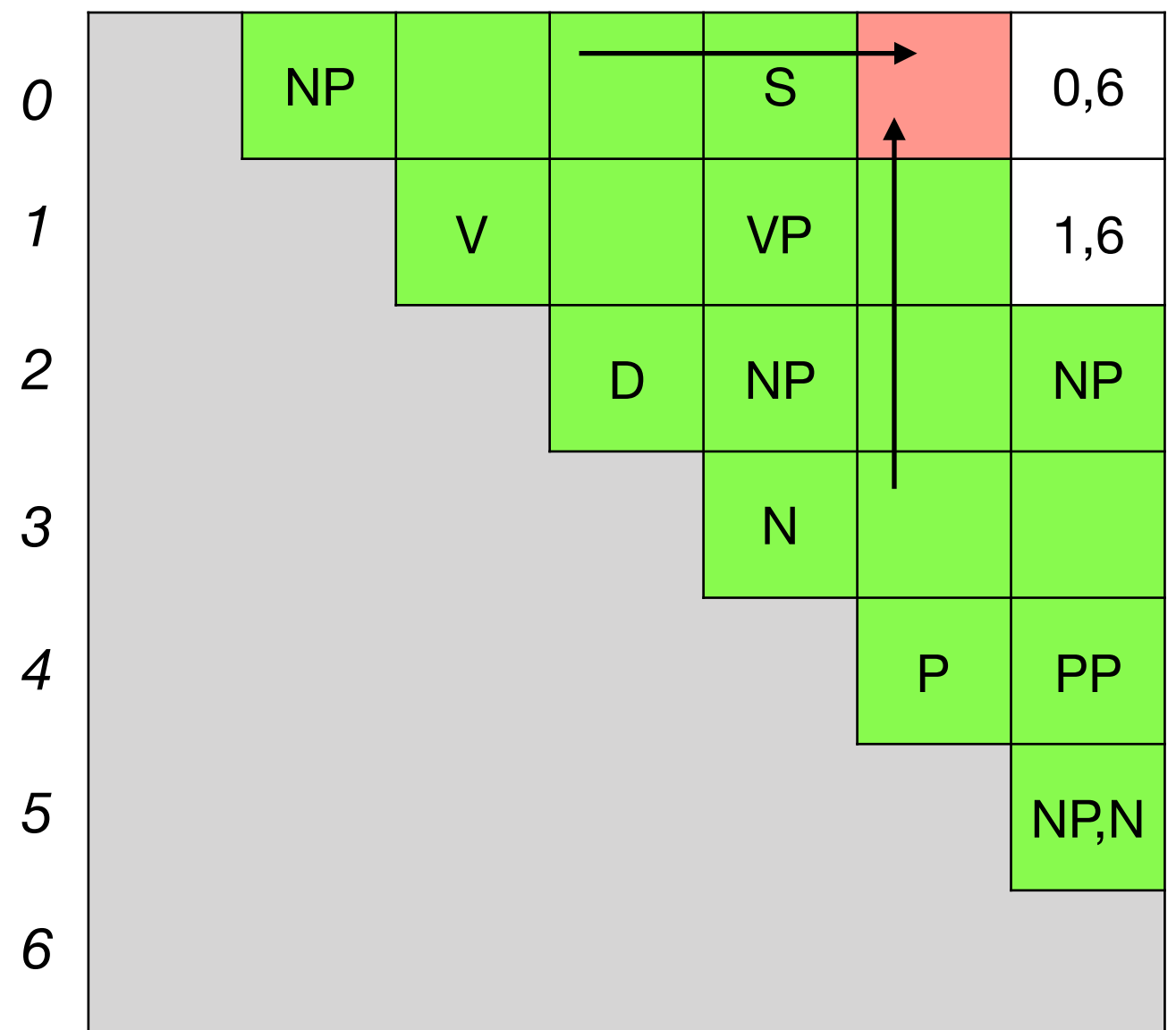
....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=5$

$i=0, k=3, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

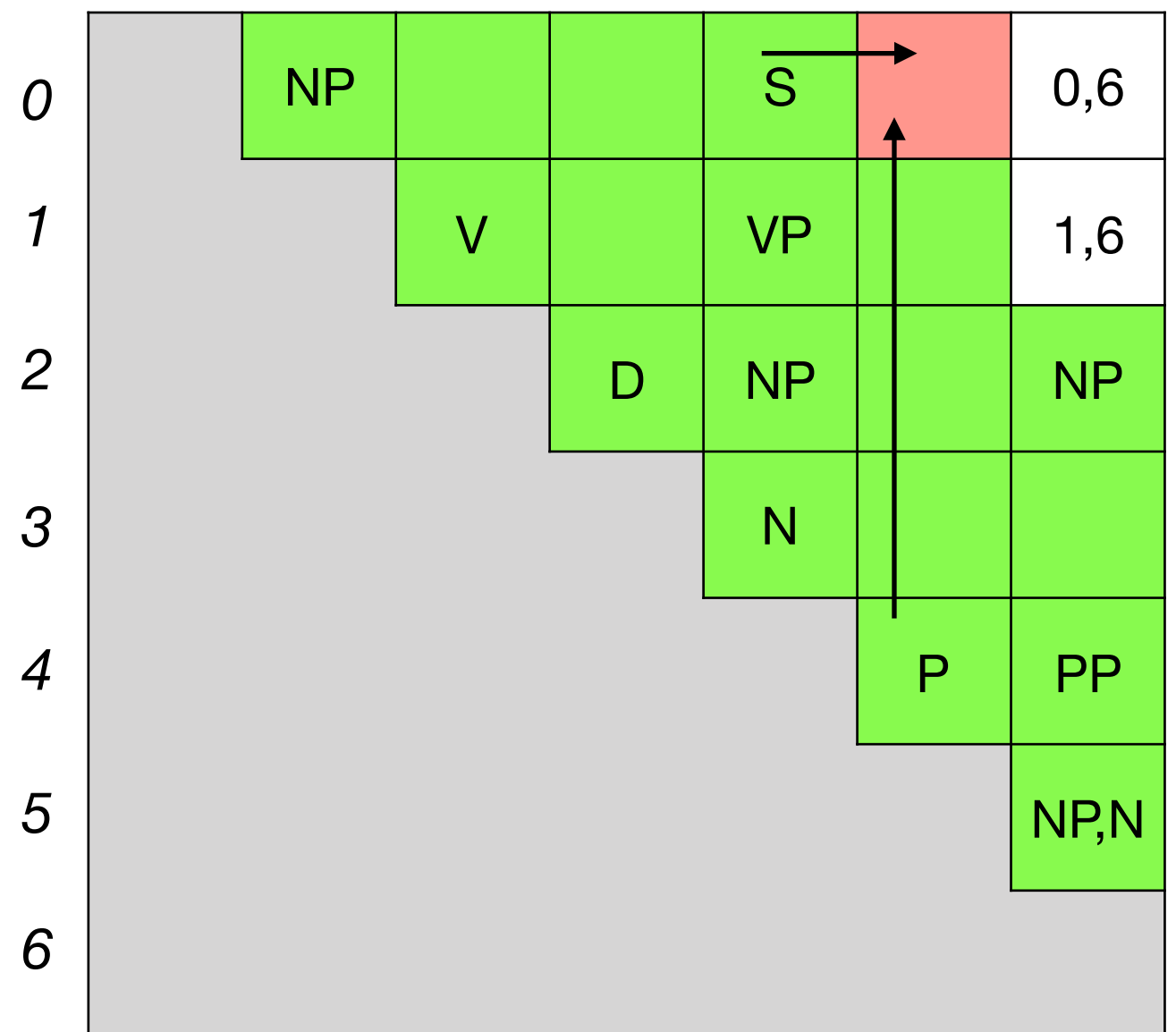
....

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with

$length=5$

$i=0, k=4, j=5$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

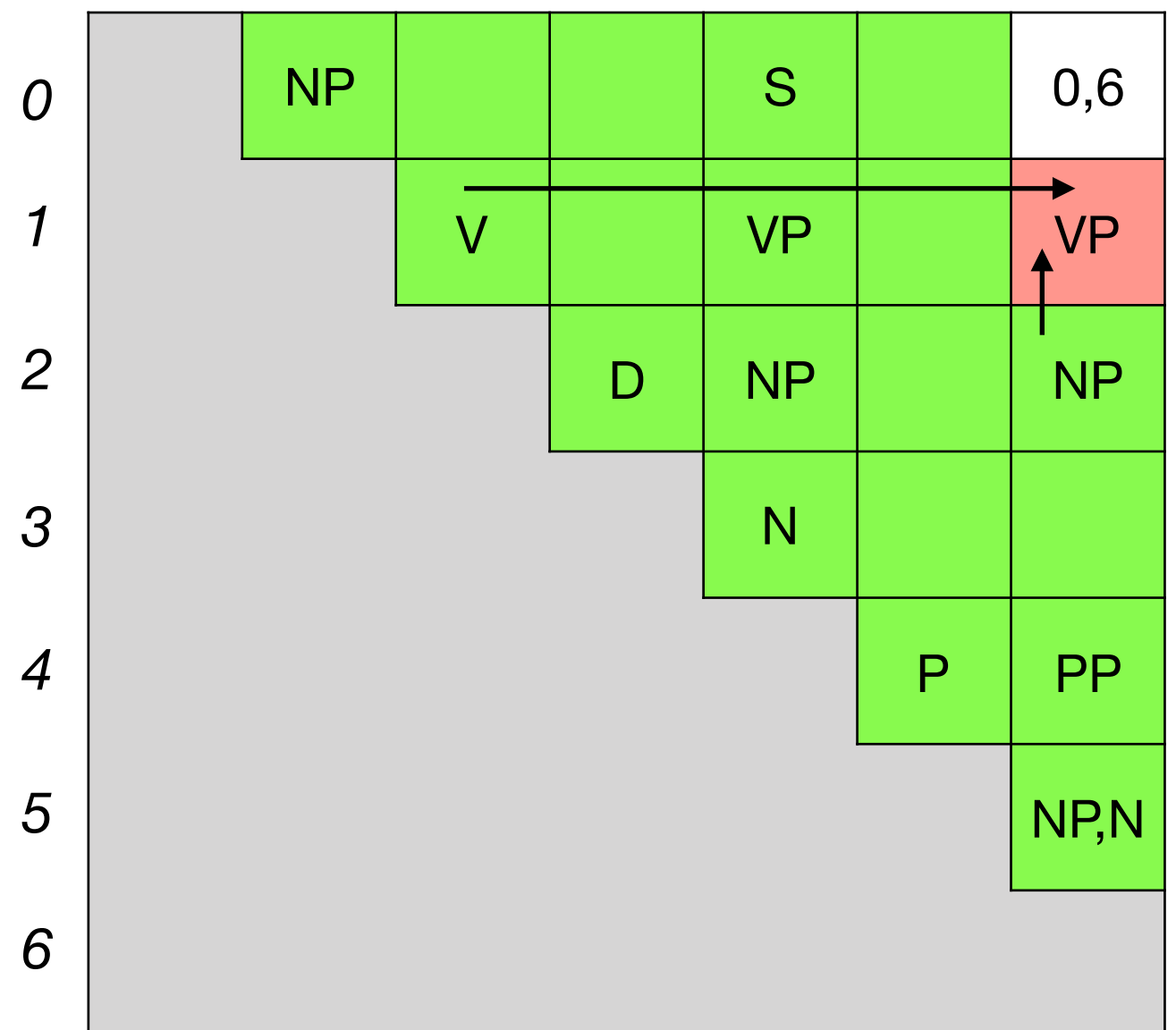
....

$S \rightarrow NP VP$	$NP \rightarrow she$
$VP \rightarrow V NP$	$NP \rightarrow glasses$
$VP \rightarrow VP PP$	$D \rightarrow the$
$PP \rightarrow P NP$	$N \rightarrow cat$
$NP \rightarrow D N$	$N \rightarrow glasses$
$NP \rightarrow NP PP$	$V \rightarrow saw$
	$P \rightarrow with$

$length=5$

$i=1, k=2, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

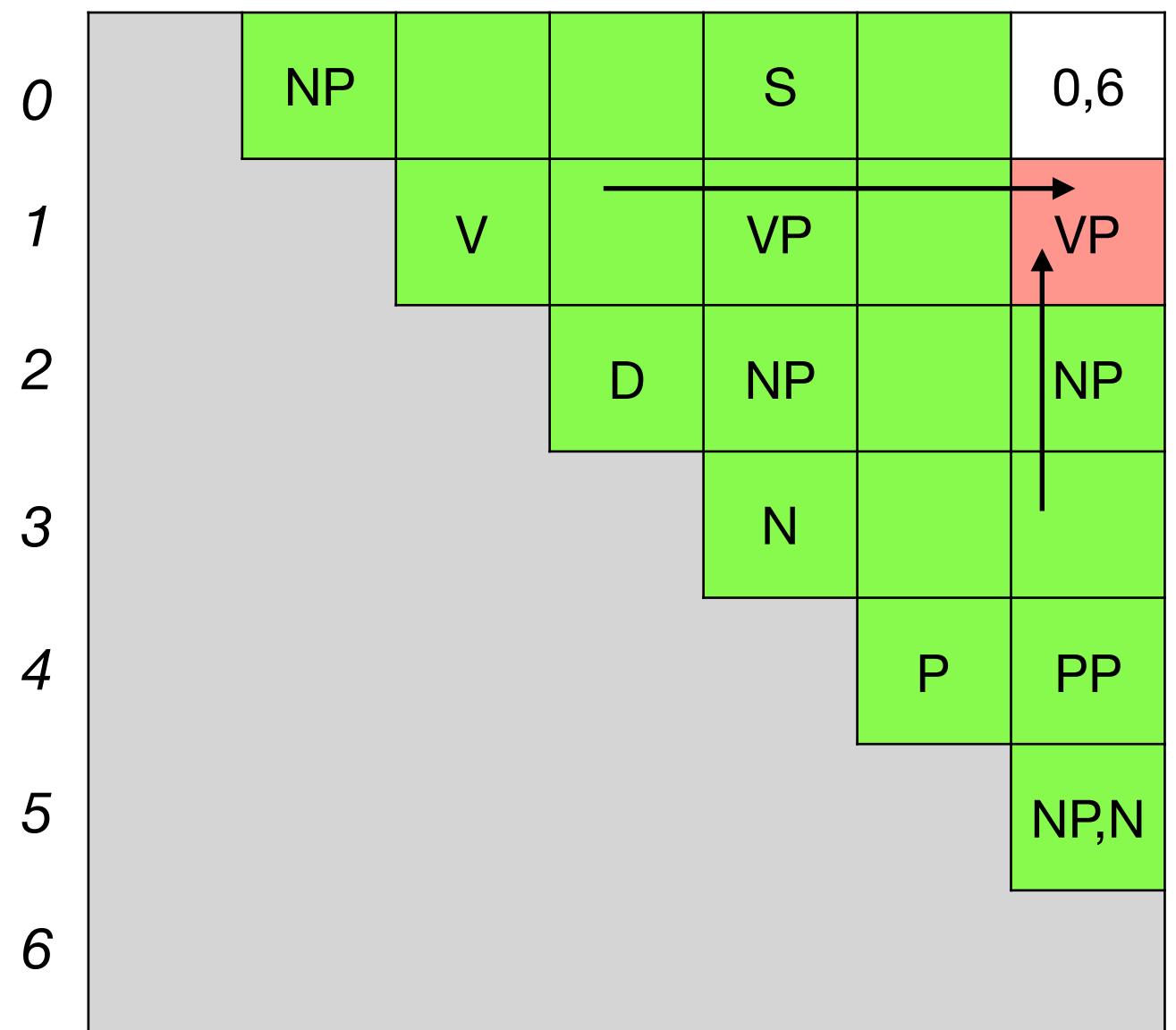
....

$S \rightarrow NP VP$	$NP \rightarrow she$
$VP \rightarrow V NP$	$NP \rightarrow glasses$
$VP \rightarrow VP PP$	$D \rightarrow the$
$PP \rightarrow P NP$	$N \rightarrow cat$
$NP \rightarrow D N$	$N \rightarrow glasses$
$NP \rightarrow NP PP$	$V \rightarrow saw$
	$P \rightarrow with$

$length=5$

$i=1, k=3, j=6$

$_0 she \ _1 saw \ _2 the \ _3 cat \ _4 with \ _5 glasses$



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

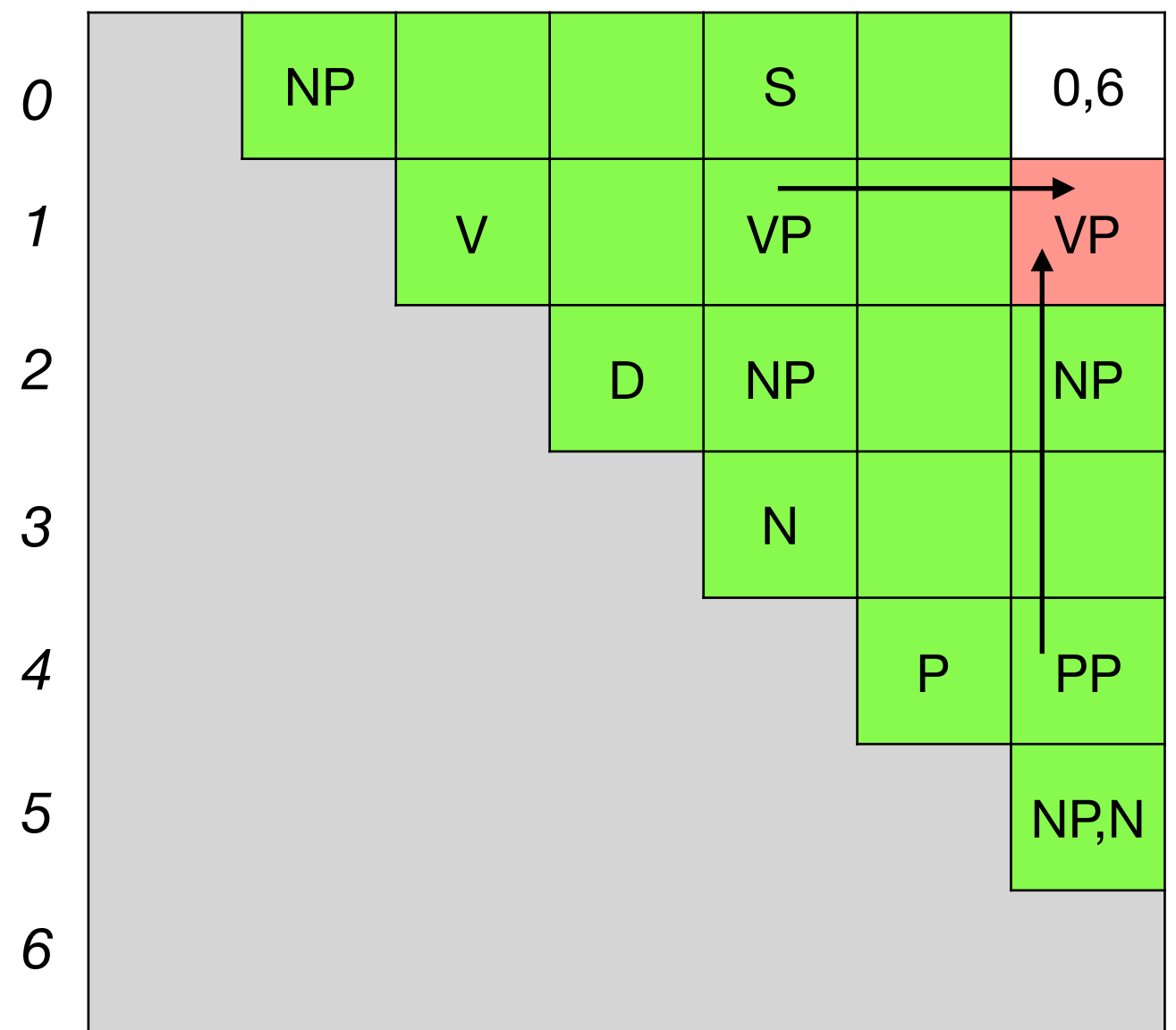
....

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with

$length=5$

$i=1, k=4, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



! We can build VP over [1,6] in two ways!

CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

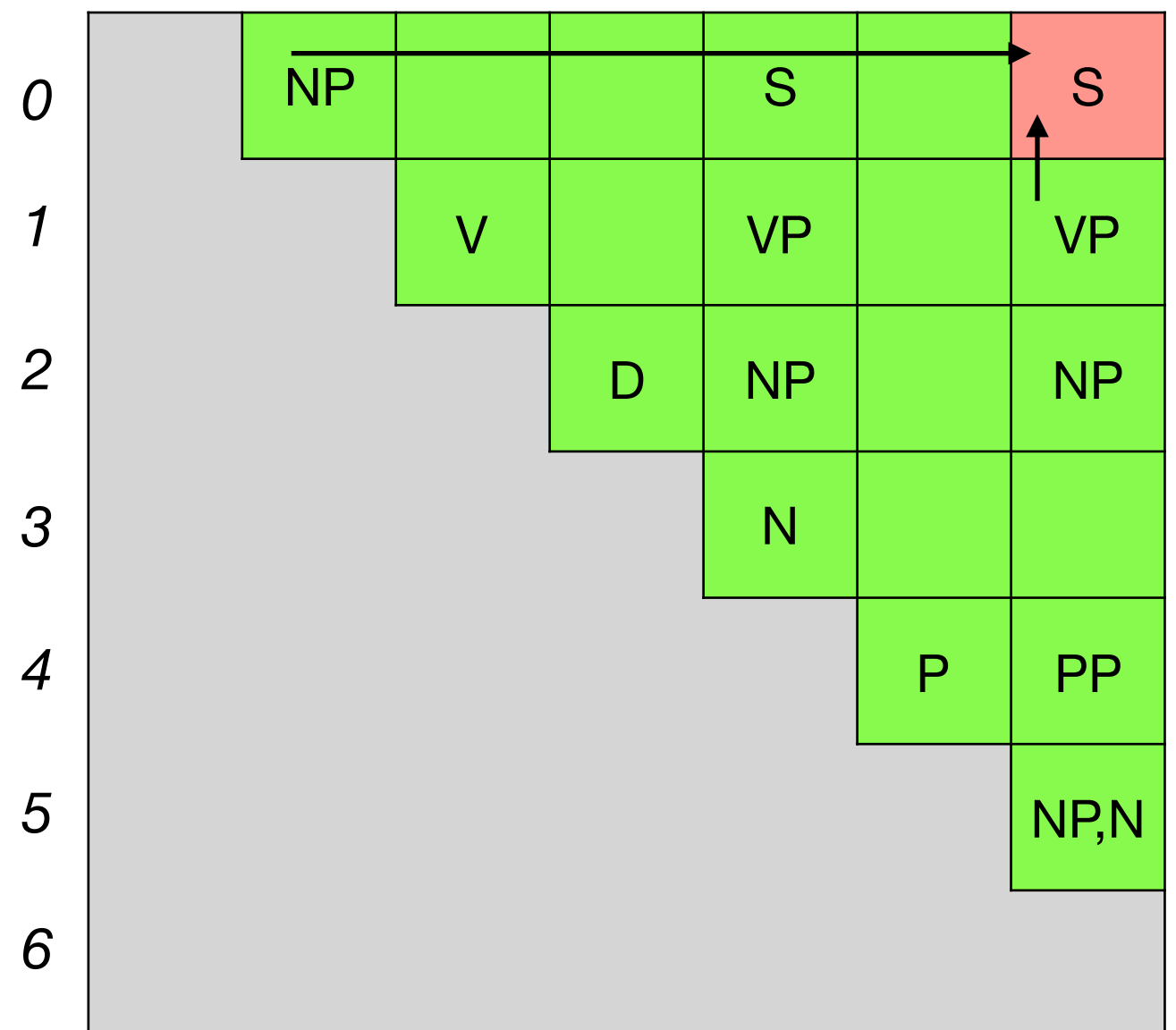
....

