COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

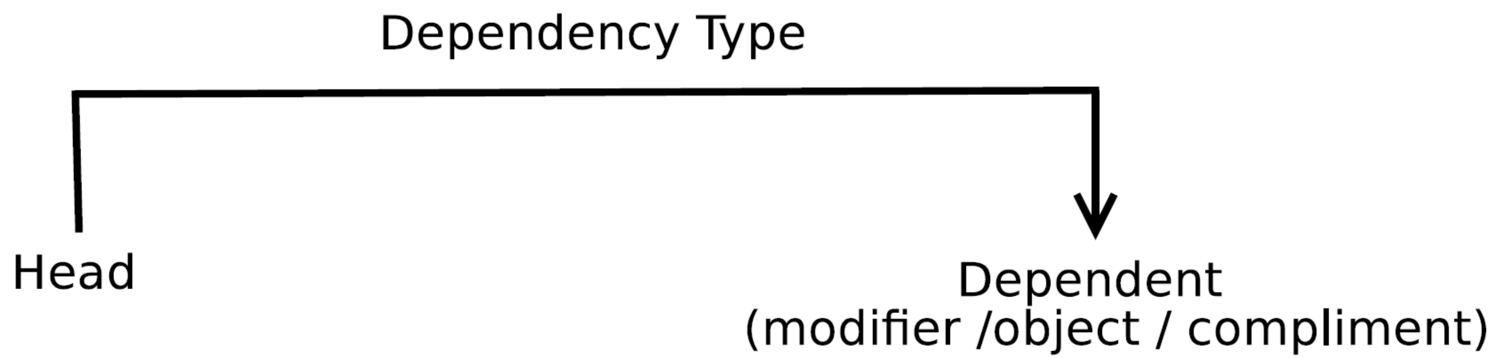
Dependency Parsing

CMSC 470

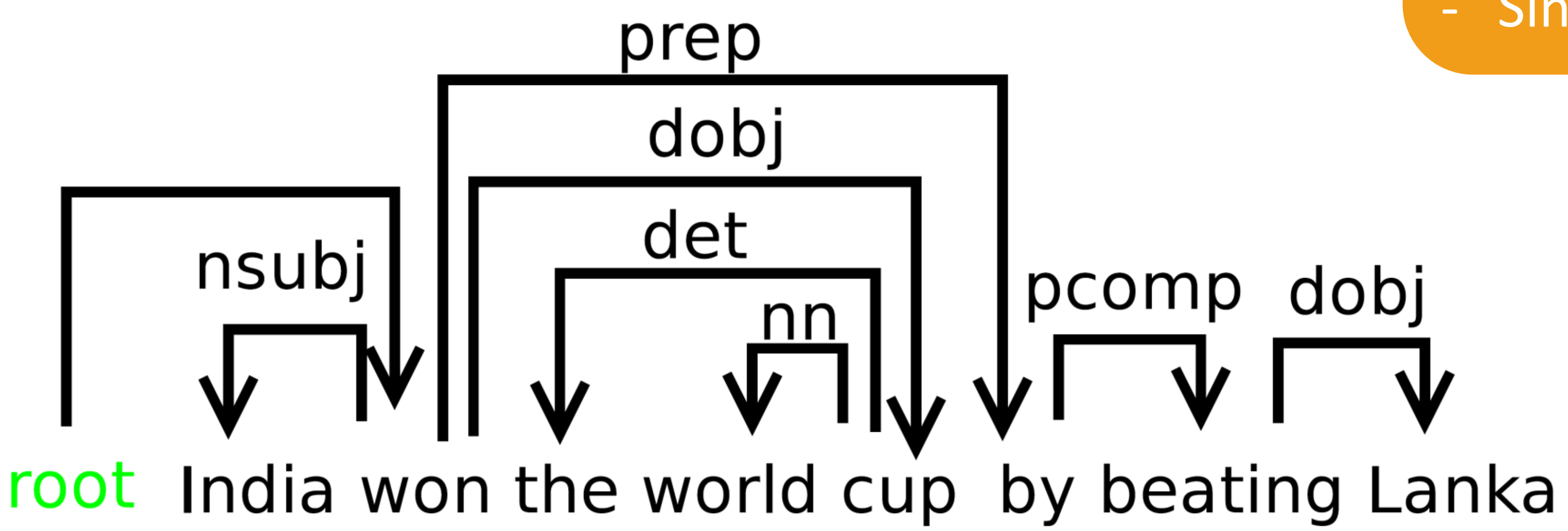
Marine Carpuat

Dependency Grammars

- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies



Example Dependency Parse



