CS310 Natural Language Processing
自然语言处理
Lecture 06 - Dependency Parsing

Instructor: Yang Xu
主讲人：徐炀
xuyang@sustech.edu.cn

Overview

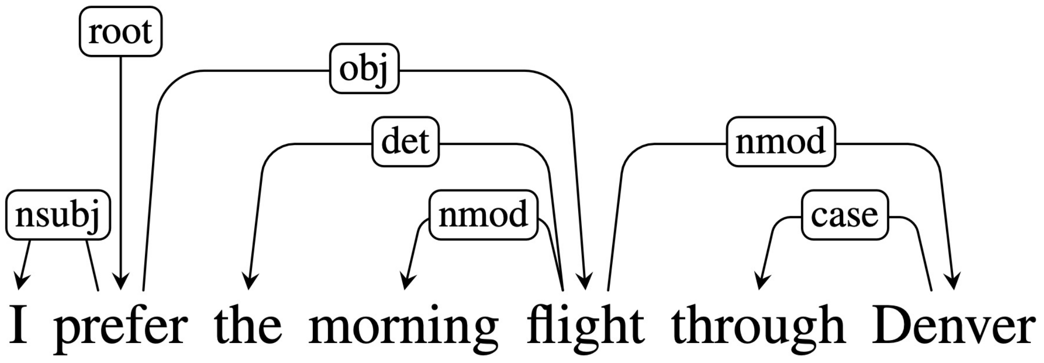
- Dependency Grammars
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluation

Dependency Grammars

- Different from context-free grammars and constituency-based representations
- **Dependency Grammars** describe syntactic structure of a sentence solely in terms of directed grammatical *relations between words*

arc: *n.* 弧线

Labeled arcs from **heads** to **dependents**



$prefer \xrightarrow{\text{nsubj}} I$ — *prefer* is the head of *I*

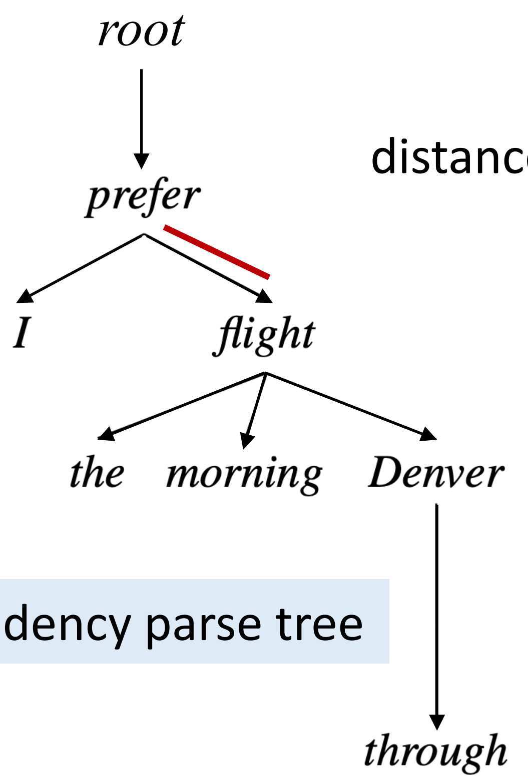
$prefer \xrightarrow{\text{obj}} flight$ — *flight* is the dependent of *prefer*

$root \rightarrow prefer$ — *root* is the head of *prefer* and also the head of the entire structure (root of the tree)

Compare to Context-Free Grammars

- Head-dependent relations **directly** encode important information that is often **buried** in the more complex constituency parses (by CFG)

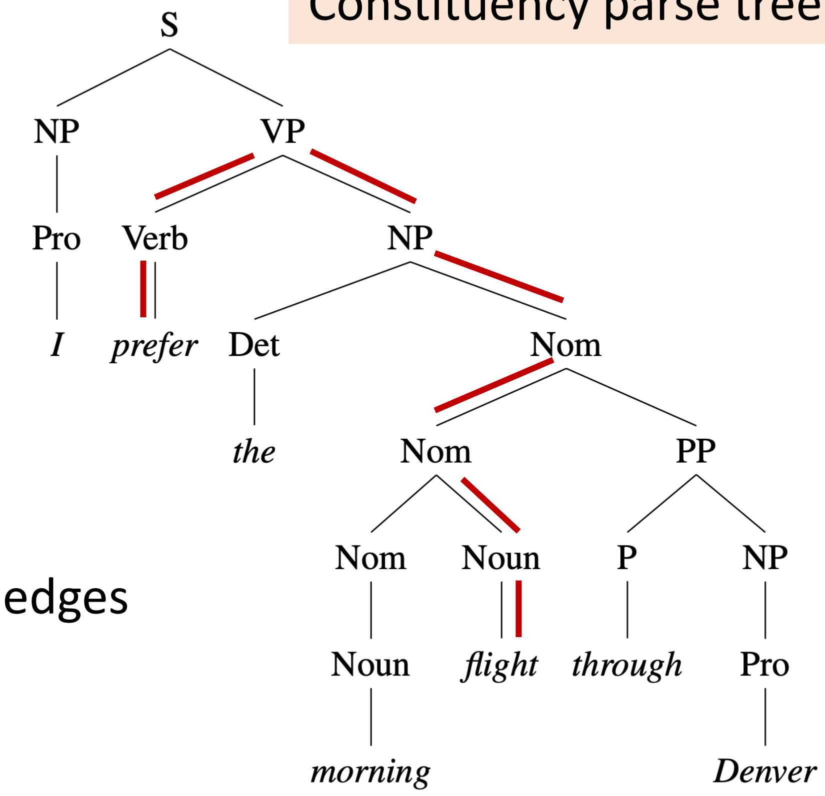
Constituency parse tree



prefer flight

distance(Verb, Obj) = 1 edge

dependency parse tree



prefer flight

distance(Verb, Obj) = 7 edges

Dependency Relations

- Dependency relations consist of a head and dependent
- The ***head*** plays the role of the *central organizing word*, and the ***dependent*** as a kind of modifier
- The relations can be classified based on the grammatical function that the dependent plays with respect to its head
  - Such as *subject*, *direct object*, *indirect object*
- Cross-linguistic standards have been developed for the taxonomies of relations: The **Universal Dependencies (UD)** project (de Marneffe et al., 2021)
  - Across >100 languages; an inventory of 37 dependency relations

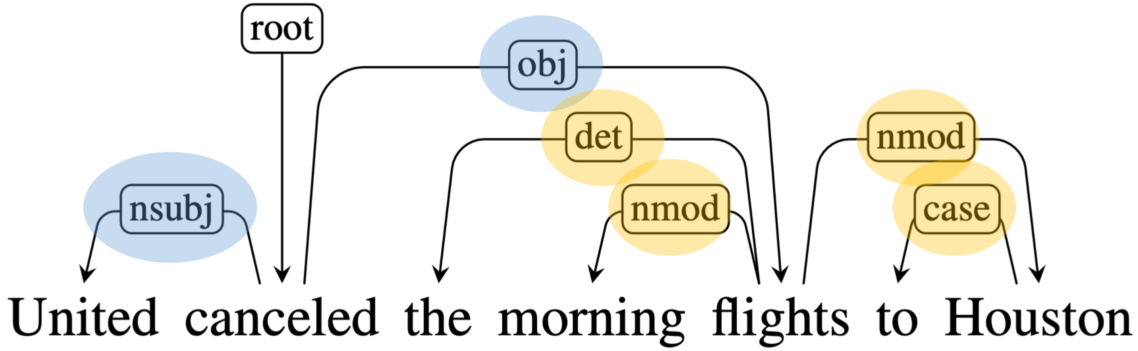
Universal Dependency Relations (Examples)

Clausal Argument Relations	**Description**
NSUBJ	Nominal subject
OBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
OBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Universal Dependency Relations (Examples)

- Two sets of most frequently used relations: **clausal** relations and **modifier** relations



clausal relations describe syntactic roles with respect to a predicate (often a verb)

modifier relations describe the ways that words can modify their heads

NSUBJ and OBJ identify the subject and direct object of the predicate *cancel*

NMOD, DET, and CASE denote modifiers of the nouns *flights* and *Houston*

Graph Formalization of Dependency

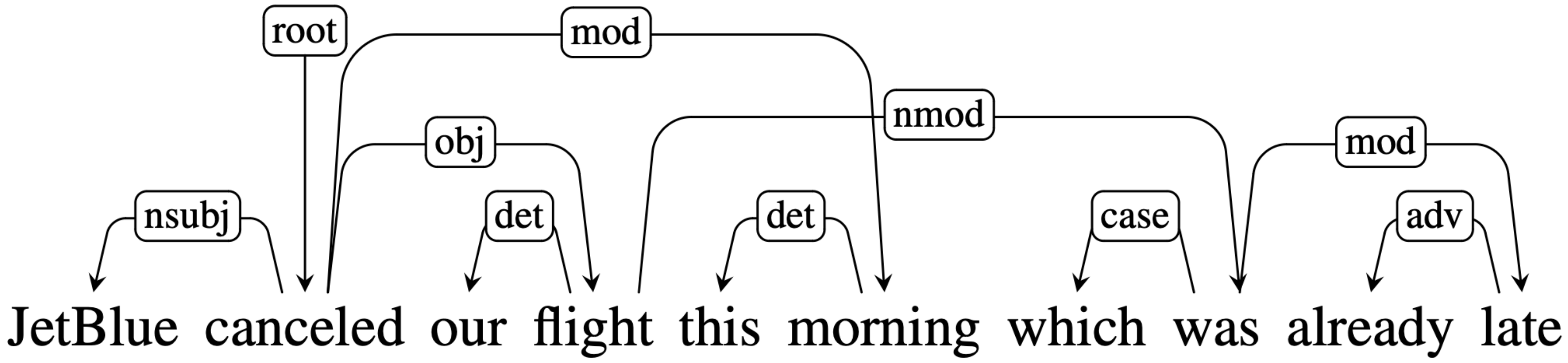
- A dependency structure can be represented as a directed graph $G = (V, A)$, with a set of vertices V, and a set of ordered pairs of vertices A, called **arcs**.
- A dependency tree satisfies the following constraints:
- 1. There is a single root node that has no incoming arcs
- 2. Each vertex has exactly one incoming arc, except for the root
- 3. There is a unique path from the root node to each vertex in V