$S \rightarrow NP VP$	$NP \rightarrow she$
$VP \rightarrow V NP$	$NP \rightarrow glasses$
$VP \rightarrow VP PP$	$D \rightarrow the$
$PP \rightarrow P NP$	$N \rightarrow cat$
$NP \rightarrow D N$	$N \rightarrow glasses$
$NP \rightarrow NP PP$	$V \rightarrow saw$
	$P \rightarrow with$

$length=5$

$i=0, k=1, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:

$j = i+length$

for $k=i+1...j-1$:

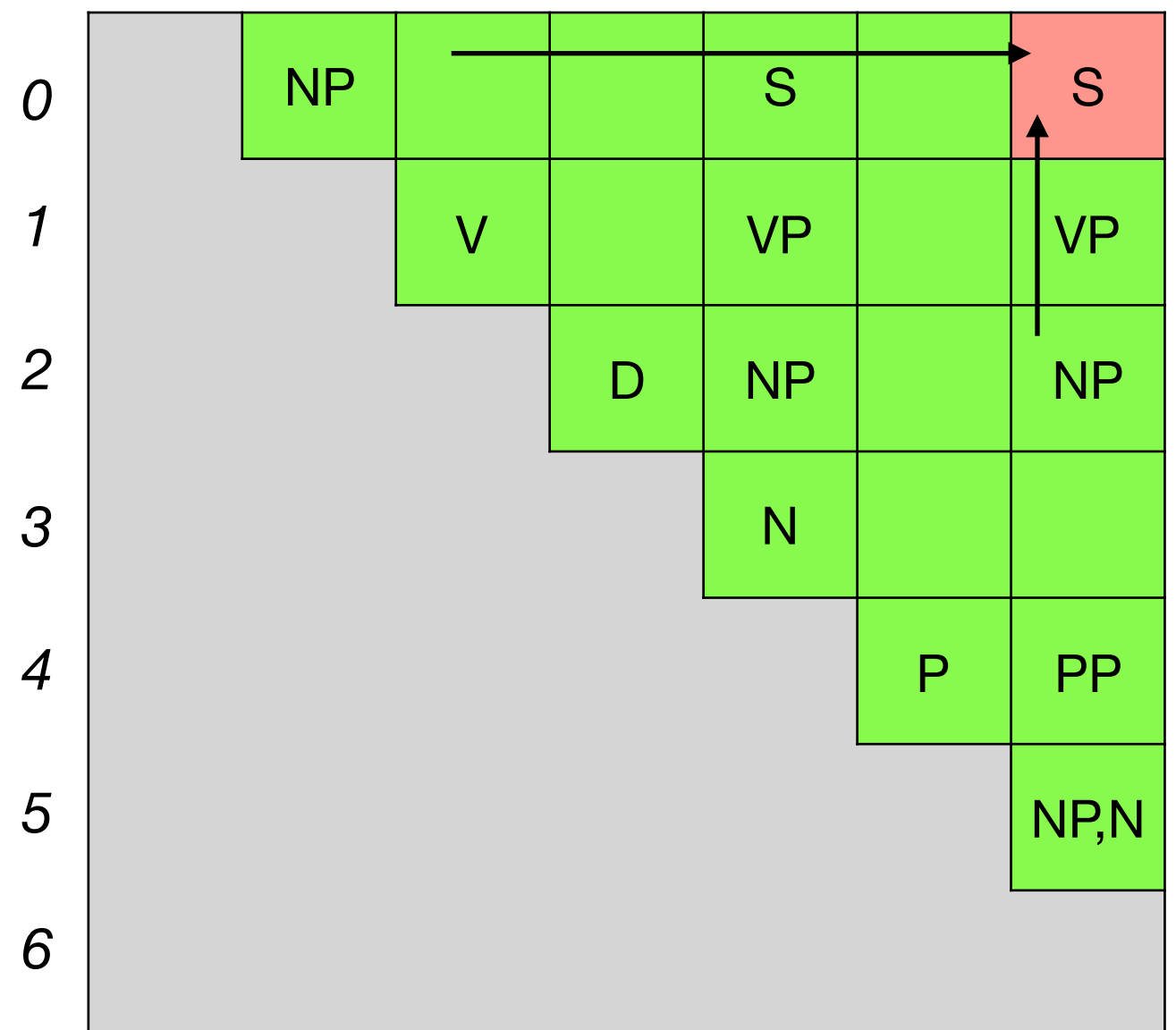
....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=5$

$i=0, k=2, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0 \dots (n-length)$:
 $j = i+length$
 for $k=i+1 \dots j-1$:

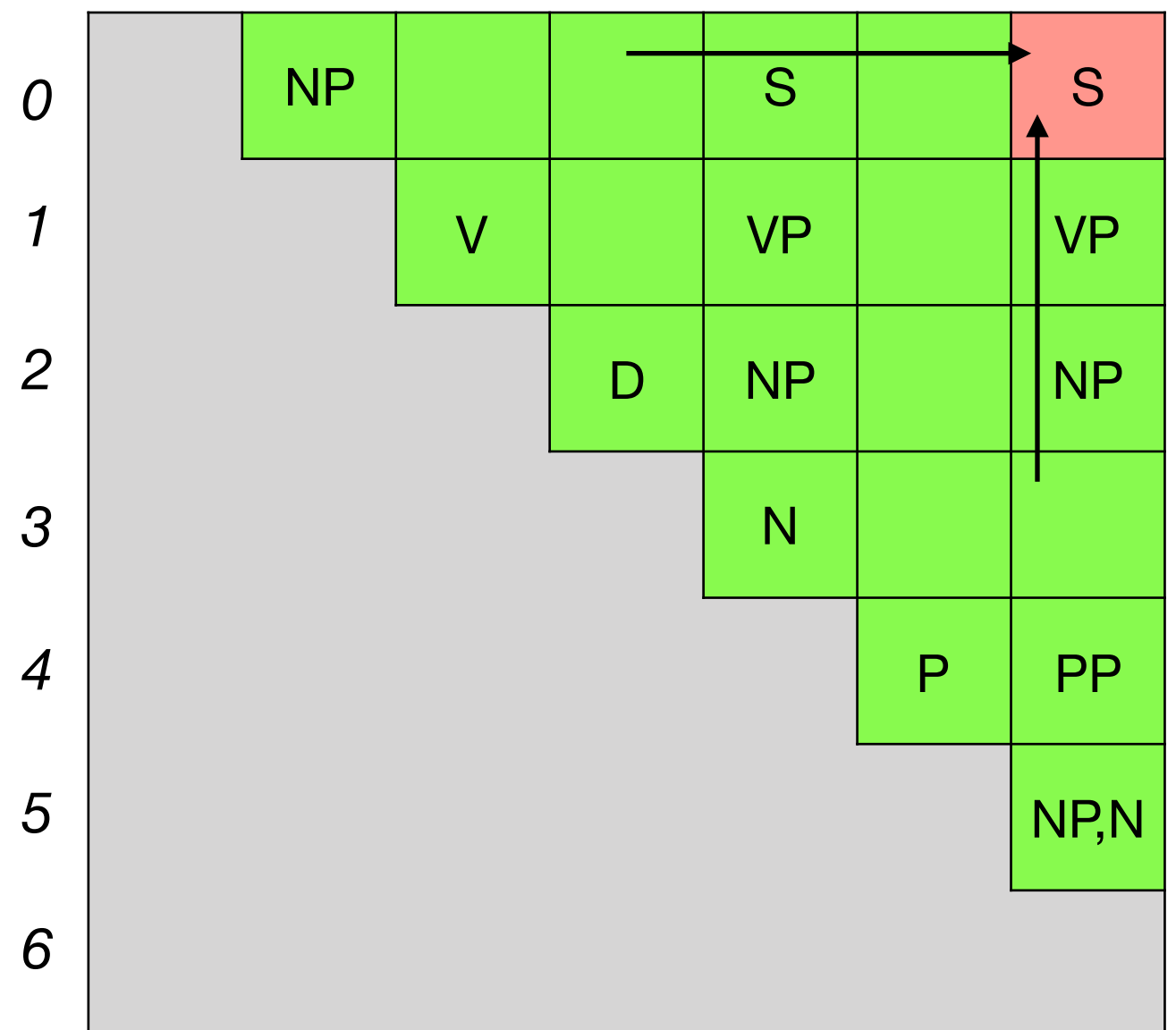
....

$S \rightarrow NP VP$	$NP \rightarrow she$
$VP \rightarrow V NP$	$NP \rightarrow glasses$
$VP \rightarrow VP PP$	$D \rightarrow the$
$PP \rightarrow P NP$	$N \rightarrow cat$
$NP \rightarrow D N$	$N \rightarrow glasses$
$NP \rightarrow NP PP$	$V \rightarrow saw$
	$P \rightarrow with$

$length=5$

$i=0, k=3, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:

$j = i+length$

for $k=i+1...j-1$:

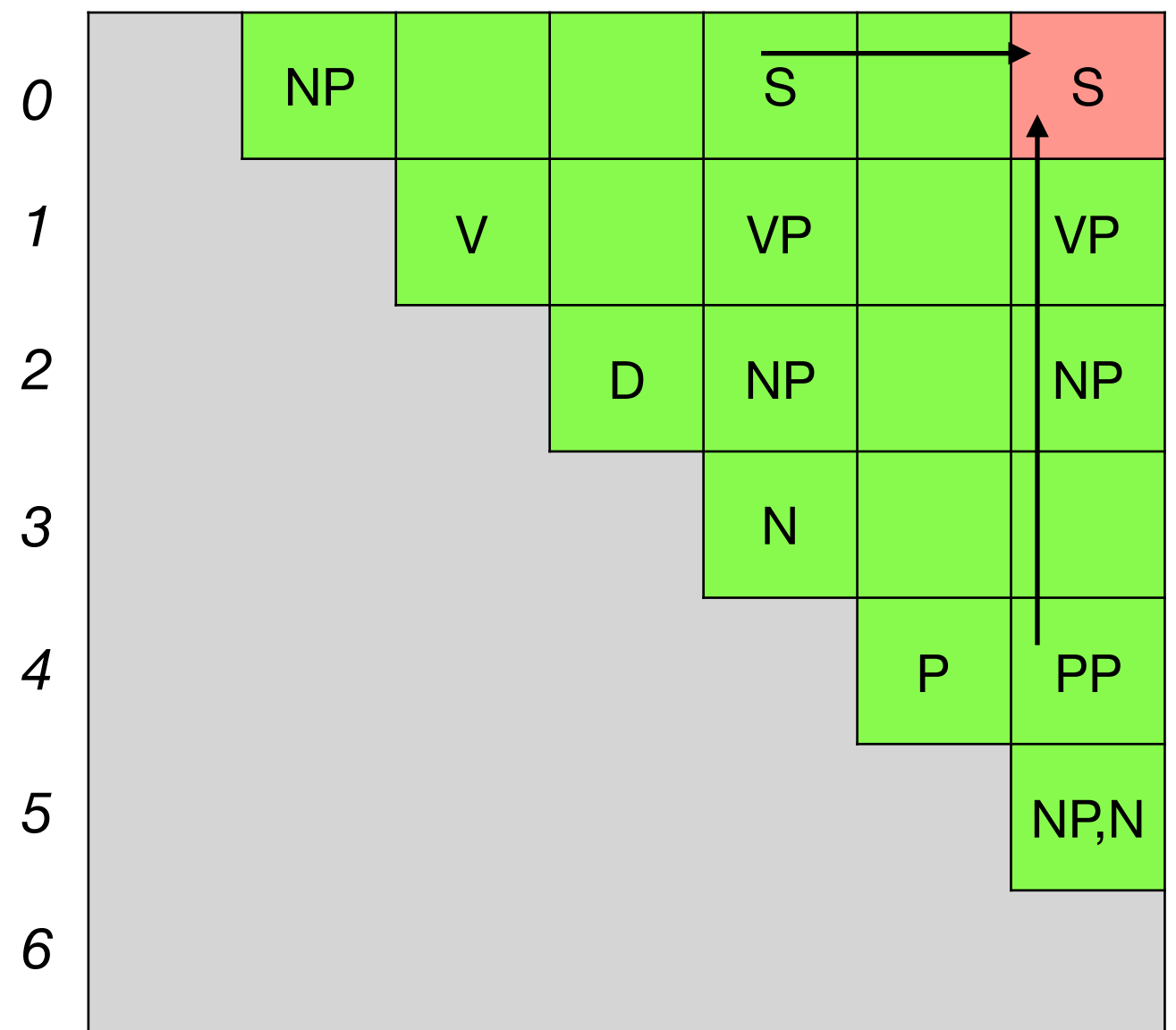
....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=5$

$i=0, k=4, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses



CKY Algorithm

for $i=0...(n-length)$:
 $j = i+length$
 for $k=i+1...j-1$:

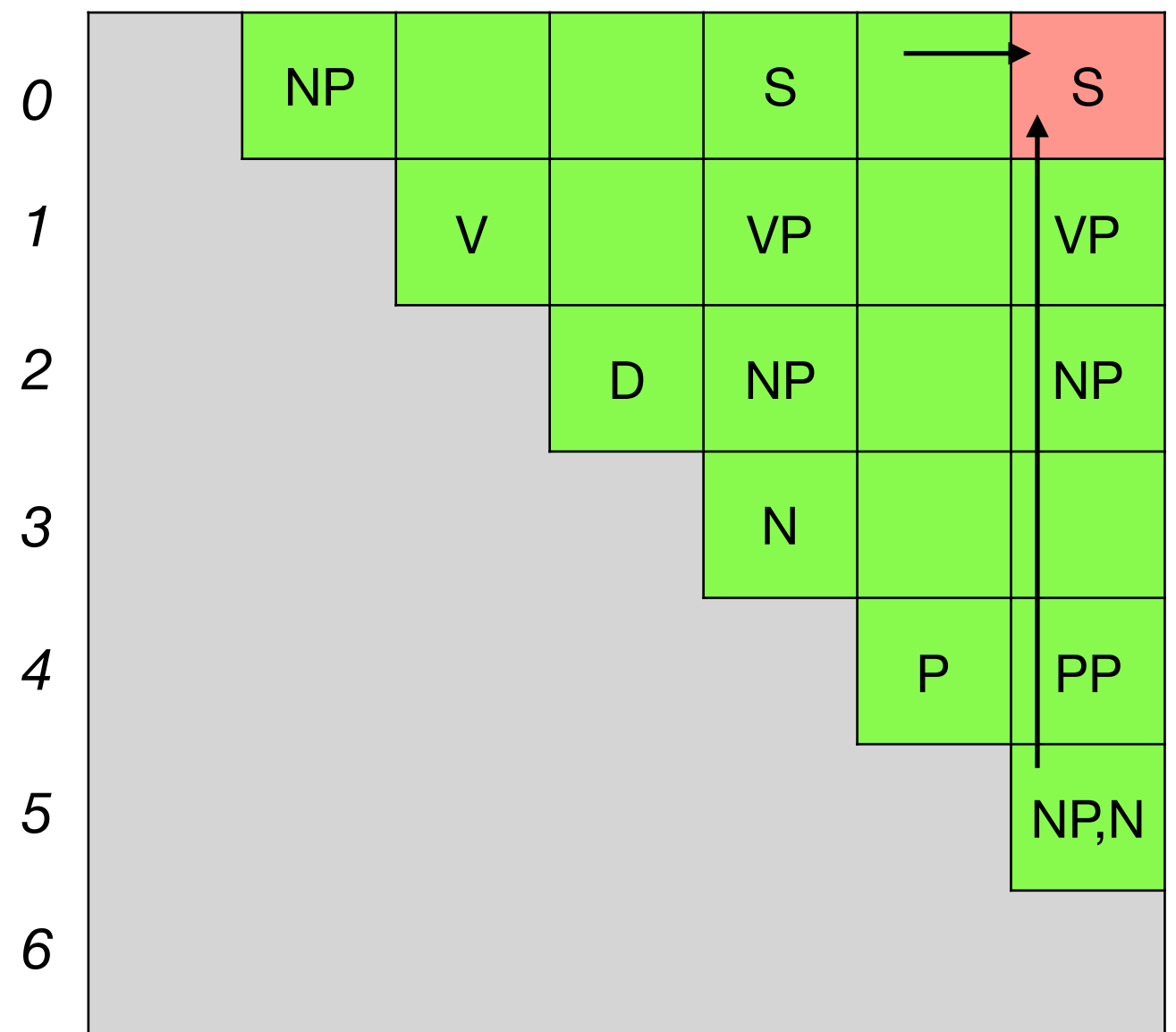
....

S \rightarrow NP VP	NP \rightarrow she
VP \rightarrow V NP	NP \rightarrow glasses
VP \rightarrow VP PP	D \rightarrow the
PP \rightarrow P NP	N \rightarrow cat
NP \rightarrow D N	N \rightarrow glasses
NP \rightarrow NP PP	V \rightarrow saw
	P \rightarrow with

$length=5$

$i=0, k=5, j=6$

$_0$ she $_1$ saw $_2$ the $_3$ cat $_4$ with $_5$ glasses

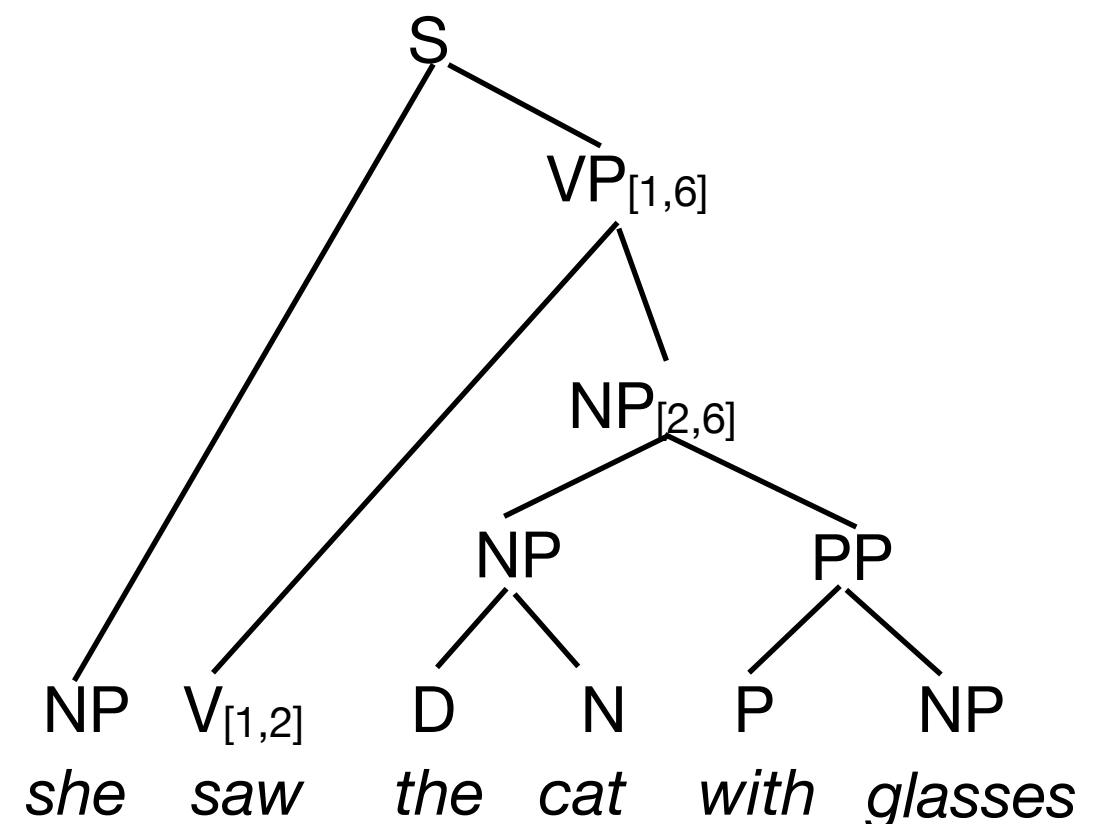
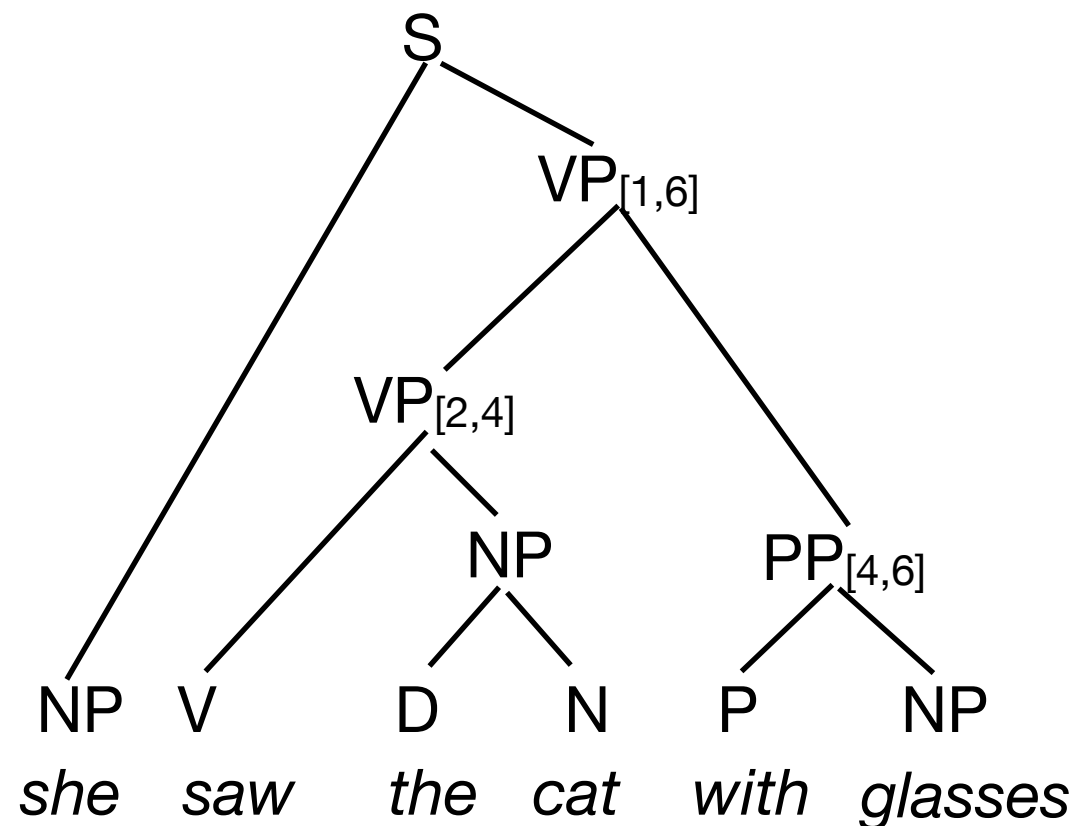


CKY Runtime

- **Input:** Grammar $G=(N, \Sigma, R, S)$, input string s of length n .
- for $i=0\dots n-1$: $O(N \times |R|)$
 $\pi[i, i+1] = \{A \mid A \rightarrow s[i]\}$
- for $length=2\dots n$: $O(N)$
 for $i=0\dots(n-length)$: $O(N)$ Total : $O(N^3 \times |R|)$
 $j = i+length$ $O(N)$
 for $k=i+1\dots j-1$:
 $M = \{A \mid A \rightarrow B C \in R \text{ and } B \in \pi[i, k] \text{ and } C \in \pi[k, j]\}$
 $\pi[i, j] = \pi[i, j] \cup M$
- if $S \in \pi[0, i+1]$ return True, otherwise False

Syntactic Ambiguity

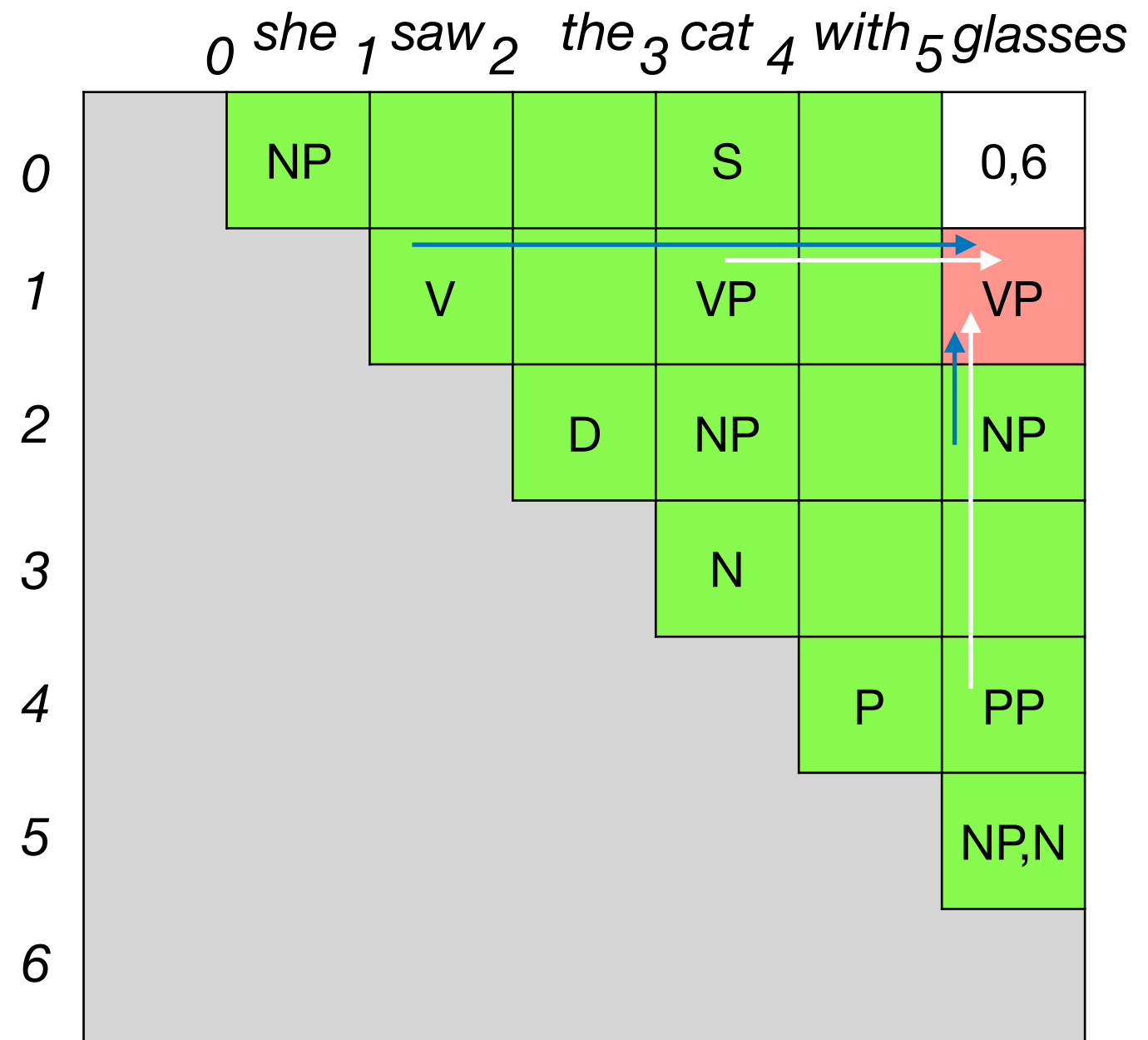
S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with



Backpointers

- The CKY algorithm presented so far determines if a sentence is recognized by a grammar.
- Also want to retrieve the parse trees!
- Instead of a set of nonterminals, store a list of instantiated rules and backpointers.

$$\left\{ \begin{array}{l} \text{VP}_{[1,6]} \rightarrow \text{V}_{[1,2]} \text{NP}_{[2,6]} \\ \text{VP}_{[1,6]} \rightarrow \text{VP}_{[1,4]} \text{PP}_{[4,6]} \end{array} \right\}$$



Retrieving Parse-Trees

- Start at the $[0,n]$ entry and recursively follow the backpointers. Return a set of subtrees from the recursion.
- How long does it take to retrieve all parse trees?
 - Worst case, there are exponentially many trees. So retrieving all of them is exponential.
- However: We can retrieve the k highest-scoring trees in polynomial time (next week).
- Retrieving ANY single parse tree takes $O(N^2)$.

