Dependency Parsing

CMSC 723 / LING 723 / INST 725

Marine Carpuat

Fig credits: Joakim Nivre, Dan Jurafsky & James Martin

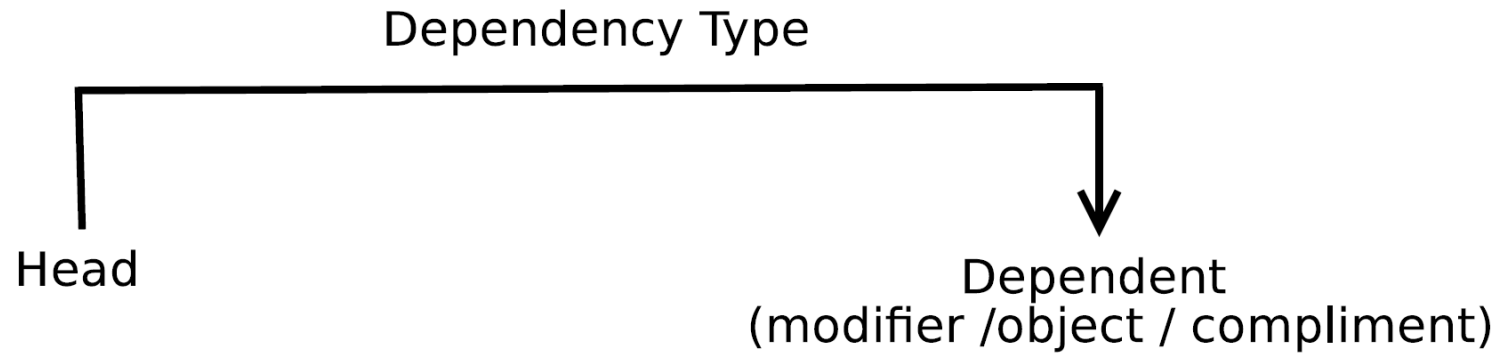
Dependency Parsing

- Formalizing dependency trees

- Transition-based dependency parsing
  - Shift-reduce parsing
  - Transition system
  - Oracle
  - Learning/predicting parsing actions

Dependency Grammars

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



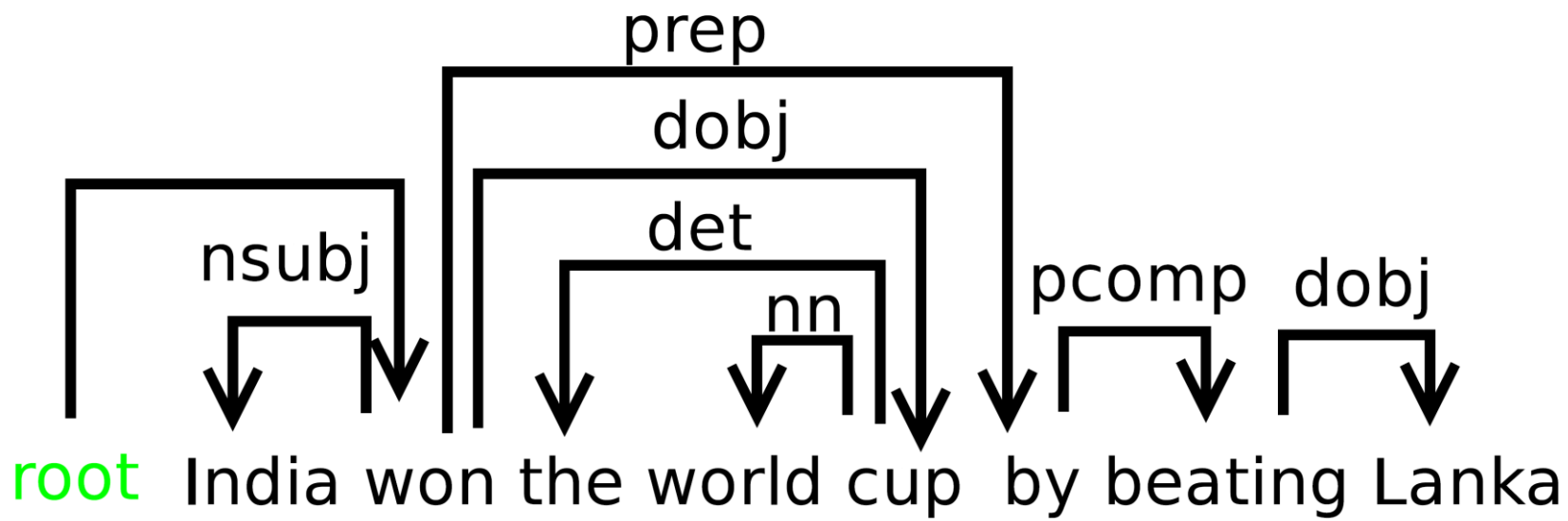
Dependency Relations

Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	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*.

