ENLP Lecture 18: Dependency Parsing

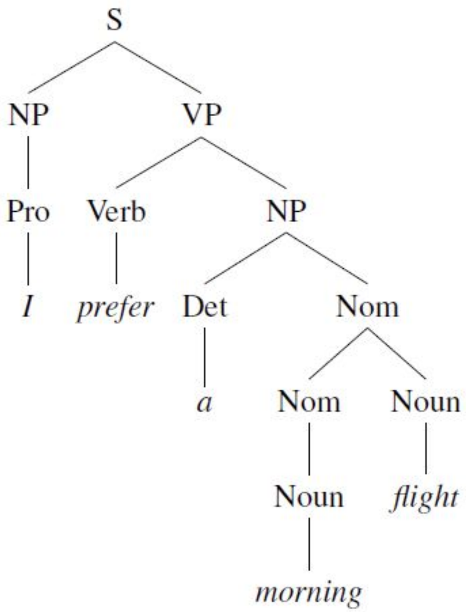
Tatsuya Aoyama

With texts/examples/figures from Dan Jurafsky & James Martin, and slides from Shira Wein
April 18, 2023

Agenda

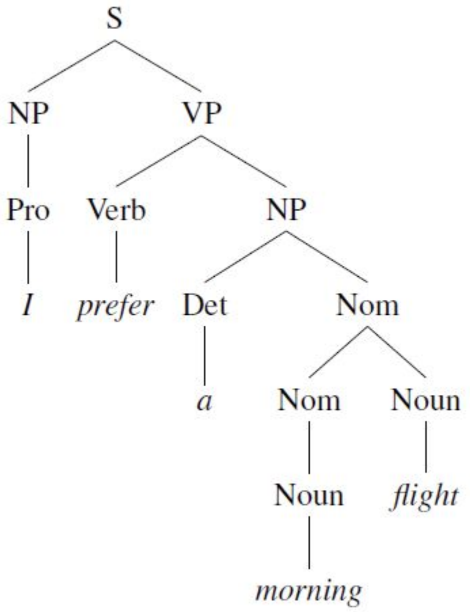
1. What is dependency grammar?
2. Inventory of dependency relations
3. Transition-based parsing
4. (very quickly) Graph-based parsing
5. (if time) Practice!

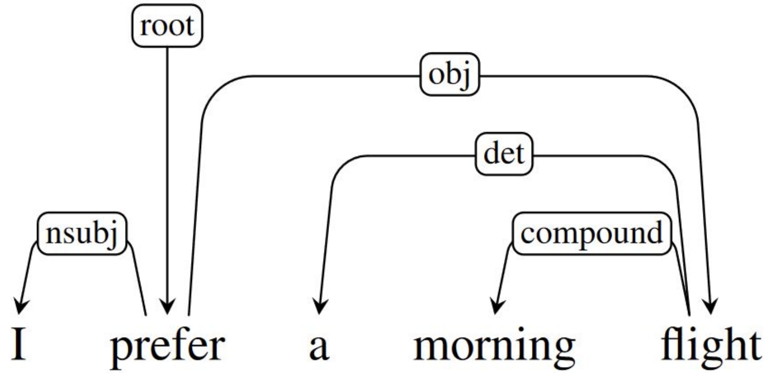
We've seen something like this:



- Many intermediate layers (S, VP, NP, ...)
- Relationships among words not necessarily clear
- Can we do this differently?

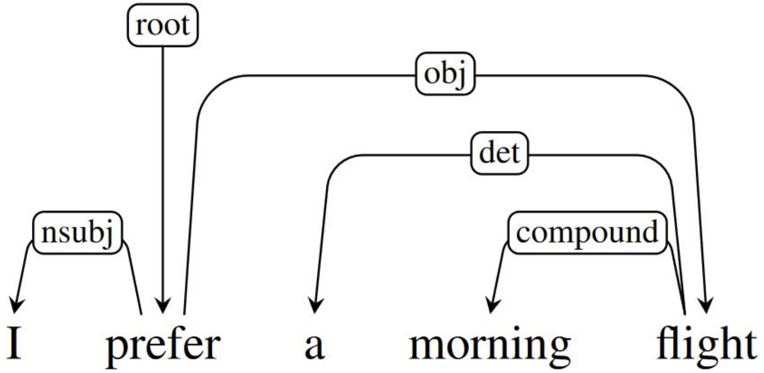
But this looks very different!





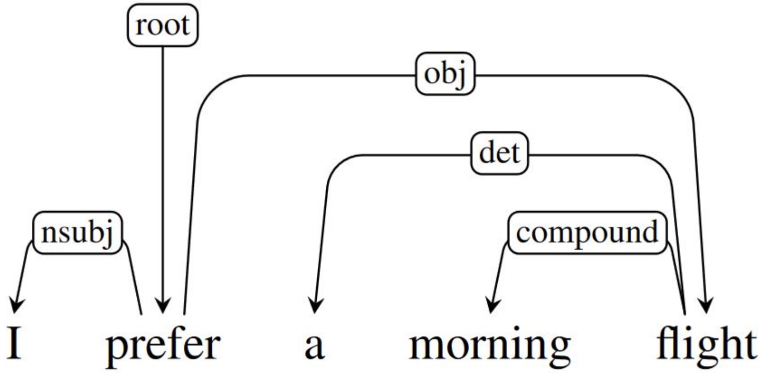
So what is dependency grammar?

- Directed binary grammatical relations between the words
- This direct encoding of the relationship between the predicates and their arguments (e.g., *prefer* takes *I* and *flight* as its arguments) is one of the reasons dependency grammar is more popular than constituency grammar in NLP!



So what is dependency grammar?

- **Arcs** go from **heads** to **dependents**
- **Exactly 1 incoming edge** for all tokens! (but can have many **outgoing** ones)
- **Root** is the head of the entire structure
- Especially useful for languages with **free word order** (constituency grammar is less suited to those)



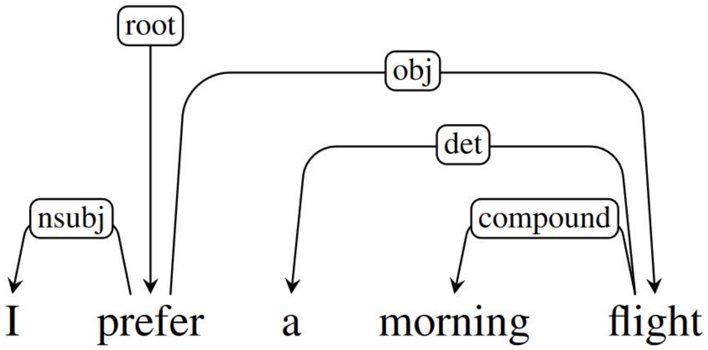
Dependency relations

- Depends on which particular framework you use
  - E.g., Universal Dependencies (UD; de Marneffe et al., 2021), Stanford Dependencies
  - We'll stick to UD in this lecture
- Different set of relation labels
- Different definition of "heads" (function heads, content heads)
  - UD is based on content heads!

Dependency relations

- Let’s unpack the example we saw

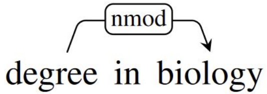
- root: d is the head of the entire sentence
- nsubj: d is a (nominal) subject of h
- obj: d is a direct object of h
- det: d is a determiner of h
- compound: d and h form a compound



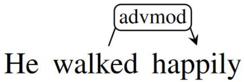
Dependency relations

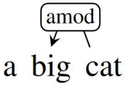
- advmod: adverb modifier
- amod: adjective modifier
- aux: auxiliary
- case: case marker (think of this as object of preposition)
- det: determiner
- iobj: indirect object
- nmod: noun modifier
- nmod:poss: possessive modifier
- nsubj: subject
- nummod: number modifier
- obj: direct object
- obl: oblique case ("prepositionally marked nominals functioning adverbially")
- root: root

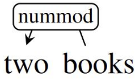
Dependency relations

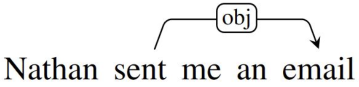


- advmod: adverb modifier
- amod: adjective modifier
- aux: auxiliary
- case: case marker
  (think of this as object of preposition)
- det: determiner
- iobj: indirect object
- nmod: noun modifier
- nmod:poss: possessive modifier
- nsubj: subject
- nummod: number modifier
- obj: direct object
- obl: oblique case ("prepositionally marked nominals functioning adverbially")
- root: root

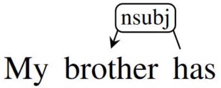


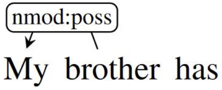


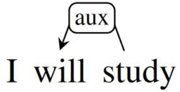


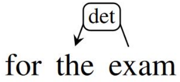


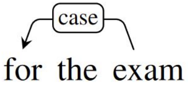
- My brother has a degree in biology.
- My brother has a degree in biology.
- My brother has a degree in biology.
- He walked happily.
- There is a big cat.
- I will study for the exam.
- I will study for the exam.
- I will study for the exam.
- I have two books.
- Nathan sent me an email on Monday.
- Nathan sent me an email on Monday.
- Nathan sent me an email on Monday.

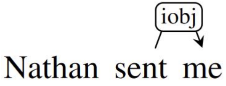


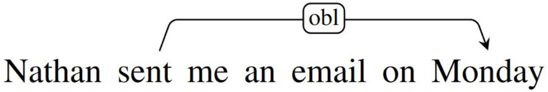












Structural ambiguity (again!)

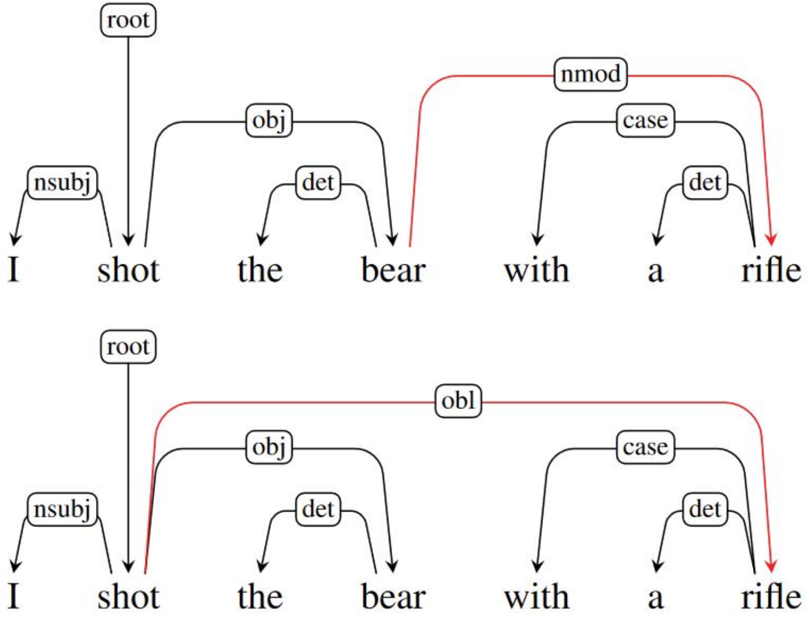
I shot the bear with a rifle

Structural ambiguity (again!)

I shot the bear with a rifle

Images from: https://www.cs.utexas.edu/~dnp/frege/subsection-178.html
https://www.istockphoto.com/vector/hunter-shoots-a-bear-gm930143498-255034331

Structural ambiguity (again!)



Images from: https://www.cs.utexas.edu/~dnp/frege/subsection-178.html
https://www.istockphoto.com/vector/hunter-shoots-a-bear-gm930143498-255034331

Dependency parsing is useful!

- Resolves attachment ambiguities that can matter for meaning
  - Grammatical structure of a sentence based on the relationships (dependencies) between the words
- Syntactic dependencies can be close to semantic relations
- Applicable across languages
- For what types of tasks might this be useful?

Information Extraction

- Can be used in information extraction to capture relationships
- **Relation extraction**: mining text to find relationships between entities
  - Who was the “doer” of the event/action? Usually nsubj!
  - Who was being acted upon? Usually obj!
  - …

Machine Translation

- When incorporated as linguistic prior during training into neural machine translation, improves performance
  - https://www.aclweb.org/anthology/P17-2012/ (from 2017)
- (Not standard practice to incorporate dependencies in MT)

Learning to Parse and Translate Improves Neural Machine Translation

Akiko Eriguchi†, **Yoshimasa Tsuruoka**†, and **Kyunghyun Cho**‡
†The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
{eriguchi, tsuruoka}@logos.t.u-tokyo.ac.jp
‡New York University, New York, NY 10012, USA
kyunghyun.cho@nyu.edu

Before we try it ourselves, some details on edges & heads

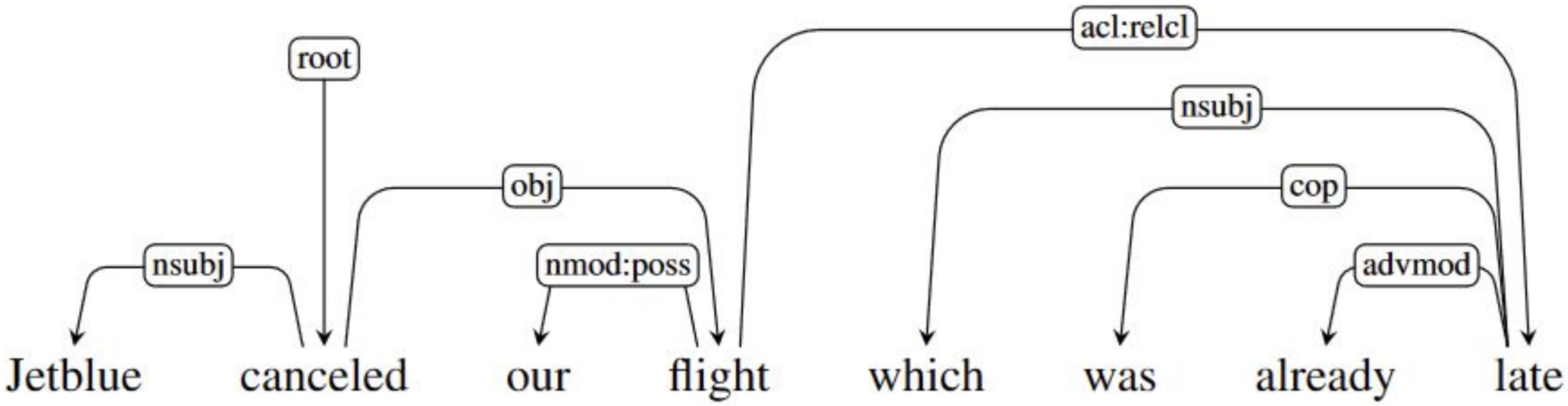
Projectivity

Formally:

- An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence.
- A dependency tree is then said to be projective if all the arcs that make it up are projective.