Dependencies (usually) form a tree:
- Connected
- Acyclic
- Single-head

Universal Dependencies project

- Set of dependency relations that are
  - Linguistically motivated
  - Computationally useful
  - Cross-linguistically applicable

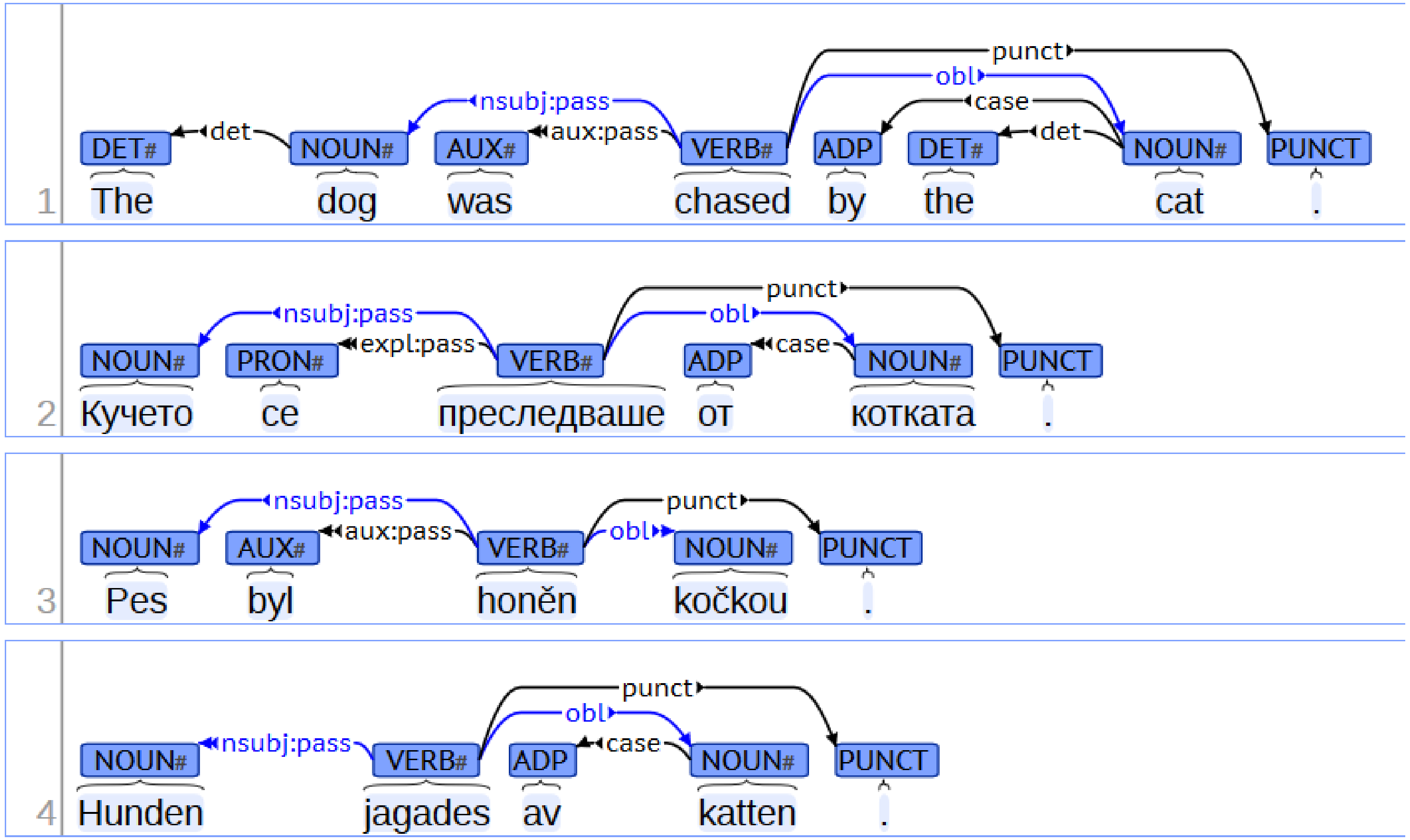
  [Nivre et al. 2016]