Figure 14.3 Examples of core Universal Dependency relations.

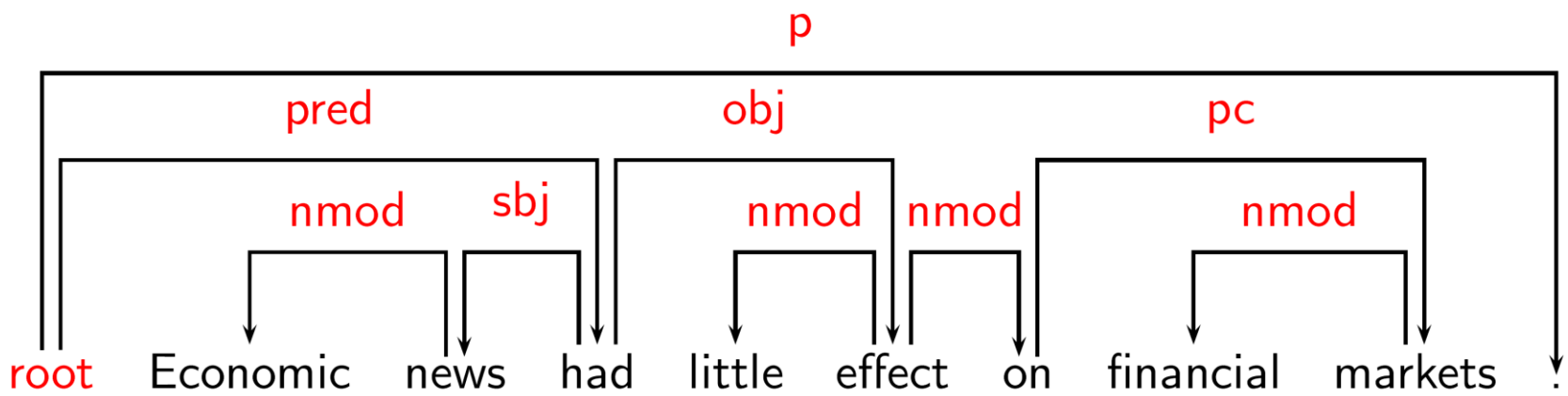
Example Dependency Parse

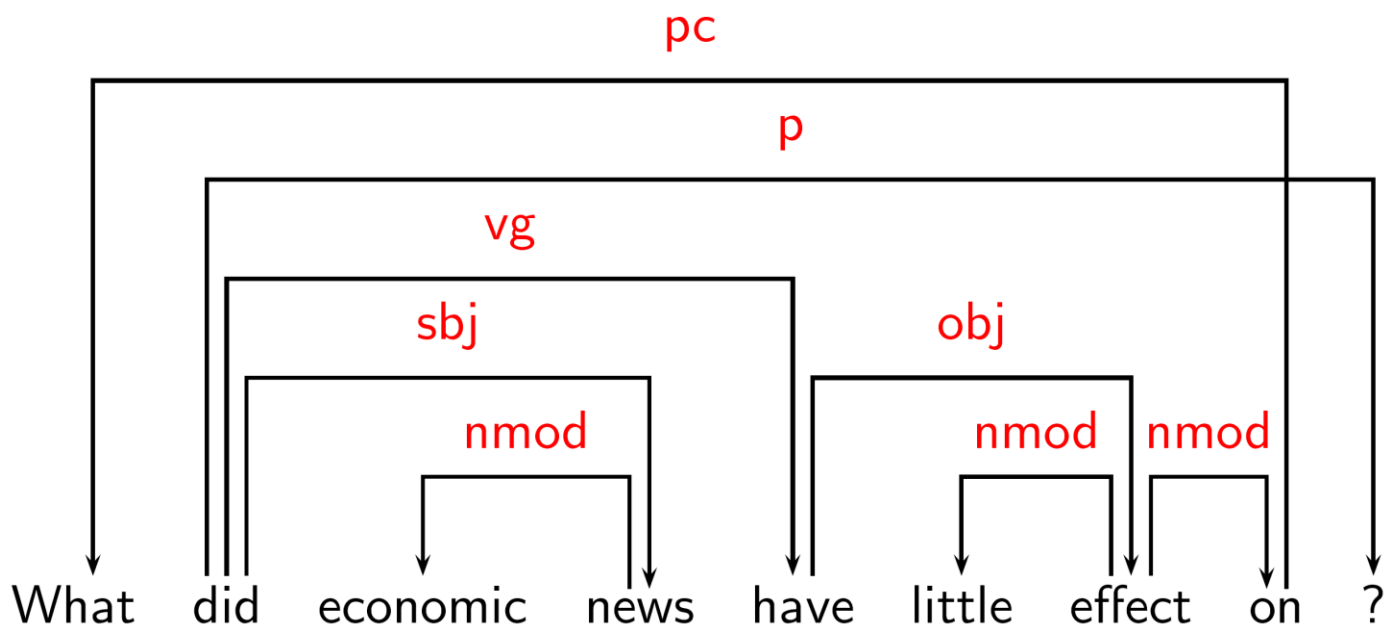


Dependency formalisms

- Most general form: a graph G = (V,A)
  - V vertices: usually one per word in sentence
  - A arcs (set of ordered pairs of vertices): head-dependent relations between elements in V
- Restricting to **trees** provide computational advantages
  - Single designated ROOT node that has no incoming arcs
  - Except for ROOT, each vertex has exactly one incoming arc
  - Unique path from ROOT to each vertex in V

- Each word has a single head
- Dependency structure is connected
- There is a single root node from which there is a unique path to each word


p
pred
obj
pc
nmod
sbj
nmod
nmod
nmod
root
Economic
news
had
little
effect
on
financial
markets
!


pc
p
vg
sbj
obj
nmod
nmod
nmod
What
did
economic
news
have
little
effect
on
?

Projectivity

- **Arc** from head to dependent is **projective**
  - If there is a path from head to every word between head and dependent
- **Dependency tree** is **projective**
  - If all arcs are projective
  - Or equivalently, if it can be drawn with no crossing edges
- Projective trees make computation easier
- But most theoretical frameworks do not assume projectivity
  - Need to capture long-distance dependencies, free word order