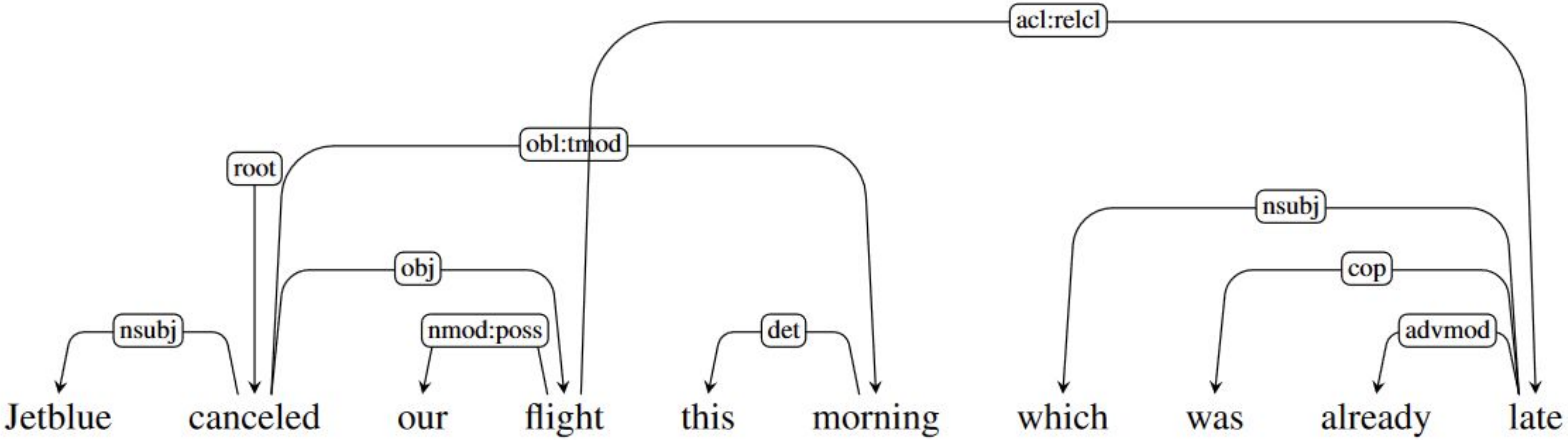
Projectivity

Is this projective?



Projectivity

This one?



Projectivity

Formally:

- An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence.
- A dependency tree is then said to be projective if all the arcs that make it up are projective.

Informally:

- A dependency tree is projective if there are no **crossing edges**.

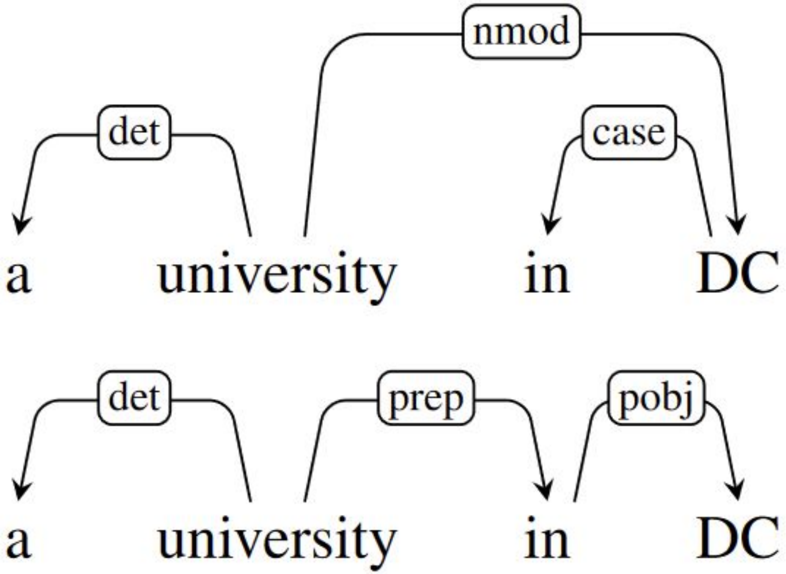
Projectivity

Why care?

- Many dependency treebanks auto-converted from constituency only allow projective trees (so the one we saw, which is a completely well-formed tree, will not be accepted in such treebanks)
- Standard transition-based parsing algorithm only produces projective trees
    - Some variants are capable of producing non-projective trees
    - Graph-based approach is generally more flexible

Heads

- Some dependency parse flavors prioritize **content** words as heads (auxiliaries, prepositions, etc. are modifiers)
- Other flavors use **functional** heads (prepositions head their objects, auxiliaries head main verbs, ...)



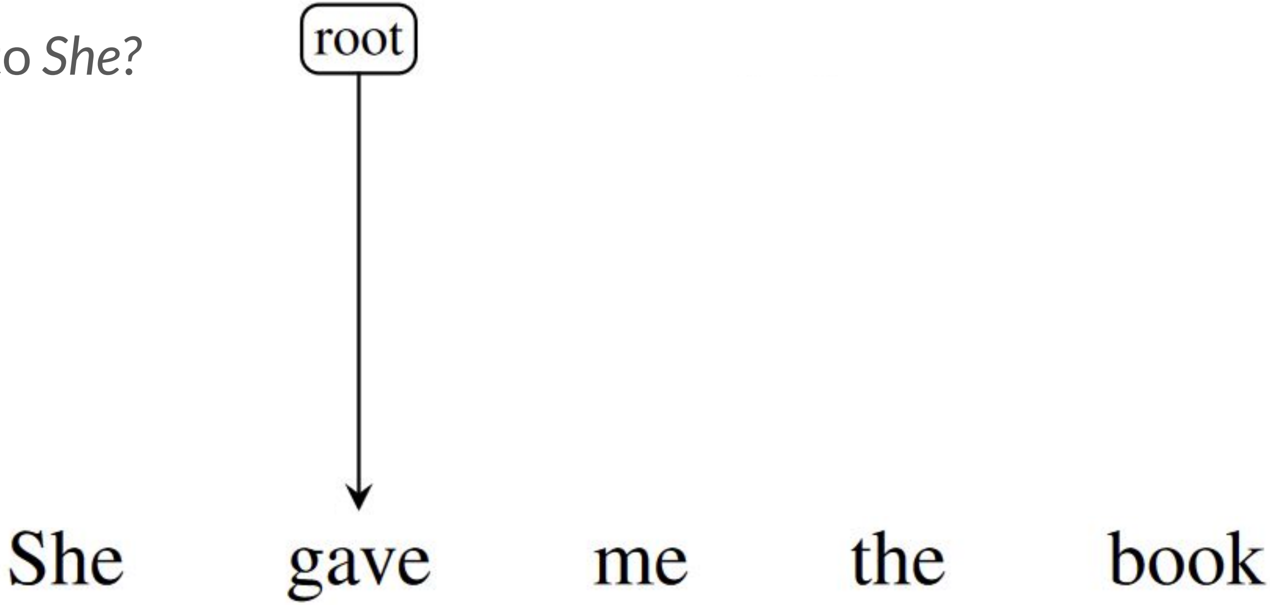
Let's try one!

root?

She gave me the book

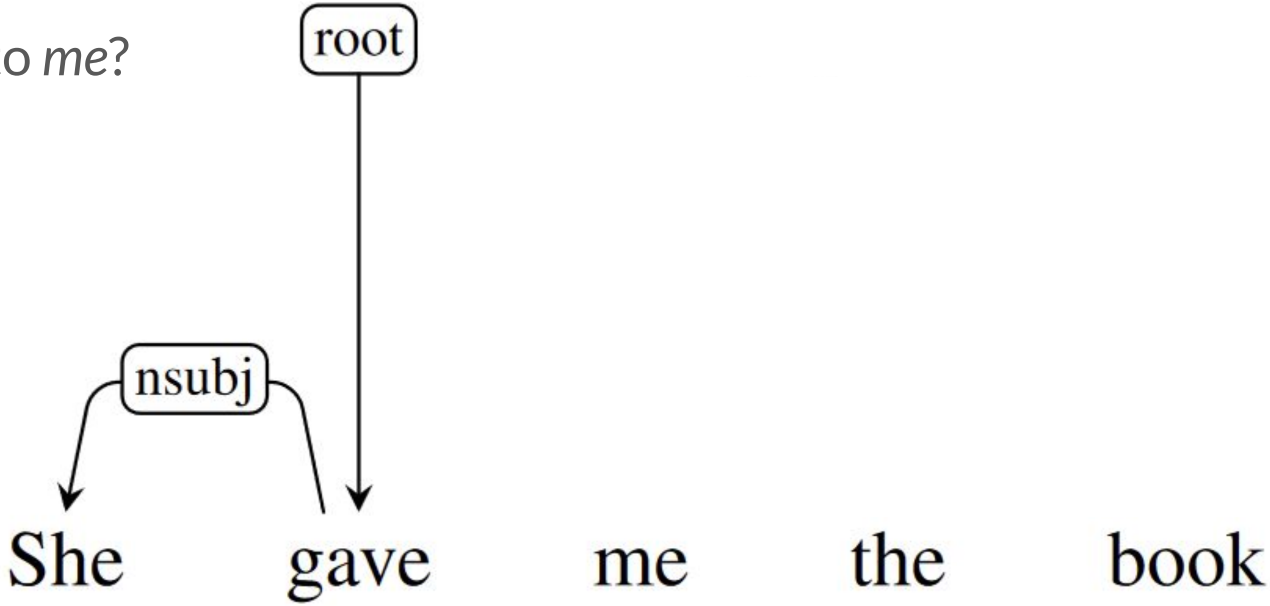
Let's try one!

Relation to *She*?



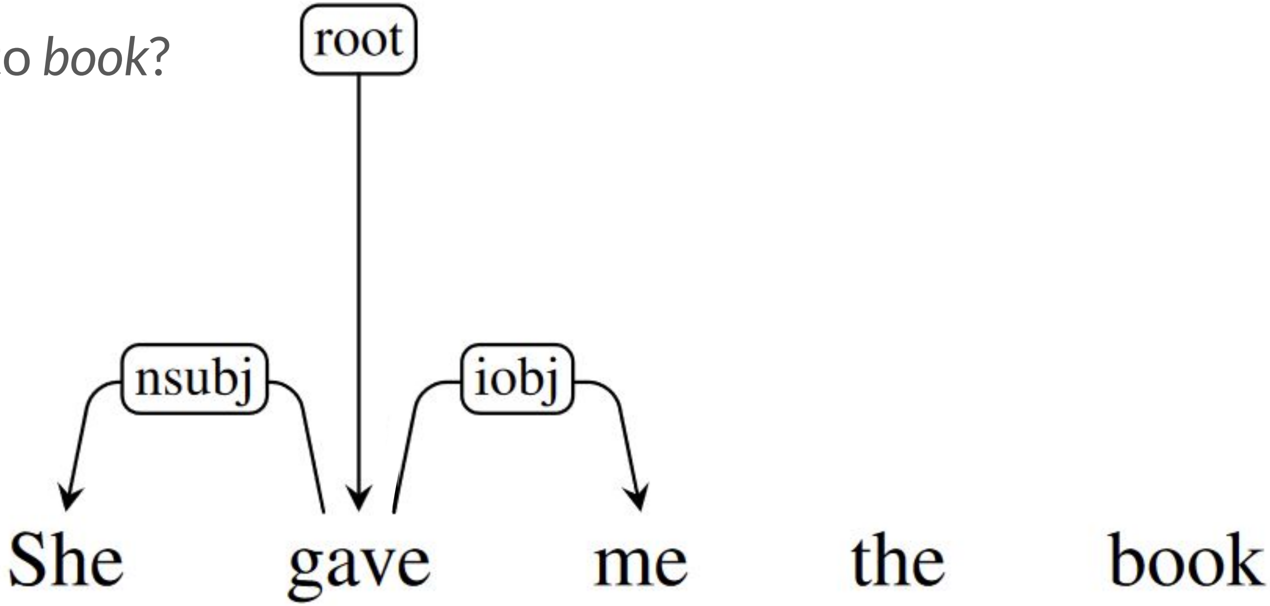
Let’s try one!

Relation to *me*?



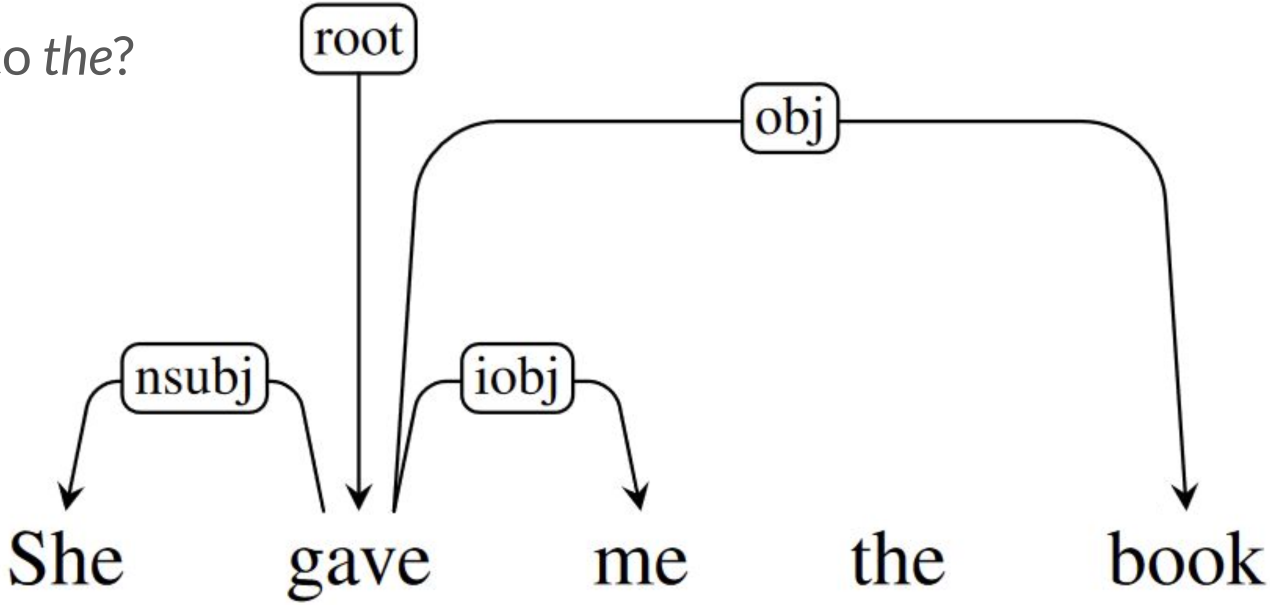
Let's try one!

Relation to *book*?