Natural Language Processing

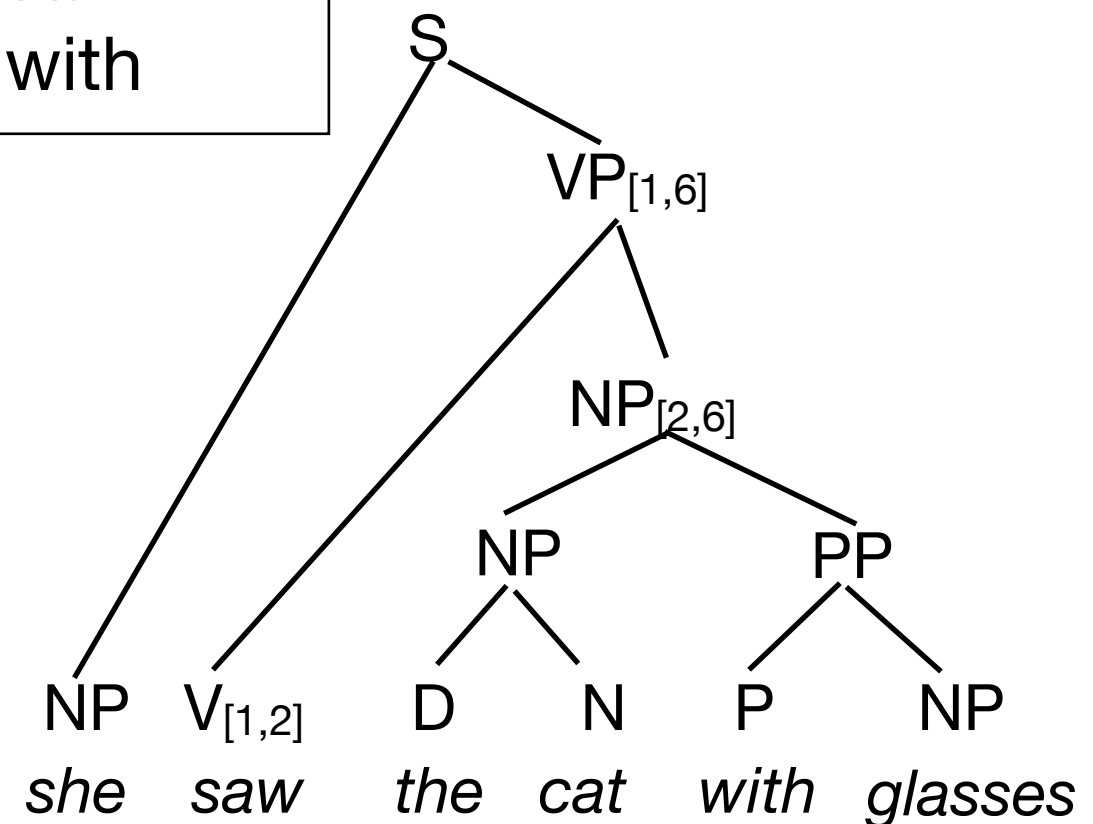
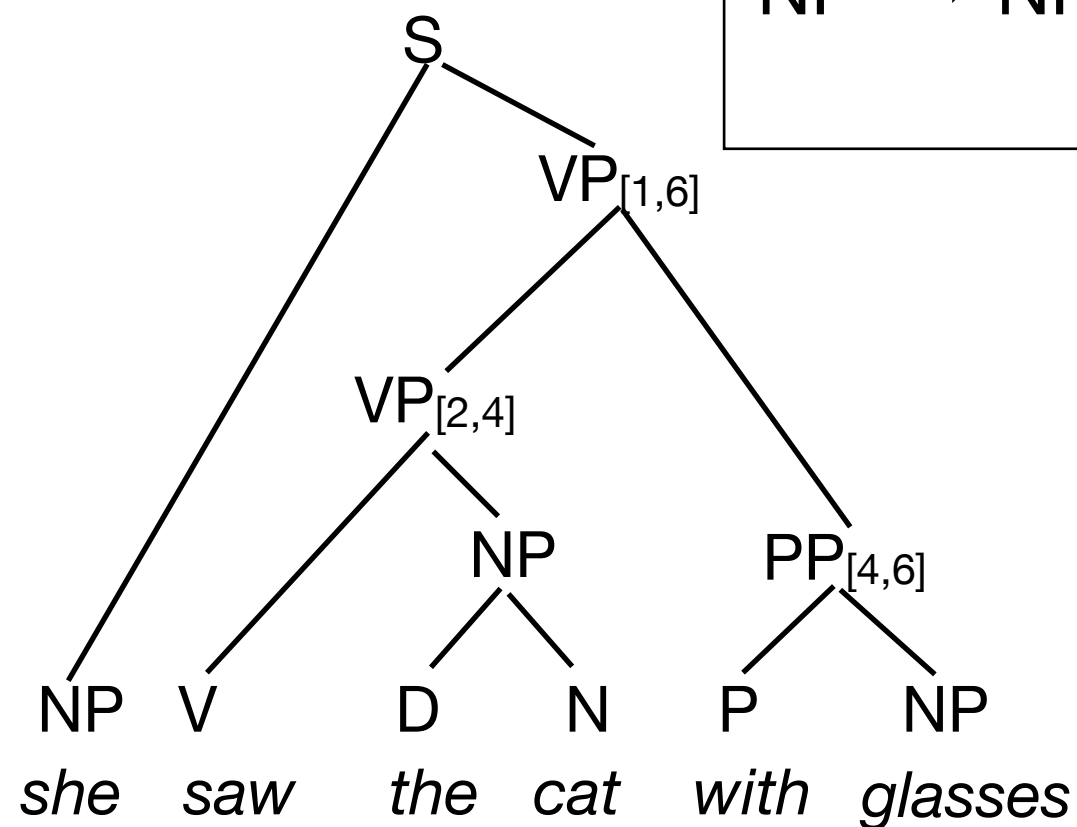
Lecture 8: More parsing with CFG & PCFG.

02/12/2018

COMS W4705
Daniel Bauer

Recall: Syntactic Ambiguity

S → NP VP	NP → she
VP → V NP	NP → glasses
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → glasses
NP → NP PP	V → saw
	P → with



Which parse tree is “better”? More probable?

Probabilities for Parse Trees

- Let \mathcal{T}_G be the (possibly infinite) set of all parse trees generated by grammar G .
- We want a model that assigns a probability to each parse tree, such that $\sum_{t \in \mathcal{T}_G} P(t) = 1$.
- We can use this model to select the most probable parse tree compatible with an input sentence.
 - This is another example of a generative model!

Selecting Parse Trees

- Let $\mathcal{T}_G(s)$ be the set of trees generated by grammar G whose *yield* (sequence of leafs) is string s .
- The most likely parse tree produced by G for string s is

$$\arg \max_{t \in \mathcal{T}_G(s)} P(t)$$

- How do we define $P(t)$?
- How do we learn such a model from training data (annotated or un-annotated).
- How do we find the highest probability tree for a given sentence? (*parsing/decoding*)

Probabilistic Context Free Grammars (PCFG)

- A PCFG consists of a Context Free Grammar $G=(N, \Sigma, R, S)$ and a probability $P(A \rightarrow \beta)$ for each production $A \rightarrow \beta \in R$.
- The probabilities for all rules with the same left-hand-side sum up to 1:

$$\sum_{A \rightarrow \beta: A=X} P(A \rightarrow \beta) = 1 \text{ for all } X \in N$$

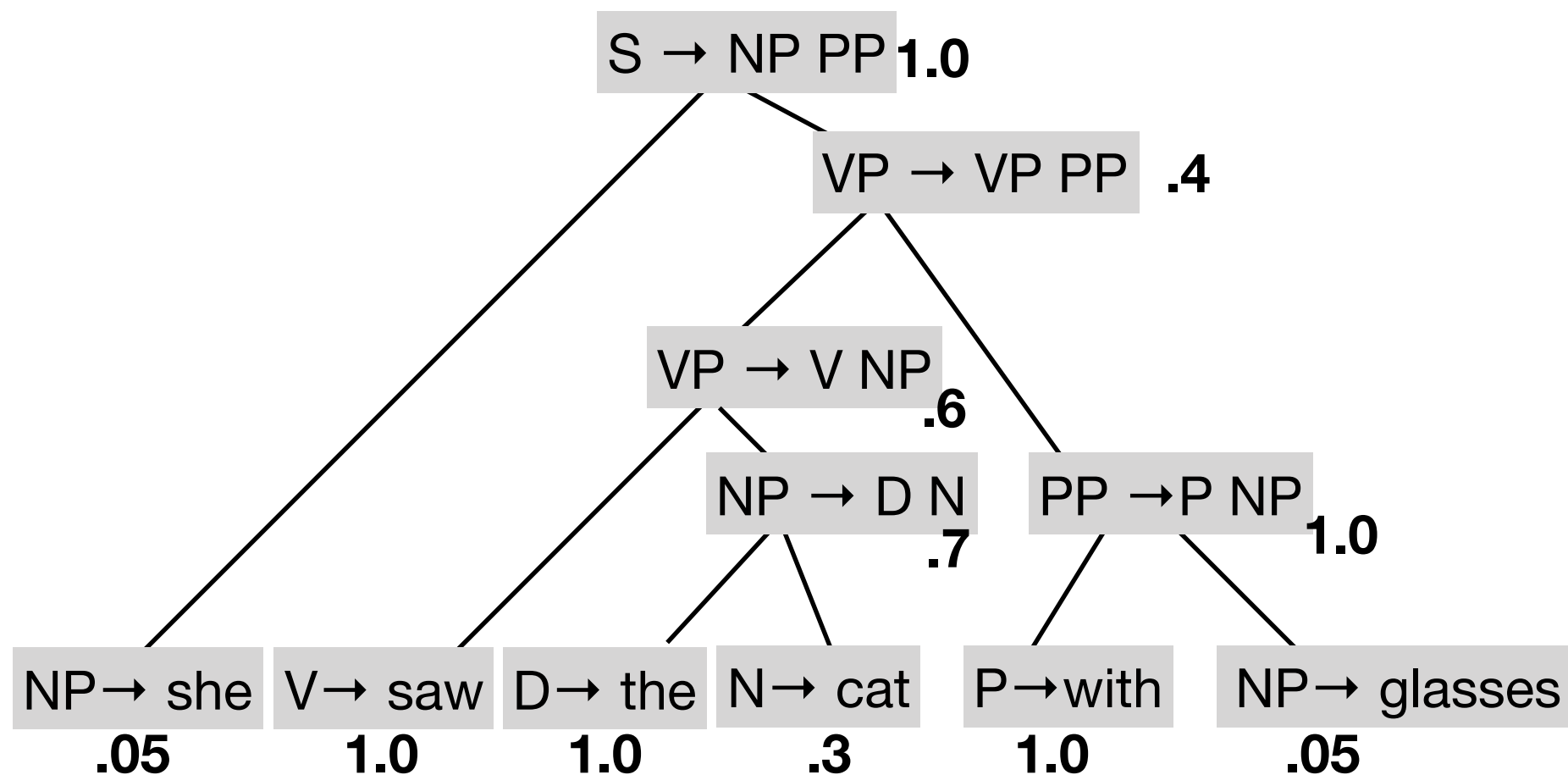
- Think of this as the conditional probability for $A \rightarrow \beta$, given the left-hand-side nonterminal A .

PCFG Example

S	→	NP VP	[1.0]	NP	→	she	[0.05]
VP	→	V NP	[0.6]	NP	→	glasses	[0.05]
VP	→	VP PP	[0.4]	D	→	the	[1.0]
PP	→	P NP	[1.0]	N	→	cat	[0.3]
NP	→	D N	[0.7]	N	→	glasses	[0.7]
NP	→	NP PP	[0.2]	V	→	saw	[1.0]
				P	→	with	[1.0]

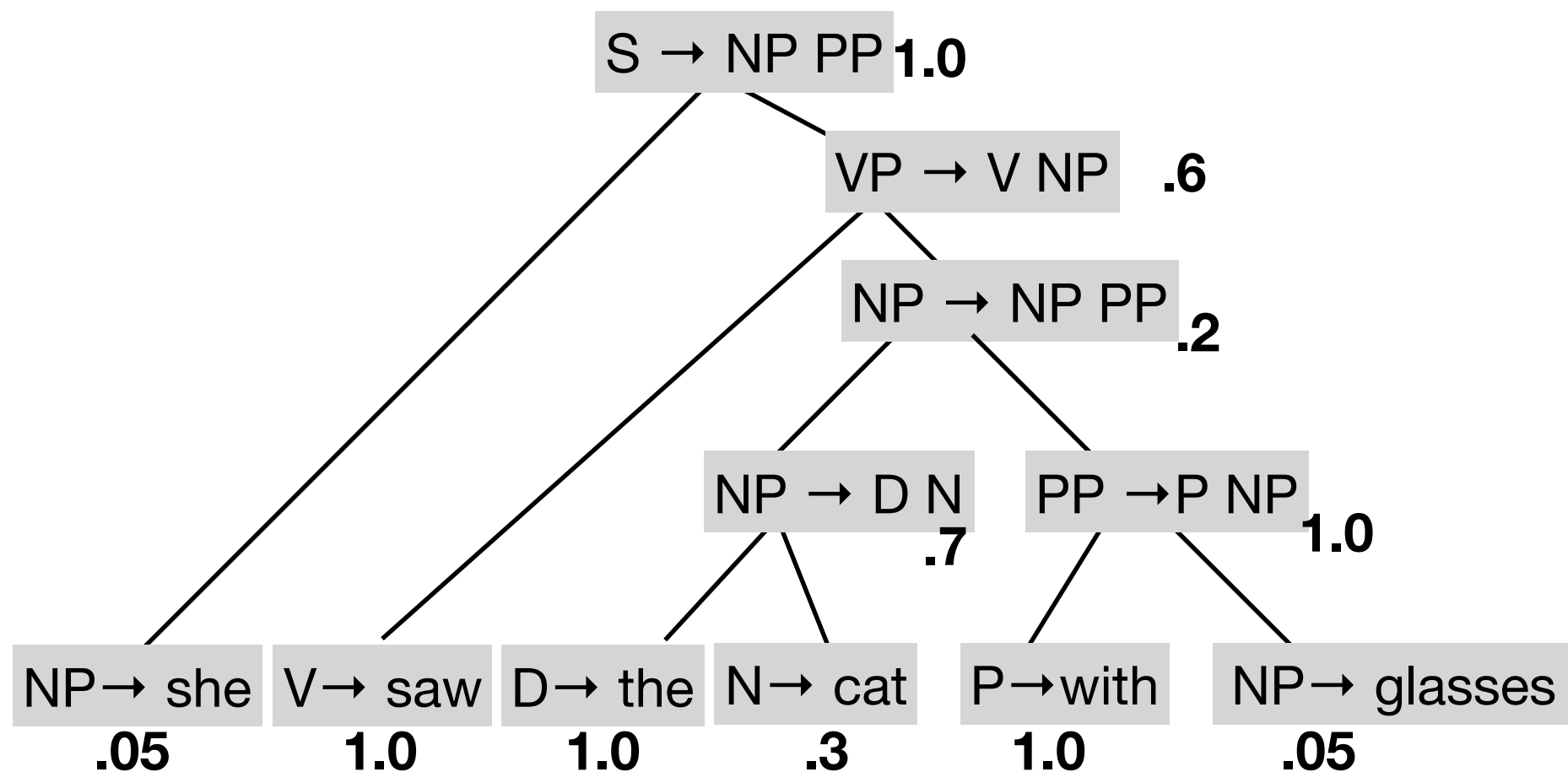
Parse Tree Probability

- Given a parse tree $t \in \mathcal{T}_G$, containing rules $A_1 \rightarrow \beta_1, \dots, A_n \rightarrow \beta_n$ the probability of t is
$$P(t) = \prod_{i=1}^n P(A_i \rightarrow \beta_i)$$



Parse Tree Probability

- Given a parse tree $t \in \mathcal{T}_G$, containing rules $A_1 \rightarrow \beta_1, \dots, A_n \rightarrow \beta_n$ the probability of t is
$$P(t) = \prod_{i=1}^n P(A_i \rightarrow \beta_i)$$



$$1 \times .05 \times .6 \times 1 \times .2 \times .7 \times 1 \times .3 \times 1 \times 1 \times .05 = 0.000063 < 0.000126$$

Estimating PCFG probabilities

- Supervised training: We can estimate PCFG probabilities from a *treebank*, a corpus manually annotated with constituency structure using maximum likelihood estimates:

$$P(A \rightarrow \beta) = \frac{\text{count}(A \rightarrow \beta)}{\text{count}(A)}$$

- Unsupervised training:
 - What if we have a grammar and a corpus, but no annotated parses?
 - Can use the **inside-outside** algorithm for parsing and do EM estimation of the probabilities (won't talk about this today).

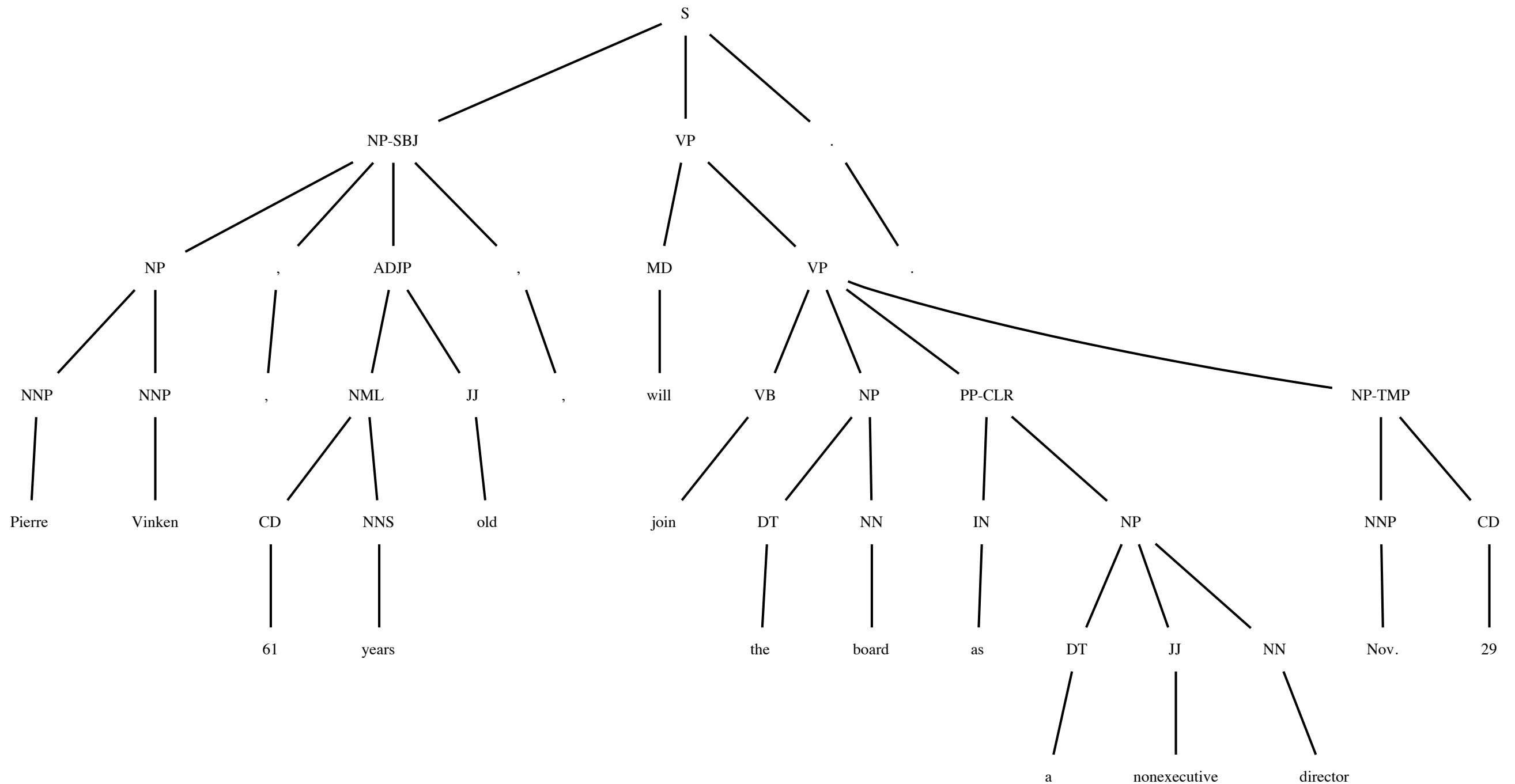
The Penn Treebank

- Syntactically annotated corpus of newspaper text (1989 Wall Street Journal Articles).
- The source text is naturally occurring but the treebank is not:
 - Assumes a specific linguistic theory (although a simple one).
 - Very flat structure (NPs, Ss, VPs).

PTB Example

```
( (S (NP-SBJ (NP (NNP Pierre) (NNP Vinken))
      ( , , )
      (ADJP (NML (CD 61) (NNS years))
      (JJ old))
      ( , , ))
  (VP (MD will)
(VP (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  ( . . )))
```

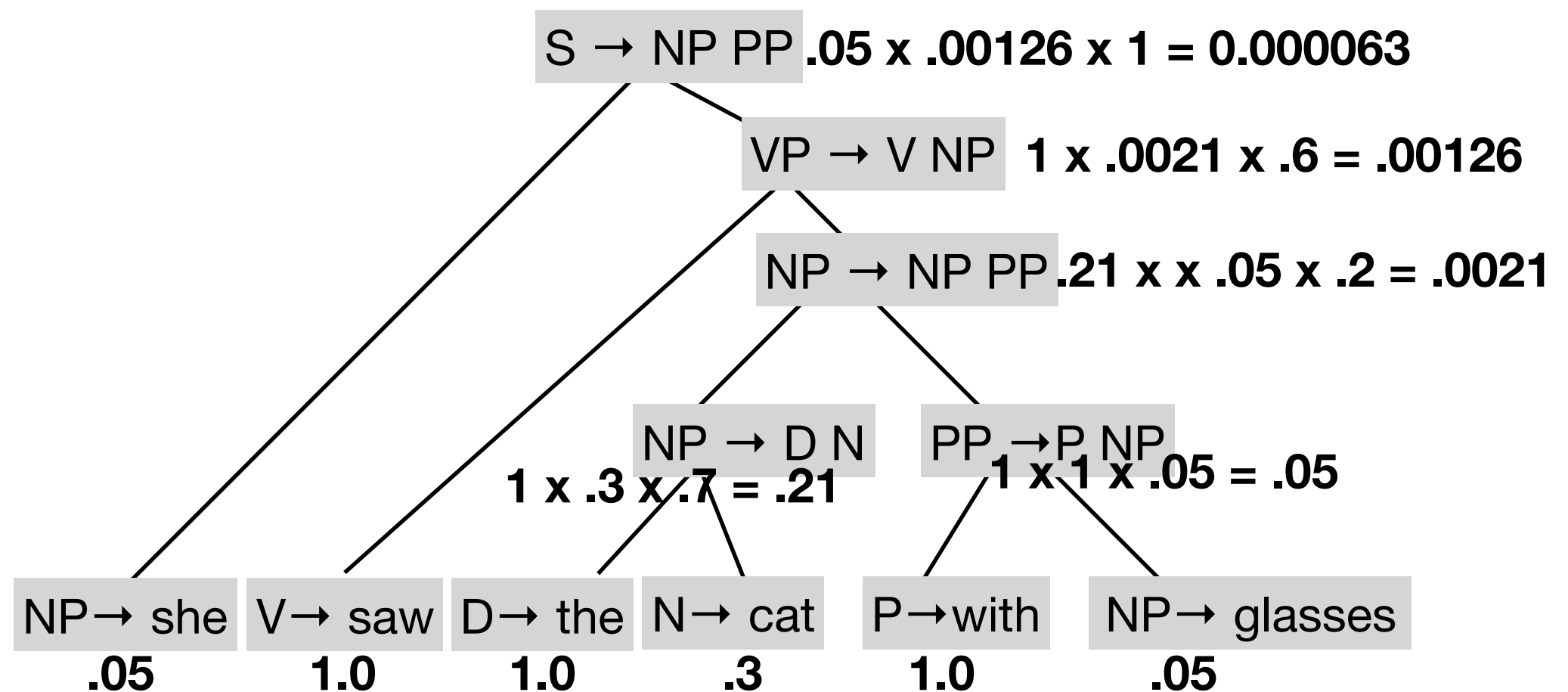
PTB Example



Parsing with PCFG

- We want to use PCFG to answer the following questions:
 - What is the total probability of the sentence under the PCFG?
 - What is the most probable parse tree for a sentence under the PCFG? (*decoding/parsing*)
- We can modify the CKY algorithm.
Basic idea: Compute these probabilities bottom-up using dynamic programming.

Computing Probabilities Bottom-Up



CKY for PCFG parsing

- Let $\pi[i,j,X]$ be the probability of the highest scoring parse tree for $s[i,j]$ starting in nonterminal X .

$$\pi[i, j, X] = \max_{t \in \mathcal{T}_G(s[i,j])} P(t)$$

- $\pi[0, \text{length}(s), S]$ is the probability of the highest-scoring parse tree for s .

- Recursive definition:

$$\text{Base case: } \pi[i, i, X] = \begin{cases} \max_{A \rightarrow s_i \in R} P(A \rightarrow s_i) & \text{if } A \rightarrow s_i \in R \\ 0 & \text{otherwise} \end{cases}$$

$$\pi[i, j, X] = \max_{\substack{k=i+1 \dots j-1, \\ A \rightarrow BC \in R}} P(A \rightarrow BC) \cdot \pi[i, k, B] \cdot \pi[k, j, C]$$

CKY for PCFG Parsing

- **Input:** PCFG $G=(N, \Sigma, R, S)$, input string s of length n .
- for $i=0 \dots n-1$: initialization
$$\pi[i, i, X] = \begin{cases} \max_{X \rightarrow s_i} & \text{if } X \rightarrow s_i \in R \\ 0 & \text{otherwise} \end{cases}$$
- for $length=2 \dots n$: main loop
 - for $i=0 \dots (n-length)$:
 - $j = i+length$
 - for $k=i+1 \dots j-1$:
 - for $X \in N$:
 - $$\pi[i, j, X] = \max_{\substack{k=i+1 \dots j-1 \\ X \rightarrow BC \in R}} P(X \rightarrow BC) \cdot \pi[i, k, B] \cdot \pi[k, j, C]$$

Use **backpointers** to retrieve the highest-scoring parse tree (see previous lecture).

Probability of a Sentence

- What if we are interested in the probability of a sentence, **not** of a single parse tree (for example, because we want to use the PCFG as a language model).
- Problem: Spurious ambiguity. Need to sum the probabilities of **all** parse trees for the sentence.
- How do we have to change CKY to compute this?

$$\pi[i, j, X] = \sum_{\substack{k=i+1 \dots j-1, \\ A \rightarrow BC \in R}} P(A \rightarrow BC) \cdot \pi[i, k, B] \cdot \pi[k, j, C]$$

Earley Parser

- CKY parser starts with words and builds parse trees bottom-up; requires the grammar to be in CNF.
- The Earley parser instead starts at the start symbol and tries to “guess” derivations top-down.
 - It discards derivations that are incompatible with the sentence.
 - The early parser sweeps through the sentence left-to-right only once. It keeps partial derivations in a table (“chart”).
 - Allows arbitrary CFGs.

