Dependency Parssing

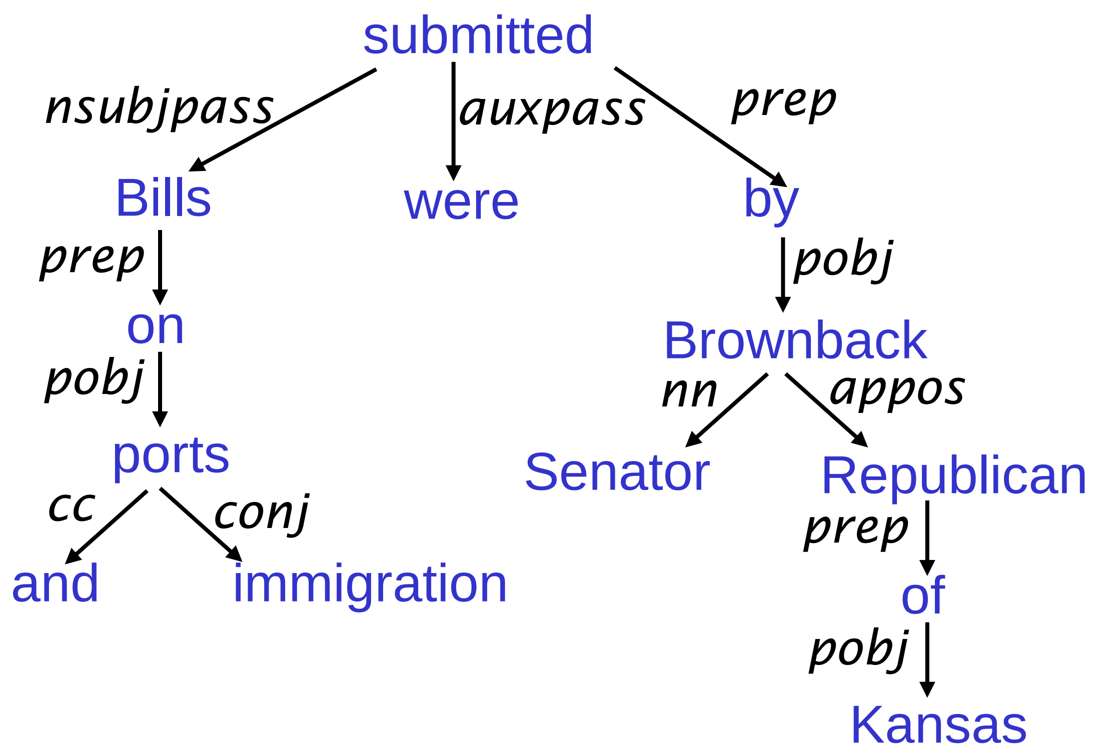
Sudeshna Sarkar

16 AUG 2019

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations

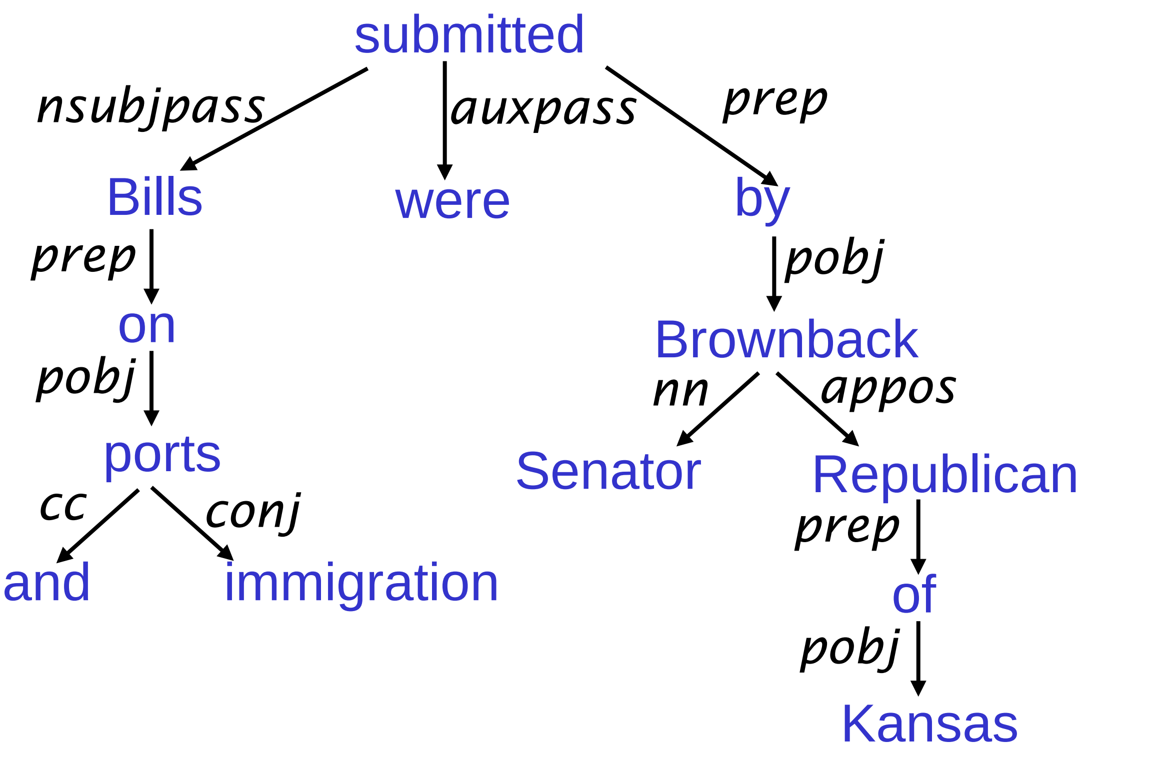


Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



Dependency Structure

1. *Look for the large barking dog by the door in a crate*
2. *Scientists study whales from space*

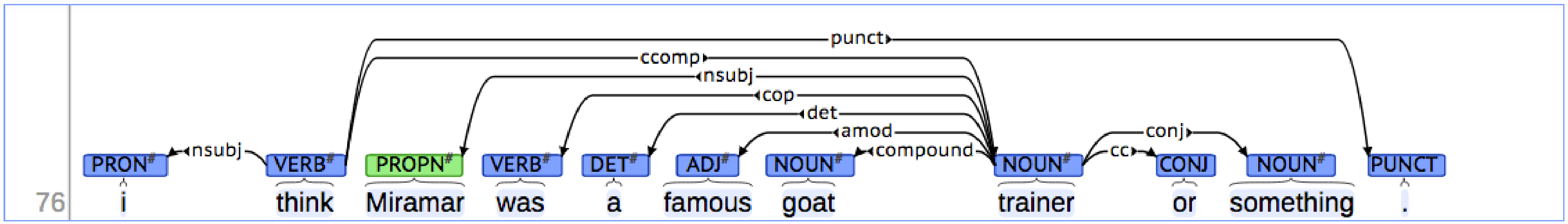