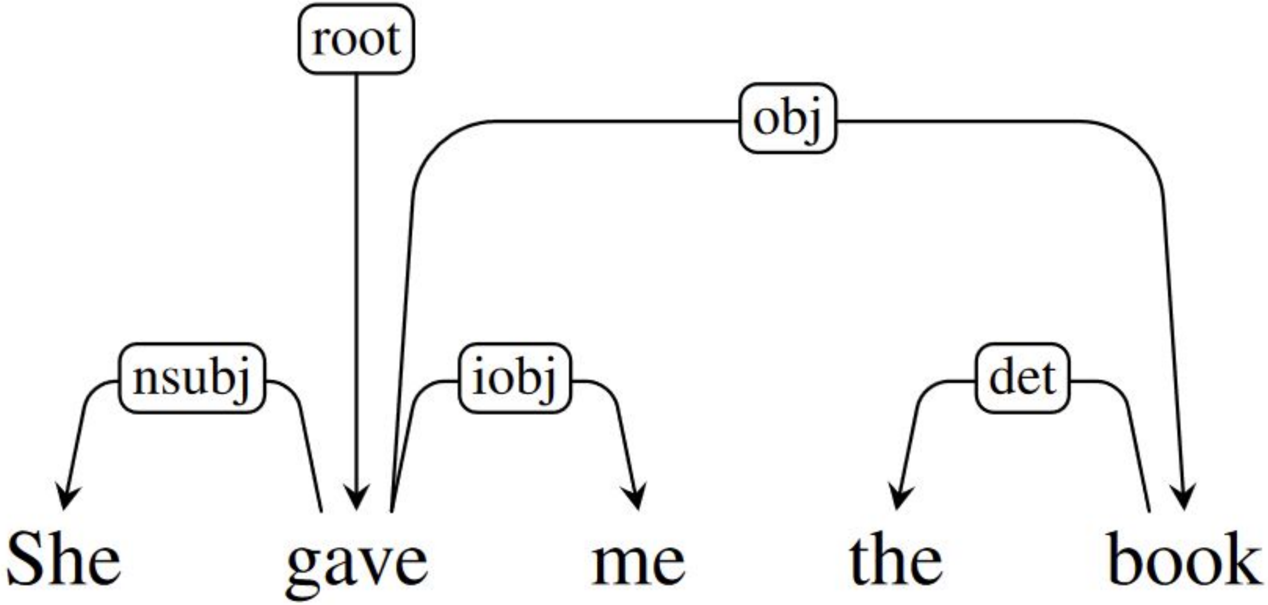


Let's try one!

Relation to *the*?



Let's try one!



Transition-based Parsing
(sort of tricky, part of A5)

Transition-based Parsing

- Process words from left to right, deciding if the two words should be attached
- Build a dependency parse using a stack and buffer
- **Input buffer**: words of the sentence
- **Stack**: to manipulate the words
- **Dependency relations:** list of relations that culminate in the dependency parse