- With above constraints: each word has single head; a word can have multiple dependents

Projectivity

- An arc is **projective** if there is a path from the head to every word that lies between the head and the dependent
- A dependent tree is said to be projective if all the arcs are projective
- Many valid constructions lead to non-projective trees, particularly in languages with relatively flexible word order

The arc *flight* → *was* is **non-projective**



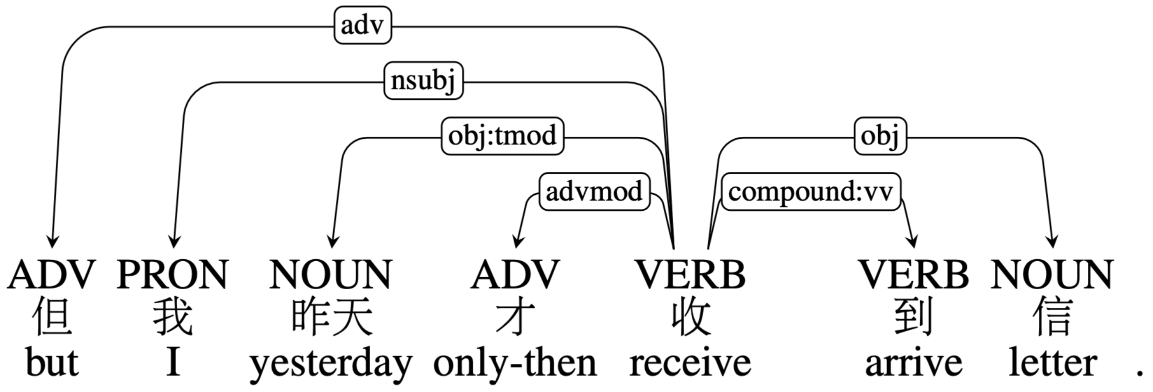
since there is no path from *flight* to the intervening words *this* and *morning*

Projectivity

- Projectivity can be detected by testing if a tree can be drawn with no crossing edges (arcs)
- Why caring about projectivity?
- ***First***, some widely used English dependency treebanks are automatically derived from constituency treebanks through the use of head-finding rules
  - Trees generated this way will always be projective; hence will be incorrect when non-projective examples are encountered
- ***Second***, there are limitations to the widely used dependency parsing algorithms
  - Transition-based parsing (covered later) can only produce projective trees
  - Motivation for more flexible graph-based parsing

Dependency Treebanks

- Treebanks play important role in training and evaluating dependency parsers, and for linguistic studies
- Treebanks are created by: human annotators; hand-corrected from the output of a parser; translated from constituency treebanks
- Largest open community project: The **Universal Dependencies (UD)** project
    - 200+ dependency treebanks in 100+ languages



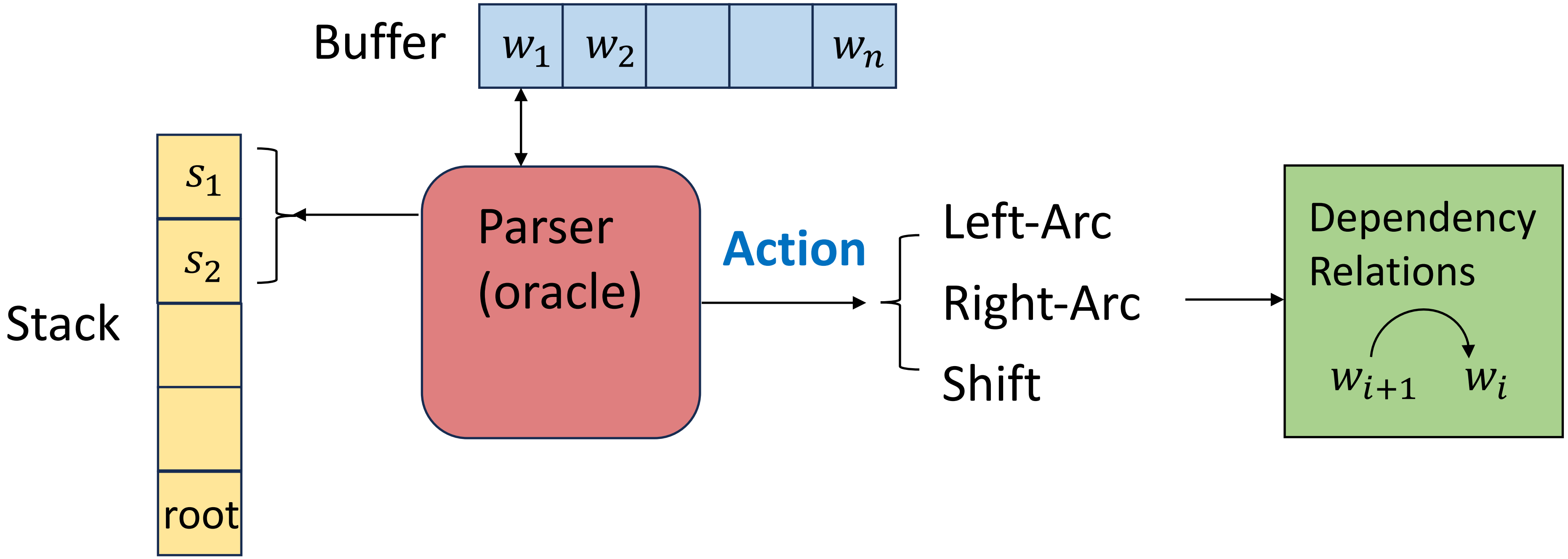
但我昨天才收到信 “But I didn’t receive the letter until yesterday”

Overview

- Dependency Grammars
- **Transition-Based Dependency Parsing**
  - Generic Parsing Process - Arc Standard Approach
  - Generating Training Data
  - Implementation
  - Advanced Methods - Arc Eager Approach
- Graph-Based Dependency Parsing
- Evaluation

Transition-Based Dependency Parsing

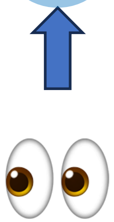
- An architecture that draws on shift-reduce parsing (a paradigm for analyzing programming languages)
- Key components: **stack**, **buffer**, and **oracle**.



Transition actions

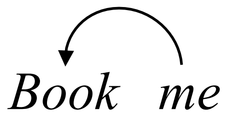
- The transition **actions** corresponds to the intuitive actions when we read a sentence in a single pass and try to create a dependency tree

Book *me* *the* …

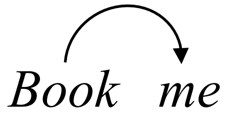


decide:

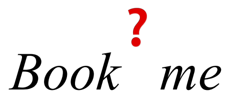
- ! Assign current word as the ***head*** of some previously seen word