Dependency Relations
(A sample from UDEP)

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

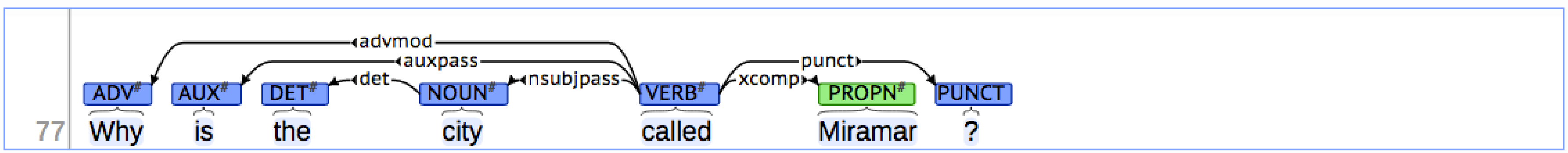
Universal Dependencies treebanks

[Universal Dependencies: http://universaldependencies.org/

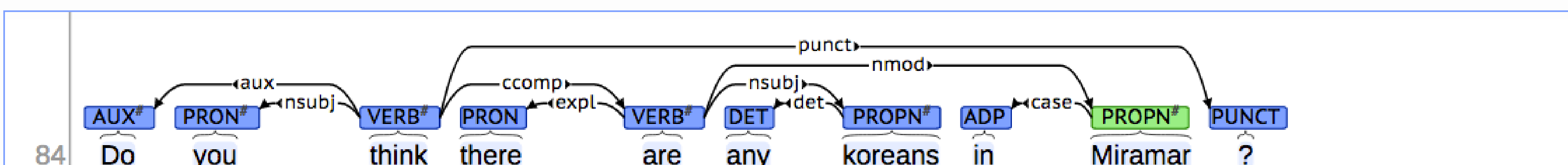
[context] [conllu]



[context] [conllu]



[context] [conllu]



Pāṇini’s grammar
(c. 5th century BCE)