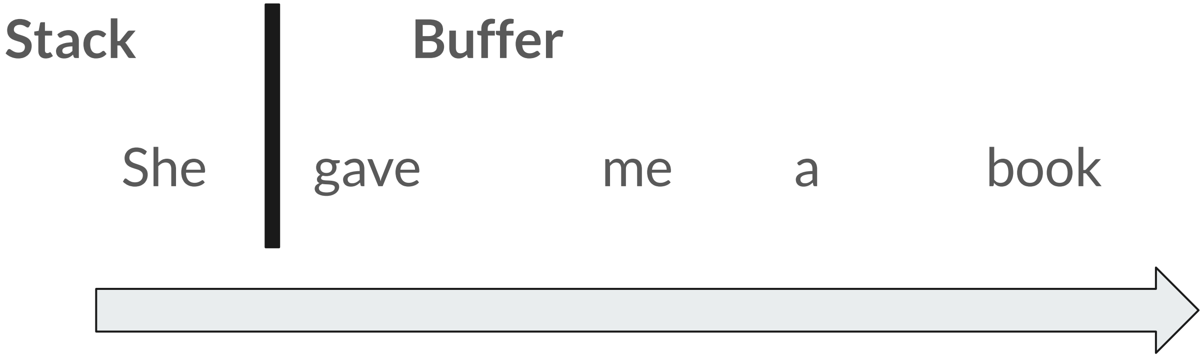
Transition-based Parsing

Stack

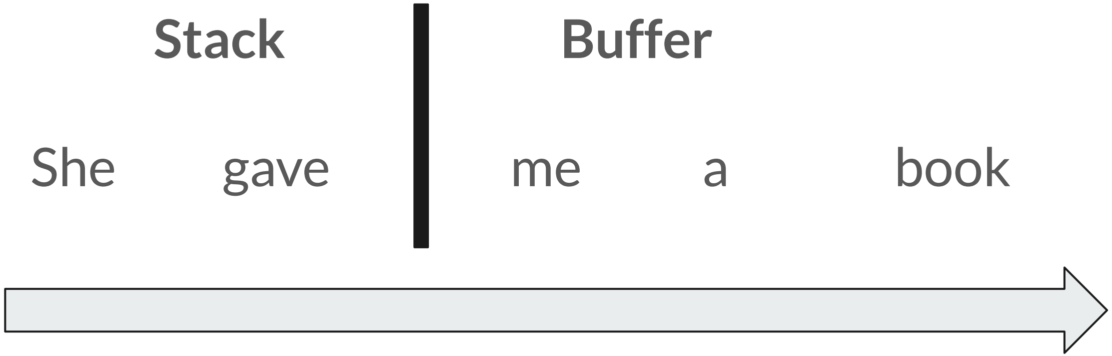
Buffer

She gave me a book

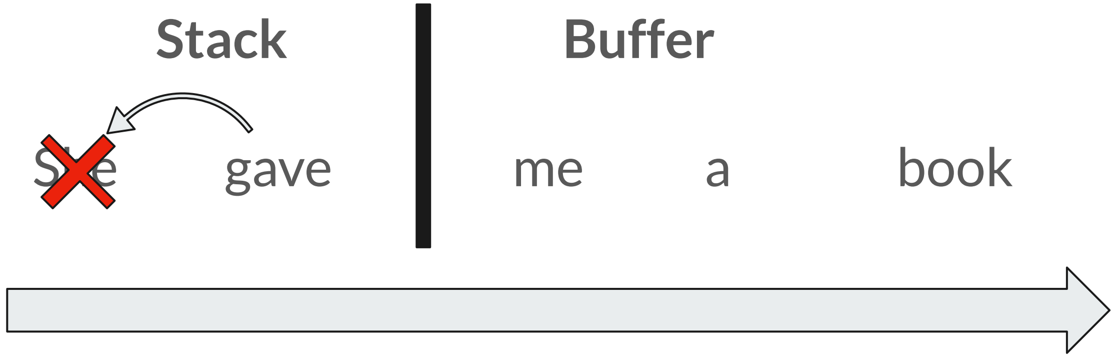
Transition-based Parsing



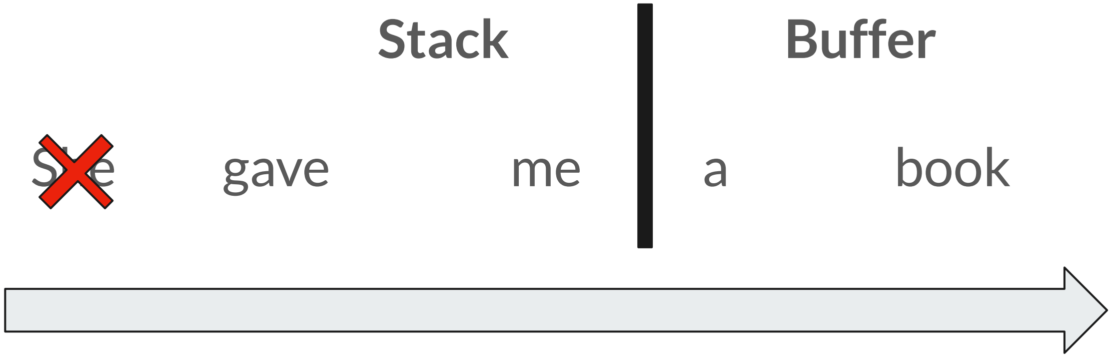
Transition-based Parsing



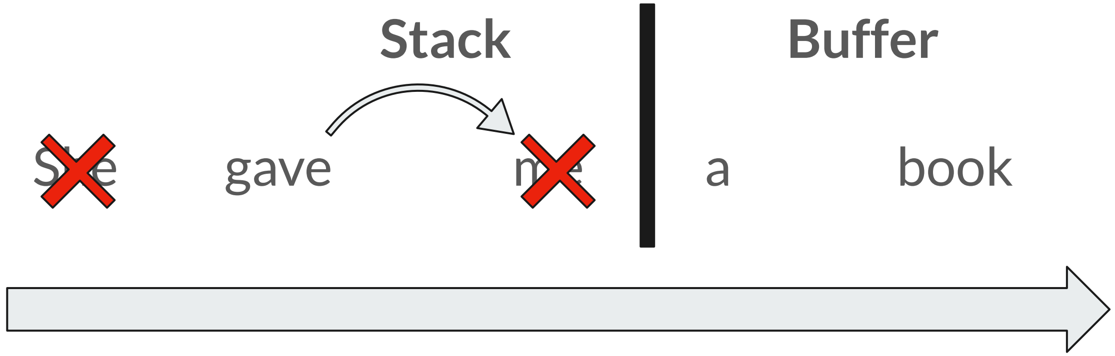
Transition-based Parsing



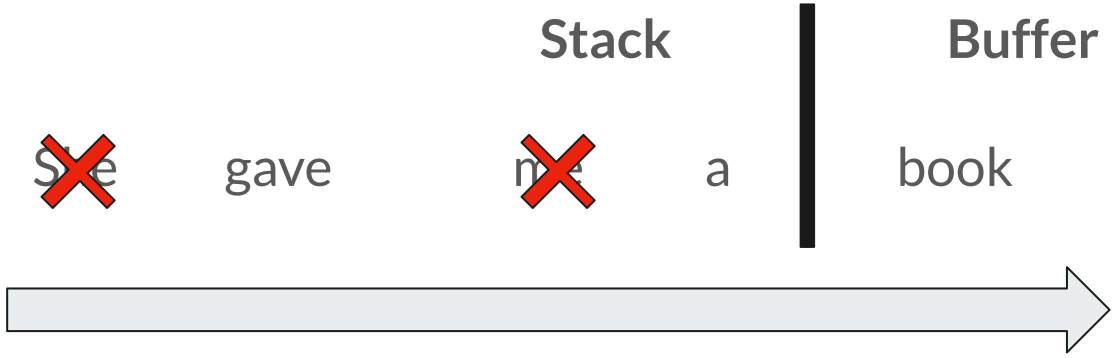
Transition-based Parsing



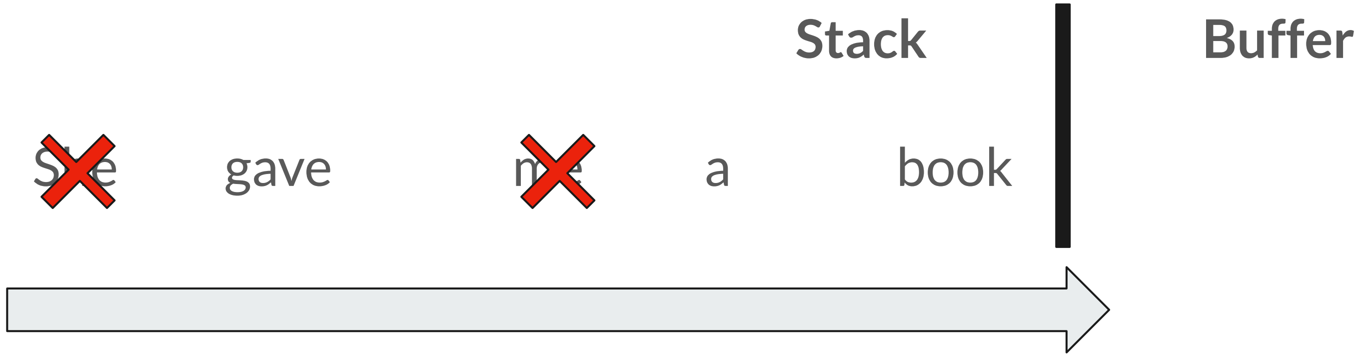
Transition-based Parsing



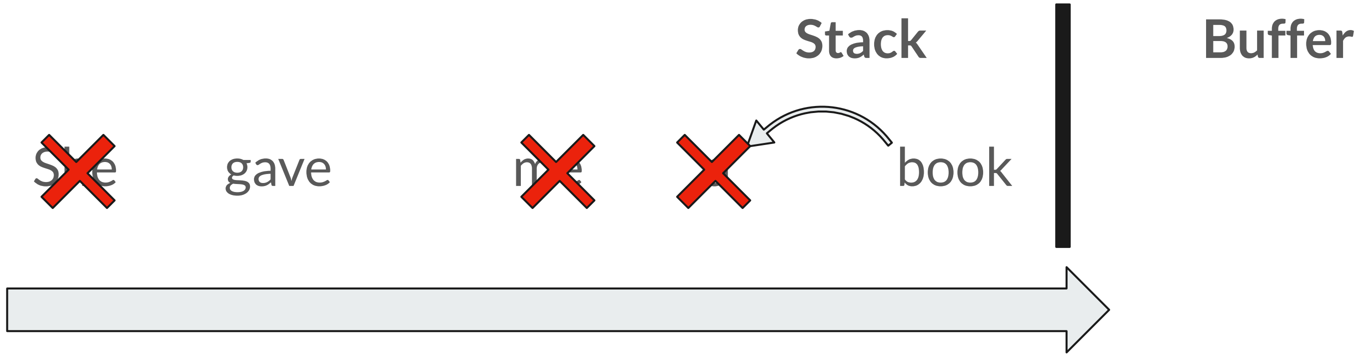
Transition-based Parsing



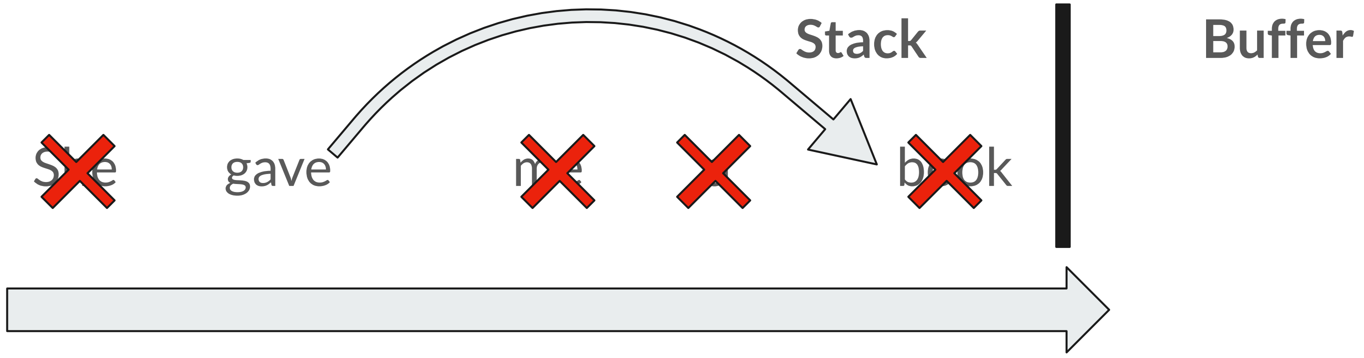
Transition-based Parsing



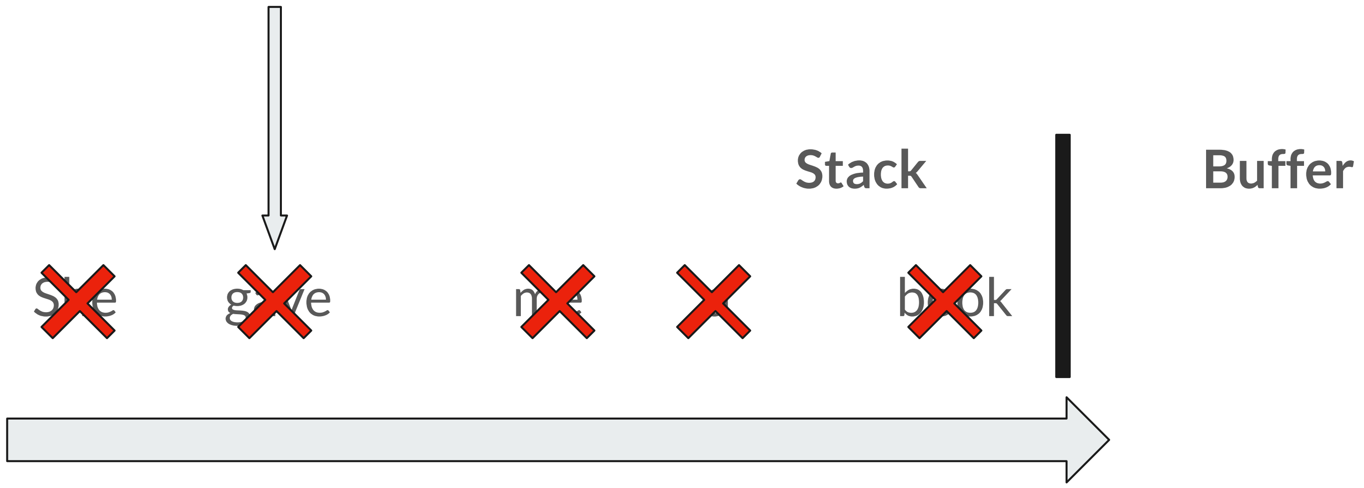
Transition-based Parsing



Transition-based Parsing



Transition-based Parsing



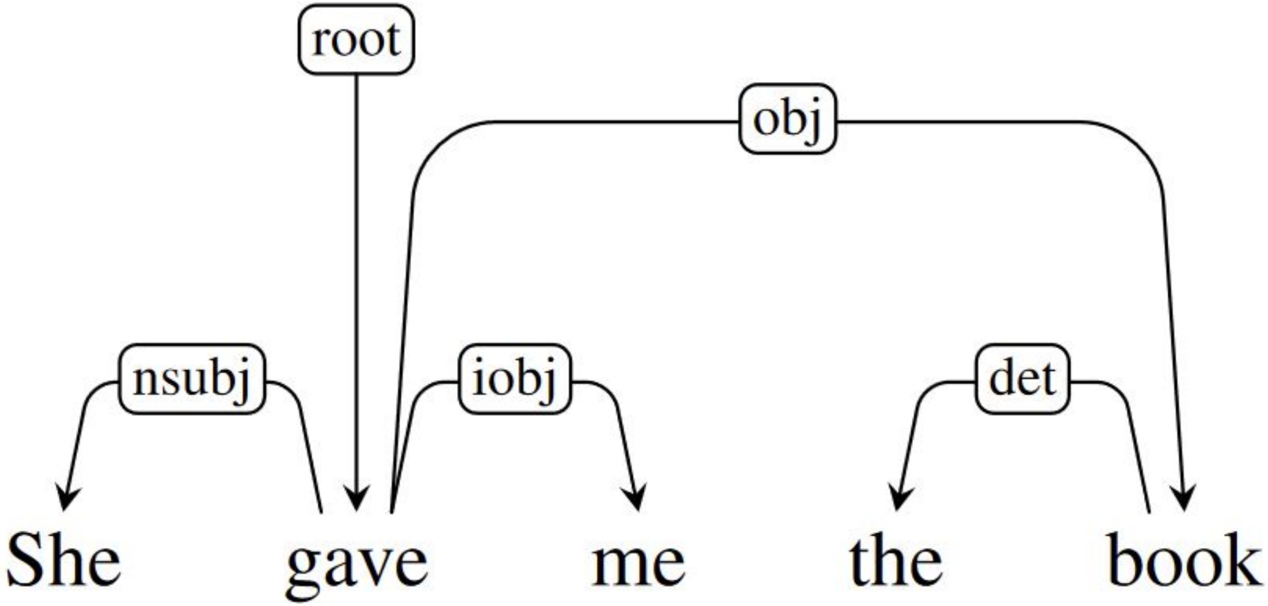
Arc-Standard Parsing

- Build relations between words using ARCS and remove word from stack once you have identified the word's parent
- **LEFTARC**
  - The word at the top of the stack is the head of the word beneath it
  - Remove second word from stack (the word you just made a dependent of the top word)
- **RIGHTARC**
  - (the reverse) The second word on the stack is the head of the word on top of the stack
  - Remove top word from stack
- **SHIFT**
  - Move the word from input buffer to the stack

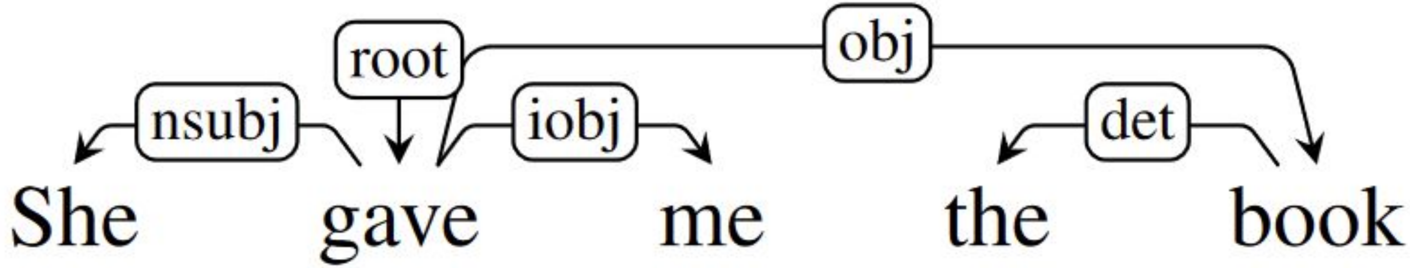
Arc-Standard Parsing

- Some restrictions!
- The root cannot be a dependent, so LEFTARC cannot be applied when the root is the second word in the stack
- There must be at least 2 words in the stack to apply LEFTARC or RIGHTARC

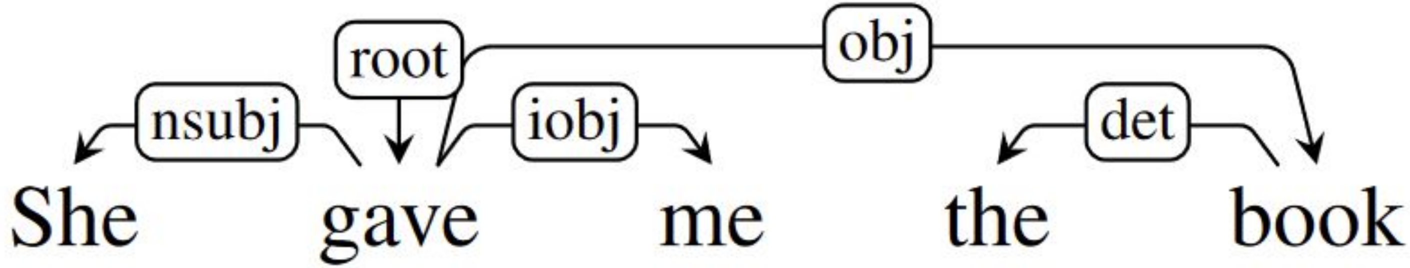
Example



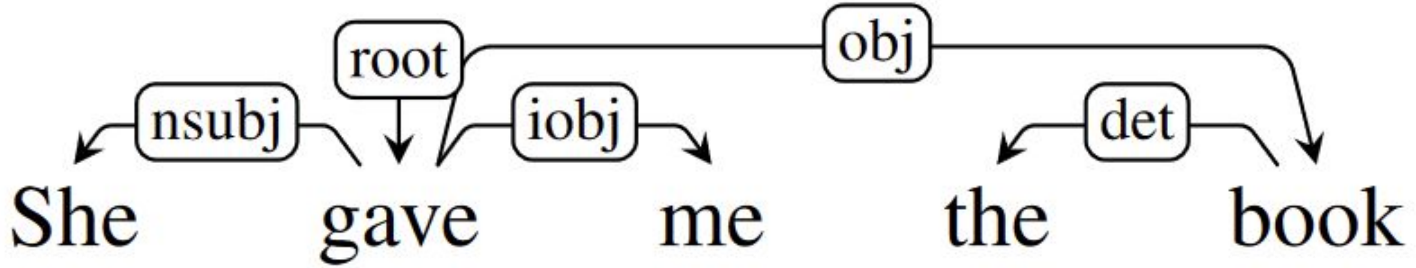
Step	Stack	Input Buffer	Action	Relation Added
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				



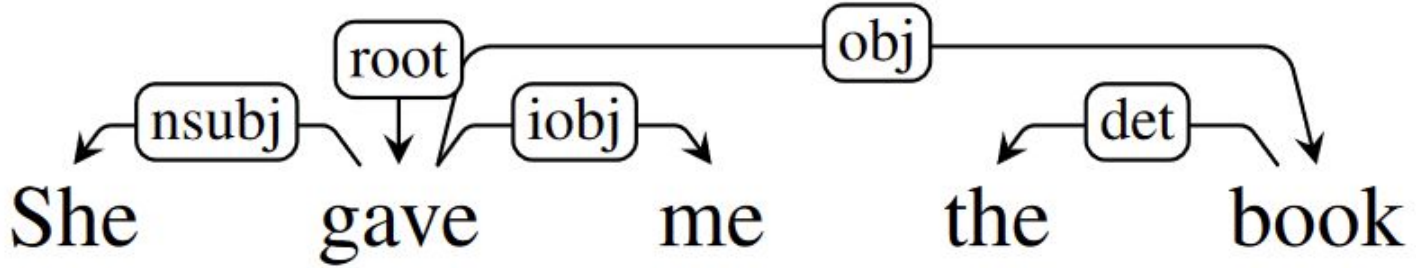
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]		
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				



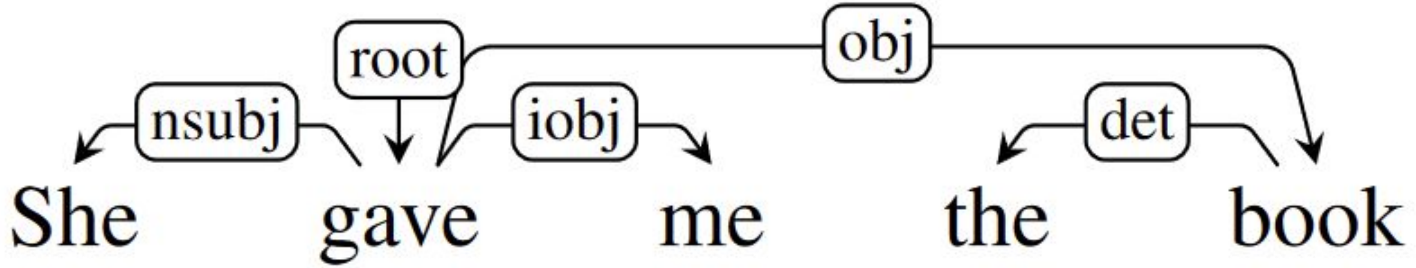
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				



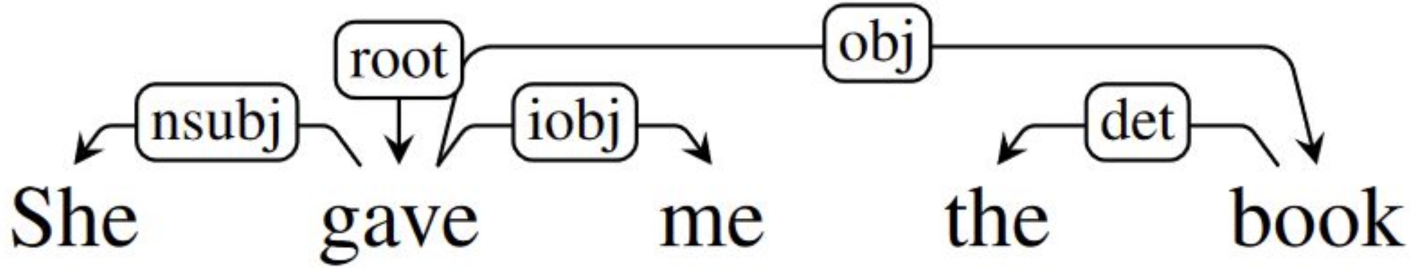
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]		
2				
3				
4				
5				
6				
7				
8				
9				
10				



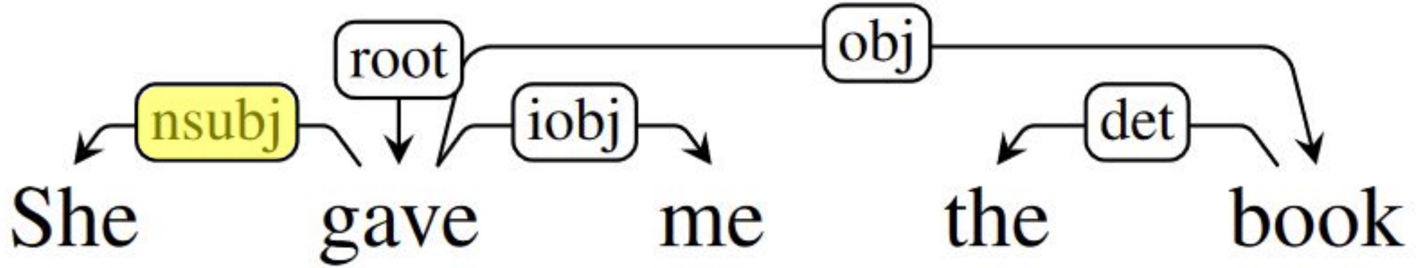
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2				
3				
4				
5				
6				
7				
8				
9				
10				



Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]		
3				
4				
5				
6				
7				
8				
9				
10				