Parser States

- Earley parser keeps track of partial derivations using parser **states / items**.
- State represent hypotheses about constituent structure based on the grammar, taking into account the input.
- Parser states are represented as **dotted rules with spans**.
 - The constituents to the left of the \cdot have already been seen in the input (corresponding to the span)

$S \rightarrow \cdot NP VP [0,0]$ *“According to the grammar, there may be an NP starting in position 0. “*

$NP \rightarrow D A \cdot N [0,2]$ *“There is a determiner followed by an adjective in $s[0,2]$ “*

$NP \rightarrow NP PP \cdot [3,8]$ *“There is complete NP in $s[3,8]$, consisting of an NP and PP”*

Earley Parser

S → NP VP	V → saw
VP → V NP	P → with
VP → VP PP	D → the
PP → P NP	N → cat
NP → D N	N → tail
NP → NP PP	N → student

S → · NP VP [0,0]

NP → · NP PP [0,0] NP → · D N [0,0]

D → · the [0,0]

Three parser operations:

1. **Predict** new subtrees top-down.

0 the 1 student 2 saw 3 the 4 cat 5 with 6 the 7 tail

Earley Parser

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

$S \rightarrow \cdot NP VP [0,0]$

$NP \rightarrow \cdot NP PP [0,0]$

$NP \rightarrow \cdot D N [0,0]$

$D \rightarrow \text{the} \cdot [0,1]$

Three parser operations:

1. Predict new subtrees top-down.
2. **Scan** input terminals.

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

$S \rightarrow \cdot NP VP [0,0]$

$NP \rightarrow \cdot NP PP [0,0]$

$NP \rightarrow \cdot D N [0,0]$

$D \rightarrow \text{the} \cdot [0,1]$

passive state

Three parser operations:

1. Predict new subtrees top-down.
2. **Scan** input terminals.

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

$NP \rightarrow \cdot NP PP [0,0]$

$S \rightarrow \cdot NP VP [0,0]$

$NP \rightarrow D \cdot N [0,1]$

$D \rightarrow \text{the} \cdot [0,1]$

passive state

Three parser operations:

1. Predict new subtrees top-down.
2. Scan input terminals.
3. **Complete** with passive states.

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

Three parser operations:

1. **Predict** new subtrees top-down.
2. Scan input terminals.
3. Complete with passive states.

$NP \rightarrow \cdot NP PP [0,0]$

$S \rightarrow \cdot NP VP [0,0]$

$NP \rightarrow D \cdot N [0,1]$

$D \rightarrow \text{the} \cdot [0,1]$

$N \rightarrow \cdot \text{cat} [1,1]$

$N \rightarrow \cdot \text{tail} [1,1]$

$N \rightarrow \cdot \text{student} [1,1]$

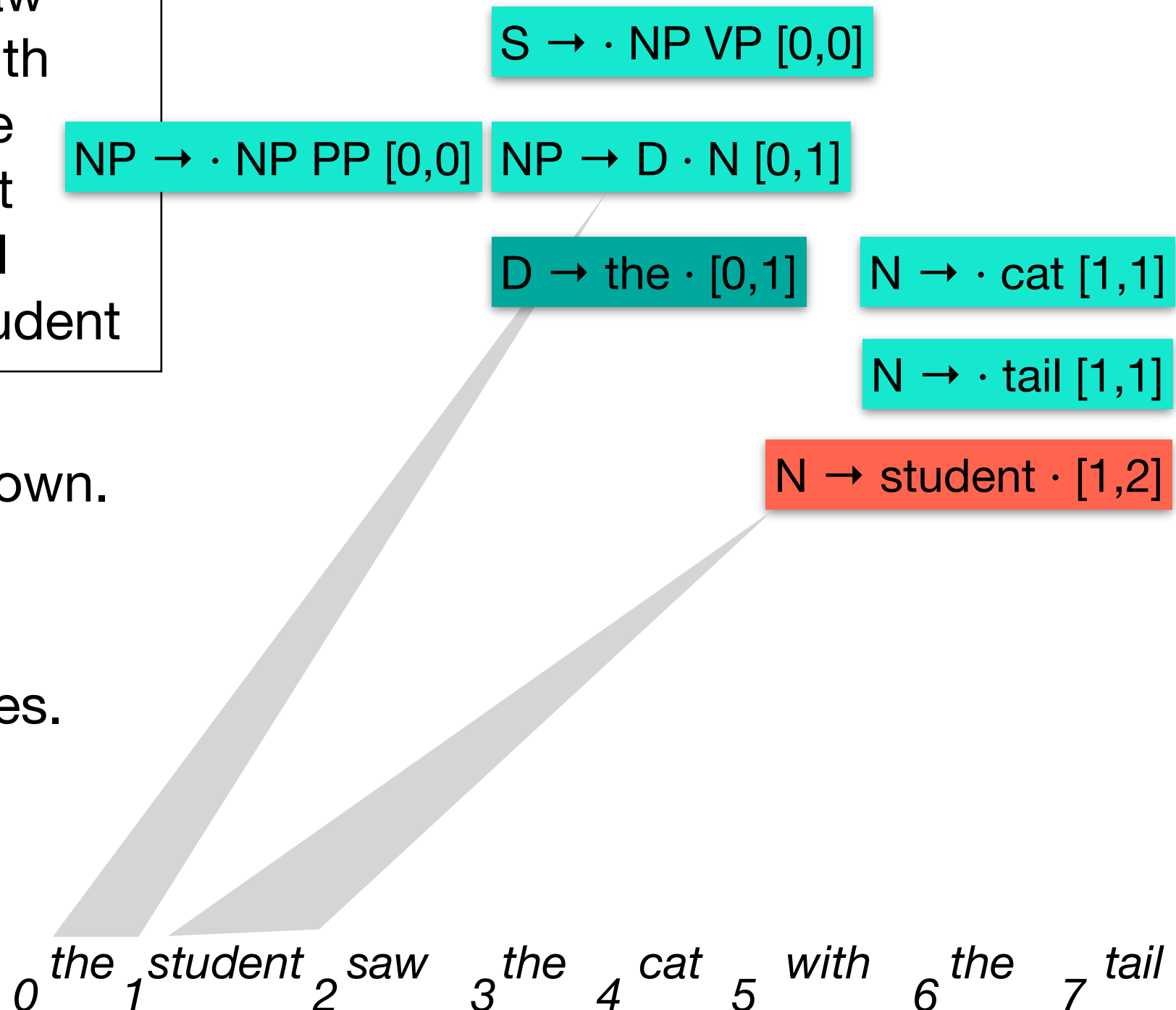
0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

Three parser operations:

1. Predict new subtrees top-down.
2. **Scan** input terminals.
3. Complete with passive states.

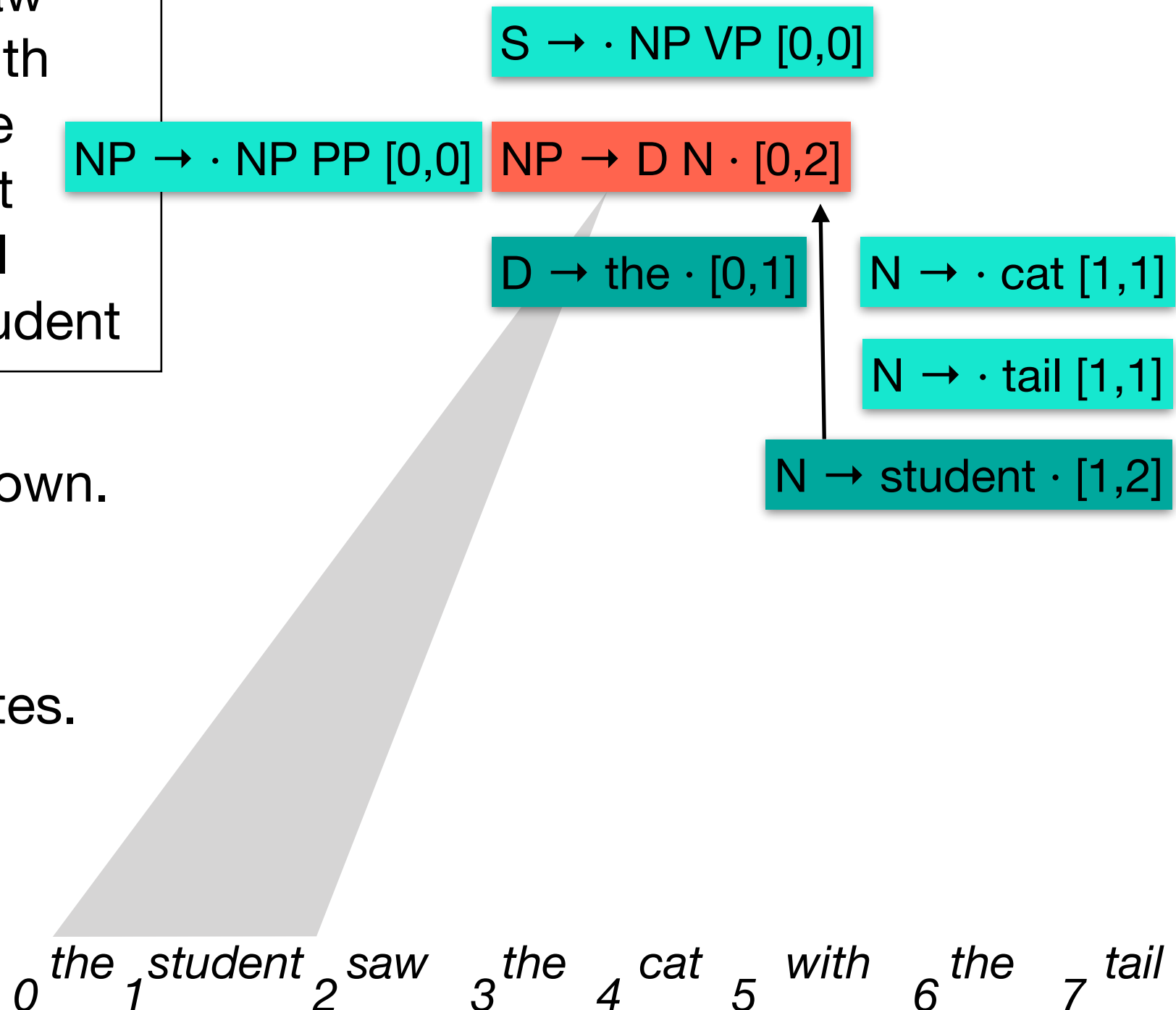


Earley Parser

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

Three parser operations:

1. Predict new subtrees top-down.
2. Scan input terminals.
3. **Complete** with passive states.

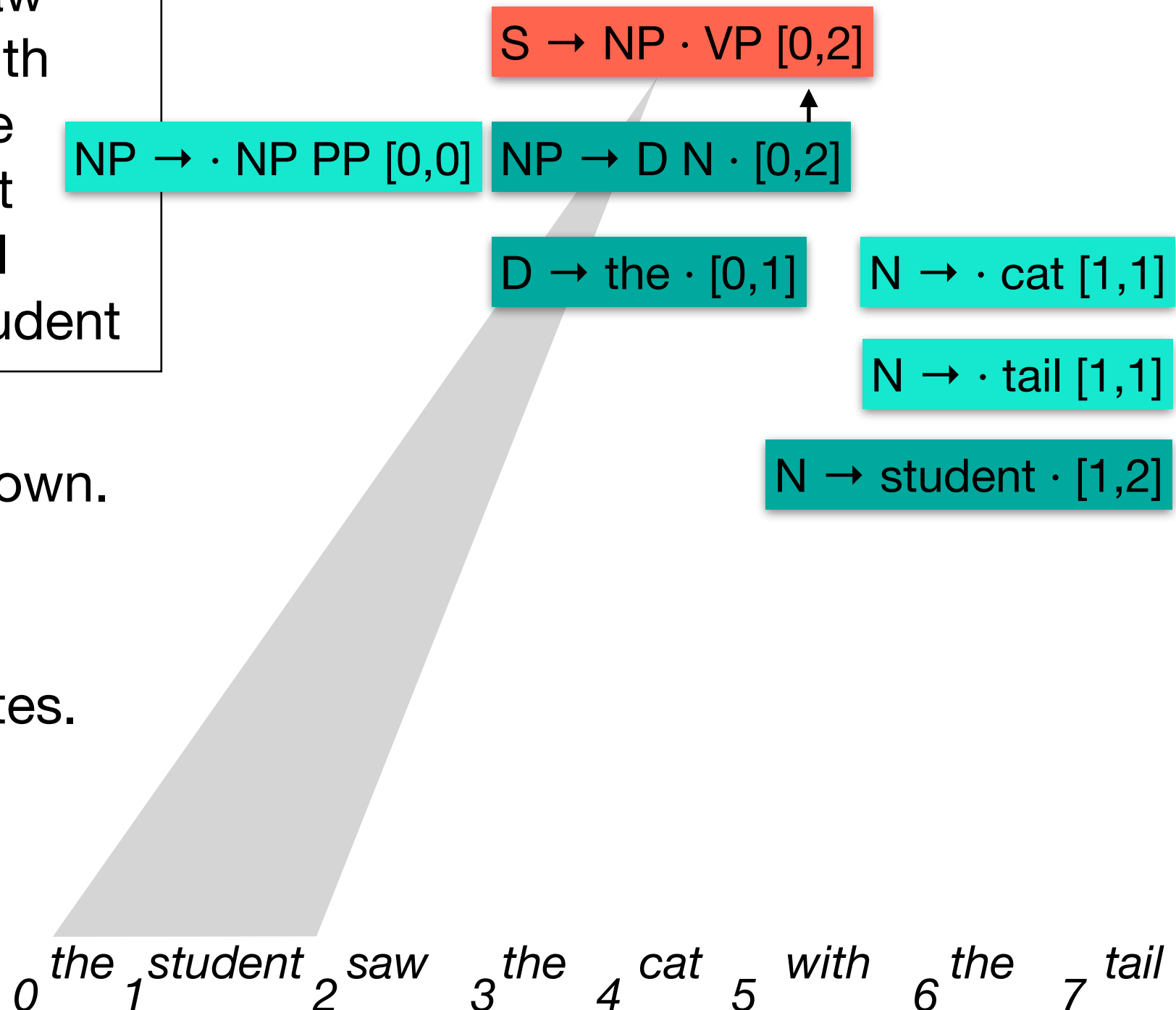


Earley Parser

$S \rightarrow NP VP$	$V \rightarrow \text{saw}$
$VP \rightarrow V NP$	$P \rightarrow \text{with}$
$VP \rightarrow VP PP$	$D \rightarrow \text{the}$
$PP \rightarrow P NP$	$N \rightarrow \text{cat}$
$NP \rightarrow D N$	$N \rightarrow \text{tail}$
$NP \rightarrow NP PP$	$N \rightarrow \text{student}$

Three parser operations:

1. Predict new subtrees top-down.
2. Scan input terminals.
3. **Complete** with passive states.



Earley Algorithm

- Keep track of parser states in a table (“chart”). $Chart[k]$ contains a list of all parser states that end in position k .
- **Input:** Grammar $G=(N, \Sigma, R, S)$, input string s of length n .
- **Initialization:** For each production $S \rightarrow \alpha \in R$
add a state $S \rightarrow \cdot \alpha [0,0]$ to $Chart[0]$.
- for $i = 0$ to n :
 - for each $state$ in $Chart[i]$:
 - if $state$ is of form $A \rightarrow \alpha \cdot s[i] \beta [k,i]$:
shift($state$)
 - elif $state$ is of form $A \rightarrow \alpha \cdot B \beta [k,i]$:
predict($state$)
 - else: // $state$ is of form $A \rightarrow \alpha \cdot [k,i]$
complete($state$)

Earley Algorithm - Shift

- The shift operation can only be applied to a state if the dot is in front of a terminal symbol that matches the next input terminal.

- function **shift**(state): *// state is of form $A \rightarrow \alpha \cdot s[i] \beta [k, i]$*
 - Add a new state $A \rightarrow \alpha s[i] \cdot \beta [k, i+1]$
to $Chart[i+1]$

Earley Algorithm - Predict

- The predict operation can only be applied to a state if the dot is in front of a non-terminal symbol.
- function **predict**(state): *// state is of form $A \rightarrow \alpha \cdot B \beta [k,i]$:*
 - Add a new state $B \rightarrow \cdot \gamma [i,i]$ to $Chart[i]$
- Note that this modifies $Chart[i]$ **while** the algorithm is looping through it.

Earley Algorithm - Complete

- The complete operation may only be applied to a passive item.

- function **complete**(state): *// state is of form $A \rightarrow \alpha \cdot [k,j]$*

- for each state $B \rightarrow \beta \cdot A \gamma [i,k]$ add a new state $B \rightarrow \beta A \cdot \gamma [i,j]$ to Chart[j]

- Note that this modifies Chart[i] **while** the algorithm is looping through it.
- Note that it is important to make a copy of the old state before moving the dot.
- This operation is similar to the combination operation in CKY!

Earley Algorithm - Runtime

- The runtime depends on the number of items in the chart (each item is “visited” exactly once).
- We proceed through the input exactly once, which takes $O(N)$.
 - For each position on the chart, there are $O(N)$ possible split points where the dot could be.
 - Each complete operation can produce $O(N)$ possible new items (with different starting points).
- Total: $O(N^3)$

Earley Algorithm - Some Observations

- How do we recover parse trees?
 - What happens in case of ambiguity?
 - Multiple ways to Complete the same state.
 - Keep back-pointers in the parser state objects.
 - Or use a separate data structure (CKY-style table or hashed states)
- How do we make the algorithm work with PCFG?
 - Easy to compute probabilities on Complete. Follow back pointer with max probability.

Chart Parsing

- Generalization of various parsing algorithms.
- Can be used for top-down and bottom-up parsing.
- Idea: Use two data structures.
 - Keep a **chart** of explored parser states.
 - Keep an **agenda** of unexplored, but reachable parser states.
 - Could use queue, stack, heap, ...
- Good framework for exploring other parsing strategies (e.g. best first parsing).

Parsing as Deduction

- It is common in the literature to write parser operations as deduction rules (as in logic).

- Predict:
$$\frac{A \rightarrow \alpha \cdot B\beta [i, j]}{B \rightarrow \cdot \gamma [j, j]} \quad \text{for all } B \rightarrow \gamma \in R$$

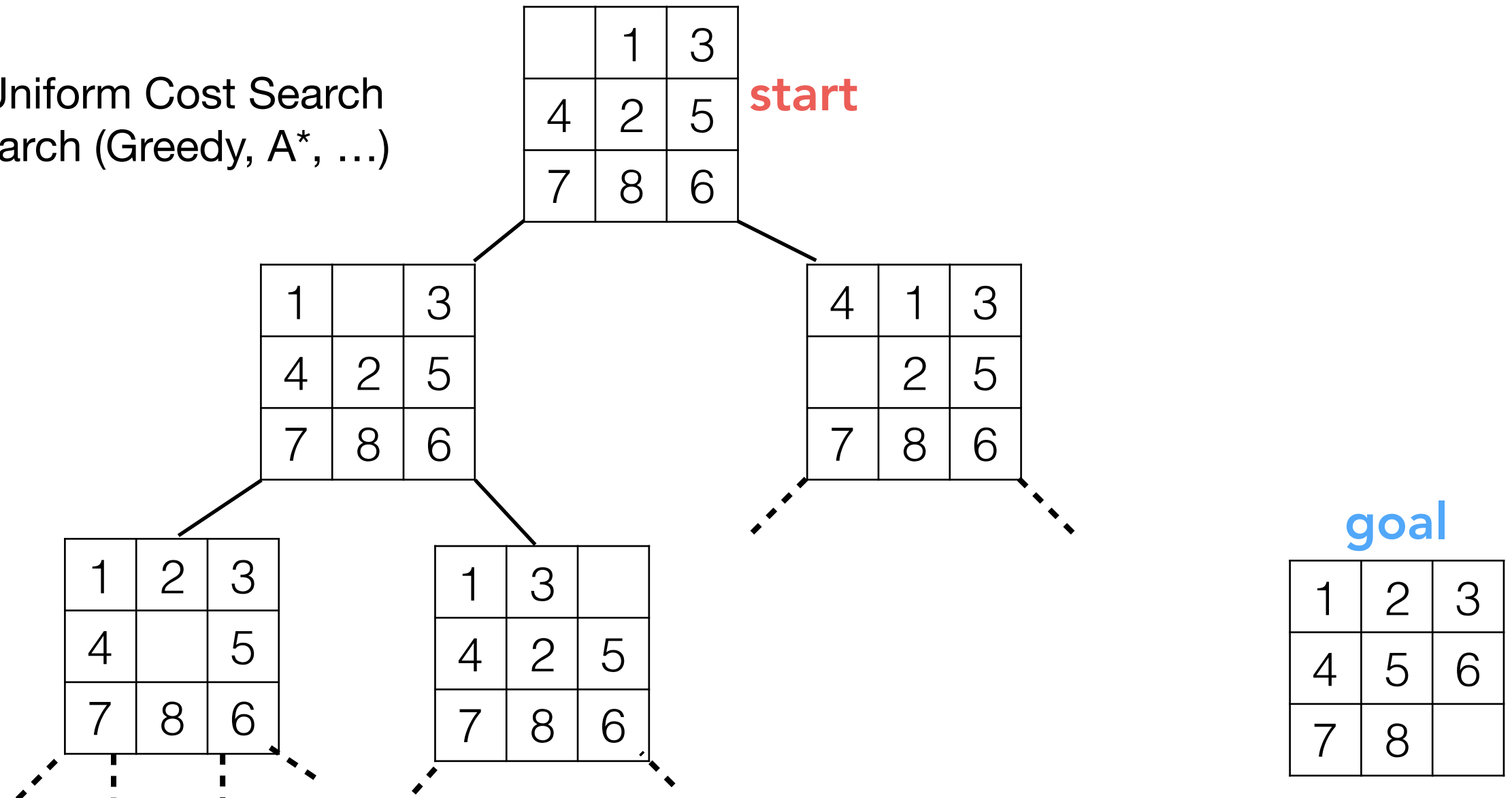
- Scan:
$$\frac{A \rightarrow \alpha \cdot s[i]\beta [i, j]}{A \rightarrow \alpha s[i] \cdot \beta [i, j + 1]}$$

- Complete:
$$\frac{A \rightarrow \alpha \cdot B\beta [i, k] \quad B \rightarrow \gamma \cdot [k, j]}{A \rightarrow \alpha B \cdot \beta [i, j]}$$

State Space Search Example: 8-puzzle

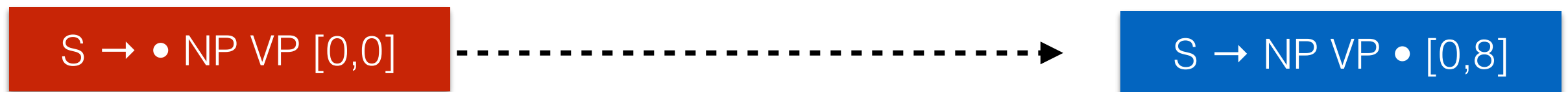
Strategies:

DFS, BFS, Uniform Cost Search
Heuristic Search (Greedy, A*, ...)



- Vertices (states) : board situations reachable from **start**.
- Edges (transitions) : permitted moves.
- Shortest path: Shortest sequence of transitions
(what if there are different costs?)

Parsing as State-Space Search



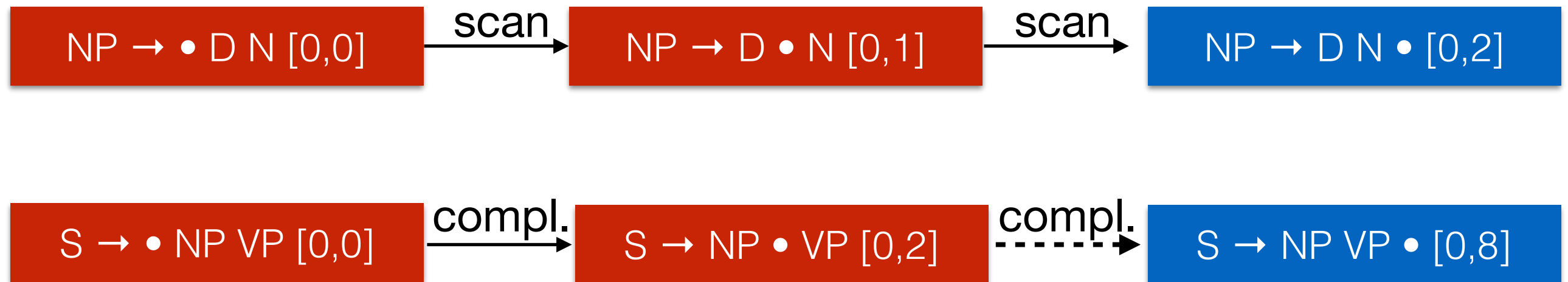
- Vertices(states) : parser states
- Transitions :??
- Shortest paths : highest probability parse tree under this item.

Parsing as State-Space Search



- Vertices(states) : parser states
- Transitions : possible **scan** operations permitted by the input.
- Shortest paths : highest probability subtree under this item.

Parsing as State-Space Search



- Vertices(states) : parser items
- Transitions : possible **scans** permitted by the input.
possible **completes** permitted by previous items on the chart.
- Shortest paths : highest probability subtree under this item.

Parsing as State-Space Search

- When we visit an active item
 - add all states resulting from **predicts** to the agenda.

$S \rightarrow \bullet NP VP [0,0]$

$NP \rightarrow \bullet NP PP [0,0]$

$NP \rightarrow \bullet D N [0,0]$

scan

$NP \rightarrow D \bullet N [0,1]$

scan

$NP \rightarrow D N \bullet [0,2]$

Parsing as State-Space Search

- When we visit an active item
 - add all states resulting from **predicts** to the agenda.

$S \rightarrow \bullet NP VP [0,0]$

$NP \rightarrow \bullet NP PP [0,0]$

$NP \rightarrow \bullet D N [0,0]$

scan

$NP \rightarrow D \bullet N [0,1]$

scan

$NP \rightarrow D N \bullet [0,2]$

Parsing as State-Space Search

- When we visit an active item
 - add all states resulting from **predicts** to the agenda.

$S \rightarrow \bullet NP VP [0,0]$

$NP \rightarrow \bullet NP PP [0,0]$

$NP \rightarrow \bullet D N [0,0]$

scan

$NP \rightarrow D \bullet N [0,1]$

scan

$NP \rightarrow D N \bullet [0,2]$

Parsing as State-Space Search

- When we visit an active item
 - add all states resulting from **predicts** to the agenda.