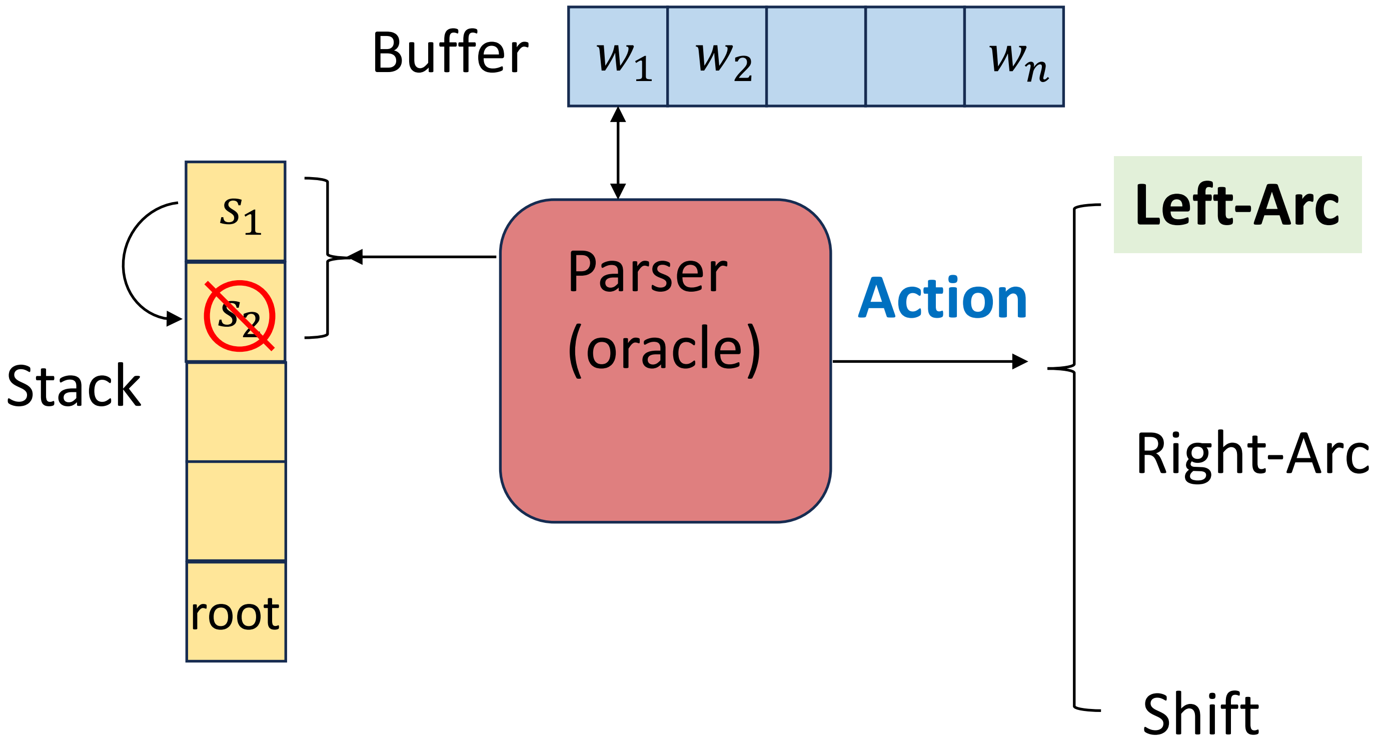
- ! Assign some previously seen word as the ***head*** of current word



- ? Not sure, so postpone the decision and store it for later processing



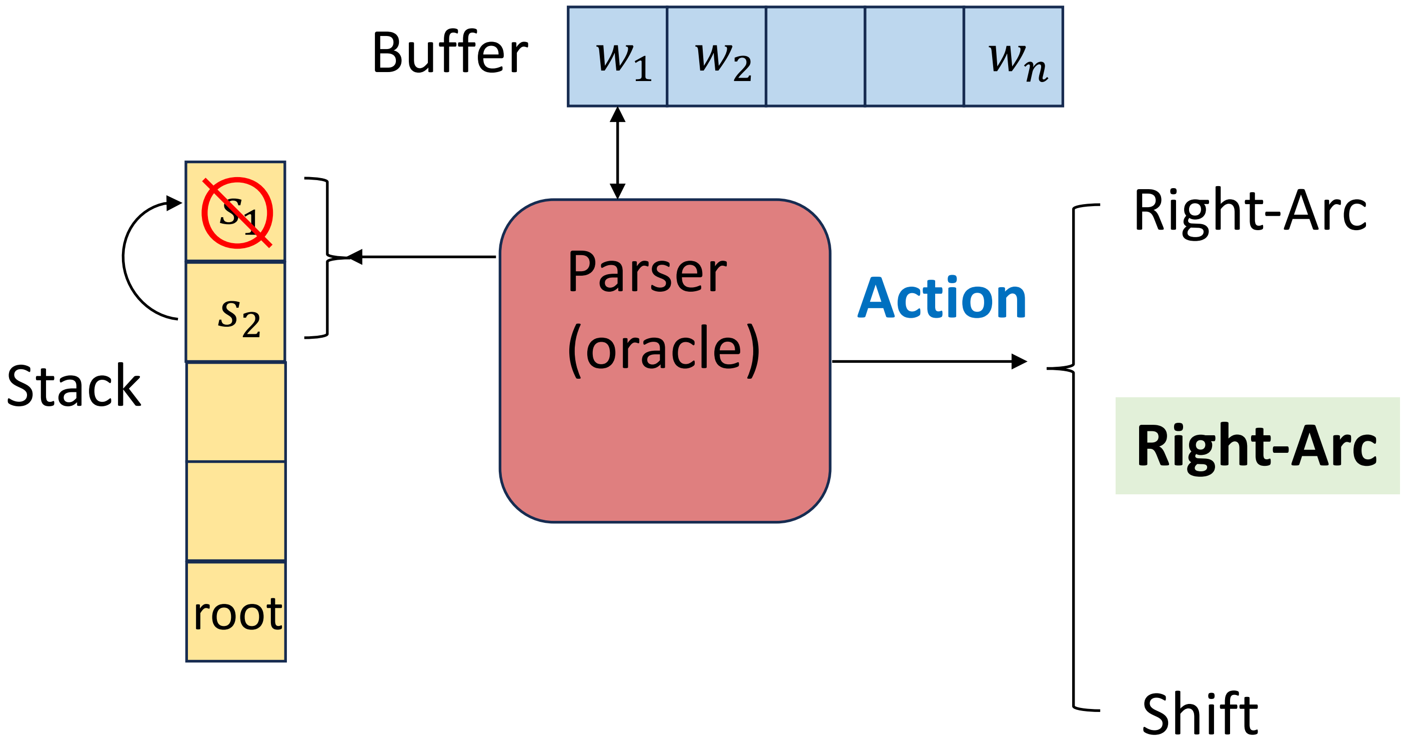
Formalize the transition actions



Assert $s_1 \rightarrow s_2$
(s_1 is head and s_2 is dependent)
Remove s_2

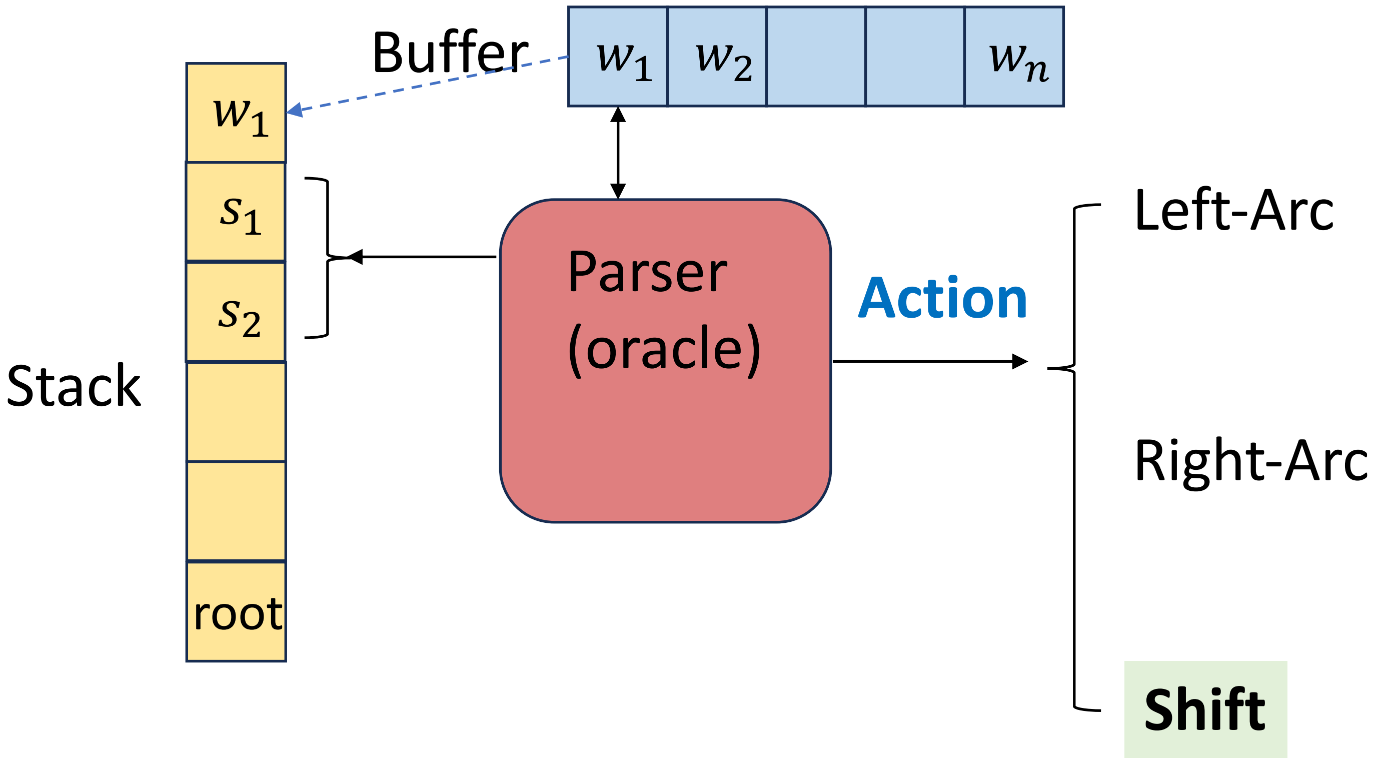
Because each word has exactly one income arc