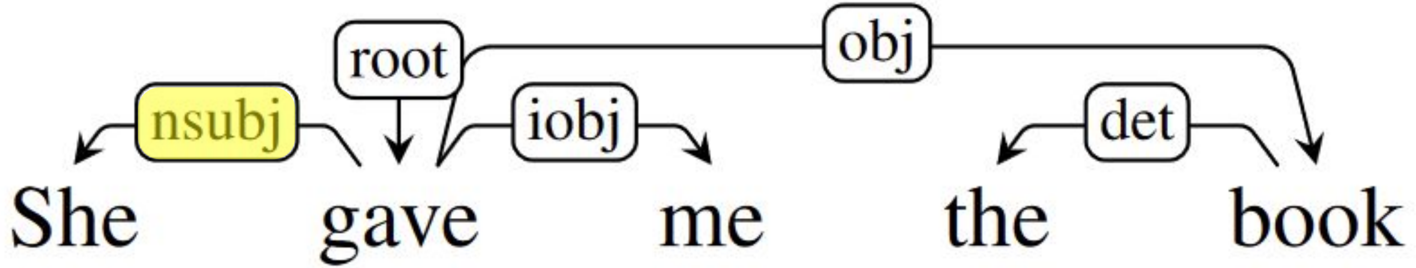


Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3				
4				
5				
6				
7				
8				
9				
10				

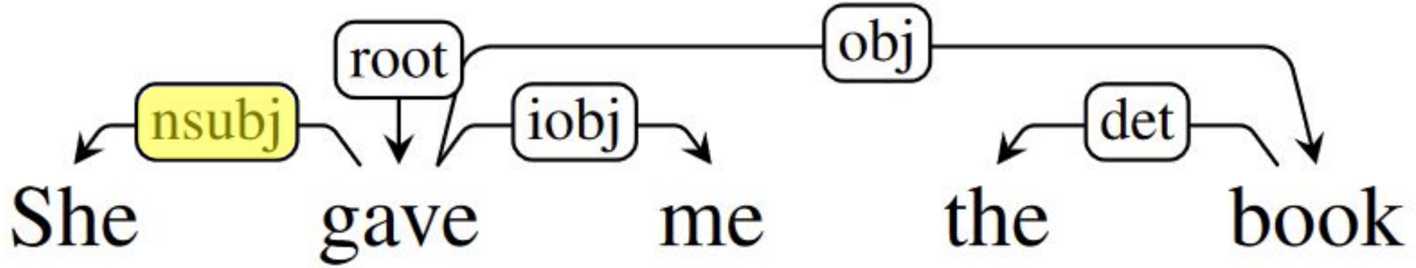


Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]		
4				
5				
6				
7				
8				
9				
10				

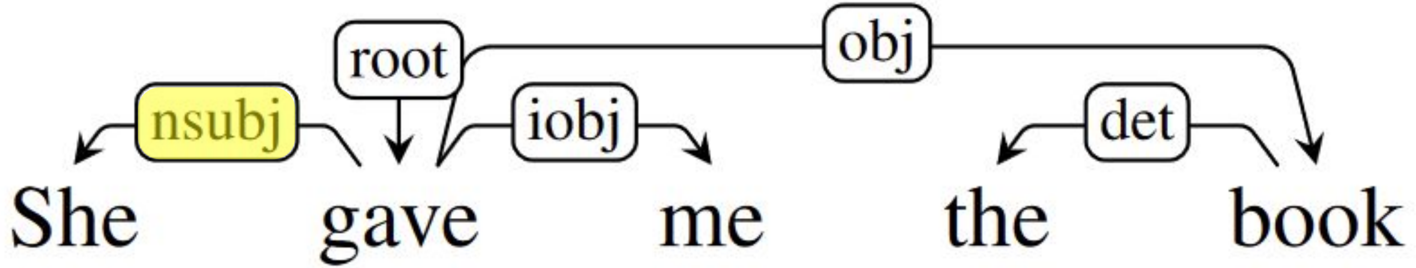
Can we add a right arc (root → gave)?



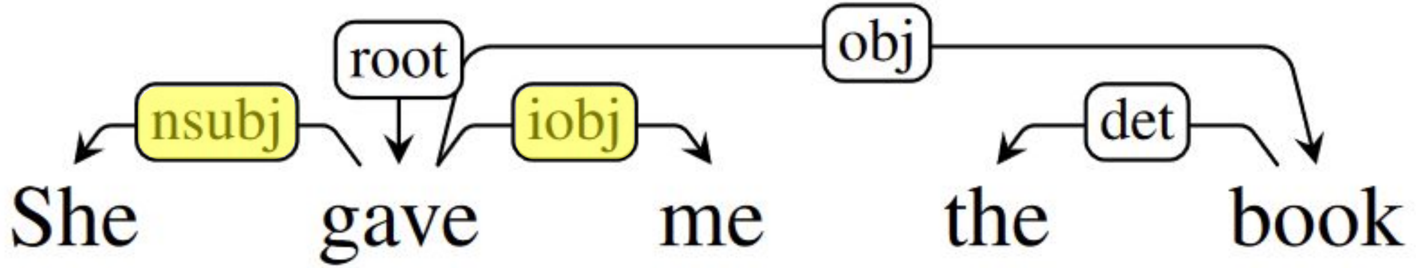
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4				
5				
6				
7				
8				
9				
10				



Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]		
5				
6				
7				
8				
9				
10				

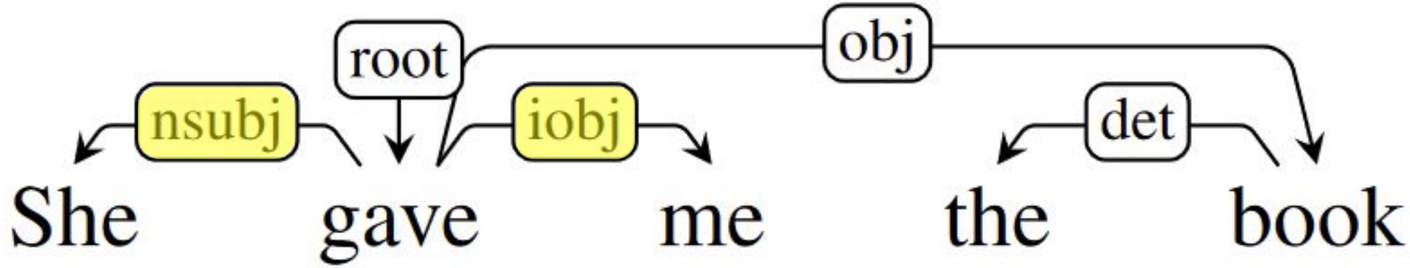


Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5				
6				
7				
8				
9				
10				

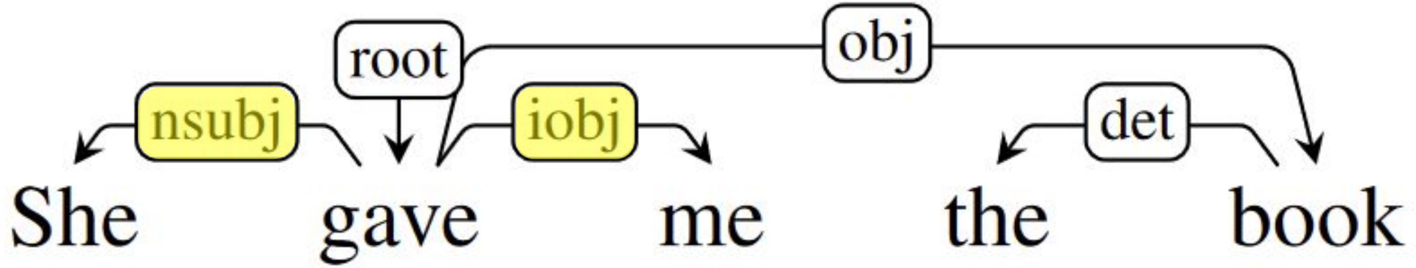


Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]		
6				
7				
8				
9				
10				

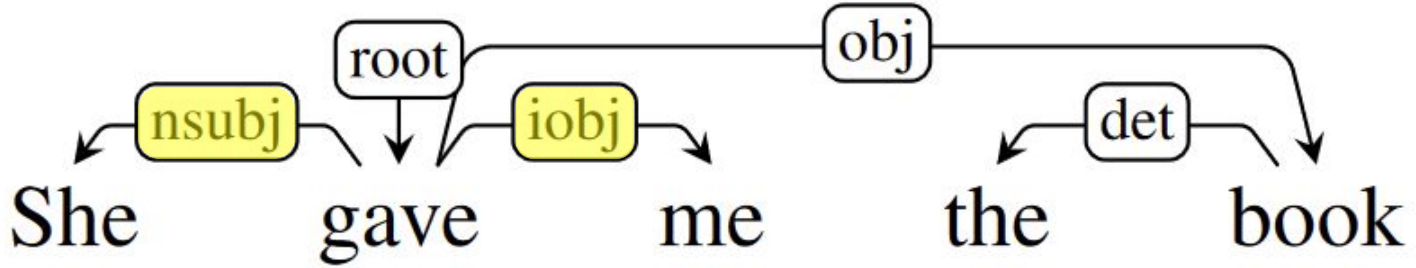
Can we add a right arc yet?



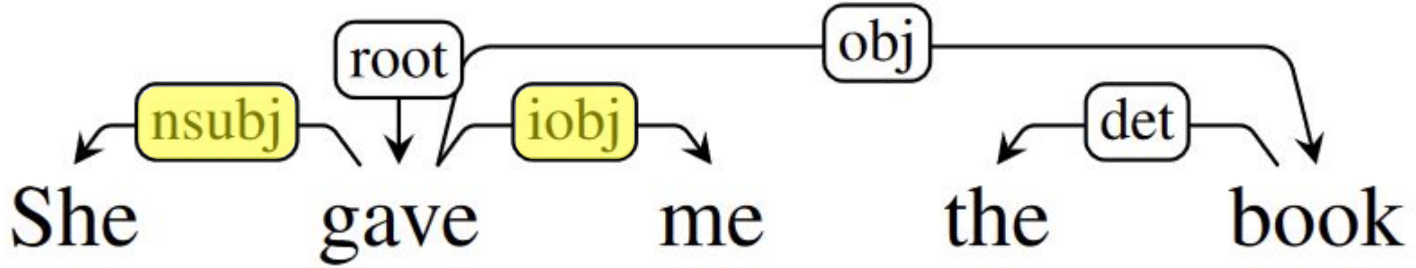
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6				
7				
8				
9				
10				



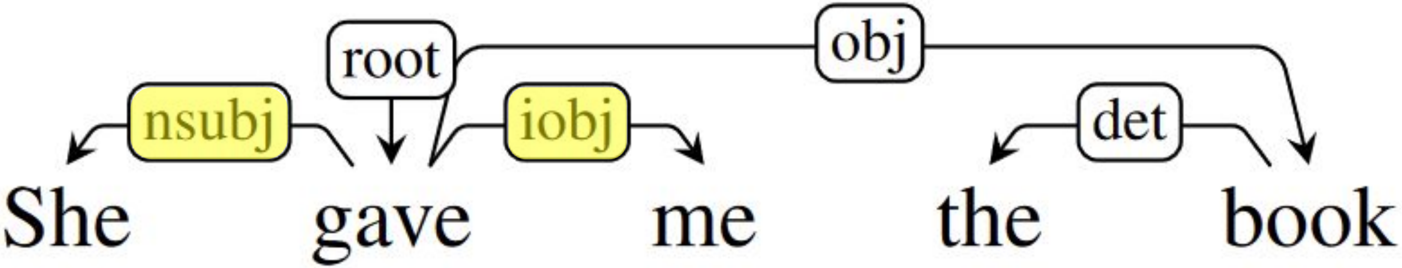
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]		
7				
8				
9				
10				



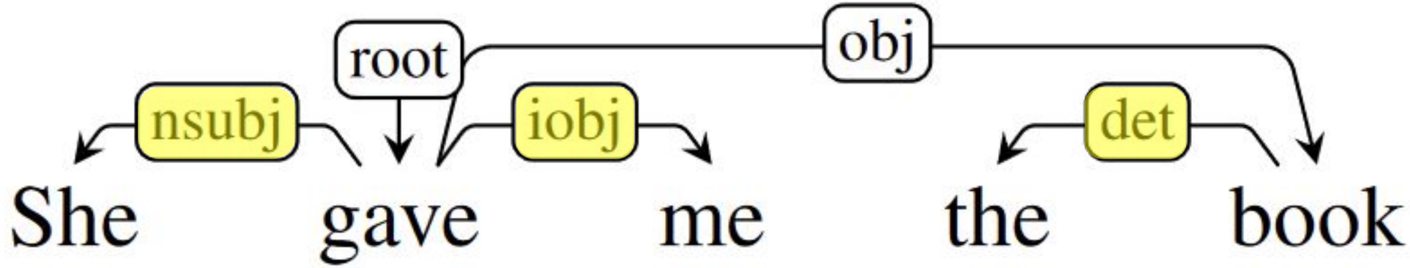
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7				
8				
9				
10				



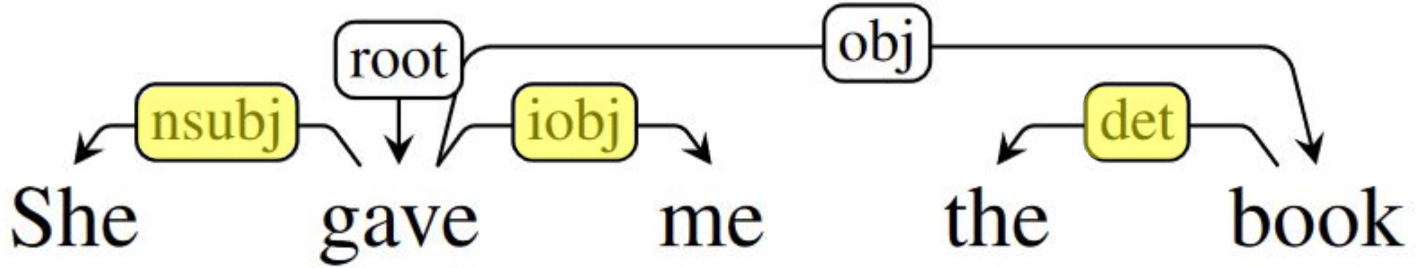
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]		
8				
9				
10				



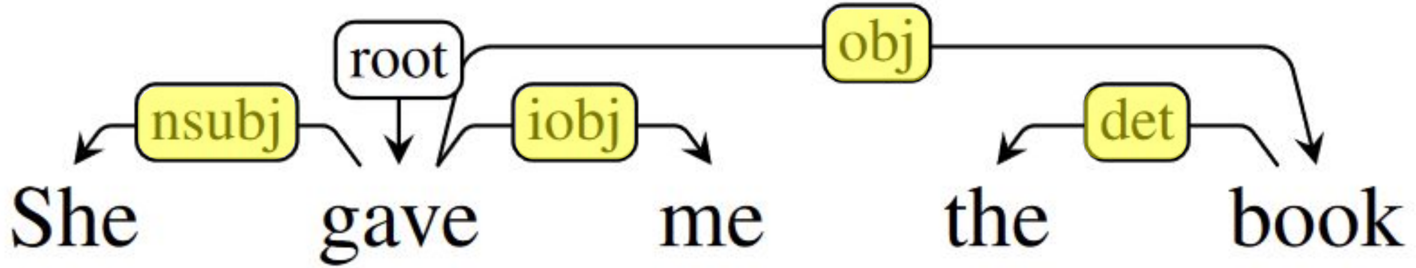
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]	LA	(the ← book)
8				
9				
10				