- 100+ dependency treebanks for more than 60 languages

universaldependencies.org

Universal Dependencies Illustrated

Parallel examples for English, Bulgarian, Czech & Swedish



https://universaldependencies.org/introduction.html

Universal Dependencies
Design principles

- UD needs to be satisfactory on linguistic analysis grounds for individual languages.
- UD needs to be good for linguistic typology, i.e., providing a suitable basis for bringing out cross-linguistic parallelism across languages and language families.
- UD must be suitable for rapid, consistent annotation by a human annotator.
- UD must be suitable for computer parsing with high accuracy.
- UD must be easily comprehended and used by a non-linguist, whether a language learner or an engineer with prosaic needs for language processing. We refer to this as seeking a *habitable* design, and it leads us to favor traditional grammar notions and terminology.
- UD must support well downstream language understanding tasks (relation extraction, reading comprehension, machine translation, …).

https://universaldependencies.org/introduction.html

Syntax in NLP

- Syntactic analysis can be useful in many NLP applications
  - Grammar checkers
  - Dialogue systems
  - Question answering
  - Information extraction
  - Machine translation
  - …
- Sequence models can go a long way but syntactic analysis is particularly useful
  - In low resource settings
  - In tasks where precise output structure matters

Syntactic analysis can help NLP tasks by

I saw a girl with a telescope

After much economic progress over the years, the country has…

The country, which has made much economic progress over the years, still has…

Providing scaffolding for semantic analysis (and representing or resolving ambiguity)

Helping generalization (e.g., by capturing long-distance dependencies)

Data-driven dependency parsing

Goal: learn a good predictor of dependency graphs

Input: sentence

Output: dependency graph/tree G = (V,A)