Dependency Grammar/Parsing History

- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
  - Basic approach of 1st millennium Arabic grammarians
- Constituency/context-free grammars is a more recent invention
  - 20th century (R.S. Wells, 1947)
- Modern dependency work often linked to work of L. Tesnière (1959)
  - Was dominant approach in "East" (Russia, China, …)
    - Good for free-er word order languages

Dependency Parsing

- Emphasis on dependency and grammatical relations
    - Subject, direct object, case, etc.
- Significant computational advantages
    - Linear vs. O(N^5) for probabilistic CFGs

Dependency Conditioning Preferences

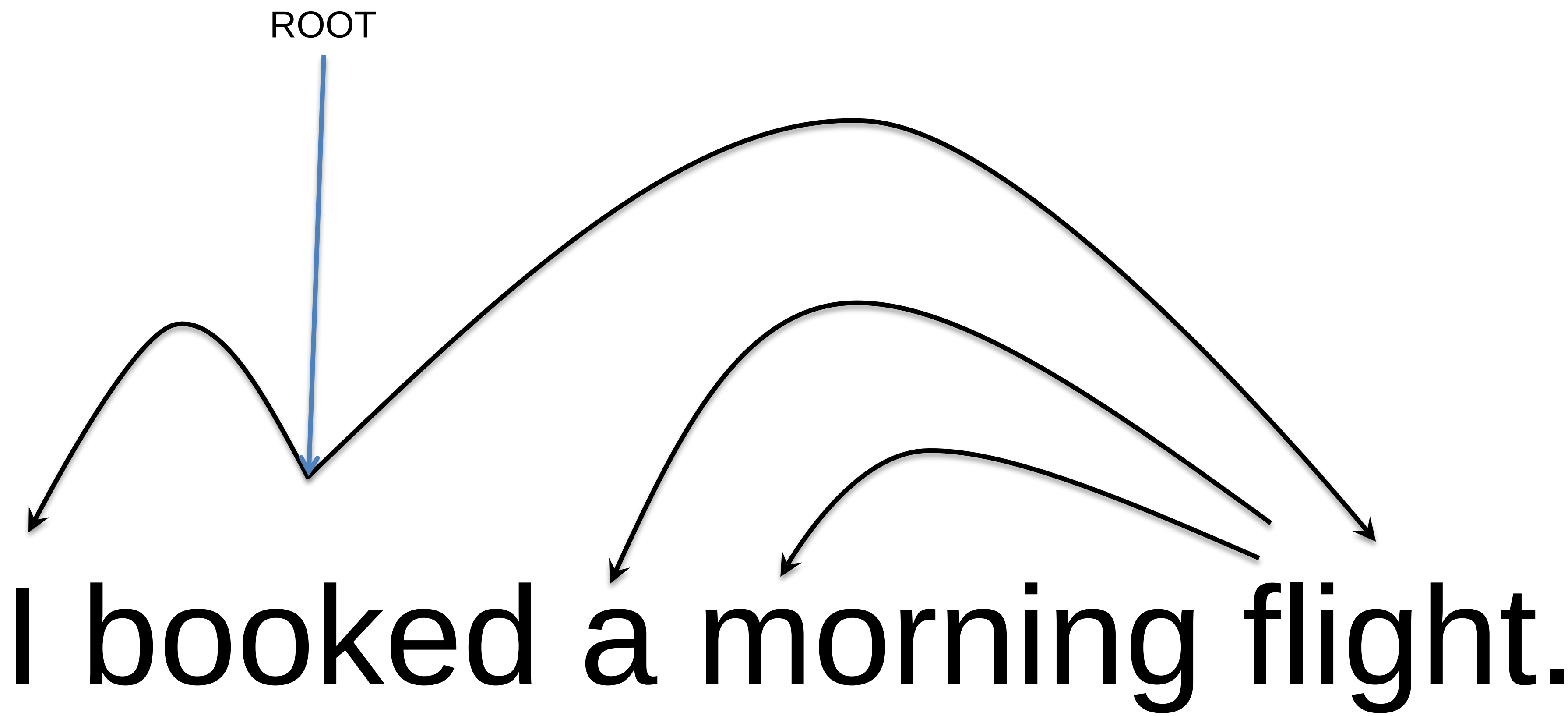
What are the sources of information for dependency parsing?

1. Bilexical affinities [issues → the] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material

   Dependencies rarely span intervening verbs or punctuation
4. Valency of heads

   How many dependents on which side are usual for a head?