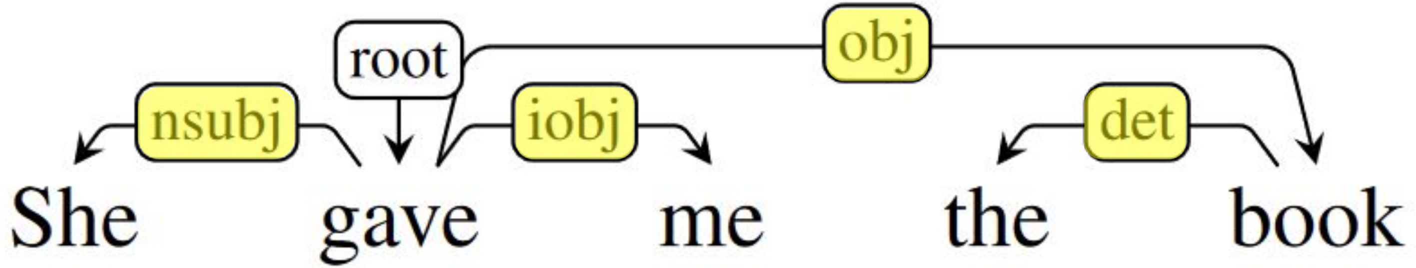
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]	LA	(the ← book)
8	[root, gave, book]	[]		
9				
10				



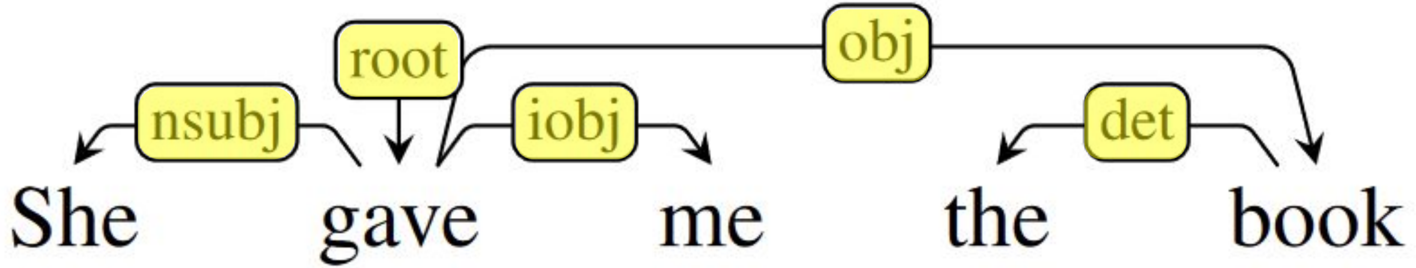
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]	LA	(the ← book)
8	[root, gave, book]	[]	RA	(gave → book)
9				
10				



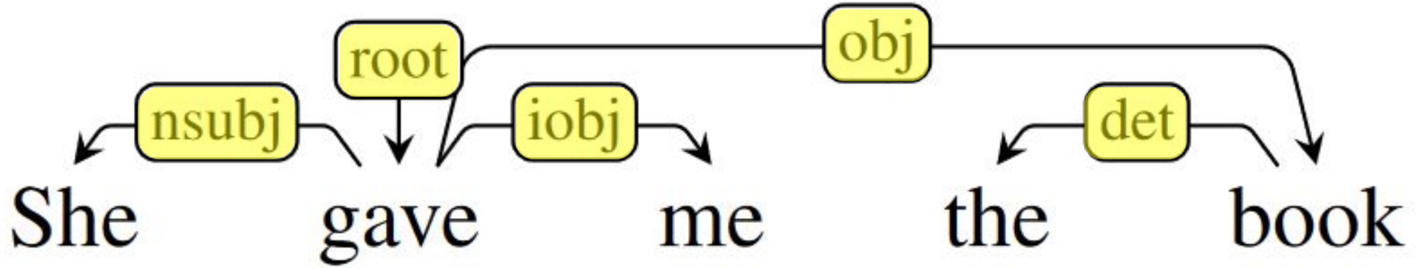
Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]	LA	(the ← book)
8	[root, gave, book]	[]	RA	(gave → book)
9	[root, gave]	[]		
10				



Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]	LA	(the ← book)
8	[root, gave, book]	[]	RA	(gave → book)
9	[root, gave]	[]	RA	(root → gave)
10				



Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]	LA	(the ← book)
8	[root, gave, book]	[]	RA	(gave → book)
9	[root, gave]	[]	RA	(root → gave)
10	[root]	[]	Done	

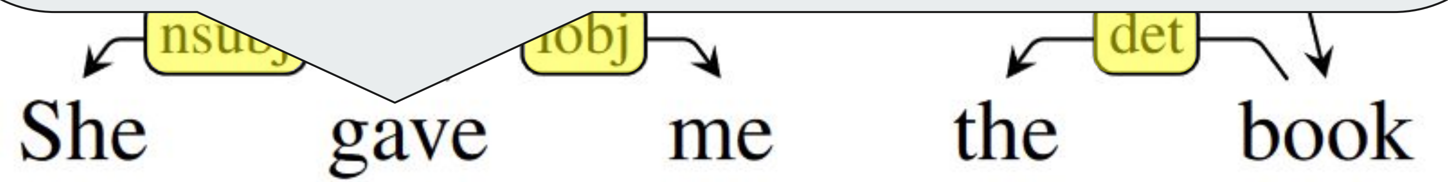


Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1				
2				ave)
3				
4				ne)
5				
6				
7				ok)
8				ook)
9				ave)
10				

We have (exactly) encoded the parse tree as a sequence of {S, LA, RA} actions! Think of it as a **program** for building the tree structure.

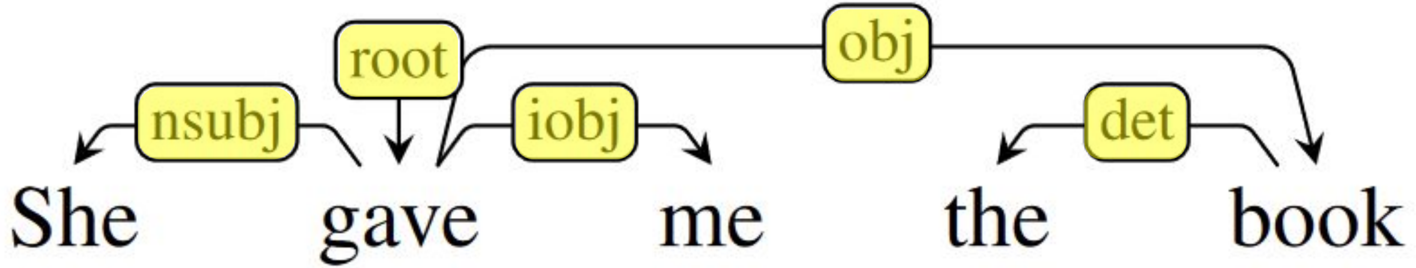
(Would also need to specify relation labels in the LA, RA actions or post hoc.)

Transition-based parsing = iteratively:
(1) consult the Oracle (algorithm giving next action)
(2) modify the Configuration (state of stack, buffer, relations) according to the action



Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, t	A	(She ← gave)
3	[root, gave]	[me, t		
4	[root, gave, me]	[the, b	A	(gave → me)
5	[root, gave]	[th		
6	[root, gave, the]	[book]		
7	[root, gave, the, book]	[]	LA	(the ← book)
8	[root, gave, book]	[]	RA	(gave → book)
9	[root, gave]	[]	RA	(root → gave)
10	[root]	[]	Done	

Would have been a mess if RA was predicted!
→ arc-eager parsing



Arc-Standard Parsing

- With the 3 Arc-Standard actions {S, LA, RA}:
  - How many transitions to parse a sentence of N words?
  - 2N: for each word, once to shift + once to attach to a head and remove from stack (LA or RA).
- Can these 3 types of actions build any tree?
  - Only **projective trees**: only adding edges at the top of the stack & permanently removing a word from the stack once attaching it to its parent ensures that all subtrees are contiguous
  - With a richer set of actions, can get non-projective trees or even graphs
- How would you implement an Oracle (choose the next action at test time)?
  - This brings us to…

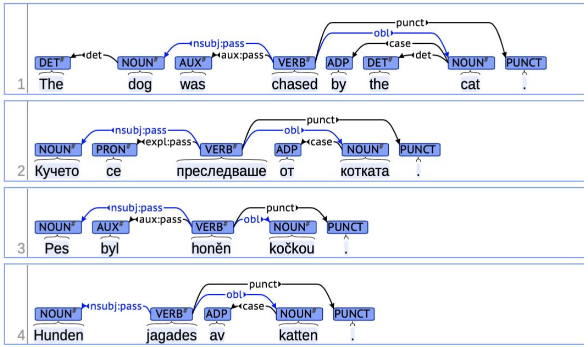
Statistical Dependency Parsing

Statistical Dependency Parsing

- Can be done by training a classifier to predict each action, using data from Treebanks
- Possible features? POS tags, word at the top of the stack, etc. — we'll come back to this in a second

Data

- Can automatically convert constituency treebanks (like the Penn Treebank) to dependencies
- Or, can use dependency treebanks like Universal Dependencies (available in many languages)
  - http://universaldependencies.org



Feature-based parser

- Any feature-based classifiers, e.g., SVM, logistic regression, …
- Feature template:

$$\langle s_1.w, op\rangle, \langle s_2.w, op\rangle \langle s_1.t, op\rangle, \langle s_2.t, op\rangle$$
$$\langle b_1.w, op\rangle, \langle b_1.t, op\rangle \langle s_1.wt, op\rangle$$

$$\langle s_1.w = \mathit{flights}, op = \mathit{shift}\rangle$$
$$\langle s_2.w = \mathit{canceled}, op = \mathit{shift}\rangle$$
$$\langle s_1.t = \mathit{NNS}, op = \mathit{shift}\rangle$$
$$\langle s_2.t = \mathit{VBD}, op = \mathit{shift}\rangle$$
$$\langle b_1.w = \mathit{to}, op = \mathit{shift}\rangle$$
$$\langle b_1.t = \mathit{TO}, op = \mathit{shift}\rangle$$
$$\langle s_1.wt = \mathit{flightsNNS}, op = \mathit{shift}\rangle$$

Neural parser

- Essentially using word embeddings as “features”!

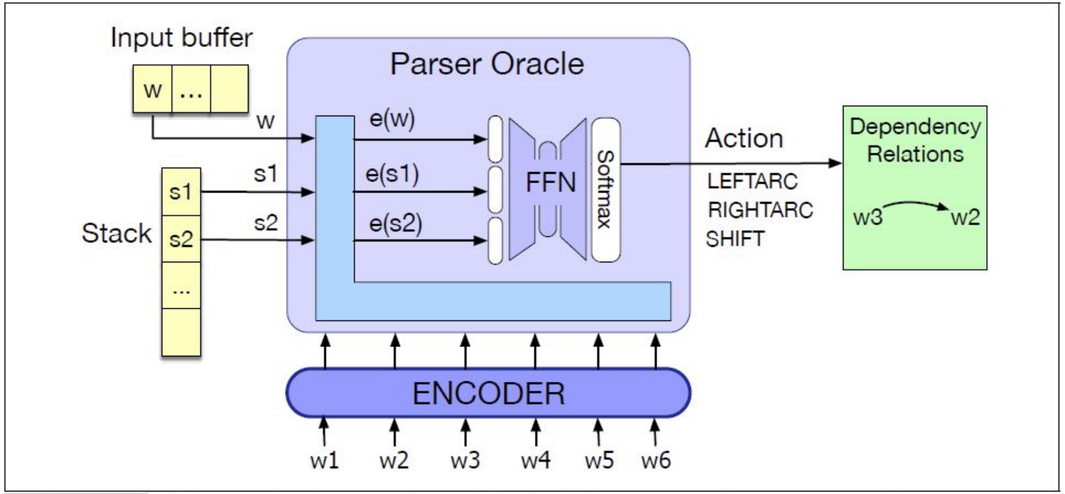
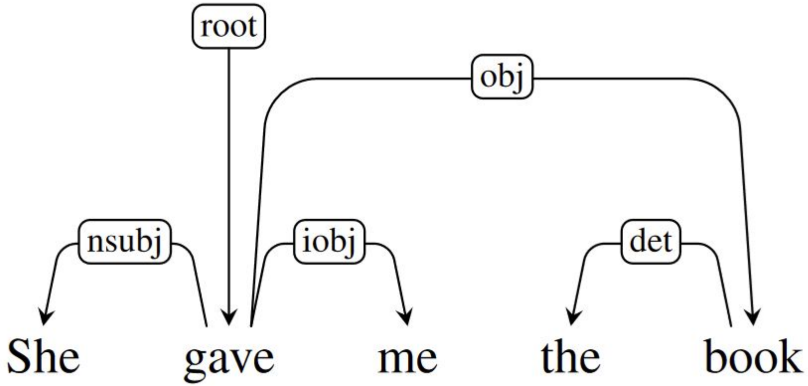


Figure 18.8 Neural classifier for the oracle for the transition-based parser. The parser takes the top 2 words on the stack and the first word of the buffer, represents them by their encodings (from running the whole sentence through the encoder), concatenates the embeddings and passes through a softmax to choose a parser action (transition).

Evaluation

- Comparing against a gold standard:
- **Unlabeled Attachment Score (UAS)**: % of words attached correctly (**correct head**)
- **Labeled Attachment Score (LAS)**: % of words attached to the **correct head with the correct relation label**



Graph-based Parsing

- Another popular approach to dependency parsing is **graph-based**
- Consider all possible trees
- Assign scores to all edges
- Best tree = largest sum of scores
- Capable of creating non-projective trees

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\operatorname{argmax}} \, \text{Score}(t, S)$$

$$\text{Score}(t, S) = \sum_{e \in t} \text{Score}(e)$$

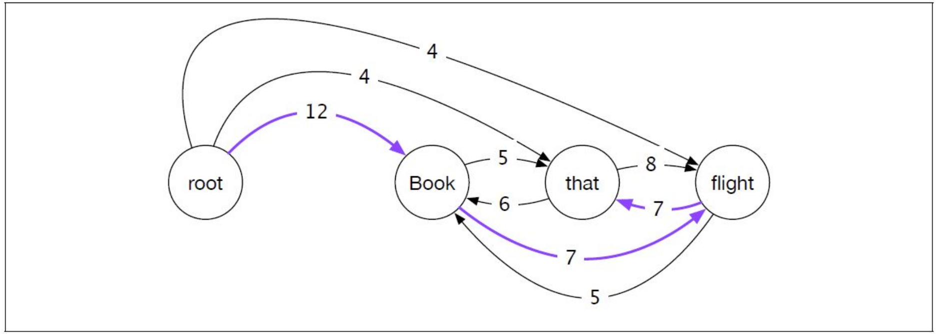


Figure 18.11 Initial rooted, directed graph for *Book that flight*.