$S \rightarrow \bullet NP VP [0,0]$

$NP \rightarrow \bullet NP PP [0,0]$

$NP \rightarrow \bullet D N [0,0]$

scan

$NP \rightarrow D \bullet N [0,1]$

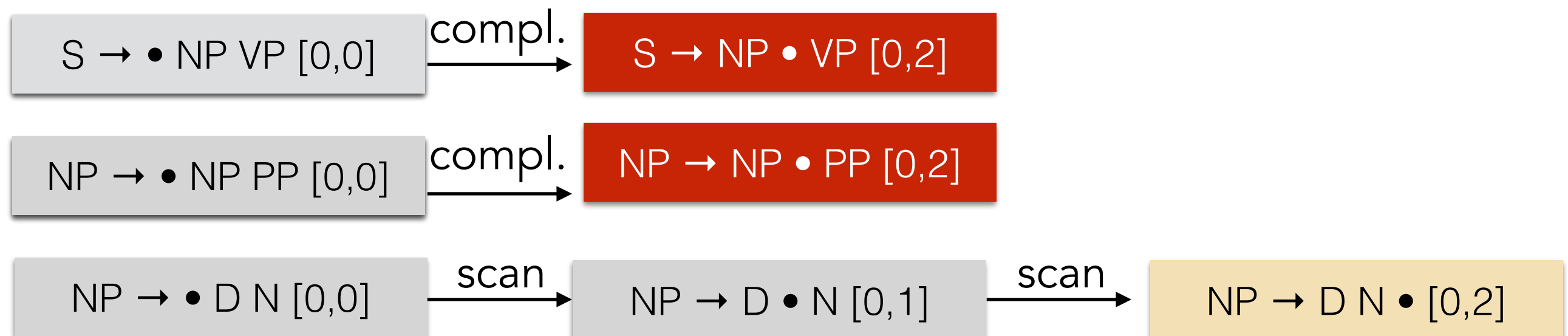
scan

$NP \rightarrow D N \bullet [0,2]$

- When we visit a passive item, find any **matching active** item on the chart and add all states resulting from **completes** with to the agenda.

Parsing as State-Space Search

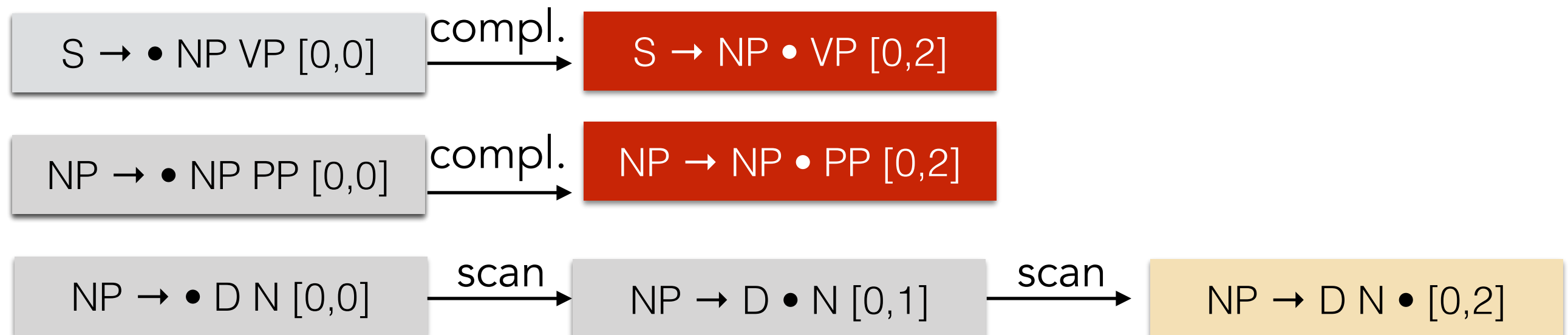
- When we visit an active item
 - add all states resulting from **predicts** to the agenda.



- When we visit a passive item, find any **matching active** item on the chart and add all states resulting from **completes** with to the agenda.

Parsing as State-Space Search

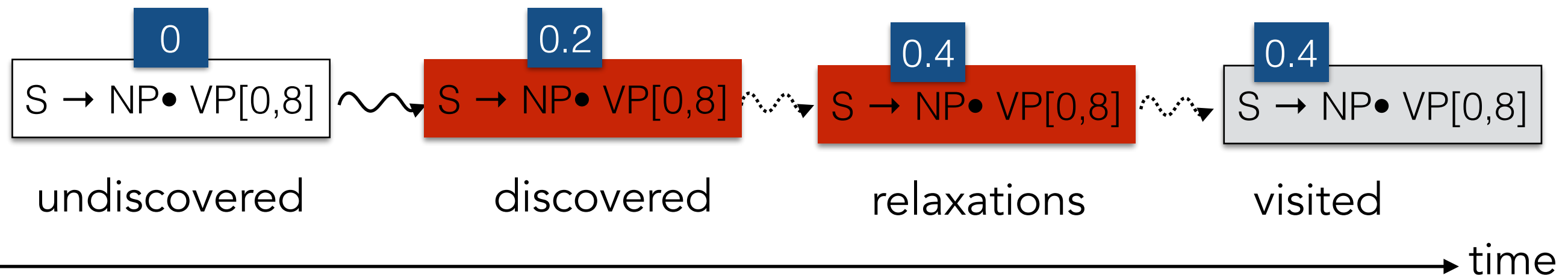
- When we visit an active item
 - add all states resulting from **predicts** to the agenda.
 - find any **matching** passive item and add states resulting from **completes** to the queue. (necessary if we don't use a queue)



- When we visit a passive item, find any **matching active** item on the chart and add all states resulting from **completes** with to the agenda.

Uniform Cost Search and PCFG Parsing

- shortest paths : highest probability subtree under this item.
- Use a priority queue (e.g. a heap) as agenda.
- Once we visit a state, we have found the highest subtree probability.



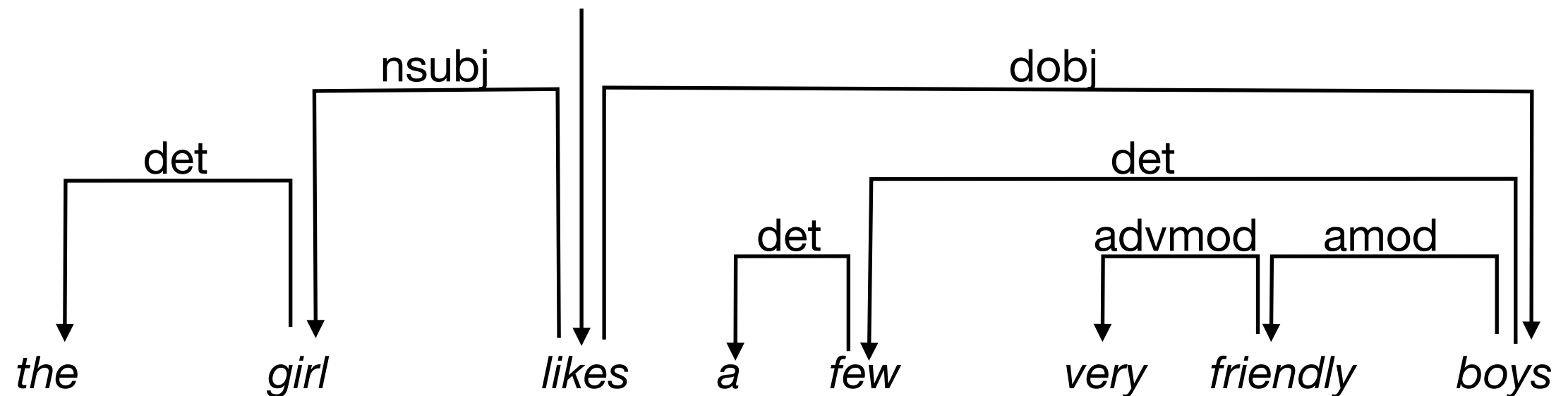
Natural Language Processing

Lecture 9 & 10: Dependency Parsing

02/14/2018 & 2/19/2018

COMS W4705
Daniel Bauer

Dependency Structure



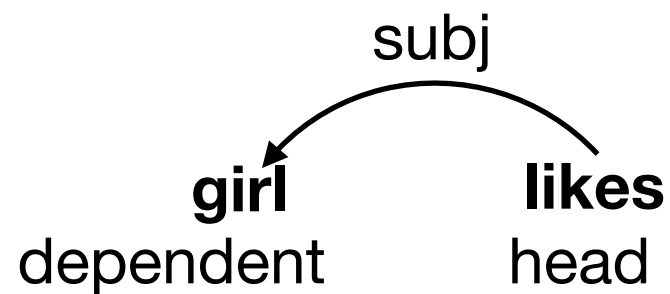
- The edges can be labeled with **grammatical relations** between words (typed dependencies):
 - Arguments (Subject, Object, Indirect Object, Prepositional Object)
 - Adjunct (Temporal, Locative, Causal, Manner...) / Modifier
 - Function words

Dependency Structure

- Long history in linguistics (Starting with Panini's Grammar of Sanskrit, 4th century BCE).
 - Modern dependency grammar originates with Tesnière and Melcuk.
- Different from phrase structure (but related via the concept of constituency and heads)
 - Focus is on grammatical relationships between words (Subject, Object, ...)
- Tighter connection to natural language semantics.

Dependency Relations

- Each dependency relation consists of a head and a dependent.



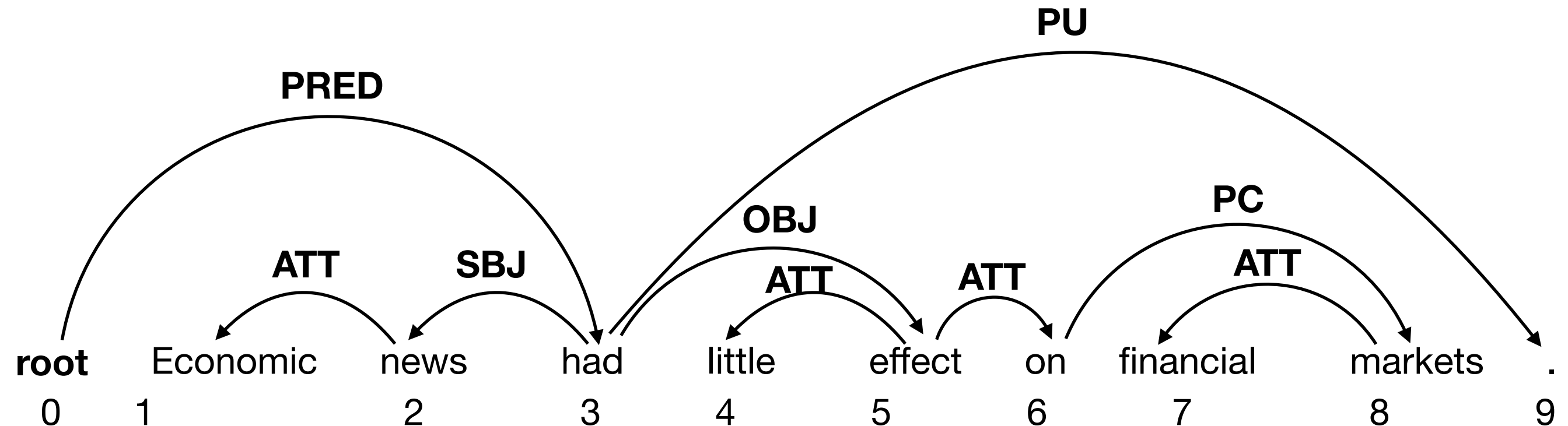
- Represent individual edges as *subj(likes-02, girl-01)*
- or as a triple (*likes, nsubj, girl*)
- And the entire sentence structure as a set of edges:

root(likes-2), subj(likes-2, girl-1), det(the-0, girl-1), obj(likes-2, boys-7),
det(boys-7, few-4), det(few-4, a-3), amod(boys-7, friendly-6), advmod(friendly-6, very-5)

Heads and Dependents

- How do we identify the the grammatical relation between head H and Dependent D (in a particular constituent C)?
 - H determines the syntactic category of C and can often replace C.
 - H determines the semantic category of C; D gives semantic specification.
 - H is obligatory; D may be optional.
 - H selects D and determines whether D is obligatory or optional.
 - The form of D depends on H (agreement or government).
 - The linear position of D is specified with reference to H.

Another Example

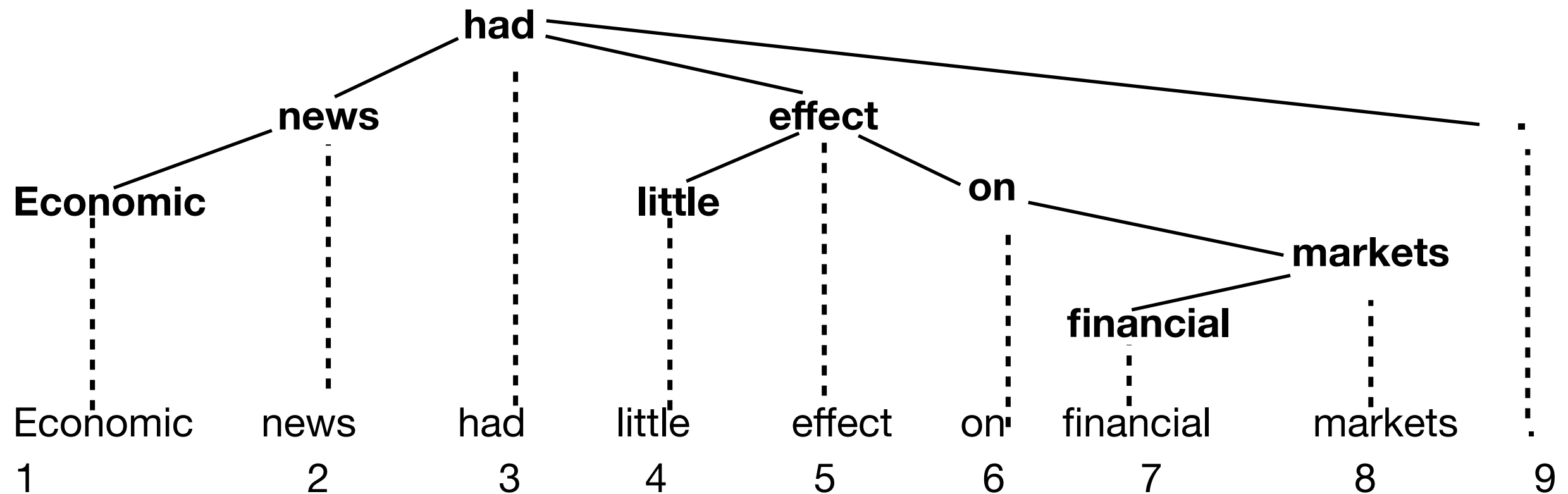


Dependency structure $G = (V_s, A)$

set of nodes $V_s = \{\mathbf{root}, \text{Economic}, \text{news}, \text{had}, \text{little}, \text{effect}, \text{on}, \text{financial}, \text{markets}, .\}$

set of edges/
arcs $A = \{(\mathbf{root}, \text{PRED}, \text{had}), (\text{had}, \text{SBJ}, \text{news}), (\text{had}, \text{OBJ}, \text{effect}), (\text{had}, \text{PU}, .),$
 $(\text{news}, \text{ATT}, \text{Economic}), (\text{effect}, \text{ATT}, \text{little}), (\text{effect}, \text{ATT}, \text{on}), (\text{on}, \text{PC}, \text{markets}),$
 $(\text{markets}, \text{ATT}, \text{financial})\}$

Another Example



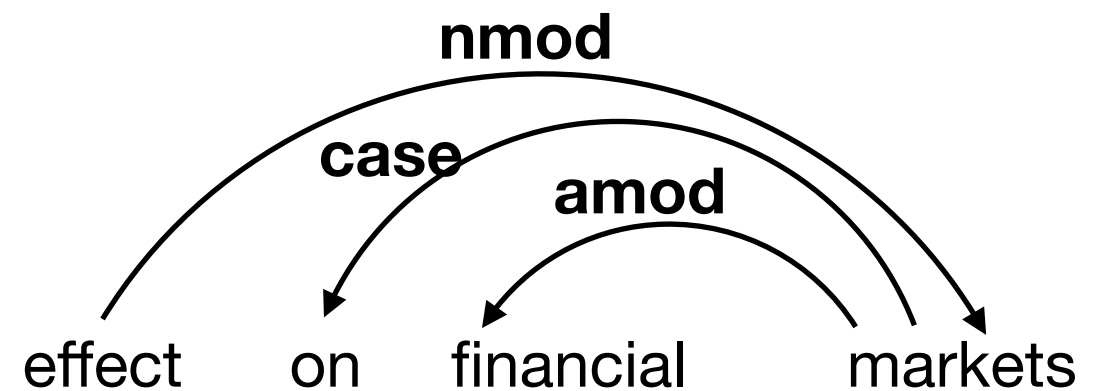
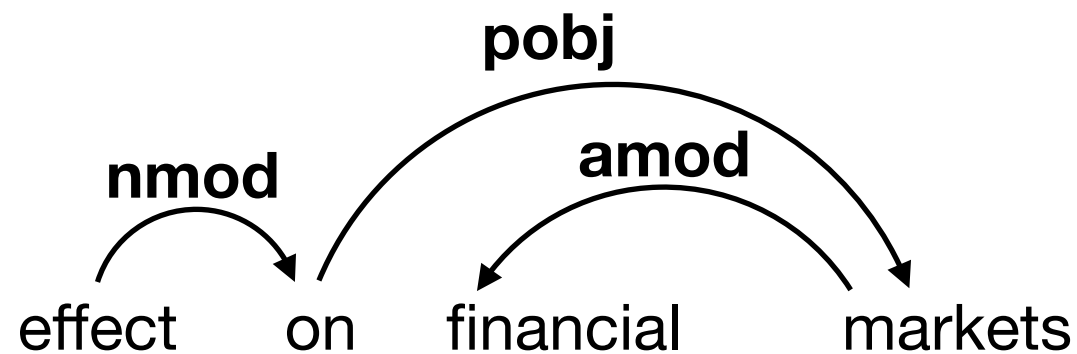
$G = (V, A)$

$V = \{\mathbf{root}, \text{Economic}, \text{news}, \text{had}, \text{little}, \text{effect}, \text{on}, \text{financial}, \text{markets}, .\}$

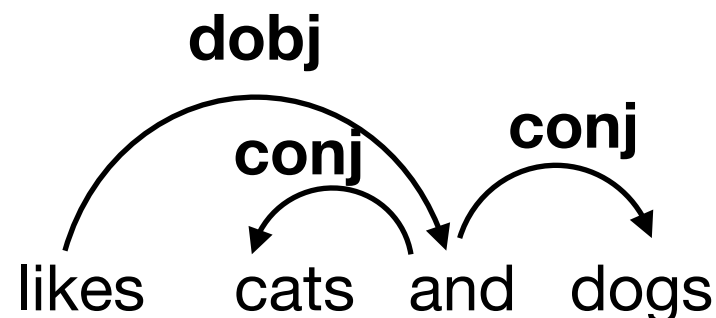
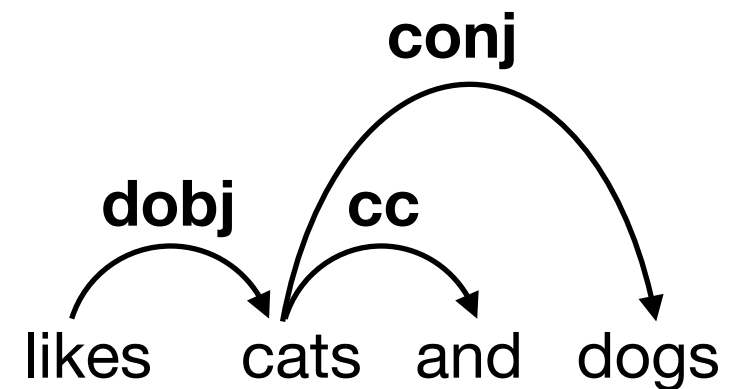
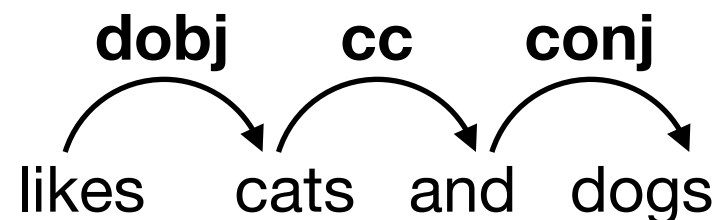
$A = \{(\text{root}, \text{PRED}, \text{had}), (\text{had}, \text{SBJ}, \text{news}), (\text{had}, \text{OBJ}, \text{effect}), (\text{had}, \text{PU}, .),$
 $(\text{news}, \text{ATT}, \text{Economic}), (\text{effect}, \text{ATT}, \text{little}), (\text{effect}, \text{ATT}, \text{on}), (\text{on}, \text{PC}, \text{markets}),$
 $(\text{markets}, \text{ATT}, \text{financial})\}$

Different Dependency Representations

- How to deal with prepositions?



- How to deal with conjunctions?



Inventory of Relations

"Universal Dependencies" (Marneffe *et al.* 2014)

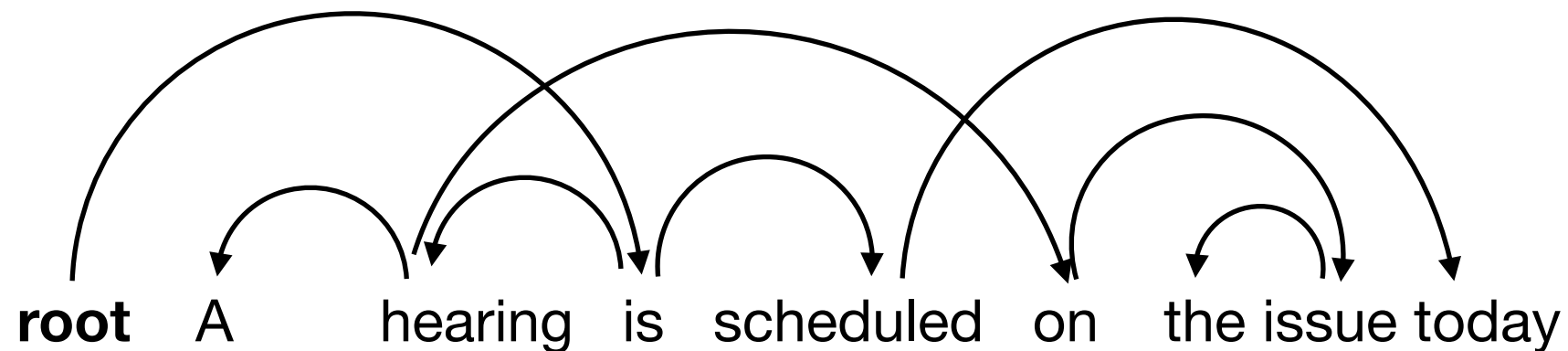
	Nominals	Clauses	Modifier words	Function Words
Core arguments	<u>nsubj</u> <u>obj</u> <u>iobj</u>	<u>csubj</u> <u>ccomp</u> <u>xcomp</u>		
Non-core dependents	<u>obl</u> <u>vocative</u> <u>expl</u> <u>dislocated</u>	<u>advcl</u>	<u>advmod</u> * <u>discourse</u>	<u>aux</u> <u>cop</u> <u>mark</u>
Nominal dependents	<u>nmod</u> <u>appos</u> <u>nummod</u>	<u>acl</u>	<u>amod</u>	<u>det</u> <u>clf</u> <u>case</u>
Coordination	MWE	Loose	Special	Other
<u>conj</u> <u>cc</u>	<u>fixed</u> <u>flat</u> <u>compound</u>	<u>list</u> <u>parataxis</u>	<u>orphan</u> <u>goeswith</u> <u>reparandum</u>	<u>punct</u> <u>root</u> <u>dep</u>

Dependency Trees

- Dependency structure is typically assumed to be a tree.
 - Root node 0 must not have a parent.
 - All other nodes must have exactly one parent.
 - The graph needs to be connected.
 - Nodes must not form a cycle.

Projectivity

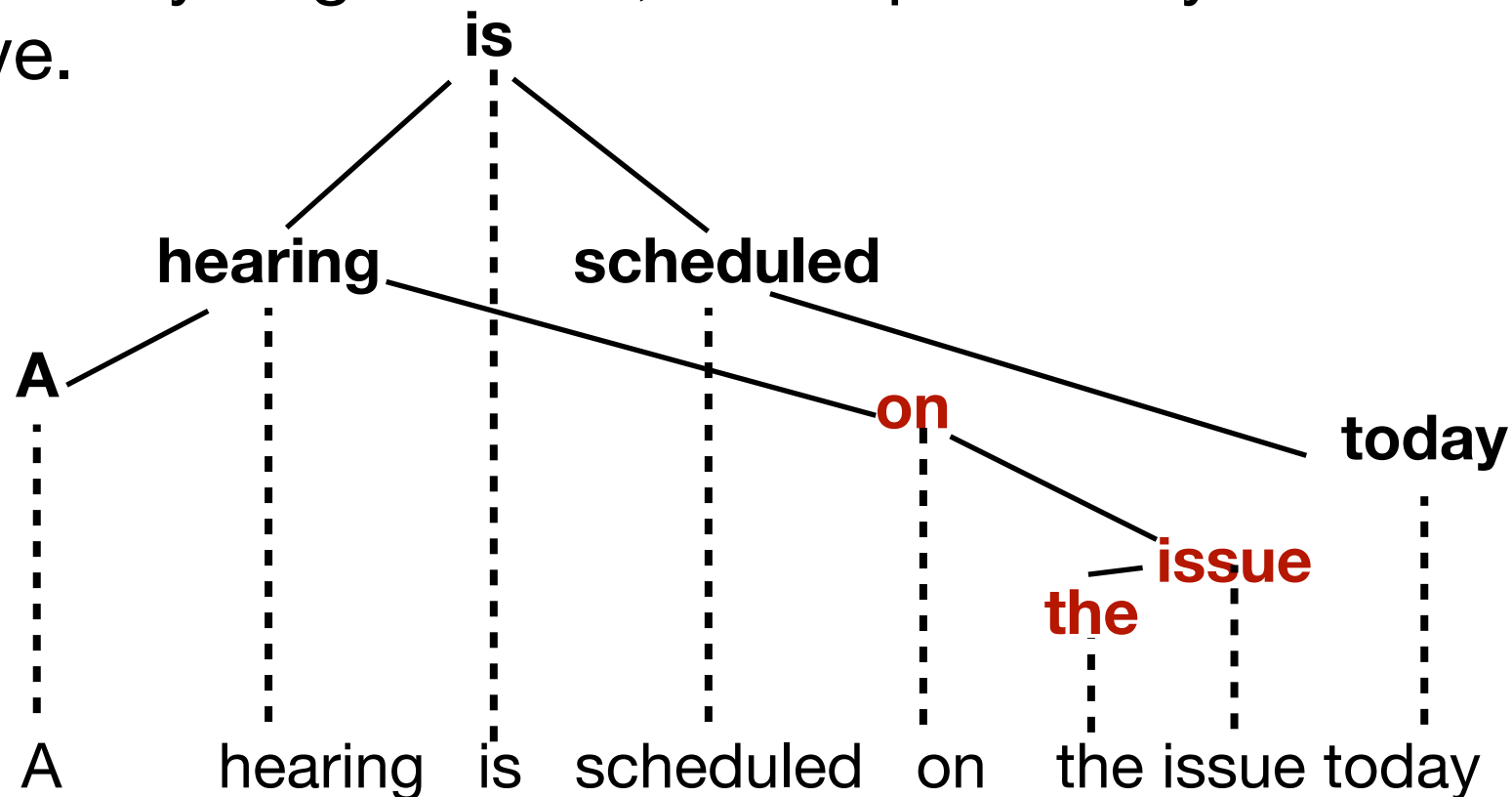
- Words in a sentence stand in a linear order.
- If dependency edges cross, the dependency structure is non-projective.