Dependency Parse

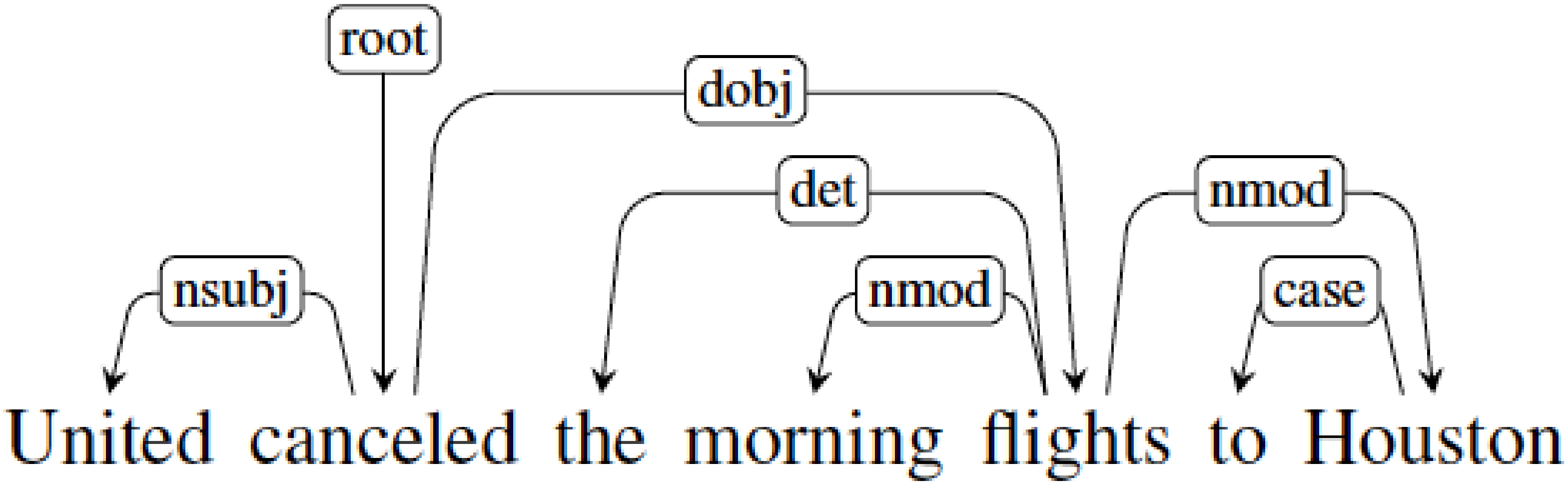


(booked, I) (booked, flight) (flight, a) (flight, morning)

Dependency Grammar

- The linguistic constraints underlying “correct trees” are usually called a dependency grammar
  - Which may or may not correspond to an explicit formal generative grammar of the kind we’ve been using

Dependency Relations



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.