Graph-based Parsing

- Feature-based scorer and neural scorer
- Popular architecture: **biaffine** (Dozat and Manning, 2017)

Model	Catalan UAS	Catalan LAS	Chinese UAS	Chinese LAS	Czech UAS	Czech LAS
Andor et al.	92.67	89.83	84.72	80.85	88.94	84.56
Deep Biaffine	**94.69**	**92.02**	**88.90**	**85.38**	**92.08**	**87.38**

Model	English UAS	English LAS	German UAS	German LAS	Spanish UAS	Spanish LAS
Andor et al.	93.22	91.23	90.91	89.15	92.62	89.95
Deep Biaffine	**95.21**	**93.20**	**93.46**	**91.44**	**94.34**	**91.65**

Table 5: Results on the CoNLL '09 shared task datasets

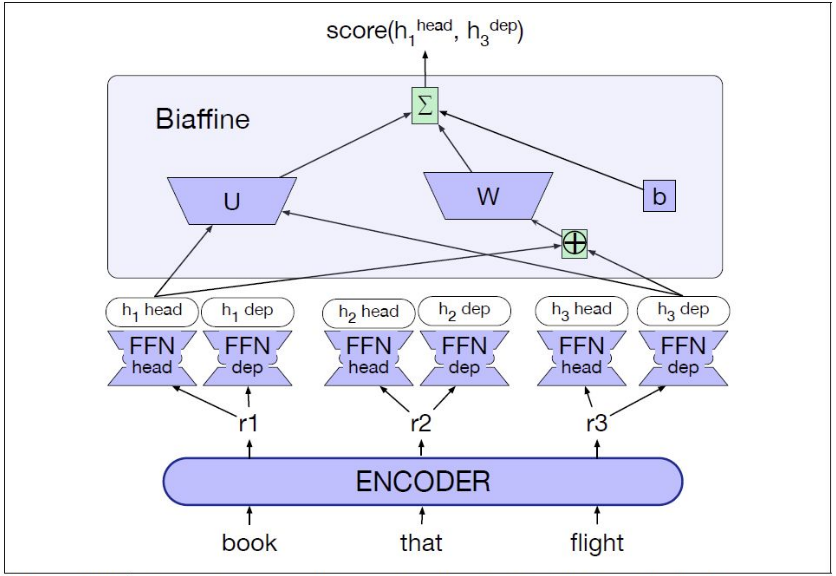


Figure 18.14 Computing scores for a single edge (book→ flight) in the biaffine parser of Dozat and Manning (2017); Dozat et al. (2017). The parser uses distinct feedforward networks to turn the encoder output for each word into a head and dependent representation for the word. The biaffine function turns the head embedding of the head and the dependent embedding of the dependent into a score for the dependency edge.

Demos

- Stanza (from Stanford): https://corenlp.run/
- AllenNLP (from Allen Institute for AI): https://demo.allennlp.org/dependency-parsing

Practice

- advmod: adverb modifier
- amod: adjective modifier
- aux: auxiliary
- case: case marker (think of this as object of preposition)
- det: determiner
- iobj: indirect object
- nmod: noun modifier
- nmod:poss: possessive modifier
- nsubj: subject
- nummod: number modifier
- obj: direct object
- obl: oblique case ("prepositionally marked nominals functioning adverbially")
- root: root

My friend from Japan will reluctantly give you 3 delicious cookies on Christmas

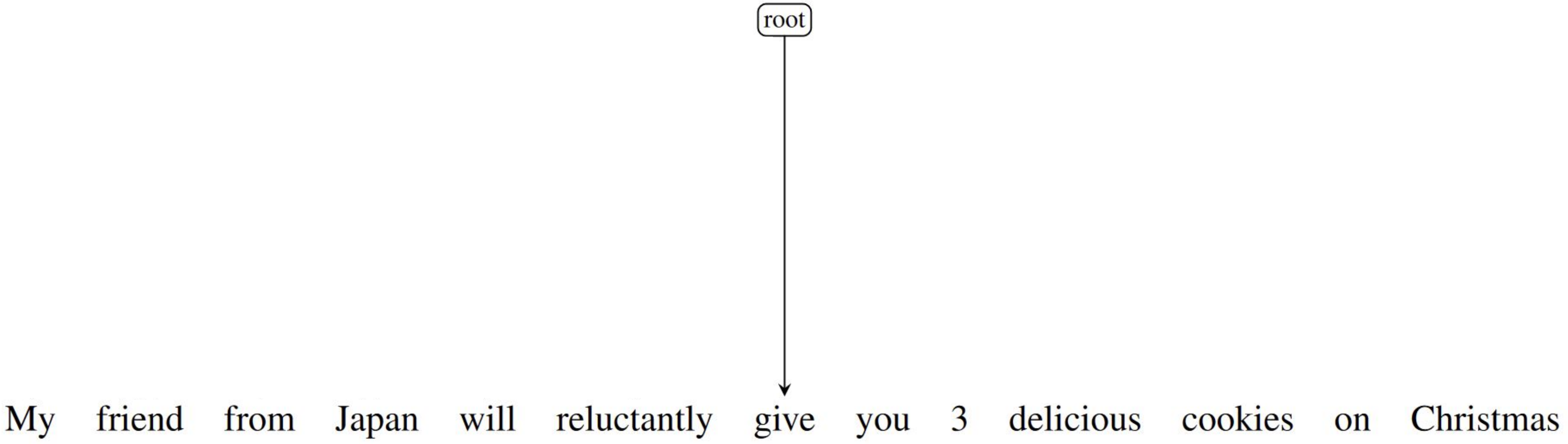
Practice

- Root?

My friend from Japan will reluctantly give you 3 delicious cookies on Christmas

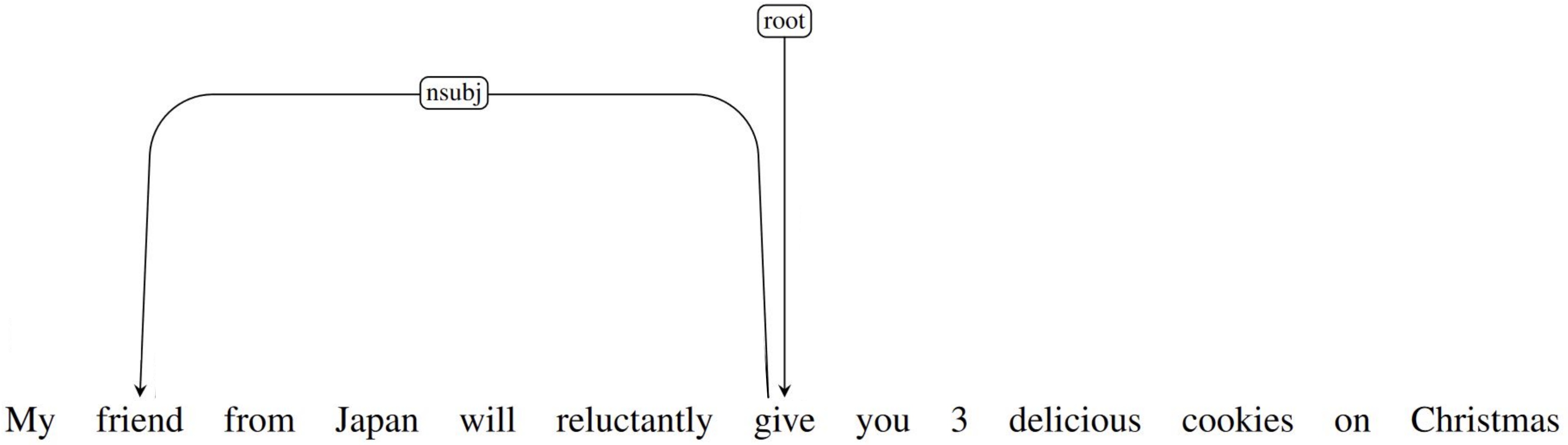
Practice

- Relation to friend?



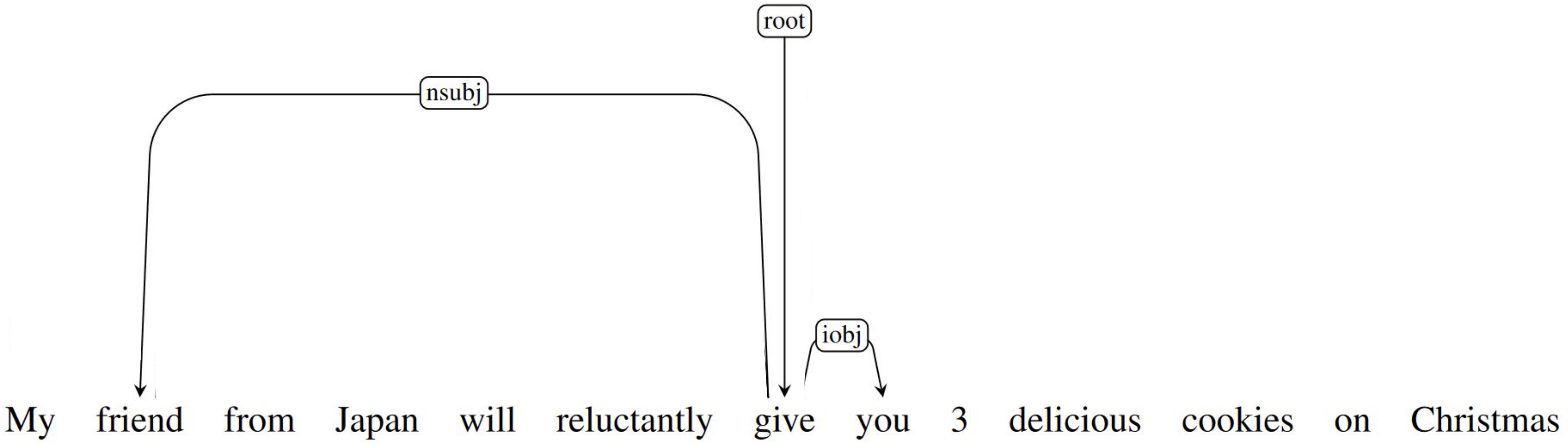
Practice

- Relation to you?



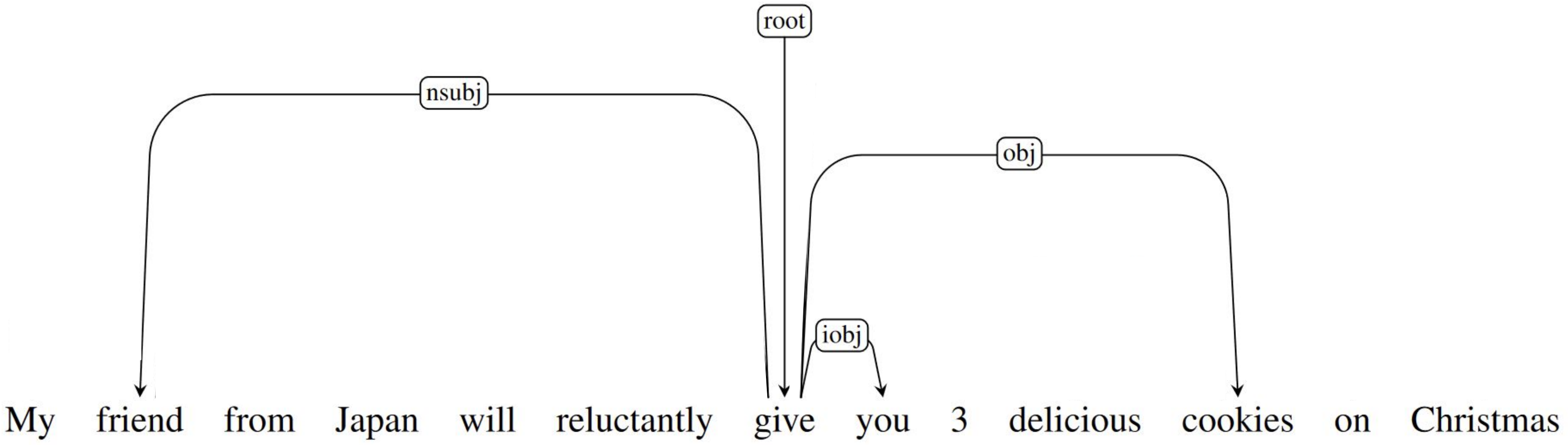
Practice

- Relation to cookies?



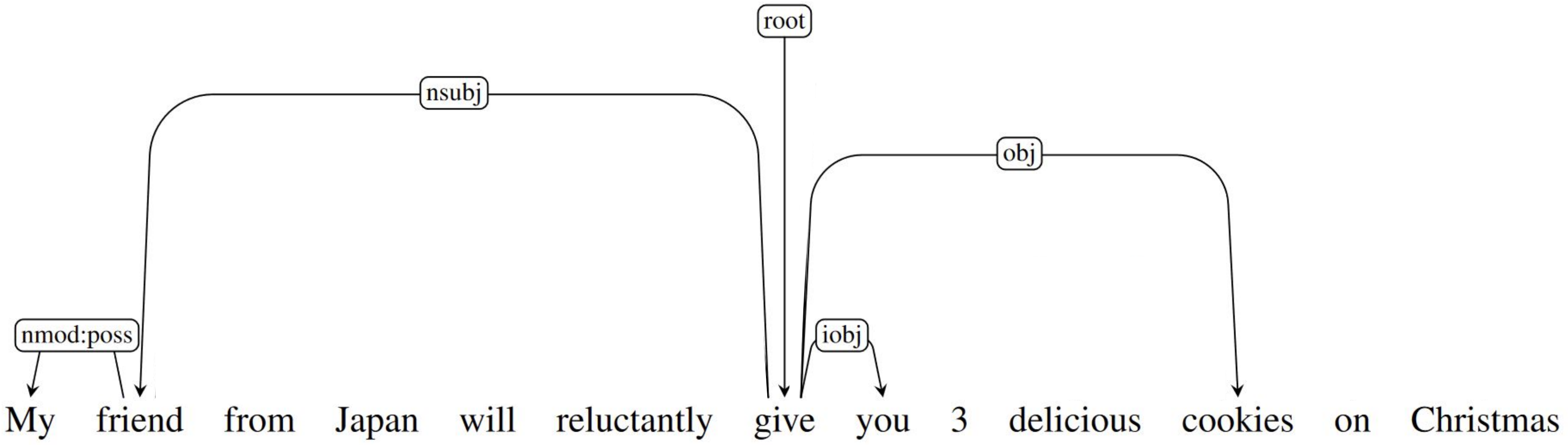
Practice

- Relation to My?