- Non-projective structures appear more frequently in some languages than others (Hungarian, German, ...)
- Some approaches to dependency parsing cannot handle non-projectivity.

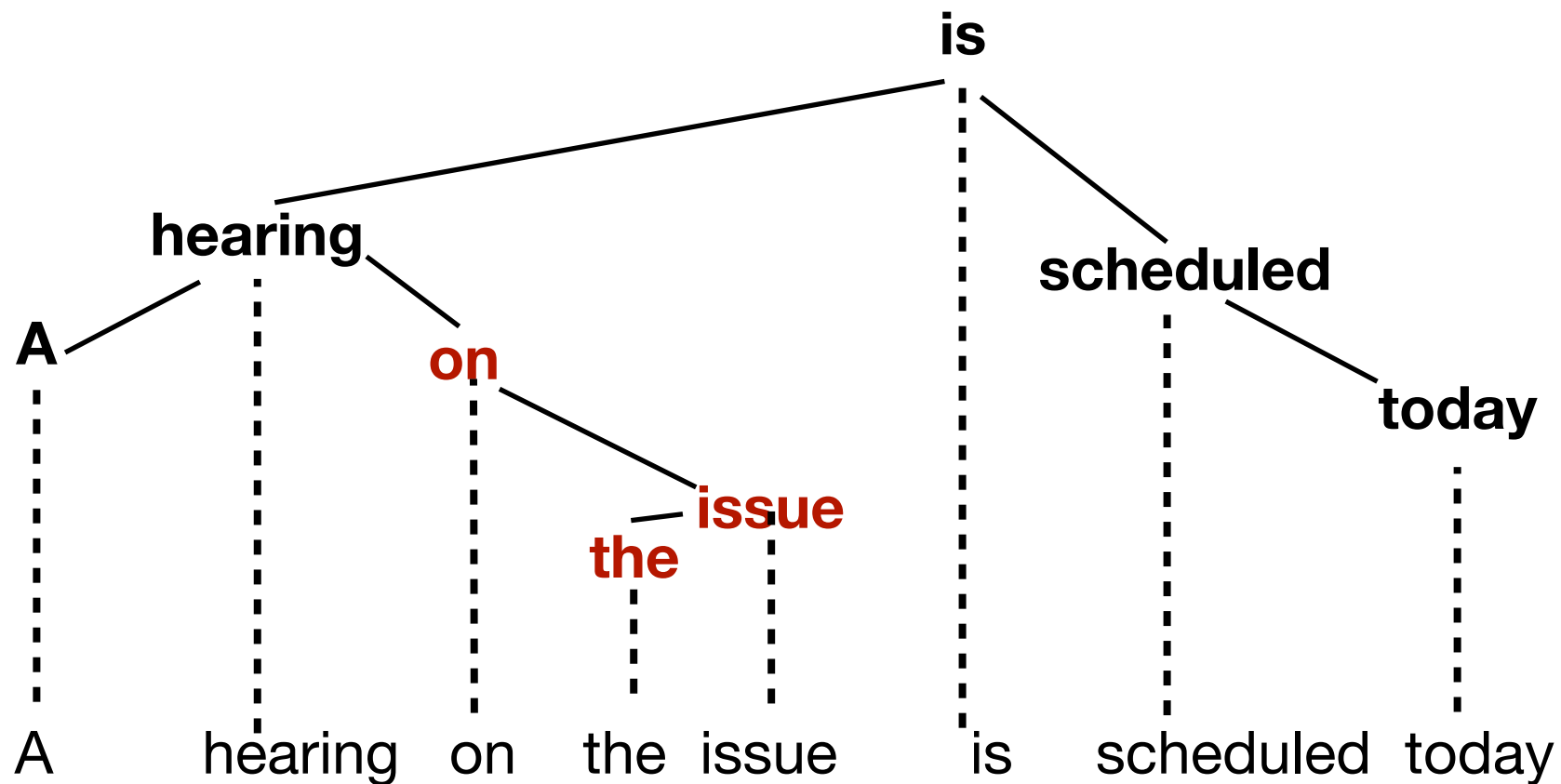
Projectivity

- Words in a sentence stand in a linear order.
- If dependency edges cross, the dependency structure is non-projective.



- Non-projective structures appear more frequently in some languages than others (Hungarian, German, ...)
- Some approaches to dependency parsing cannot handle non-projectivity.

Projectivity



An edge (i, r, j) in a dependency tree is projective if there is a directed path from i to k for all $i < k < j$ (if $i < j$) or $j < k < i$ ($j < i$).

Dependency Parsing

- Input:
 - a set of nodes $V_s = \{w_0, w_1, \dots, w_m\}$ corresponding to the input sentence $s = w_1, \dots, w_m$ (0 is the special **root** node)
 - an inventory of labels $R = \{\text{PRED}, \text{SBJ}, \text{OBJ}, \text{ATT}, \dots\}$
- Goal: Find a set of labeled, directed edges between the nodes, such that the resulting graph forms a **correct dependency tree** over V_s .

↑
structural constraints

Dependency Parsing

- What information could we use?
 - bi-lexical affinities
 - *financial markets, meeting... scheduled*
 - dependency distance (prefer close words?)
 - Intervening words
 - *had little effect, little gave effect*
 - subcategorization/valency of heads.

Subcategorization/Valency

- Verbs may take a different number of arguments of different syntactic types in different positions (applies to other types of words as well).
 - *The baby slept.* * *The baby slept the house.*
 - *He pretended to sleep.* * *He pretended the cat.*
 - *Godzilla destroyed the city.* * *Godzilla destroyed.*
 - *Jenny gave the book to Carl.* * *Jenny gave the book.*
 - ... examples for *ask, promise, bet, load,...*

Dependency Parsing

- As with other NLP problems, we can think of dependency parsing as a kind of search problem:
 - Step 1: Define the space of possible analyses for a sentence
 - Step 2: Select the best analysis from this search space.
- Need to define the search space, search algorithm, and a way to determine the "best" parse.

Dependency Parsing

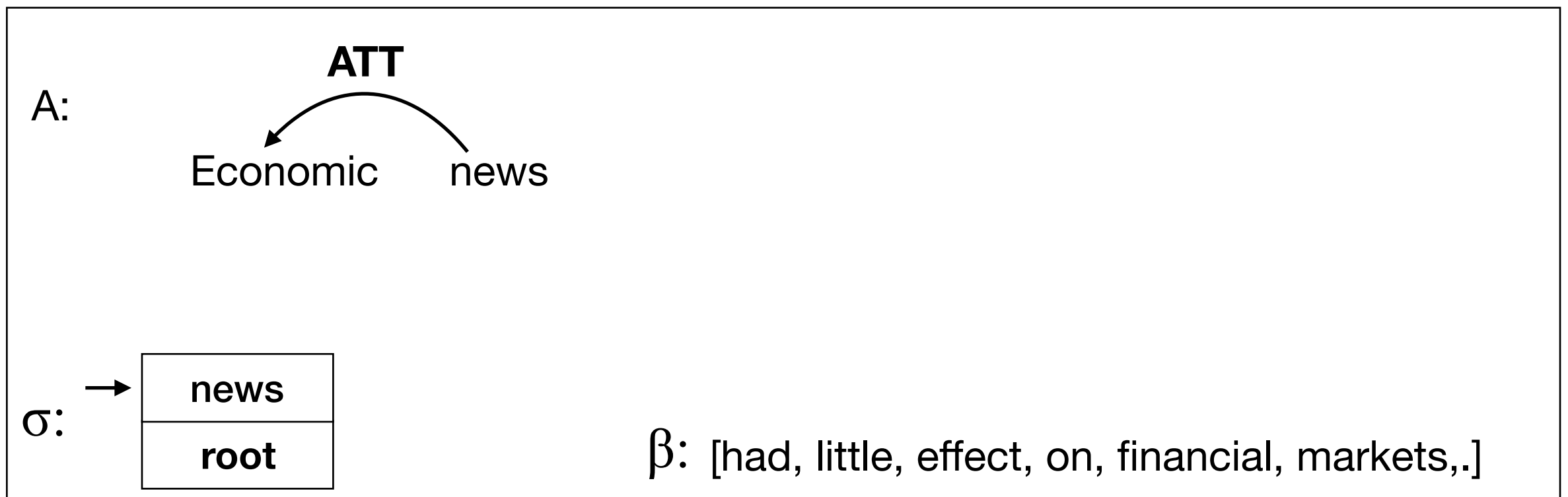
- Approaches to Dependency Parsing:
 - Grammar-based
 - Data-based
 - Dynamic Programming (e.g. Eisner 1996,)
 - Graph Algorithms (e.g. McDonald 2005, MST Parser)
 - **Transition-based (e.g. Nivre 2003, MaltParser)**
 - Constraint satisfaction (Karlsson 1990)

Transition-Based Dependency Parsing

- Defines the search space using parser states (configurations) and operations on these states (transitions).
- Start with an initial configuration and find a sequence of transitions to the terminal state.
- Uses a greedy approach to find the best sequence of transitions.
- Uses a discriminative model (classifier) to select the next transition.

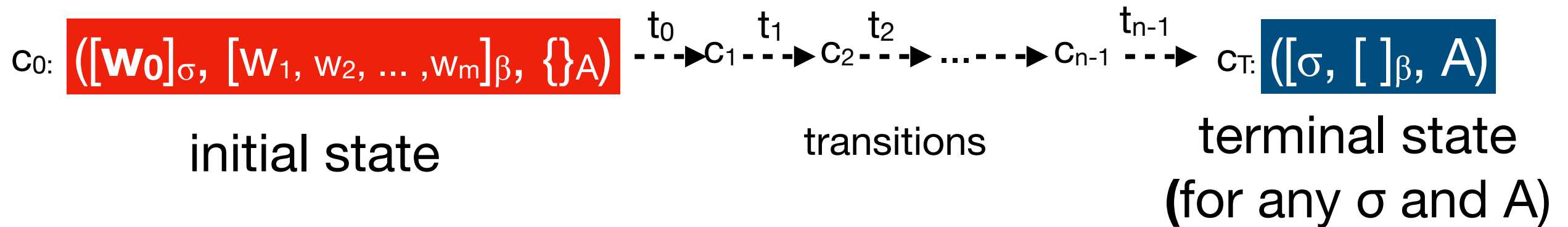
Transition-Based Parsing - States

- A parser state (configuration) is a triple $c = (\sigma, \beta, A)$
 - σ - is a **stack** of words $w_i \in V_S$
 - β - is a **buffer** of words $w_i \in V_S$
 - A - is a set of dependency arcs (w_i, r, w_j)



$([\mathbf{root}, \mathbf{news}]_\sigma, [\text{had, little, effect, on, financial, markets,.}]_\beta, \{ (\text{news}, \text{ATT}, \text{Economic}) \}_A$

Transition-Based Parsing - initial and terminal state



- Start with initial state c_0 .
- Apply sequence of transitions, t_0, \dots, t_{n-1} .
- Once a terminal state C_T is reached, return final parse A from state C_T .

Transition-Based Parsing - Transitions ("Arc-Standard")

- **Shift:**

Move next word from the buffer to the stack

$$(\sigma, w_i \mid \beta, A) \Rightarrow (\sigma \mid w_i, \beta, A)$$

- **Left-Arc (for relation r):**

Build an edge from the next word on the buffer to the top word on the stack.

$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_j \mid \beta, A \cup \{w_j, r, w_i\})$$

- **Right-Arc (for relation r)**

Build an edge from the top word on the stack to the next word on the buffer.

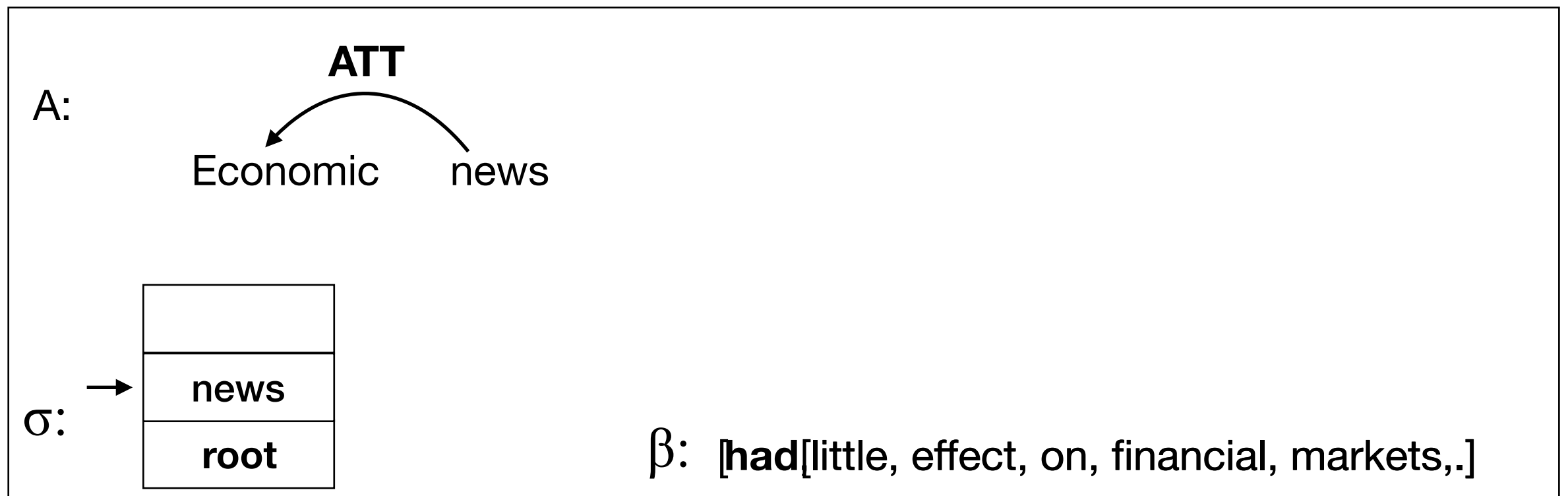
$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_i \mid \beta, A \cup \{w_i, r, w_j\})$$

Transition-Based Parsing - Transitions

- **Shift**

Move next word from the buffer to the stack

$$(\sigma, w_i \mid \beta, A) \Rightarrow (\sigma \mid w_i, \beta, A)$$



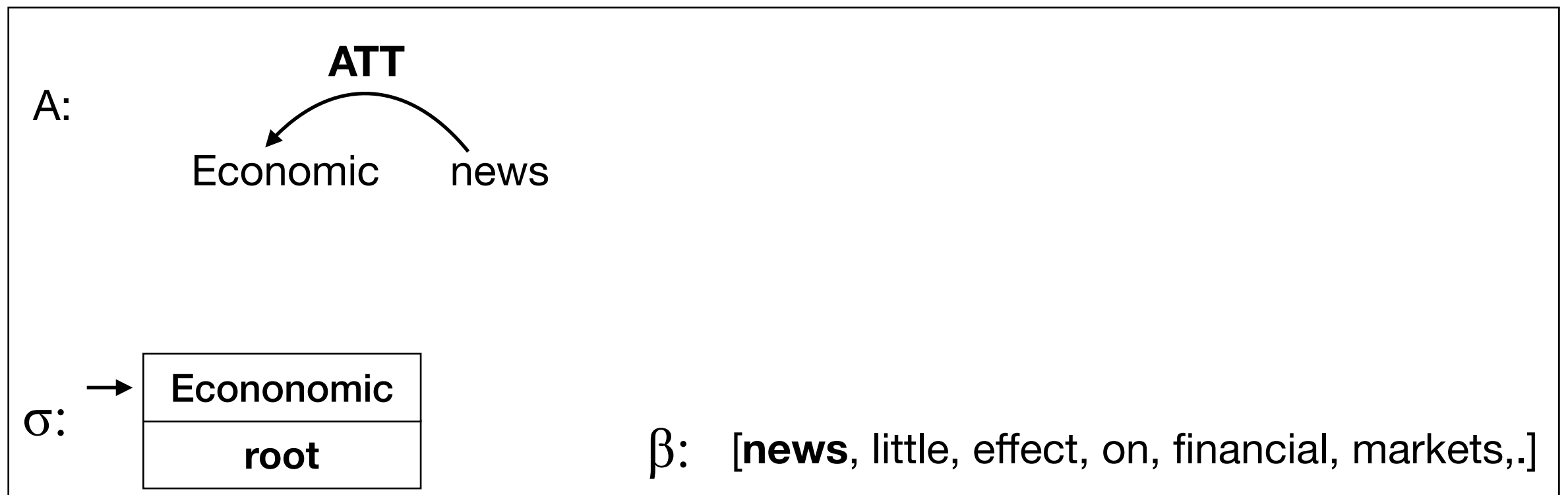
Transition-Based Parsing - Transitions

- **Arc-left_r**

Build an edge from the next word on the buffer to the top word on the stack.

$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_j \mid \beta, A \cup \{w_j, r, w_i\})$$

Not allowed if $i=0$ (root may not have a parent)



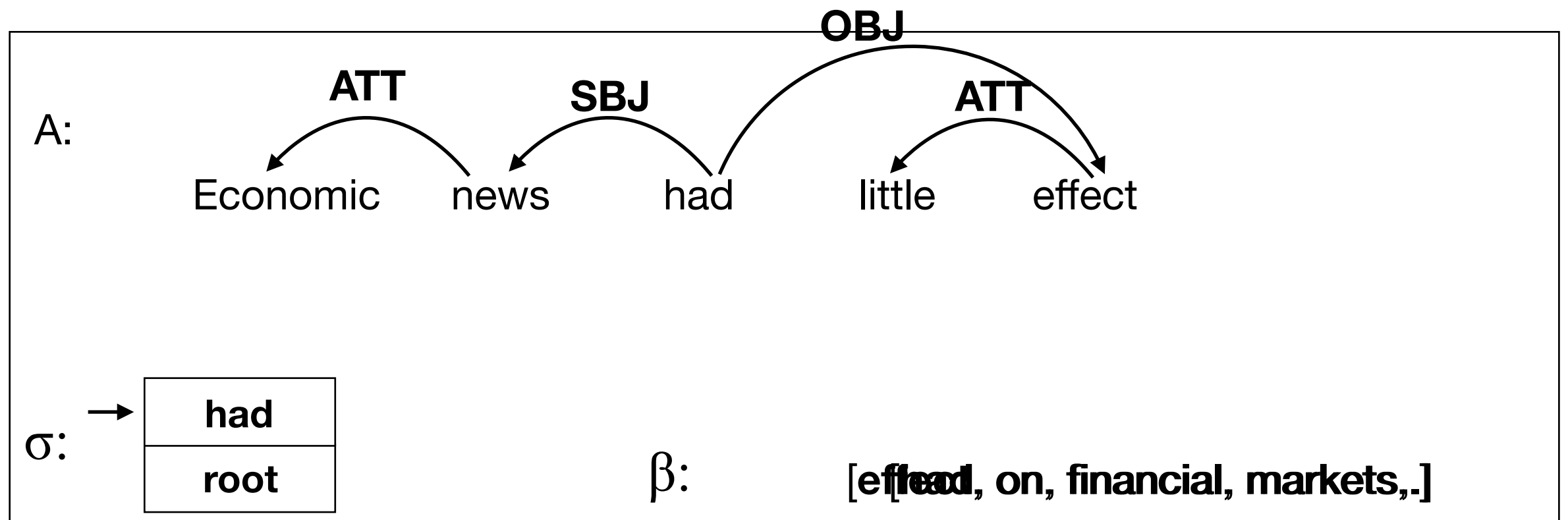
note: w_j remains in the buffer

Transition-Based Parsing - Transitions

- **Arc-right_r**

Build an edge from the next word on the buffer to the top word on the stack.

$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_i \mid \beta, A \cup \{w_j, r, w_i\})$$

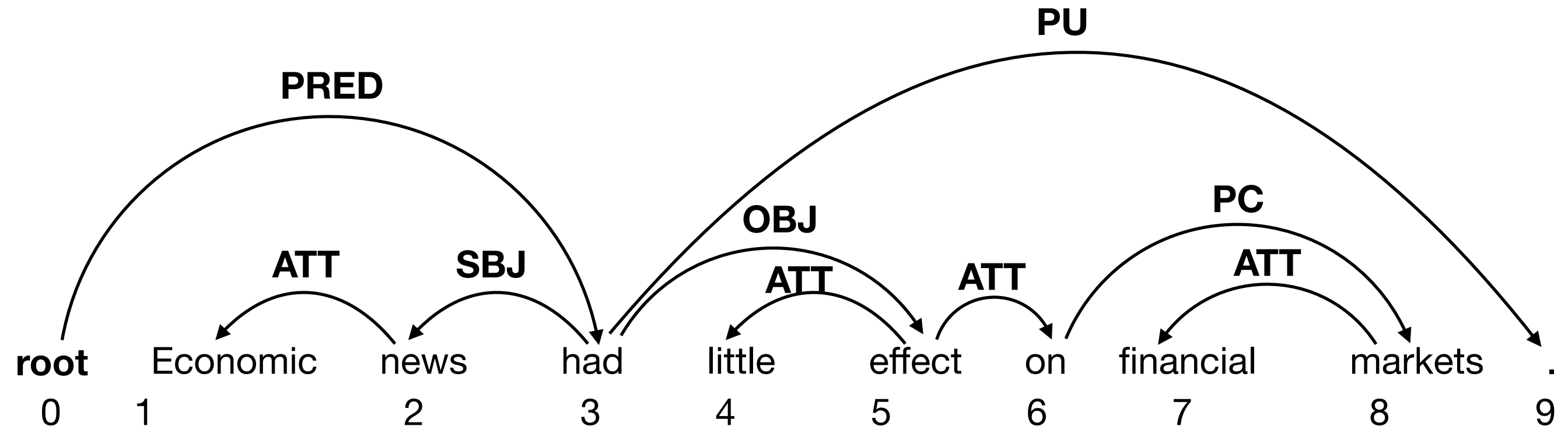


note: w_i is moved from the top of the stack back to the buffer!

Transition-Based Parsing - Some Observations

- Does the transition system contain dead ends? (states from which a terminal state cannot be reached)? No!
- What is the role of the buffer?
 - Contains words that can become dependents of a right-arc.
Keep unseen words.
- What is the role of the stack?
 - Keep track of nodes that can become dependents of a left-arc.
- Once a word disappears from the buffer and the stack it cannot be part of any further edge!