Can be framed as a structured prediction task

- very large output space
- with interdependent labels

2 dominant approaches: transition-based parsing and graph-based parsing

Transition-based dependency parsing

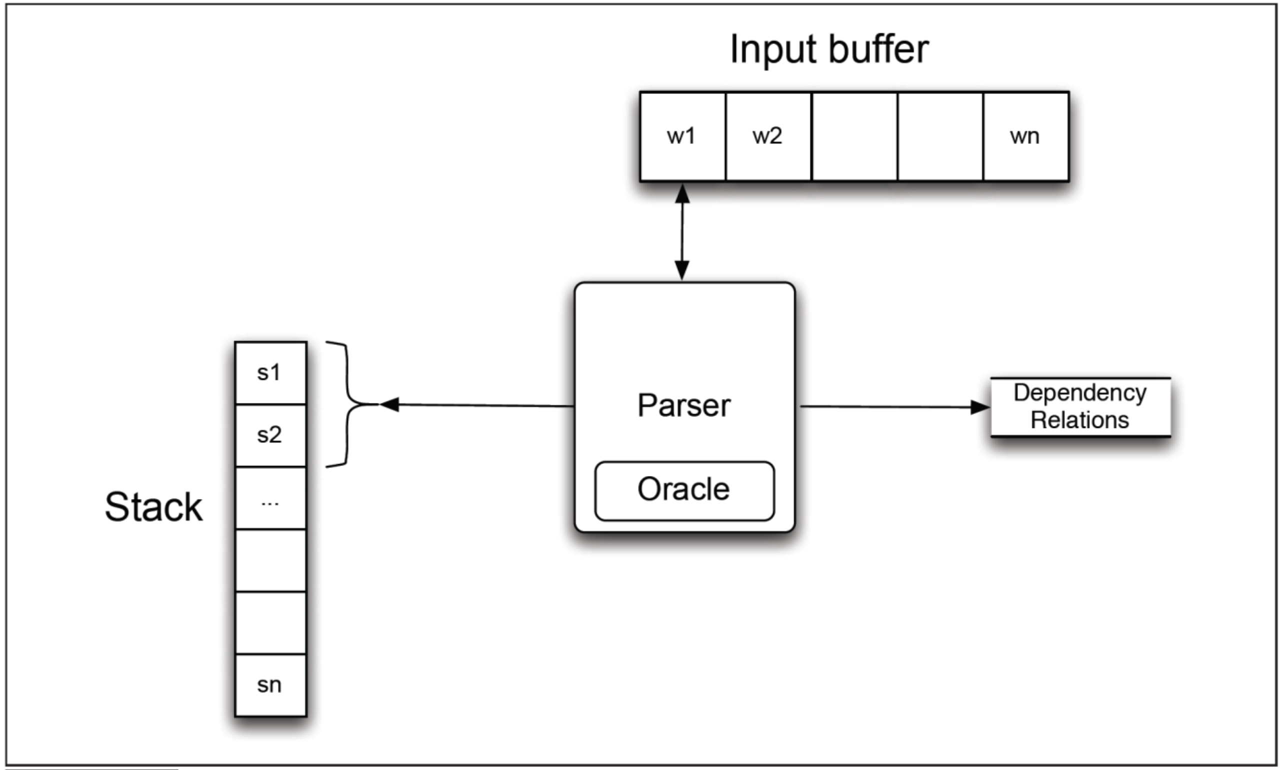


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

- Builds on shift-reduce parsing
  [Aho & Ullman, 1972]
- **Configuration**
  - **Stack**
  - **Input buffer** of words
  - Set of dependency relations
- Goal of parsing
  - find a final configuration where
  - all words accounted for
  - Relations form dependency tree

Defining Transitions

- **Transitions**
  - Are functions that produce a new configuration given current configuration
  - Parsing is the task of finding a sequence of transitions that leads from start state to desired goal state
- **Start state**
  - Stack initialized with ROOT node
  - Input buffer initialized with words in sentence
  - Dependency relation set = empty
- **End state**
  - Stack and word lists are empty
  - Set of dependency relations = final parse

Arc Standard Transition System defines 3 transition operators [Covington, 2001; Nivre 2003]