Formalize the transition actions



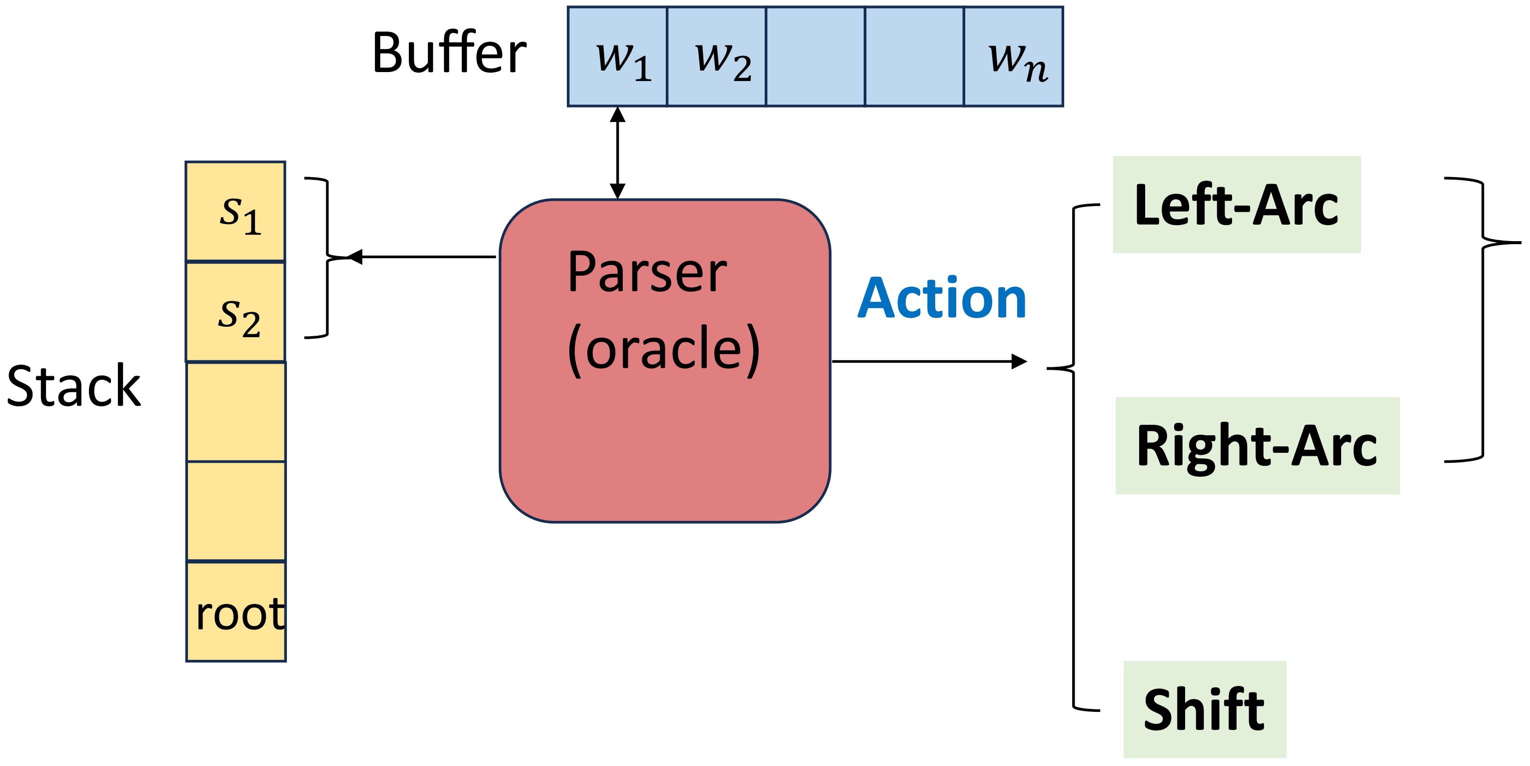
Assert $s_2 \rightarrow s_1$
(s_2 is head and s_1 is dependent)
Remove s_1

Formalize the transition actions



Remove the word from the front of **buffer** and push it to **stack**

Formalize the transition actions



Reduce operations:
"reduce" means combining elements on stack

Preconditions:

- Both Left-Arc and Right-Arc require *two* elements to be on stack
- Left-Arc cannot be applied when ROOT is the second element

Describe a generic transition-based parser

- **Configuration**: Describe the **current state** of parser with the **stack**, an input **buffer** of words, and a set of **relations** representing the dependency tree.

function DEPENDENCYPARSE(*words*) **returns** dependency tree

```
state ← {[root], [words], [] }  ; initial configuration
while state not final
   t ← ORACLE(state)          ; choose a transition operator to apply
   state ← APPLY(t, state)  ; apply it, creating a new state
return state
```

Initial configuration state:

- stack: [root]
- buffer: $[w_1, w_2, \ldots, w_n]$
- relations: []

Final configuration state:

- stack: [root]
- buffer: []
- relations: a set of relations representing the final parse

Parsing ⇒ making a sequence of transitions through the space of possible configurations

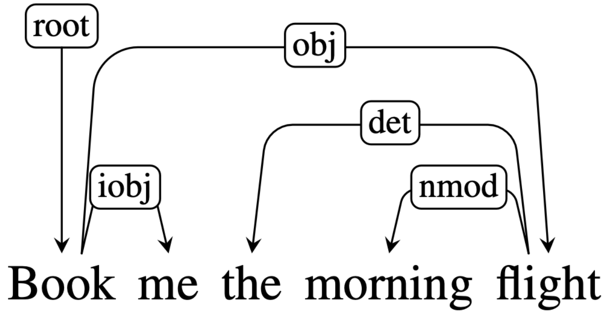
Parsing Example

Input sentence: Book me the morning flight

Initial configuration state:

- stack: [**root**]
- buffer: [book, me, the, morning, flight]
- relations: []

Goal:



Add relation (book → me)

Step	Stack	Buffer	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	Shift	
1	[root, book]	[me, the, morning, flight]	Shift	
2	[root, book, me]	[the, morning, flight]	Right-Arc	(book → me)

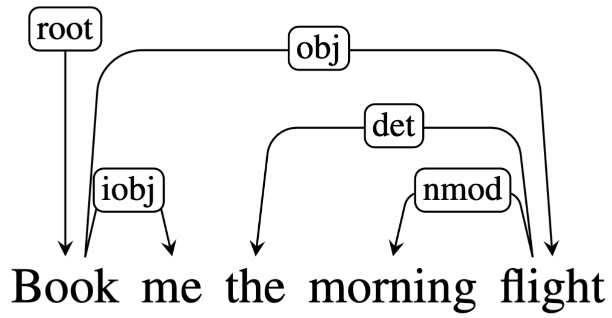
Pop *me* from the stack

Assign *book* as the head of *me*

Parsing Example

Input sentence: Book me the morning flight

Goal:



Step	Stack	Buffer	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	Shift	
1	[root, book]	[me, the, morning, flight]	Shift	
2	[root, book, me]	[the, morning, flight]	Right-Arc	(book → me)
3	[root, book]	[the, morning, flight]	Shift	
4	[root, book, the]	[morning, flight]	Shift	
5	[root, book, the, morning]	[flight]	Shift	
6	[root, book, the, morning, flight]	[]	Left-Arc	(morning ⟵ flight)

Assign *flight* as the head of *morning*

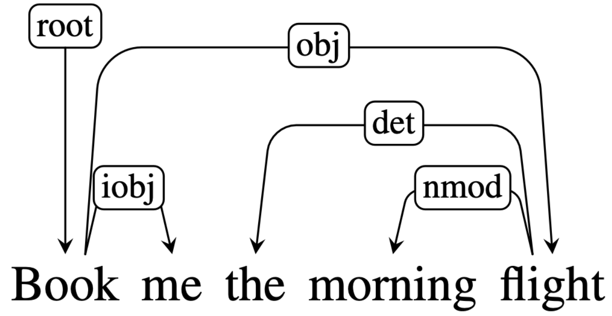
Add relation

Pop *morning* from the stack

Parsing Example

Input sentence: Book me the morning flight

Goal:



Step	Stack	Buffer	Action	Relation Added
6	[root, book, the, morning, flight]	[]	Left-Arc	(morning ⟵ flight)
7	[root, book, ~~the~~, flight]	[]	Left-Arc	(the ⟵ flight)

Pop *the* from the stack

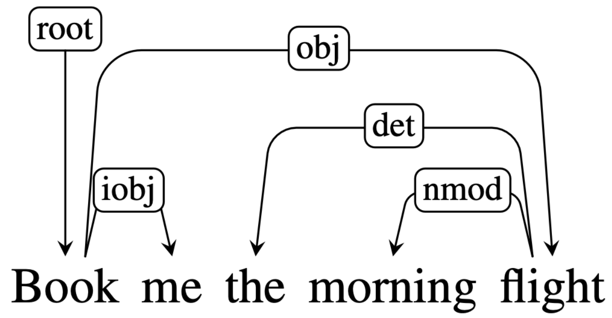
Assign *flight* as the head of *the*

Add relation

Parsing Example

Input sentence: Book me the morning flight

Goal:



Step	Stack	Buffer	Action	Relation Added
6	[root, book, the, morning, flight]	[]	Left-Arc	(morning ⟵ flight)
7	[root, book, the, flight]	[]	Left-Arc	(the ⟵ flight)
8	[root, book, ~~flight~~]	[]	Right-Arc	(book ⟶ flight)

Assign *book* as the head of *flight*

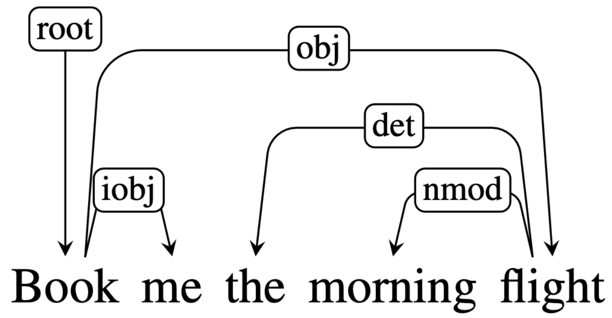
Add relation

Pop *flight* from the stack

Parsing Example

Input sentence: Book me the morning flight

Goal:



Step	Stack	Buffer	Action	Relation Added
6	[root, book, the, morning, flight]	[]	Left-Arc	(morning ⟵ flight)
7	[root, book, the, flight]	[]	Left-Arc	(the ⟵ flight)
8	[root, book, flight]	[]	Right-Arc	(book → flight)
9	[root, book]	[]	Right-Arc	(root → book)

Assign *root* as the head of *book*

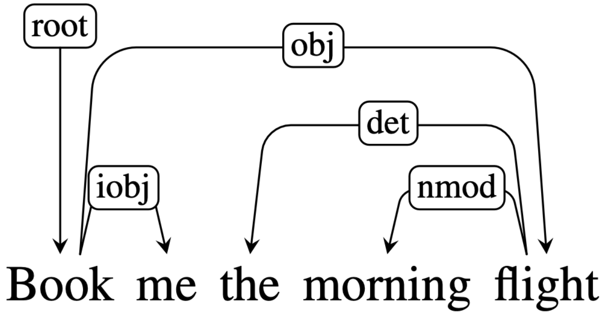
Add relation

Pop *book* from the stack

Parsing Example

Input sentence: Book me the morning flight

Goal:



Step	Stack	Buffer	Action	Relation Added
6	[root, book, the, morning, flight]	[]	Left-Arc	(morning ⟵ flight)
7	[root, book, the, flight]	[]	Left-Arc	(the ⟵ flight)
8	[root, book, flight]	[]	Right-Arc	(book ➝ flight)
9	[root, book]	[]	Right-Arc	(root ➝ book)
9	[root]	[]	Done	

Parse is done when **stack** only contains **root** and **buffer** is empty

Recap of Generic Parser

- Known as **arc standard** approach to transition-based parsing
- Only asserts relations between the top elements on stack
- Once an element has been assigned its head, it is removed from the stack and is not available for further processing

- There are alternative transition systems that have different parsing behaviors (the **arc eager** approached covered later)
- But arc standard is quite effective and easy to implement

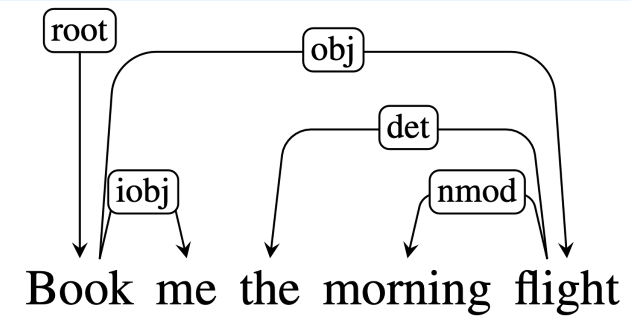
Several Notable Things

- The sequence of transition actions is not the only one leading to a reasonable parse
- We assume the oracle always provides the correct action
  - which is **unlikely** to be true in practice
  - Incorrect choices will lead to incorrect parses since the parser **has no opportunity to go back and make alternative choices**
- In the previous example, dependency relations are without labels
  - To produce labeled parses, need to parameterize the Left-Arc and Right-Arc operators:
  - such as, Left-Arc(NSUBJ), Right-Arc(OBJ)
  - Makes the job of oracle more difficult

Train the Oracle

- Oracle's job: select the appropriate transition action based on the configuration: **stack (S)**, **buffer (B)**, and the already assigned **relations (R)**
- Trained by supervised machine learning
- **Training data**: **configurations** ($\boldsymbol{x}$) annotated with the correction **transition action** (y)
- ($\boldsymbol{x}$, y) are drawn from existing dependency trees

Treebank data:



How to convert?

Training data with paired ($\boldsymbol{x}$, y):

($\boldsymbol{x}_1$ = S_1, B_1, R_1 y_1 = Shift)

($\boldsymbol{x}_2$ = S_2, B_2, R_2 y_2 = Shift)

($\boldsymbol{x}_2$ = S_3, B_3, R_3 y_3 = Shift)

…

Procedure of Generating Training Data

- Given current configuration and a gold-standard **reference** parse
- Choose **Left-Arc** if:
  it produces a correct head-dependent relation
- Choose **Right-Arc** if:
  1) it produces a correct head-dependent relation **and**
  2) all dependents of the top word of stack have already been assigned
- Choose **Shift** otherwise

The restriction on Right-Arc is to prevent a word being popped from stack before all of its dependent have been assigned

Formally, given

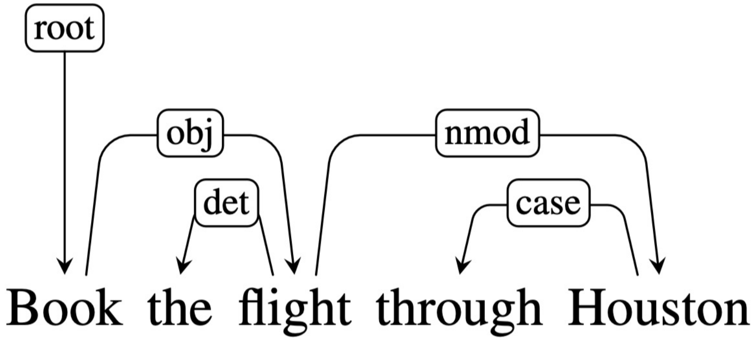
- a stack $\boldsymbol{S}$
- a reference parse with a set of relations $\boldsymbol{R}_{\text{ref}}$
- a set of currently already assigned relations $\boldsymbol{R}_{\text{c}}$

Choose transition action as follows:

- $\text{Left}-\text{Arc}(r)$: **if** $\left(S_1 \xrightarrow{r} S_2\right) \in R_{\text{ref}}$
- $\text{Right}-\text{Arc}(r)$: **if** $\left(S_2 \xrightarrow{r} S_1\right) \in R_{\text{ref}}$ **and**
  $\forall r'$, w.r.t. $\left(S_1 \xrightarrow{r'} w\right) \in R_{\text{ref}} \Rightarrow \left(S_1 \xrightarrow{r'} w\right) \in R_{\text{c}}$
- Shift: **otherwise**

Procedure of Generating Training Data

- Gold-standard sentence: Book the flight through Houston
- with reference parse:



Relations in this reference parse

$R_{\text{ref}} = \{\text{root} \rightarrow \text{book},$
$\text{book} \xrightarrow{\text{obj}} \text{flight},$
$\text{flight} \xrightarrow{\text{det}} \text{the},$
$\text{flight} \xrightarrow{\text{nmod}} \text{Houston},$
$\text{Houston} \xrightarrow{\text{case}} \text{through}\}$

Initial configuration stage:
stack = [root], **buffer** = [book, the, …],
currently assigned relations $\boldsymbol{R}_{\text{c}} = \{\}$

Procedure of Generating Training Data

Step	Stack	Buffer
0	[root]	[book, the, flight, through, Houston]
1	[root, book]	[the, flight, through, Houston]

$\boldsymbol{R}_{\mathrm{c}} = \{\}$

$\boldsymbol{R}_{\mathrm{ref}} = \{\mathrm{root} \rightarrow \mathrm{book}, \mathrm{book} \xrightarrow{\mathrm{obj}} \mathrm{flight}, \mathrm{flight} \xrightarrow{\mathrm{det}} \mathrm{the}, \mathrm{flight} \xrightarrow{\mathrm{nmod}} \mathrm{Houston}, \mathrm{Houston} \xrightarrow{\mathrm{case}} \mathrm{through}\}$

At Step 1:
Left-Arc is not applicable because $(\mathrm{book} \rightarrow \mathrm{root}) \notin R_{\mathrm{ref}}$

Right-Arc is could be applicable since $(\mathrm{root} \rightarrow \mathrm{book}) \in R_{\mathrm{ref}}$, however *book* has not been assigned with its dependent yet:

as we find $\left(\mathrm{book} \xrightarrow{\mathrm{obj}} \mathrm{flight}\right) \in R_{\mathrm{ref}}$
but $\left(\mathrm{book} \xrightarrow{\mathrm{obj}} \mathrm{flight}\right) \notin R_{\mathrm{c}}$