Another Example



$$G = (V_s, A)$$

$V_s = \{\mathbf{root}, \text{Economic}, \text{news}, \text{had}, \text{little}, \text{effect}, \text{on}, \text{financial}, \text{markets}, \text{.}\}$

$A = \{(\text{root}, \text{PRED}, \text{had}), (\text{had}, \text{SBJ}, \text{news}), (\text{had}, \text{OBJ}, \text{effect}), (\text{had}, \text{PU}, \text{.}),$
 $(\text{news}, \text{ATT}, \text{Economic}), (\text{effect}, \text{ATT}, \text{little}), (\text{effect}, \text{ATT}, \text{on}), (\text{on}, \text{PC}, \text{markets}),$
 $(\text{markets}, \text{ATT}, \text{financial})\}$

Transition-Based Parsing - Complete Example

initial state

next transition: shift

A:

σ :

root

β : [Economic, news, had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next-transition: Left-Arc_{ATT}

A:

σ :

Economic
root

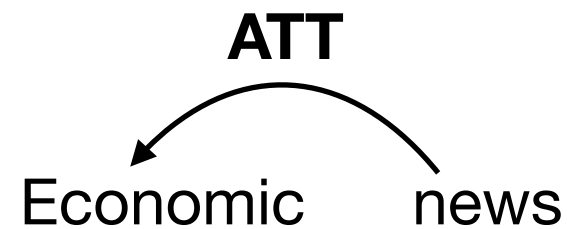
β :

[news, had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ :

root

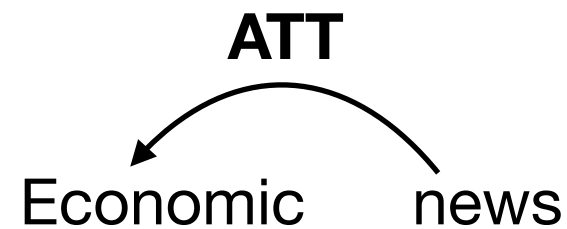
β :

[news, had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: **Left-Arc_{SBJ}**

A:



σ :

news
root

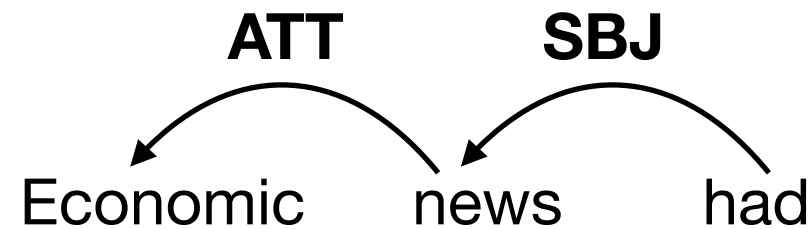
β :

[had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ :

root

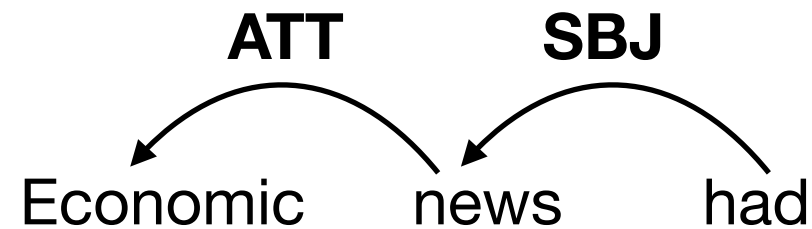
β :

[had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ :

had
root

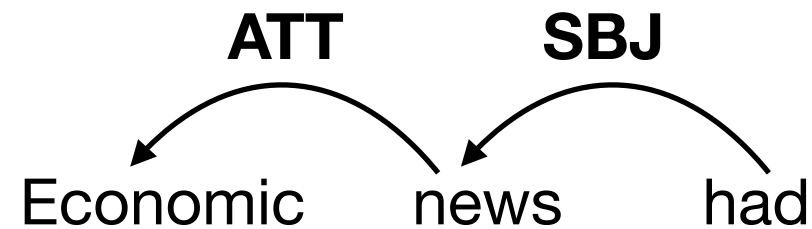
β :

[little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: Left-Arc_{SBJ}

A:



σ :

little
had
root

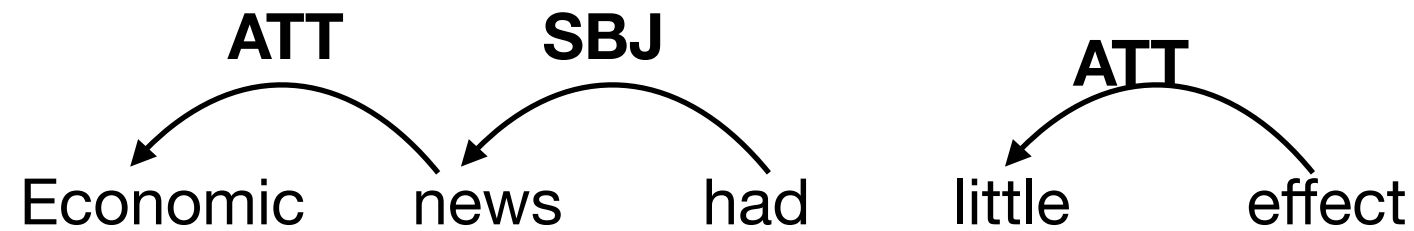
β :

[effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ :

had
root

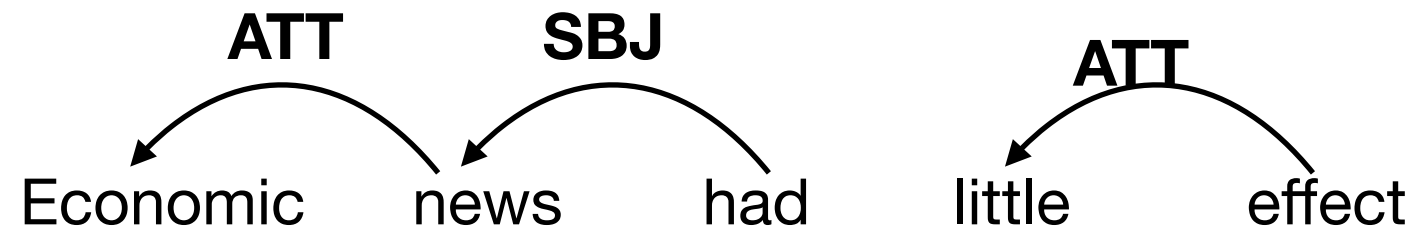
β :

[effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ :

effect
had
root

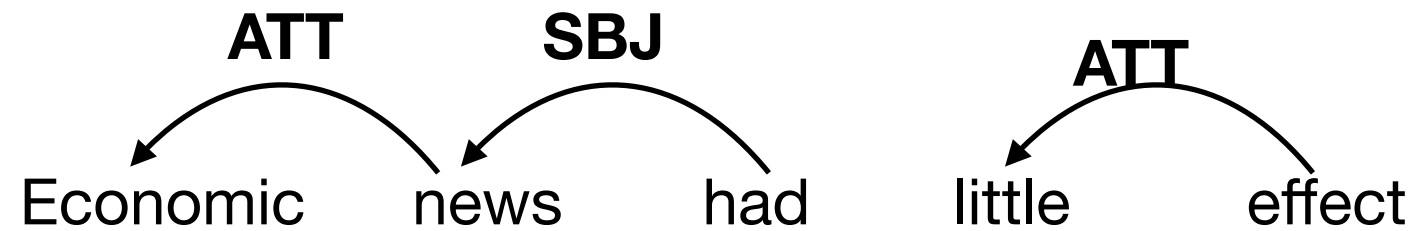
β :

[on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ :

on
effect
had
root

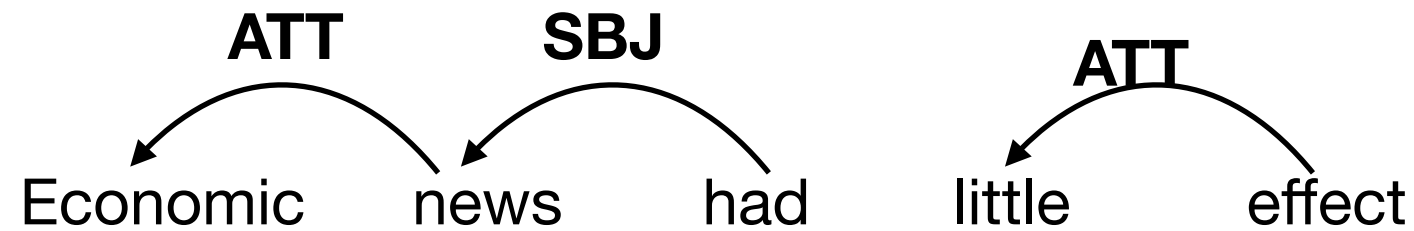
β :

[financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: Left-Arc_{ATT}

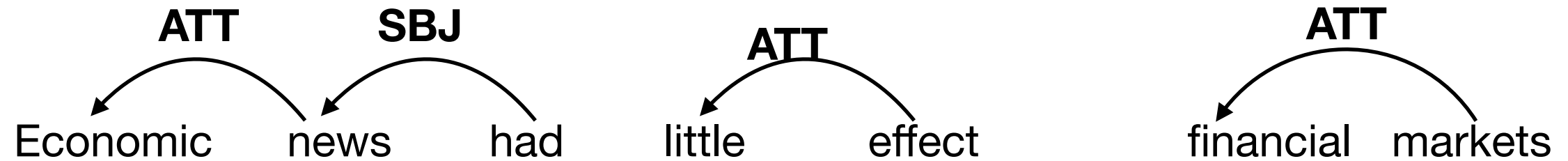
A:



Transition-Based Parsing - Oracle Example

next transition: Right-Arc_{PC}

A:



σ :

on
effect
had
root

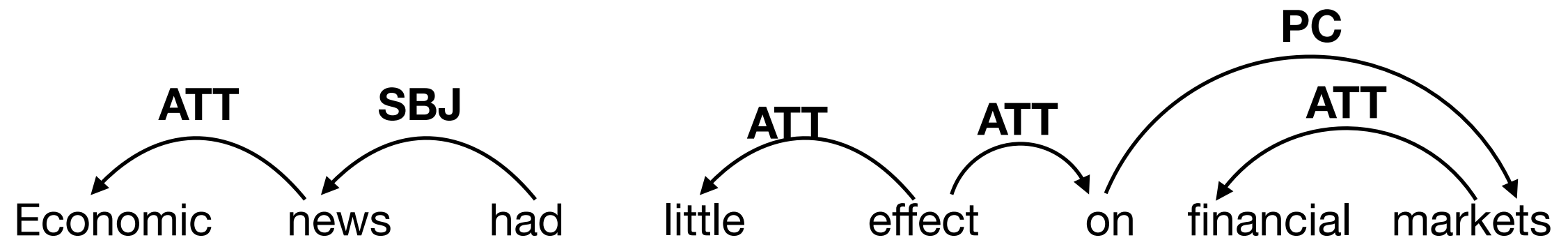
β :

[markets,.]

Transition-Based Parsing - Oracle Example

next transition: **Right-Arc_{OBJ}**

A:



σ :

had
root

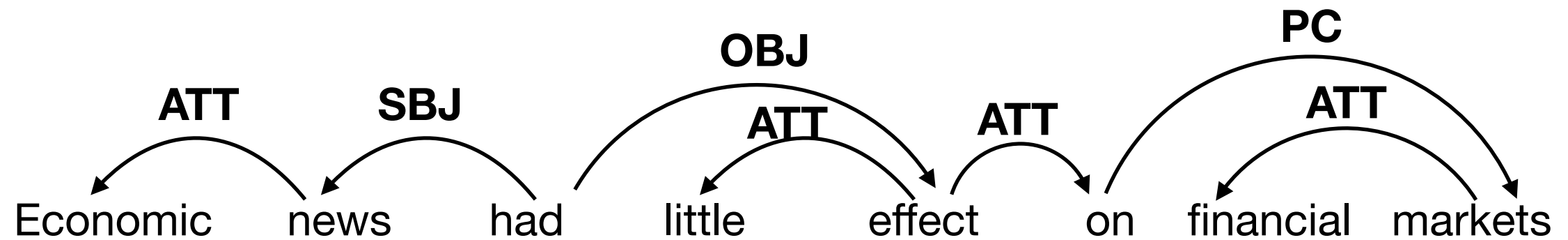
β :

[effect,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ :

root

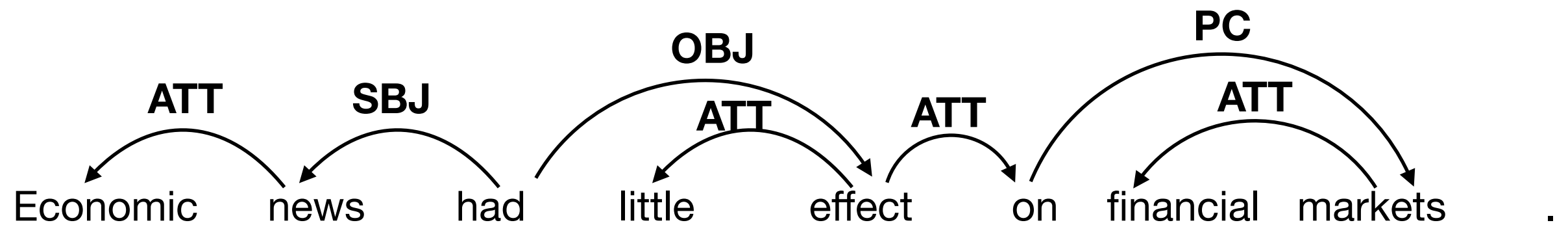
β :

[had,.]

Transition-Based Parsing - Oracle Example

next transition: **Right-Arc_{PU}**

A:



σ :

had
root

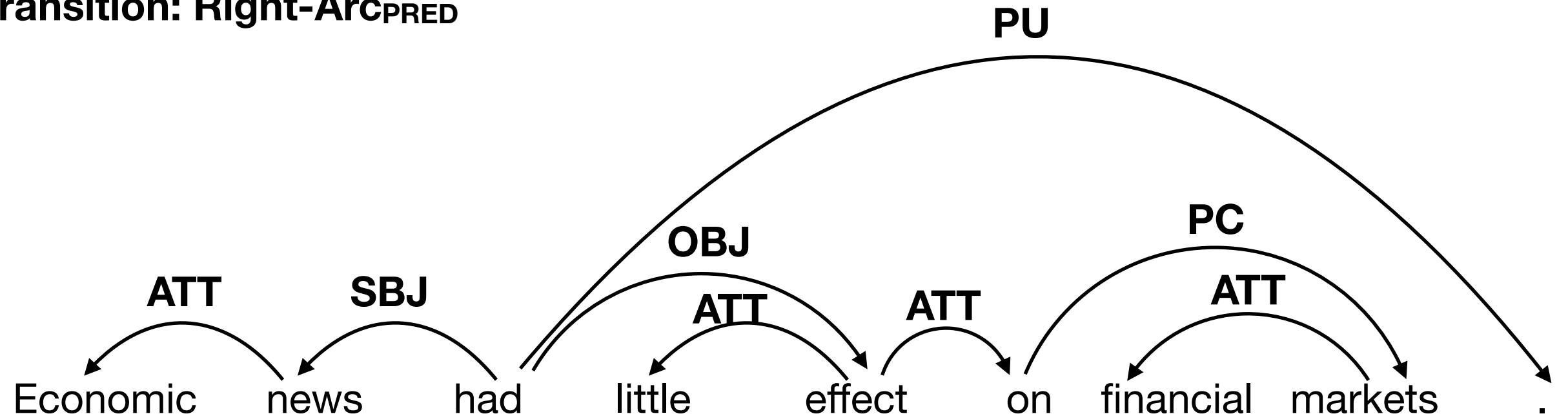
β :

[.]

Transition-Based Parsing - Oracle Example

next transition: Right-Arc_{PRED}

A:



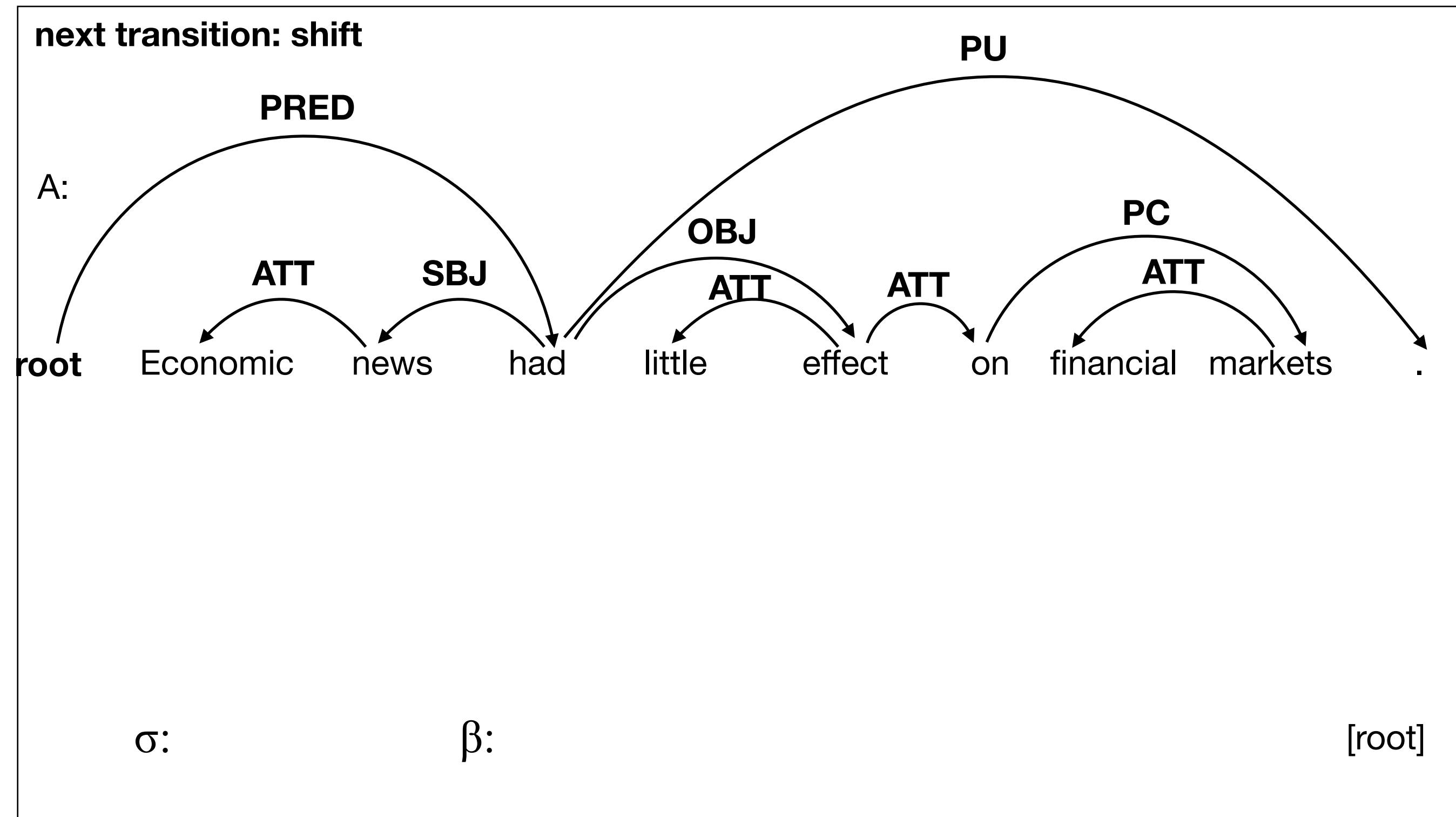
σ :

root

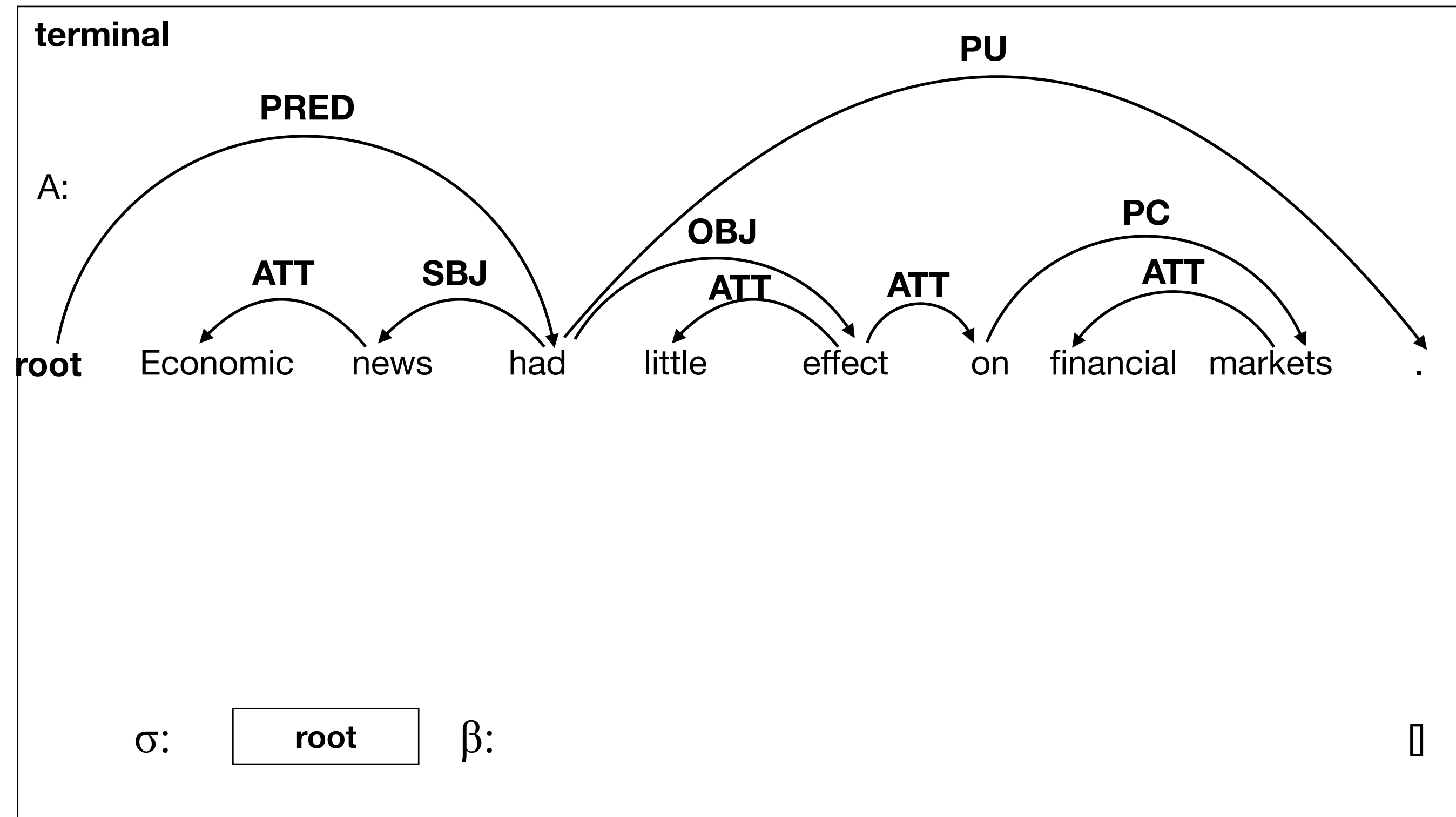
β :

[had]

Oracle Example



Transition-Based Parsing - Oracle Example



Properties of the transition system

- The time required to parse w_1, \dots, w_m with an oracle is $O(m)$. Why?
- Bottom-up approach: A node must *collect* all its children before its parent. Why?
- Can only produce projective trees. Why?
- This algorithm is complete (all projective trees over w_1, \dots, w_m can be produced by some sequence of transitions)
- Soundness: All terminal structures are projective forests (but not necessarily trees)

Deciding the Next Transition

- Instead of the unrealistic oracle, predict the next transition (and relation label) using a discriminative classifier.
 - Could use perceptron, log linear model, SVM, Neural Network, ...
 - This is a greedy approach (could use beam-search too).
 - If the classifier takes $O(1)$, the runtime for parsing is still $O(m)$ for m words.
- Questions:
 - What features should the classifier use?
Local features from each state, but ideally want to model entire history of transitions leading to the state.
 - How to train the model?

Extracting Features

- Need to define a feature function that maps states to feature vectors.
- Each feature consists of:
 1. an address in the state description:
(identifies a specific word in the configuration, for example "top of stack").
 2. an attribute of the word in that address:
(for example POS, word form, lemma, word embedding, ...)

Example Features

Table 3.2: Typical feature model for transition-based parsing with rows representing address functions, columns representing attribute functions, and cells with + representing features.

Address	Attributes				
	FORM	LEMMA	POSTAG	FEATS	DEPREL
STK[0]	+	+	+	+	
STK[1]			+		
LDEP(STK[0])					+
RDEP(STK[0])					+
BUF[0]	+	+	+	+	
BUF[1]	+		+		
BUF[2]			+		
BUF[3]			+		
LDEP(BUF[0])					+
RDEP(BUF[0])					+

Training the Model

- Training data: Manually annotated (dependency) treebank
 - *Prague Dependency Treebank*
English/Czech parallel data, dependencies for full PTB WSJ.
 - *Universal Dependencies Treebank*
Treebanks for more than 80 languages (varying in size)
(<http://universaldependencies.org/>)
- Problem: We have not actually seen the transition sequence, only the dependency trees!
- Idea: Construct oracle transition sequences from the dependency tree.
Train the model on these transitions.

Constructing Oracle Transitions

- Start with initial state $([\mathbf{w}_0]_\sigma, [w_1, w_2, \dots, w_m]_\beta, \{\}_A)$.
- Then predict the next transition using the annotated dependency tree A_d

$$o(c = (\sigma, \beta, A)) = \begin{cases} \text{LEFT-ARC}_r & \text{if } (\beta[0], r, \sigma[0]) \in A_d \\ \text{RIGHT-ARC}_r & \text{if } (\sigma[0], r, \beta[0]) \in A_d \text{ and, for all } w, r', \\ & \text{if } (\beta[0], r', w) \in A_d \text{ then } (\beta[0], r', w) \in A \\ \text{SHIFT}_r & \text{otherwise} \end{cases}$$

"Arc-Standard" Transitions

- **Shift:**

Move next word from the buffer to the stack

$$(\sigma, w_i \mid \beta, A) \Rightarrow (\sigma \mid w_i, \beta, A)$$

- **Left-Arc (for relation r):**

Build an edge from the next word on the buffer to the top word on the stack.

$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_j \mid \beta, A \cup \{w_j, r, w_i\})$$

- **Right-Arc (for relation r)**

Build an edge from the top word on the stack to the next word on the buffer.

$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_i \mid \beta, A \cup \{w_i, r, w_j\})$$

"Arc-Eager" Transitions

- **Shift:**

Move next word from the buffer to the stack

$$(\sigma, w_i \mid \beta, A) \Rightarrow (\sigma \mid w_i, \beta, A)$$

- **Left-Arc (for relation r):**

Build an edge from the next word on the buffer to the top word on the stack.

$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_j \mid \beta, A \cup \{(w_j, r, w_i)\})$$

Precondition: $(w_j, *, w_i)$ is not yet in A .

- **Right-Arc (for relation r)**

Build an edge from the top word on the stack to the next word on the buffer.

$$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma \mid w_i \mid w_j, \beta, A \cup \{w_i, r, w_j\})$$

- **Reduce**

Remove a completed node from the stack.

$$(\sigma \mid w_i, \beta, A) \Rightarrow (\sigma, \beta, A)$$

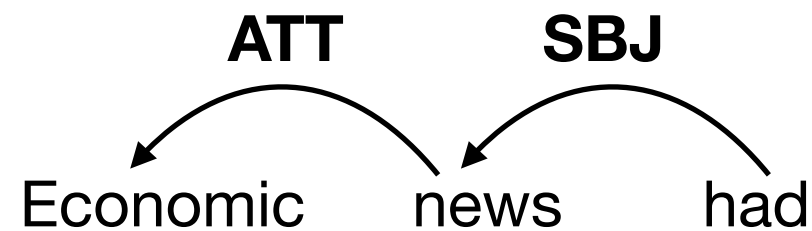
Precondition: there is some $(*, *, w_i)$ in A .

Arc-Eager Example

next transition: $\text{RightArc}_{\text{pred}}$

Can immediately attach *had* to root.

A:



σ :

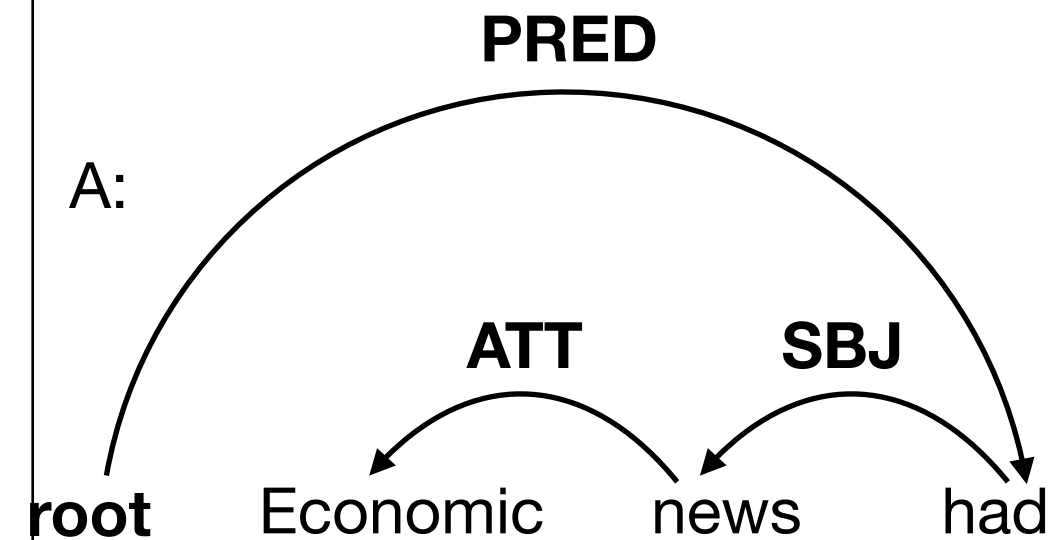
root

β :

[had, little, effect, on, financial, markets,.]