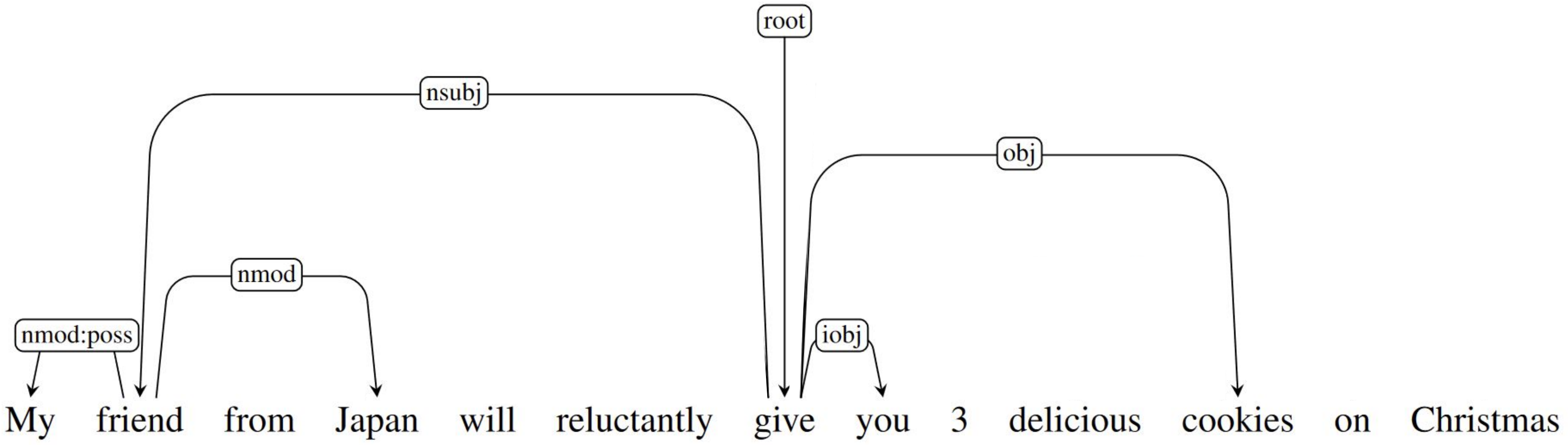
Practice

- Relation to Japan?



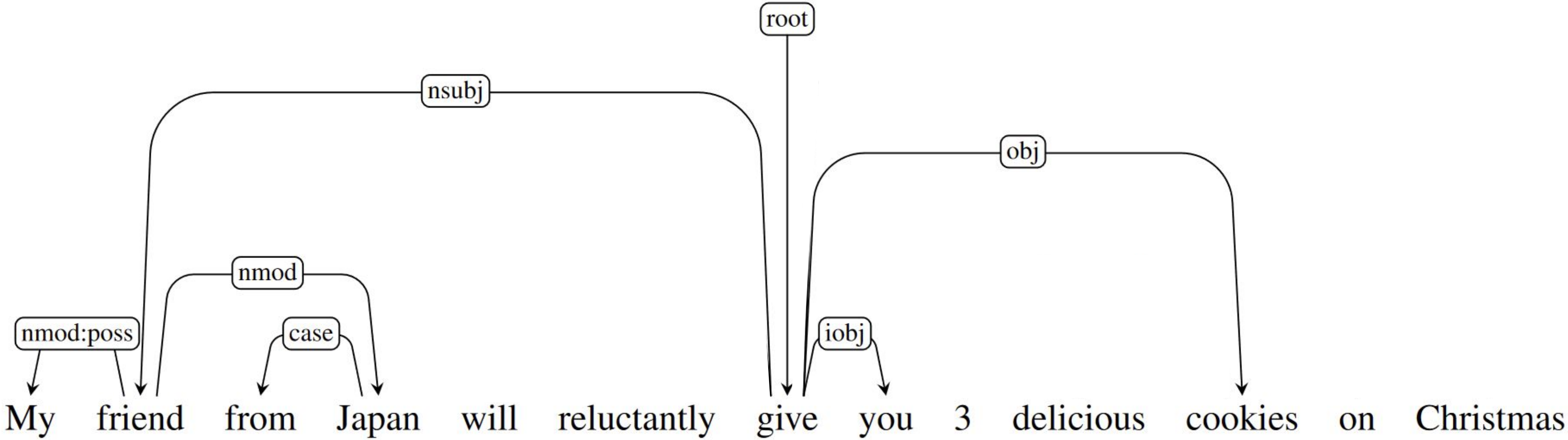
Practice

- Relation to from?



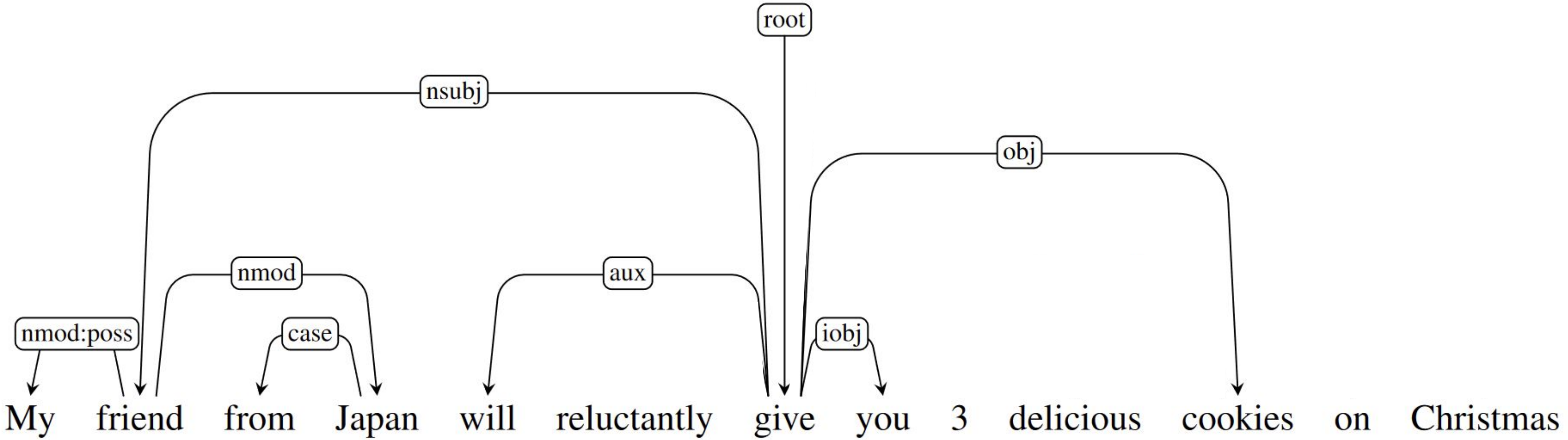
Practice

- Relation to will?



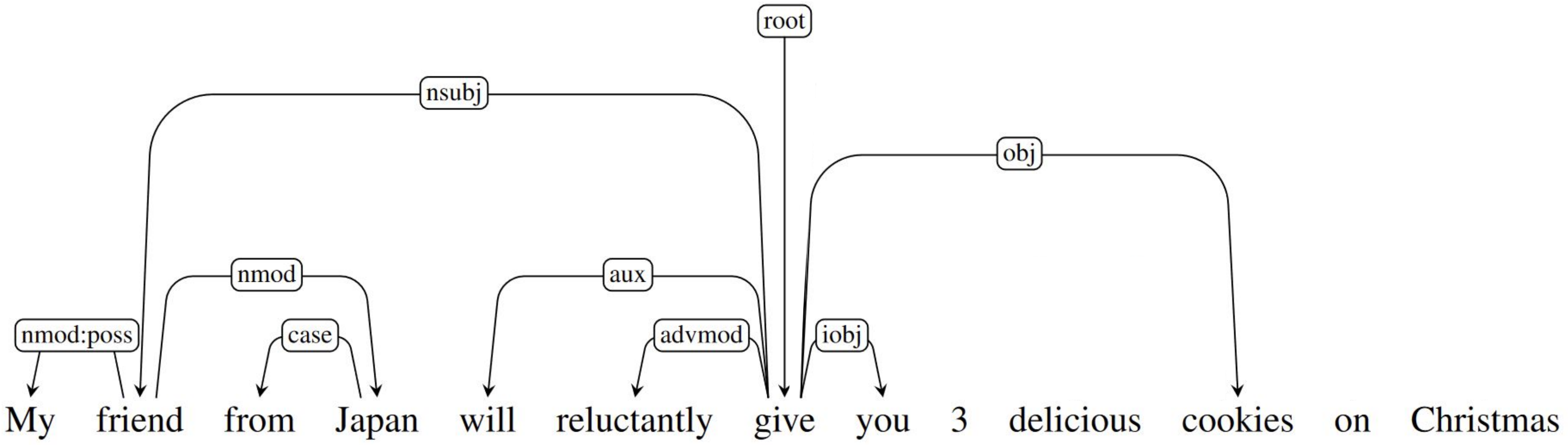
Practice

- Relation to reluctantly?



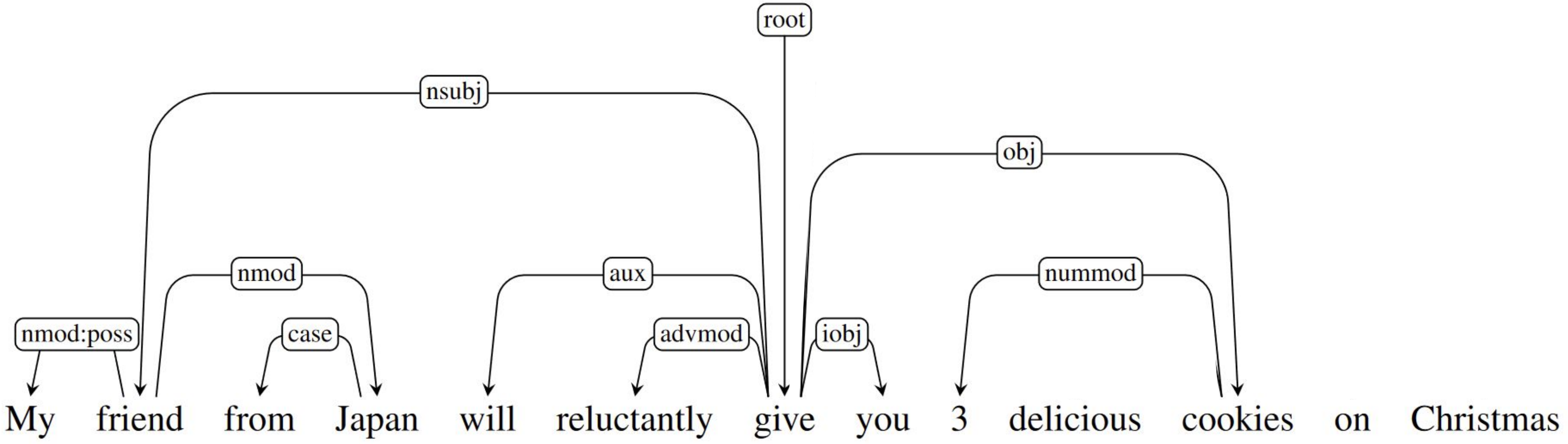
Practice

- Relation to 3?



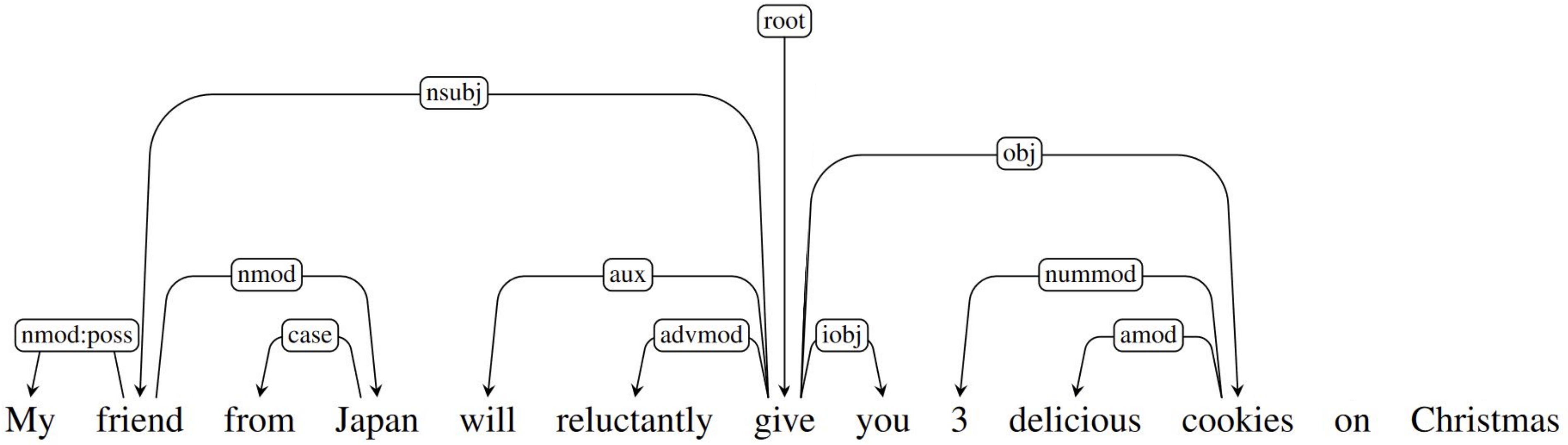
Practice

- Relation to delicious?



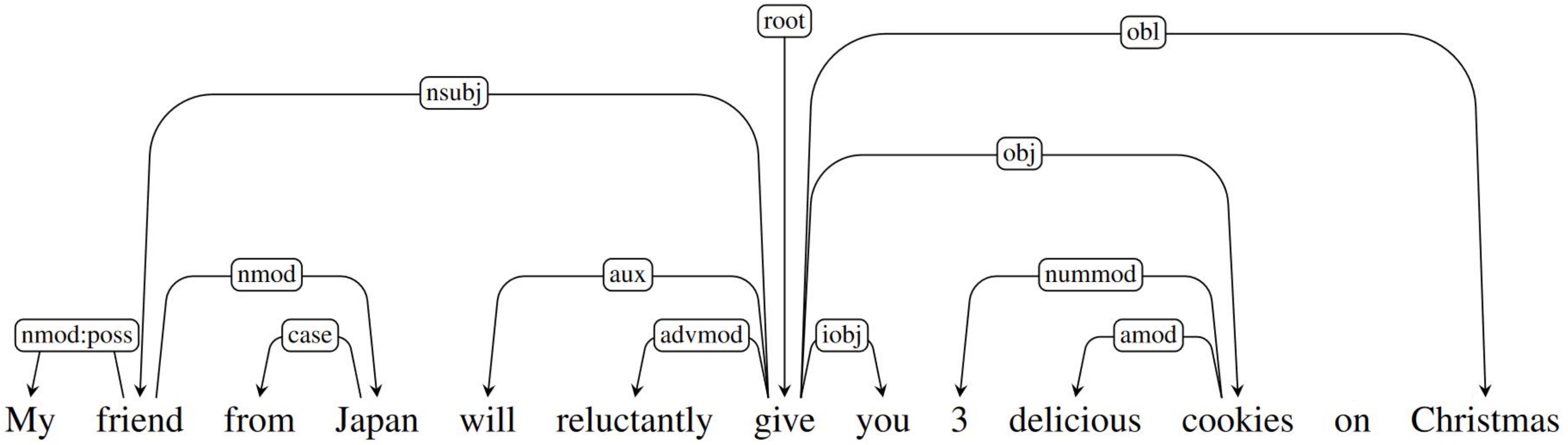
Practice

- Relation to Christmas?



Practice

- Relation to on?



Practice

- Relation to on?