So **Shift** is the only possible action

Choose transition action as follows:

- $\mathrm{Left{-}Arc}(r)$: **if** $\left(S_1 \xrightarrow{r} S_2\right) \in R_{\mathrm{ref}}$
- $\mathrm{Right{-}Arc}(r)$: **if** $\left(S_2 \xrightarrow{r} S_1\right) \in R_{\mathrm{ref}}$ **and** $\forall r'$, w.r.t. $\left(S_1 \xrightarrow{r'} w\right) \in R_{\mathrm{ref}} \Rightarrow \left(S_1 \xrightarrow{r'} w\right) \in R_{\mathrm{c}}$
- Shift: **otherwise**

Procedure of Generating Training Data

Step	Stack	Buffer
0	[root]	[book, the, flight, through, Houston]
1	[root, book]	[the, flight, through, Houston]
2	[root, book, the]	[flight, through, Houston]
3	[root, book, the, flight]	[through, Houston]

$\boldsymbol{R}_{\mathrm{c}} = \{\}$

$\boldsymbol{R}_{\mathrm{ref}} = \{\mathrm{root} \rightarrow \mathrm{book}, \mathrm{book} \xrightarrow{\mathrm{obj}} \mathrm{flight}, \mathrm{flight} \xrightarrow{\mathrm{det}} \mathrm{the}, \mathrm{flight} \xrightarrow{\mathrm{nmod}} \mathrm{Houston}, \mathrm{Houston} \xrightarrow{\mathrm{case}} \mathrm{through}\}$

Choose transition action as follows:

- $\mathrm{Left{-}Arc}(r)$: **if** $\left(S_1 \xrightarrow{r} S_2\right) \in R_{\mathrm{ref}}$
- $\mathrm{Right{-}Arc}(r)$: **if** $\left(S_2 \xrightarrow{r} S_1\right) \in R_{\mathrm{ref}}$ **and** $\forall r'$, w.r.t. $\left(S_1 \xrightarrow{r'} w\right) \in R_{\mathrm{ref}} \Rightarrow \left(S_1 \xrightarrow{r'} w\right) \in R_{\mathrm{c}}$
- Shift: **otherwise**

At Step 3:

Left-Arc is applicable because $\left(\mathrm{flight} \xrightarrow{\mathrm{det}} \mathrm{the}\right) \in R_{\mathrm{ref}}$

then **Left-Arc** is the transition action for this step and $\mathrm{flight} \xrightarrow{\mathrm{det}} \mathrm{the}$ will be added to $\boldsymbol{R}_{\mathrm{c}}$

Procedure of Generating Training Data

Step	Stack	Buffer
0	[root]	[book, the, flight, through, Houston]
1	[root, book]	[the, flight, through, Houston]
2	[root, book, the]	[flight, through, Houston]
3	[root, book, the, flight]	[through, Houston]
4	[root, book, flight]	[through, Houston]

$\boldsymbol{R}_{\mathrm{c}} = \{\text{flight} \xrightarrow{\text{det}} \text{the}\}$

$\boldsymbol{R}_{\mathrm{ref}} = \{\text{root} \rightarrow \text{book}, \text{book} \xrightarrow{\text{obj}} \text{flight}, \text{flight} \xrightarrow{\text{det}} \text{the}, \text{flight} \xrightarrow{\text{nmod}} \text{Houston}, \text{Houston} \xrightarrow{\text{case}} \text{through}\}$

At Step 4:

Left-Arc is not applicable because $(\text{flight} \rightarrow \text{book}) \notin R_{\text{ref}}$ So, only **shift** is viable

We might be tempted to choose Right-Arc and add $(\text{book} \rightarrow \text{flight})$

But we cannot because flight has a dependent that has not been assigned yet: $\text{flight} \xrightarrow{\text{nmod}} \text{Houston}$

If Right-Arc was chosen, then it presents attachment from $Houston$ to $flight$ in later steps

Procedure of Generating Training Data

- So far, 5 training data instances generated:

Step 0:
$$\left(\begin{array}{ll} x_1 = \{\text{S: [root]}, & y_1 = \text{Shift} \\ \quad\quad \text{B: [book, the, flight, through, Houston]}, & \\ \quad\quad R_\text{c}: \{\}\} & \end{array} \right)$$

⋮

Step 3:
$$\left(\begin{array}{ll} x_4 = \{\text{S: [root, book, the, flight]}, & y_4 = \text{Left-Arc} \\ \quad\quad \text{B: [through, Houston]}, & \\ \quad\quad R_\text{c}: \{\}\} & \end{array} \right)$$

Step 4:
$$\left(\begin{array}{ll} x_5 = \{\text{S: [root, book, flight]}, & y_5 = \text{Shift} \\ \quad\quad \text{B: [through, Houston]}, & \\ \quad\quad R_\text{c}: \{\text{flight} \xrightarrow{\text{det}} \text{the}\}\} & \end{array} \right)$$

Complete Generated Data

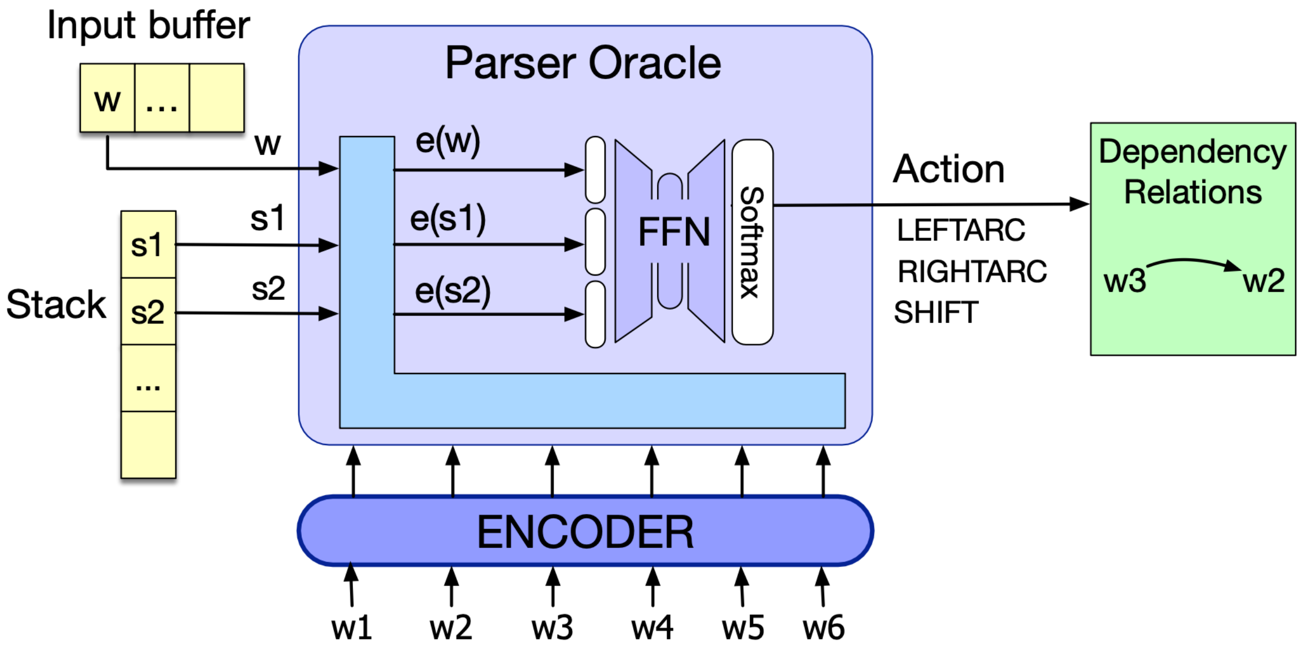
Step	Stack	Buffer	Action
0	[root]	[book, the, flight, through, Houston]	Shift
1	[root, book]	[the, flight, through, Houston]	Shift
2	[root, book, the]	[flight, through, Houston]	Shift
3	[root, book, the, flight]	[through, Houston]	Left-Arc
4	[root, book, flight]	[through, Houston]	Shift
5	[root, book, flight, through]	[Houston]	Shift
6	[root, book, flight, through, Houston]	[]	Left-Arc
7	[root, book, flight, Houston]	[]	Right-Arc
8	[root, book, flight]	[]	Right-Arc
9	[root, book]	[]	Right-Arc
10	[root]	[]	Done

Recap for Generating Training Data

- The whole process is a *simulation* of how a parser works
- Simulate the action of a correct parser with a reference dependency tree
- A training data instance is a configuration-transition (action) pair
- Record the correct parser action at each step as progressing through each training example