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

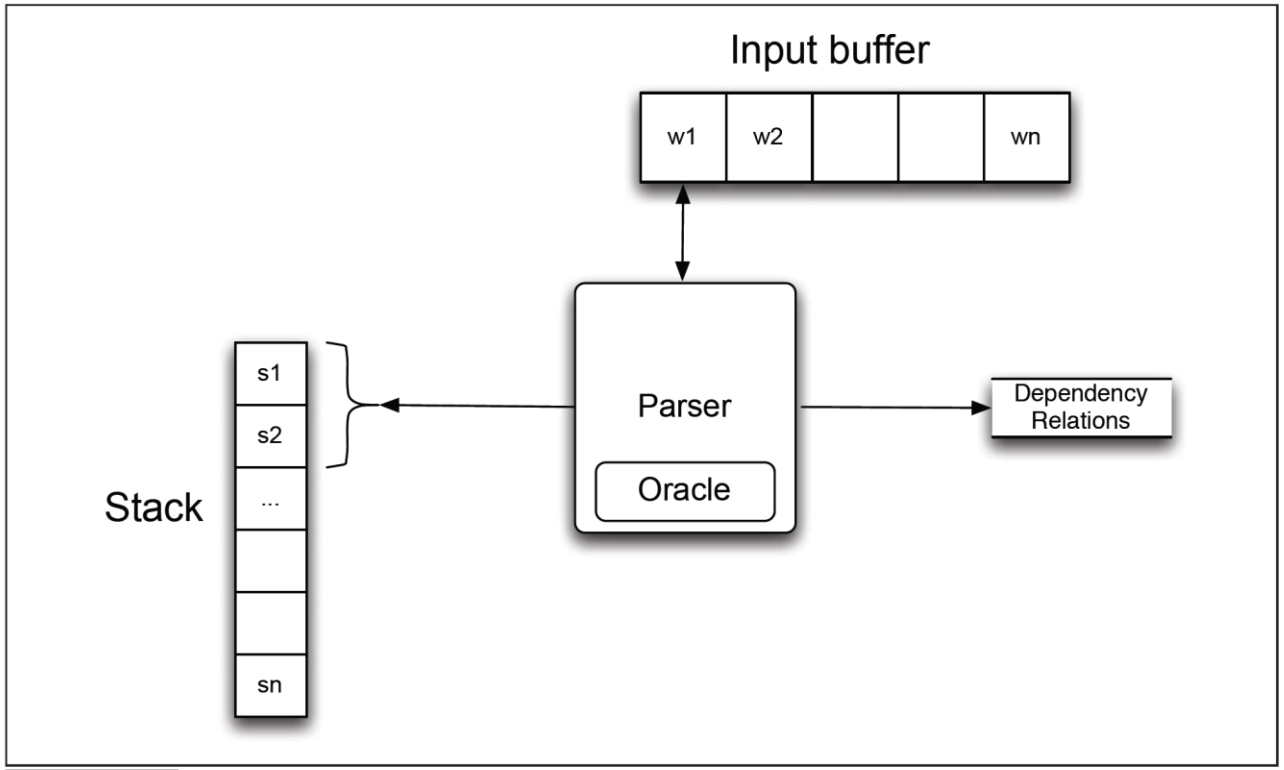


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

- Builds on shift-reduce parsing
  [Aho & Ullman, 1927]
- **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

Transition operators

- Transitions: 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]
- LEFT-ARC:
  - create head-dependent rel. between word at top of stack and 2nd word (under top)
  - remove 2nd word from stack
- RIGHT-ARC:
  - Create head-dependent rel. between word on 2nd word on stack and word on top
  - Remove word at top of stack
- SHIFT
  - Remove word at head of input buffer
  - Push it on the 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

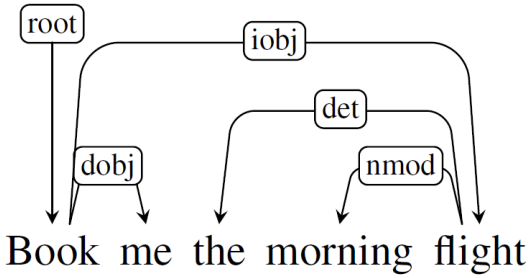
```
function DEPENDENCYPARSE(words) returns dependency tree

  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