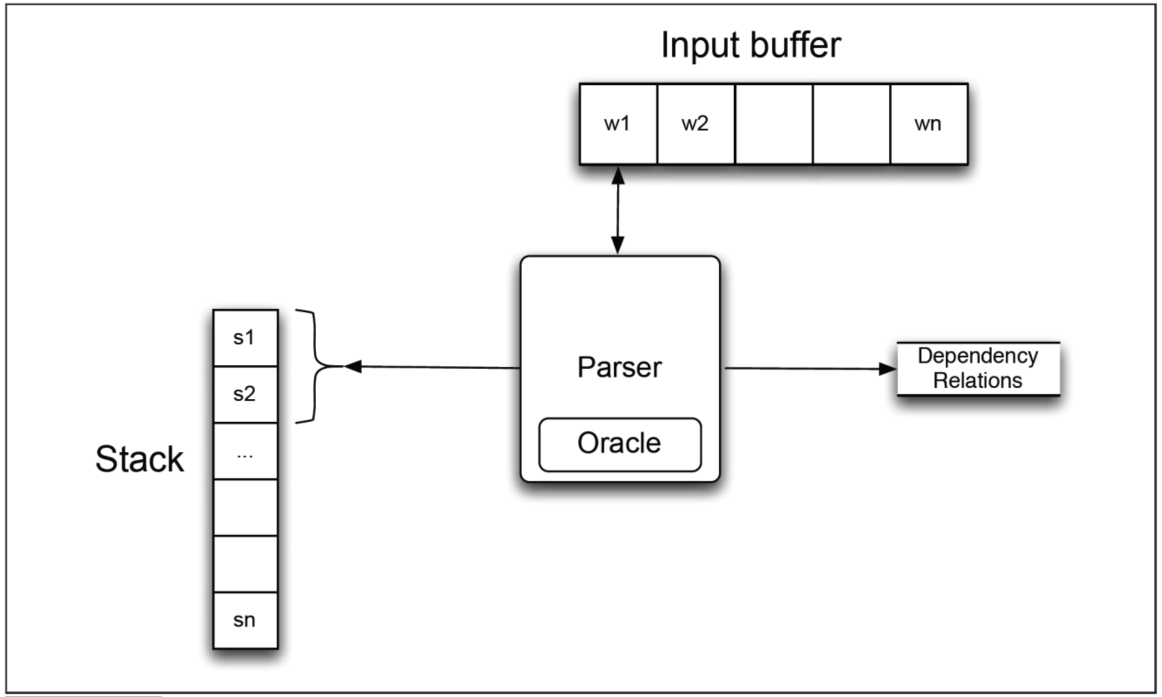


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

SHIFT

- Remove word at head of input buffer
- Push it on the stack

LEFT-ARC

- create head-dependent relation between word at top of stack and 2nd word (under top)
- remove 2nd word from stack

RIGHT-ARC

- Create head-dependent relation between word on 2nd word on stack and word on top
- Remove word at top of stack

Arc standard transition systems

- Preconditions
  - ROOT cannot have incoming arcs
  - LEFT-ARC cannot be applied when ROOT is the 2nd element in stack
  - LEFT-ARC and RIGHT-ARC require 2 elements in stack to be applied

Transition-based Dependency Parser

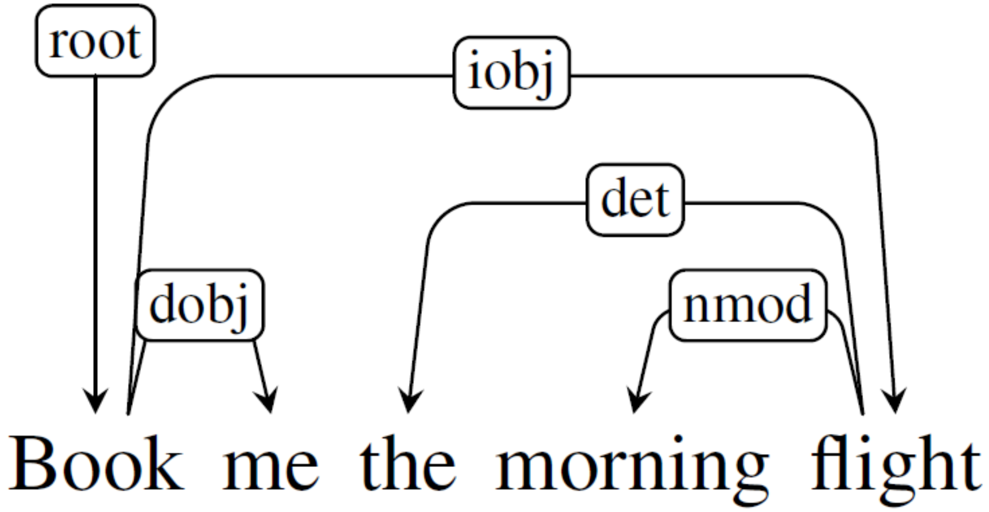
```
function DEPENDENCYPARSE(words) returns dependency tree

  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
```

Figure 14.6 A generic transition-based dependency parser

Properties of this algorithm:

- Linear in sentence length
- A greedy algorithm
- Output quality depends on oracle

Exercise: find a sequence of transitions to generate this parse



SHIFT

- Remove word at head of input buffer
- Push it on the stack

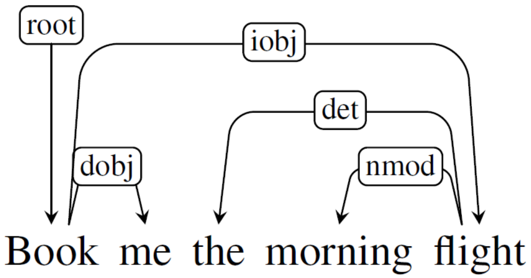
LEFT-ARC

- create head-dependent relation between word at top of stack and 2nd word (under top)
- remove 2nd word from stack

RIGHT-ARC

- Create head-dependent relation between word on 2nd word on stack and word on top
- Remove word at top of stack