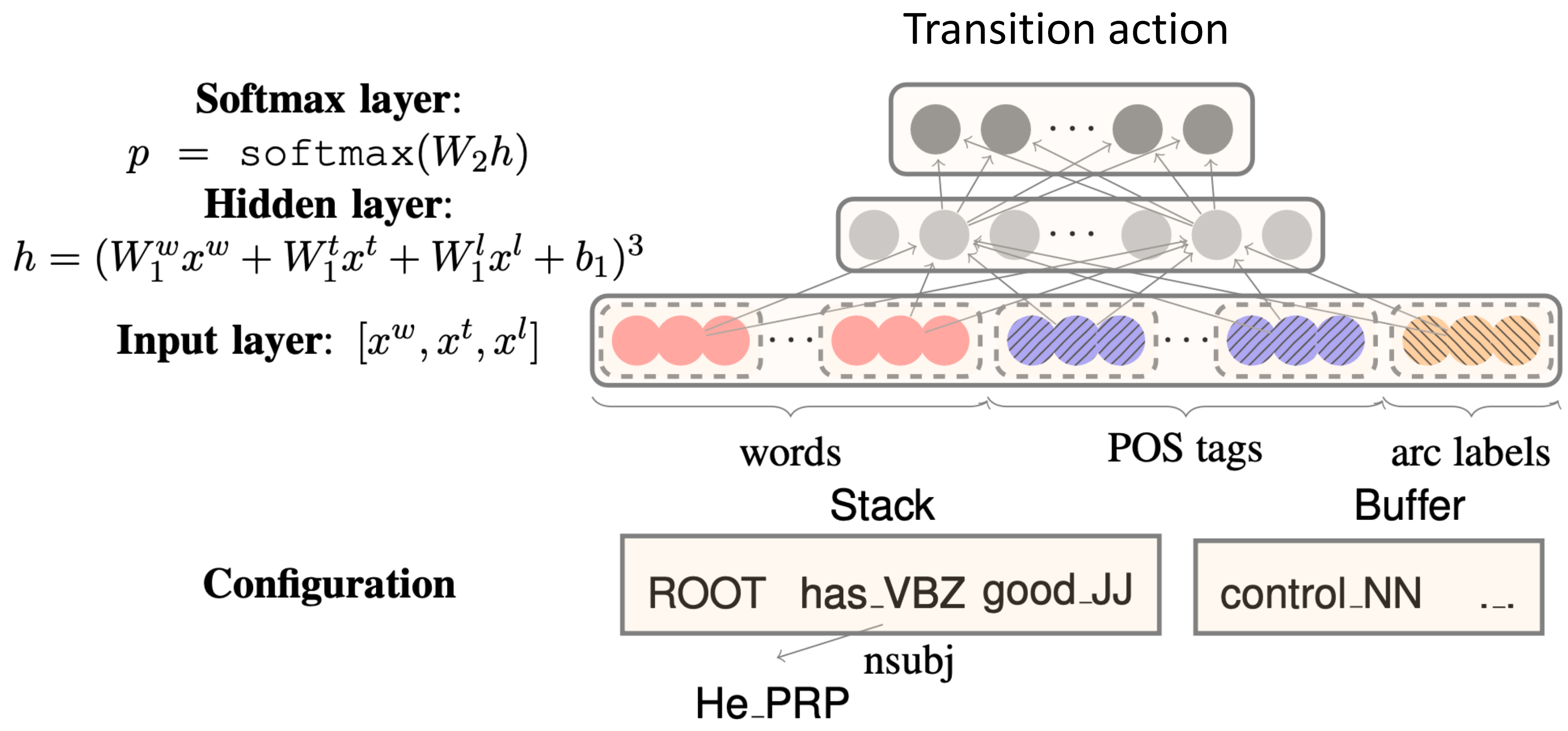
Implementing the Oracle: A Neural Classifier

- **A standard architecture**:
- Pass the sentence through a neural encoder
- Take the vector representation of the top 2 words on stack and the first word on buffer, *concatenate* them
- Pass to a feedforward network to predict the transition action (learning done with cross-entropy loss)



Reference: Chen and Manning, 2014; Kiperwasser and Goldberg, 2016; Kulmizev et al., 2019

A Feed-Forward Neural Implementation



Features used:

- The top 3 words on the stack and buffer, respectively: $s_0, s_1, s_2, b_0, b_1, b_2$
- The POS tags of the words
- …

Reference: Chen, D., & Manning, C. D. (2014, October). A fast and accurate dependency parser using neural networks.

Decoding/Inference Stage: Greedy Parsing

Algorithm 1 Greedy transition-based parsing

Input: sentence $s = w_1, \ldots, x_w,\ t_1, \ldots, t_n$, parameterized function $\text{SCORE}_\theta(\cdot)$ with parameters θ.
$c \leftarrow \text{INITIAL}(s)$
while not $\text{TERMINAL}(c)$ **do**
  $\hat{t} \leftarrow \arg\max_{t \in \text{LEGAL}(c)} \text{SCORE}_\theta(\phi(c), t)$
  $c \leftarrow \hat{t}(c)$
return $tree(c)$

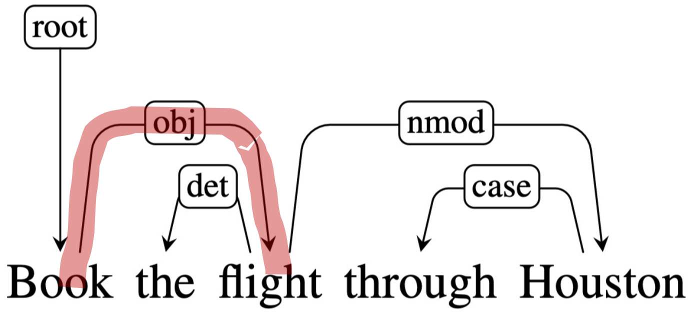
Figure from Kiperwasser and Goldberg (2016)

- TERMINAL(c) can be manually decided
- $t \in \text{LEGAL}(c)$ means not all predictions from the oracle are legal transitions: Left-Arc or Right-Arc on empty stack; Left-Arc to root; etc.

Advanced Methods: Alternative Transition System

- Alternative to arc standard: **arc eager** transition system
- Where “eager” is from -- **assert Right-Arc much sooner** than arc standard

Revisit arc standard



book $\xrightarrow{\text{obj}}$ flight is assigned at step 8

But *book* and *flight* first met much earlier at step 4

Step	Stack	Buffer	Action
0	[root]	[book, the, flight, through, Houston]	Shift
1	[root, book]	[the, flight, through, Houston]	Shift
2	[root, book, the]	[flight, through, Houston]	Shift
3	[root, book, the, flight]	[through, Houston]	Left-Arc
4	[root, book, flight]	[through, Houston]	Shift
5	[root, book, flight, through]	[Houston]	Shift
6	[root, book, flight, through, Houston]	[]	Left-Arc
7	[root, book, flight, Houston]	[]	Right-Arc
8	[root, book, flight]	[]	Right-Arc
9	[root, book]	[]	Right-Arc
10	[root]	[]	Done

Advanced Methods: Alternative Transition System

- The reason book $\xrightarrow{\text{obj}}$ flight cannot be assigned at Step 4 is due to the presence of the modifier *through Houston*.
- In arc-standard approach, a dependent (*flight*) is removed from stack as soon as it is assigned its head (*book*)

If *flight* had been assigned book as its head at Step 4, it would no longer be available to serve as the head of *Houston*!

So we have to delay this action to Step 8

3	[root, book, the, flight]	[through, Houston]	Left-Arc
4	[root, book, flight]	[through, Houston]	Shift
5	[root, book, flight, through]	[Houston]	Shift
6	[root, book, flight, through, Houston]	[]	Left-Arc
7	[root, book, flight, Houston]	[]	Right-Arc
8	[root, book, flight]	[]	Right-Arc

While this delay doesn't cause any issues in this example, in general the longer a word has to wait to get assigned its head, the more opportunities there are for something to go awry (wrong)

Arc Eager Transition System

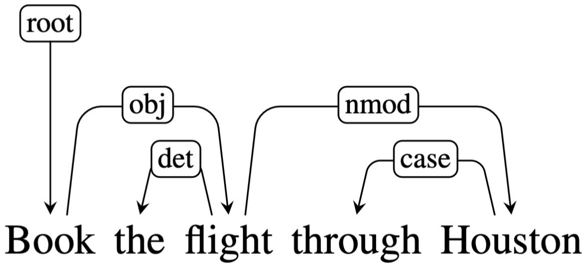
- Arc eager system addresses the issue by allowing words to be attached to their heads as early as possible
- *before* all the subsequent dependents have been seen
- Make changes to the Left-Arc and Right-Arc operators and add a new Reduce operator

- **Left-Arc**: Assert a head-dependent relation buffer[1] $\longrightarrow$ stack[1]; pop stack
- **Right-Arc**: Assert a head-dependent relation stack[1] $\longrightarrow$ buffer[1]; shift buffer[1] to stack
- **Shift**: move buffer[1] to stack
- **Reduce**: Pop stack

Difference from arc standard:

- Left-Arc and Right-Arc are applied to the top of **stack** and the front of **buffer**, instead of the top two elements of stack
- Right-Arc now moves the dependent to stack from buffer rather than removing it
  thus, making it still available to serve as head for subsequent words