- Assume an oracle
- Parsing complexity
  - Linear in sentence length!
- Greedy algorithm
  - Unlike Viterbi for POS tagging

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.

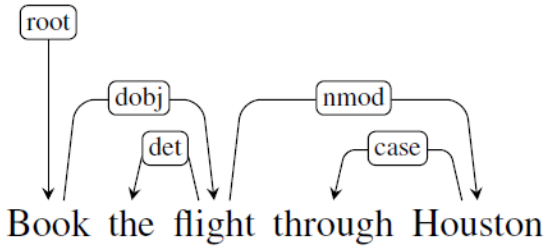
Where to we get an oracle?

- 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

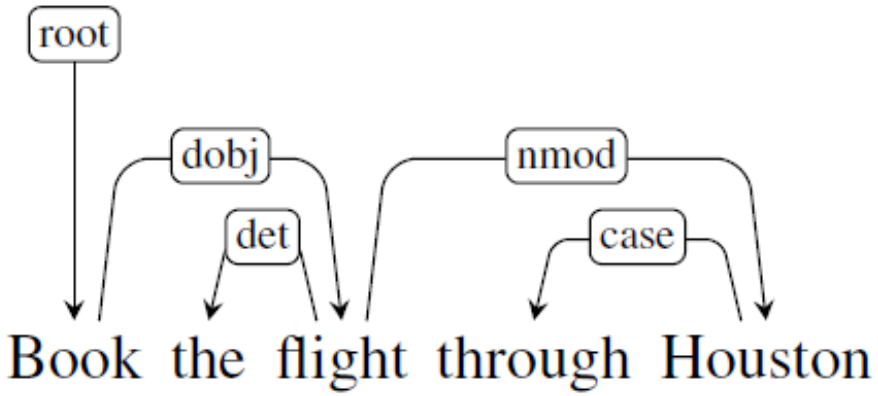
LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$
RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t.(S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$
SHIFT: **otherwise**

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

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

$$\langle s_1 t.s_2 t = NNSVBD, op = shift \rangle$$

Dependency Parsing

- Formalizing dependency trees

- Transition-based dependency parsing
  - Shift-reduce parsing
  - Transition system
  - Oracle
  - Learning/predicting parsing actions