Transition-Based Parsing Illustrated



Step	Stack	Word List	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]	RIGHTARC	(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]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

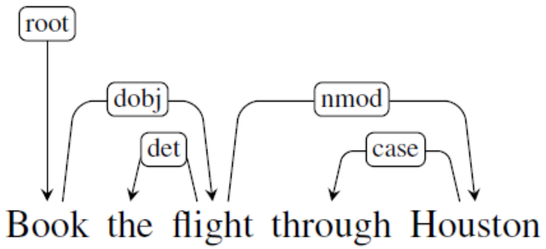
Figure 14.7 Trace of a transition-based parse.

Where do we get an oracle?

- Multiclass classification problem
  - Input: current parsing state (e.g., current and previous configurations)
  - Output: one transition among all possible transitions
  - Q: size of output space?
- Supervised classifiers can be used
  - E.g., perceptron
- Open questions
  - What are good features for this task?
  - Where do we get training examples?

Generating Training Examples

- What we have in a treebank



- What we need to train an oracle
  - Pairs of configurations and predicted parsing action

Step	Stack	Word List	Predicted 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]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 14.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

Generating training examples

- Approach: simulate parsing to generate reference tree
- Given
  - A current config with stack S, dependency relations Rc
  - A reference parse (V,Rp)
- Do

LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$

RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t. (S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$

SHIFT: **otherwise**

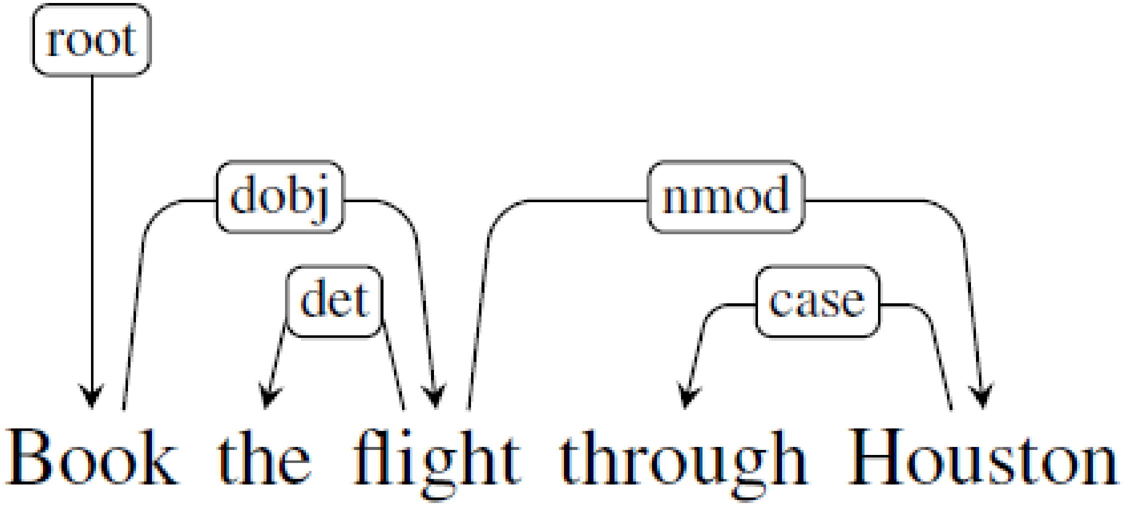
Additional condition on RightArc makes sure a word is not removed from stack before its been attached to all its dependent

Let's try it out

LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$

RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t.(S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$

SHIFT: **otherwise**



Features

- Configuration consist of stack, buffer, current set of relations
- Typical features
  - Features focus on top level of stack
  - Use word forms, POS, and their location in stack and buffer

Features example

- Given configuration

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United)
		(flights → morning)
		(flights → the)

- Example of useful features

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

$$\langle s_1t.s_2t = NNSVBD, op = shift \rangle$$

Features example

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Figure 14.9 Standard feature templates for training transition-based dependency parsers. In the template specifications s_n refers to a location on the stack, b_n refers to a location in the word buffer, w refers to the wordform of the input, and t refers to the part of speech of the input.