Arc Eager Parsing Example



Step	Stack	Buffer	Action
0	[root]	[book, the, flight, through, Houston]	Right-Arc
1	[root, book]	[the, flight, through, Houston]	Shift
2	[root, book, the]	[flight, through, Houston]	Left-Arc
3	[root, book]	[flight, through, Houston]	Right-Arc
4	[root, book, flight]	[through, Houston]	Shift
5	[root, book, flight, through]	[Houston]	Left-Arc
6	[root, book, flight]	[Houston]	Right-Arc
7	[root, book, flight, Houston]	[]	Reduce
8	[root, book, flight]	[]	Reduce
9	[root, book]	[]	Reduce
10	[root]	[]	Done

Overview

- Dependency Grammars
- Transition-Based Dependency Parsing
- **Graph-Based Dependency Parsing**
- Evaluation

Graph-Based Dependency Parsing

- Transition-based parsing has trouble when heads are very far from dependents
- Graph-based parsing avoid this difficulty by scoring entire trees, rather than relying on greedy local decisions; can produce non-projective trees
- **Idea**: Search through the space of possible trees to find a tree that maximizes some score over the given sentence

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \text{Score}(t, S)$$

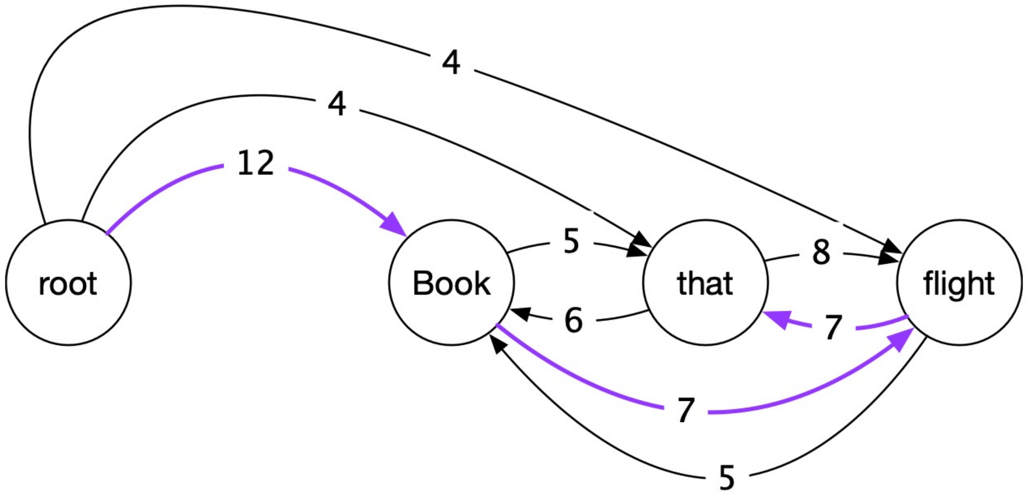
Given a sentence S
$\mathcal{G}_S$: the space of all possible trees for S

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

Score is edge-factored: sum of the scores of edges comprising the tree

Graph-Based Dependency Parsing

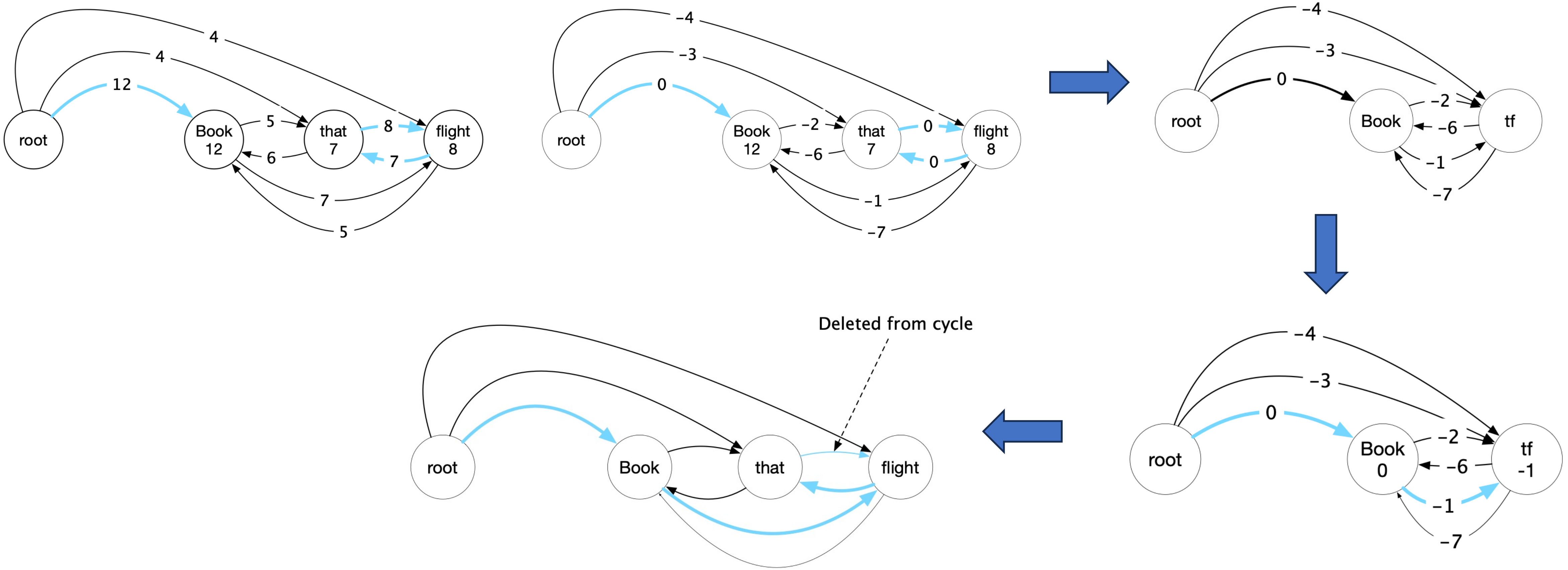
- Using Graph traversal algorithm: Maximum spanning tree problem
- Creating a graph G which is a *fully-connected*, weighted, directed graph where the vertices are the input words and the directed edges represent **all possible head-dependent** assignments



The weights reflect the score for each possible head-dependent relation assigned by some scoring algorithm

Graph-Based Dependency Parsing

- Find the maximum spanning tree by eliminating cycles

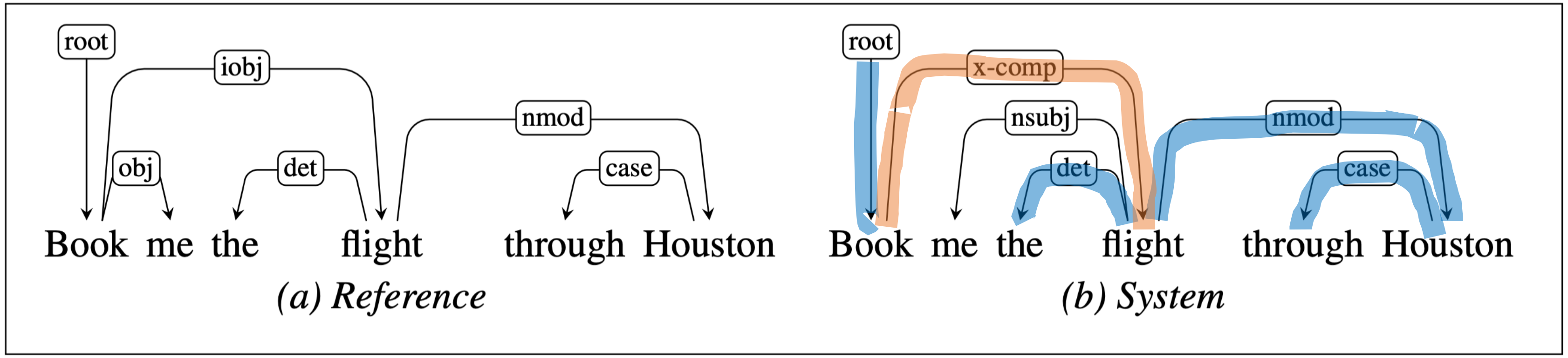


Overview

- Dependency Grammars
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- **Evaluation**

Evaluation of Dependency Parsers

- Labeled attachment accuracy (**LAC**) and unlabeled attachment accuracy (**UAC**)
- LAC: Proper assignment of word to its head along with the correct dependency relation
- UAC: Simply focuses on the correctness of the assigned head, ignoring the dependency relation



The parser correctly finds 4 of the 6 relations in the reference parse ⇒ **LAS = 2/3**

But $book \xrightarrow{\text{x−comp}} flight$ is only wrong in label but still correct in head-dependent ⇒ **UAS = 5/6**

Recap

- Head-dependent relations are a good proxy for the semantic relationship between predicates and arguments
- Dependency grammars are currently more common than constituency grammars in NLP
- Transition-based parsing is a greedy algorithm
  - Limits: can only product projective trees
- Graph-based parsing scores entire tree
  - More accurate for long sentences
  - Can produce non-projective trees

To-Do List

- Start working on A4
- Read Chapter 10 of SLP3: Transformers and Large Language Models
- Attend Lab 7

References

- Covington, M. 2001. A fundamental algorithm for dependency parsing. *Proceedings of the 39th Annual ACM Southeast Conference*.
- Chen, D., & Manning, C. D. (2014, October). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 740-750).
- Kiperwasser, E., & Goldberg, Y. (2016). Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, *4*, 313-327.