Arc-Eager Example

next transition: shift



σ :

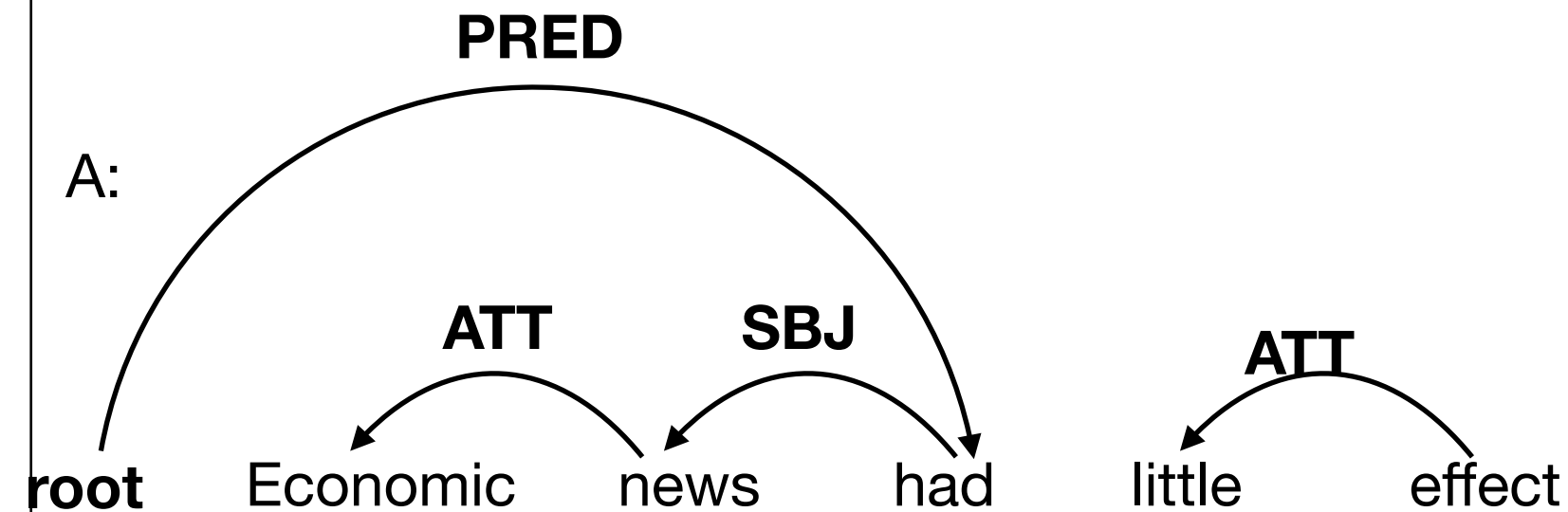
had
root

β :

[little, effect, on, financial, markets,.]

Arc-Eager Example

next transition: LeftArc_{ATT}



σ :

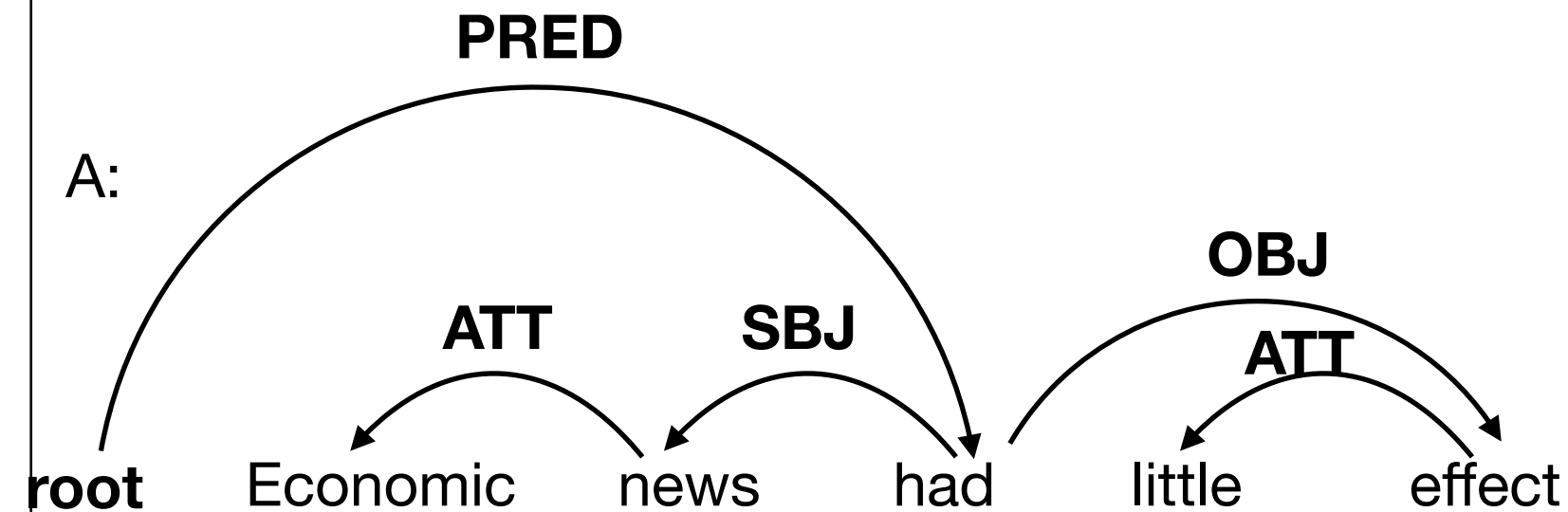
little
had
root

β :

[effect, on, financial, markets,.]

Arc-Eager Example

next transition: $\text{RightArc}_{\text{OBJ}}$



σ :

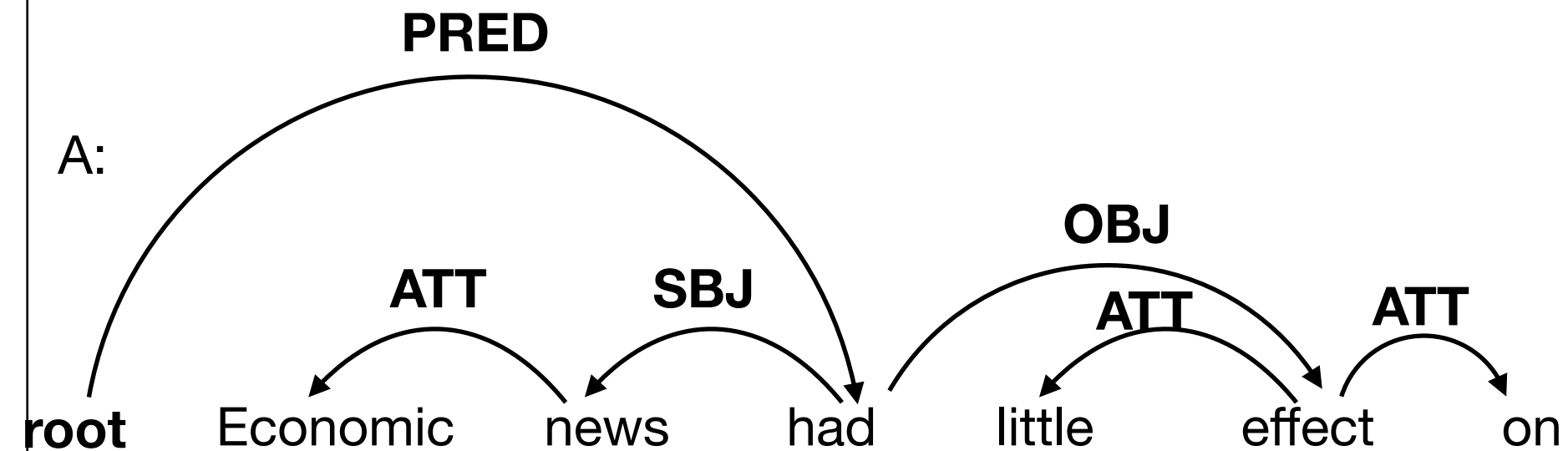
effect
little
had
root

β :

[on, financial, markets,.]

Arc-Eager Example

next transition: shift



σ :

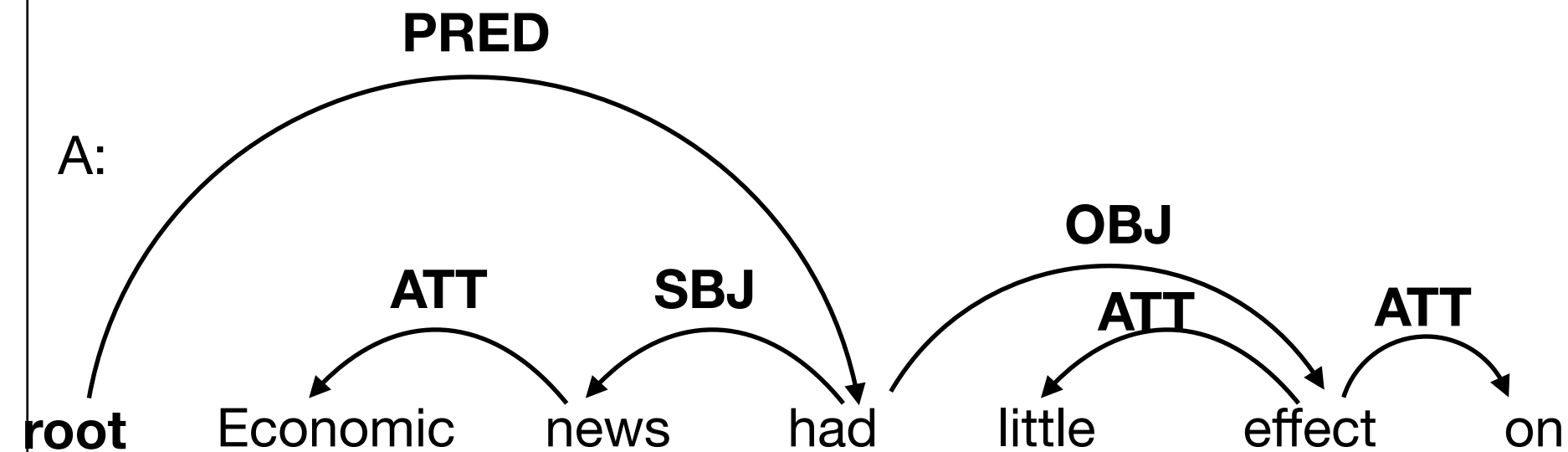
on
effect
little
had
root

β :

[financial, markets,.]

Arc-Eager Example

next transition: LeftArc_{ATT}



σ :

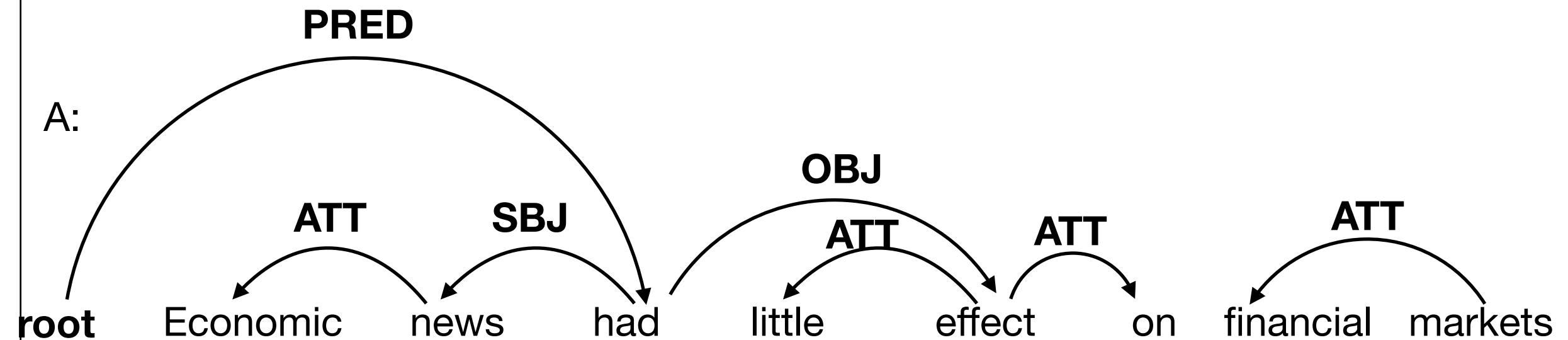
financial
on
effect
little
had
root

β :

[markets,.]

Arc-Eager Example

next transition: RightArc_{PC}



σ :

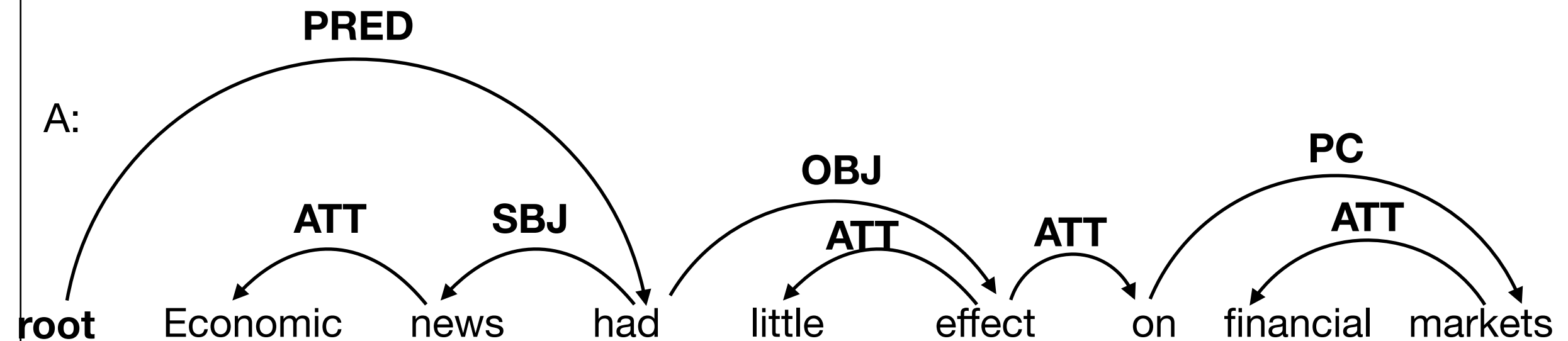
on
effect
little
had
root

β :

[markets,.]

Arc-Eager Example

next transition: Reduce



σ :

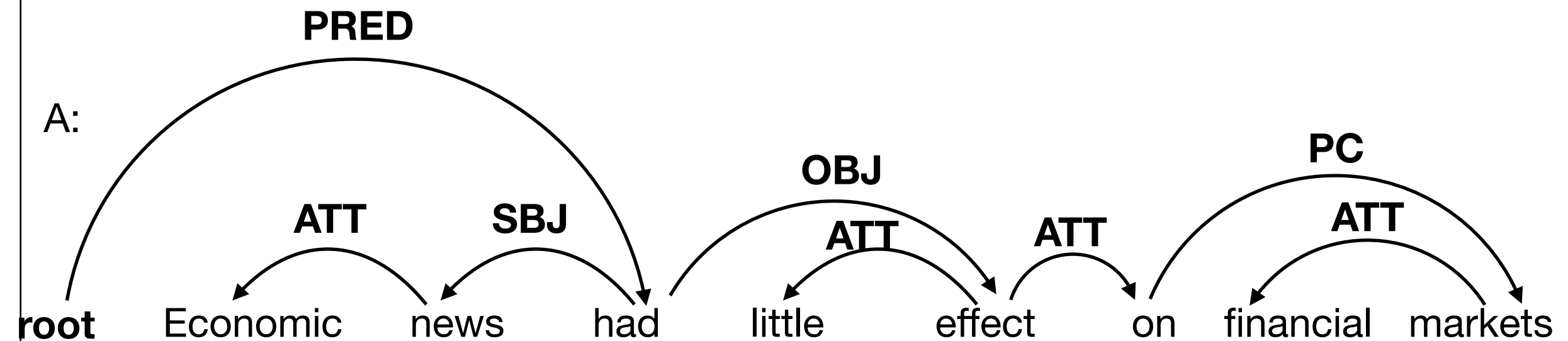
markets
on
effect
little
had
root

β :

[.]

Arc-Eager Example

next transition: Reduce



σ :

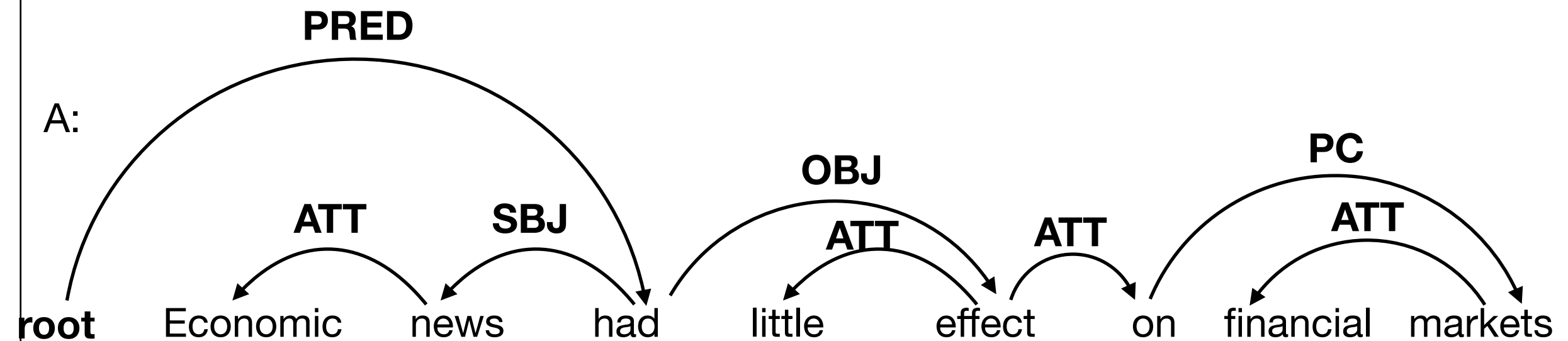
on
effect
little
had
root

β :

[.]

Arc-Eager Example

next transition: Reduce



σ :

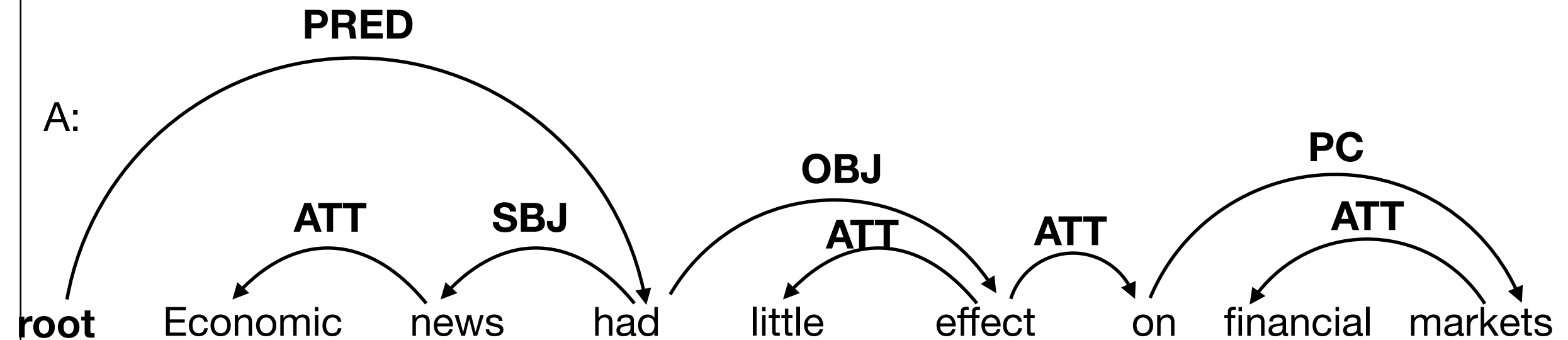
effect
little
had
root

β :

[.]

Arc-Eager Example

next transition: Reduce



σ :

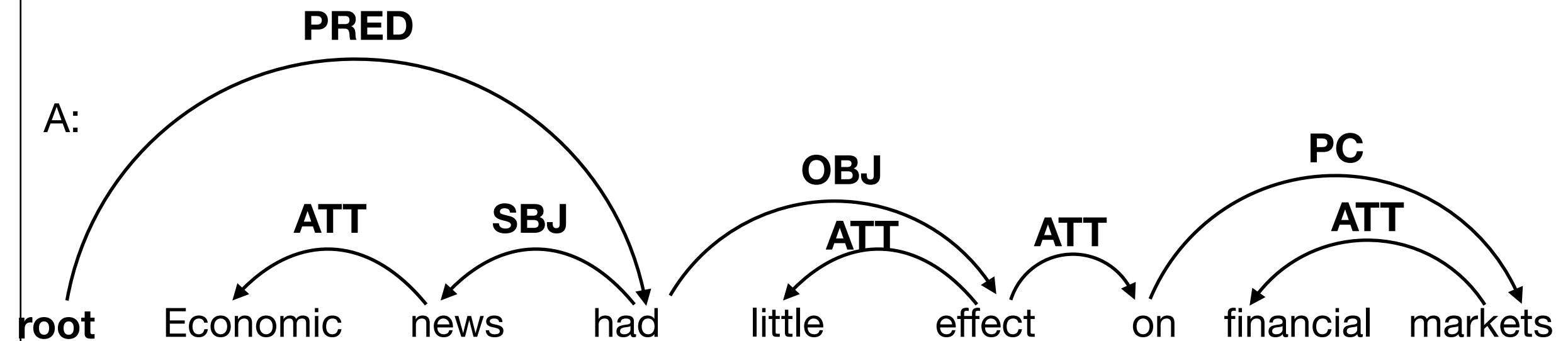
little
had
root

β :

[.]

Arc-Eager Example

next transition: Reduce



σ :

had
root

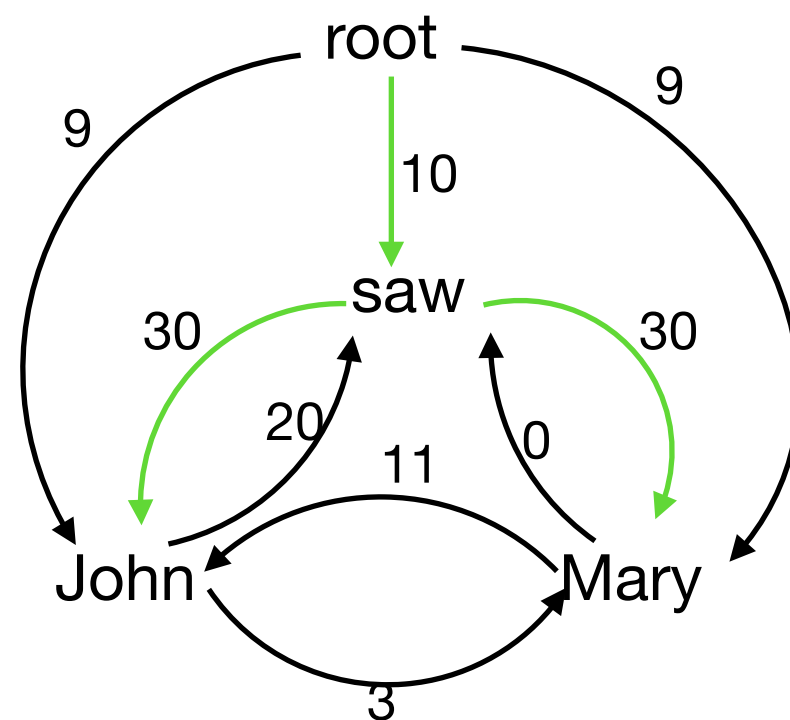
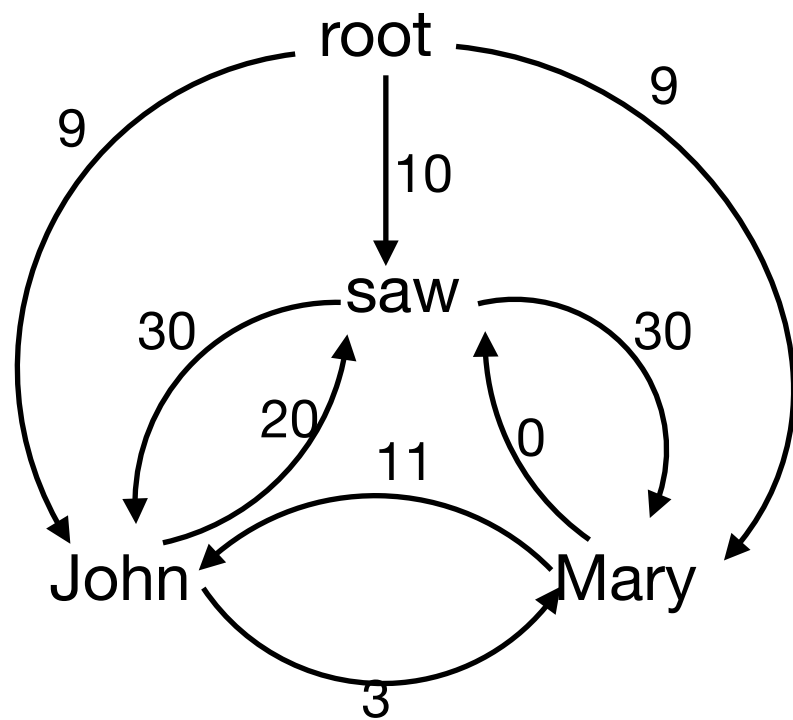
β :

[.]

Graph-Based Approach

- Transition Based Parsing can only produce projective dependency structures? Why?
- Graph-based approaches do not have this restriction.
- Basic idea:
 - Each word is a vertex. Start with a completely connected graph.
 - Use standard graph algorithms to compute a Maximum Spanning Tree:
 - Need a model that assigns a score to each edge ("edge-factored model").
$$score(G) = \sum_{(w_i, r, w_j) \in A} \lambda(w_i, r, w_j)$$

MST Example



total score: 70

Computing the MST

- For undirected graphs, there are two common algorithms:
 - Kruskal's and Prim's, both run in $O(E \log V)$
- For dependency parsing we deal with directed graphs, so these algorithms are not guaranteed to find a tree.
 - Instead use Chu–Liu–Edmonds' algorithm, which runs in $O(EV)$ (naive implementation) or $O(E \log V)$ (with optimizations).