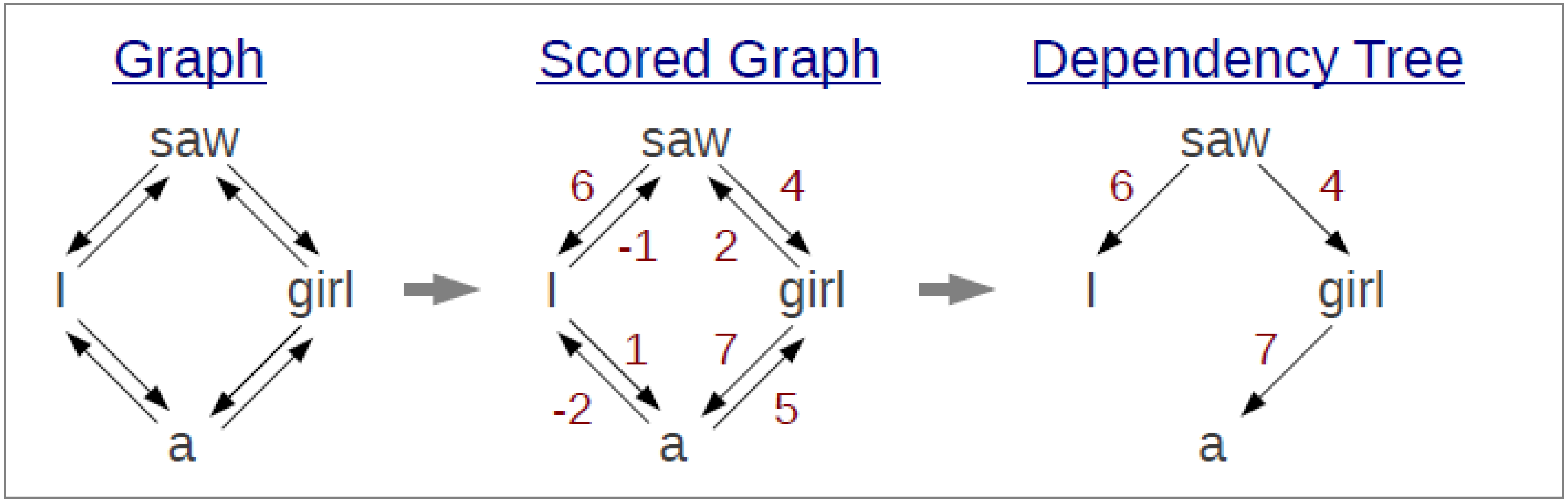
Dependency Parsing

- Given an input sentence, draw edges between pairs of words, and label them.
  - Result should be a tree.
  - The edge-labels between word pairs should convey the correct syntactic relation.

1. Could construct a tree one edge at a time.
   - Transition parsing.
2. Could construct a fully connected tree, and prune it.
   - Graph-based methods.

Maximum Spanning Tree

- Each dependency is an edge in a directed graph
- Assign each edge a score (with machine learning)
- Keep the tree with the highest score

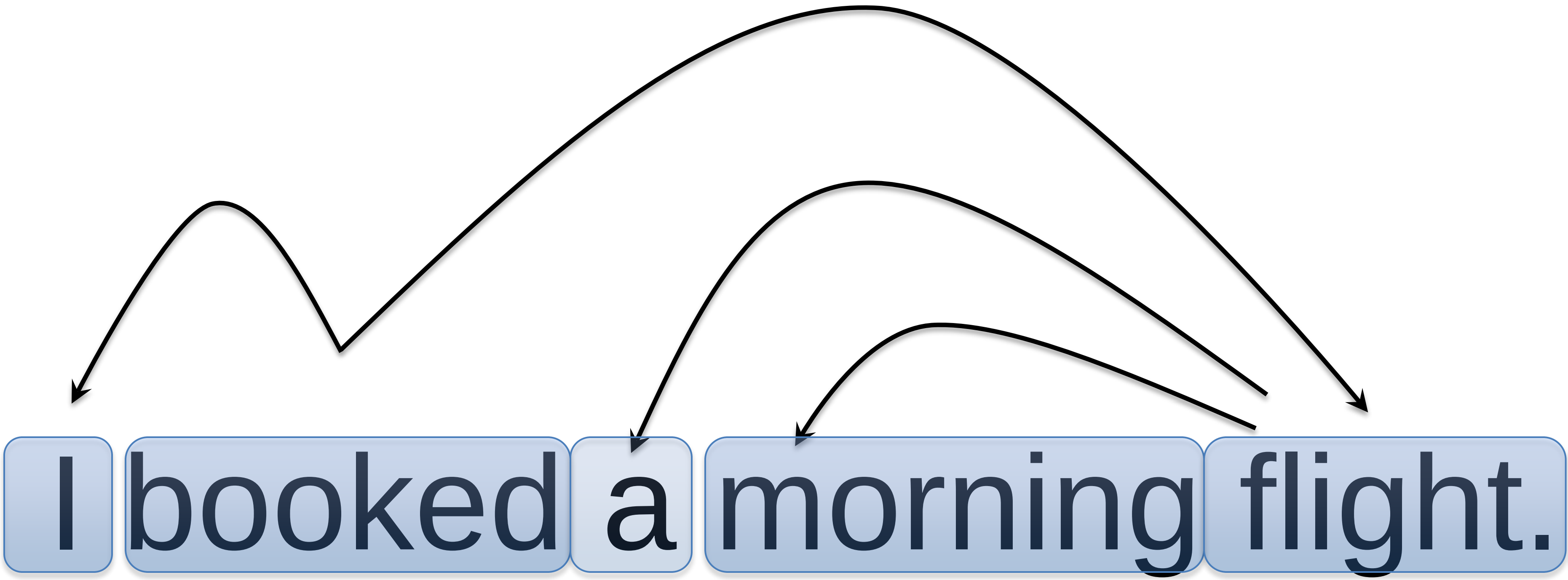


(Chu-Liu-Edmonds Algorithm)

Transition-Based Parsing

- Transition-based parsing is a greedy word-by-word approach to parsing
  - A single dependency tree is built up an arc at a time as we move left to right through a sentence
  - No backtracking
  - ML-based classifiers are used to make decisions as we move through the sentence

Dependency Parse

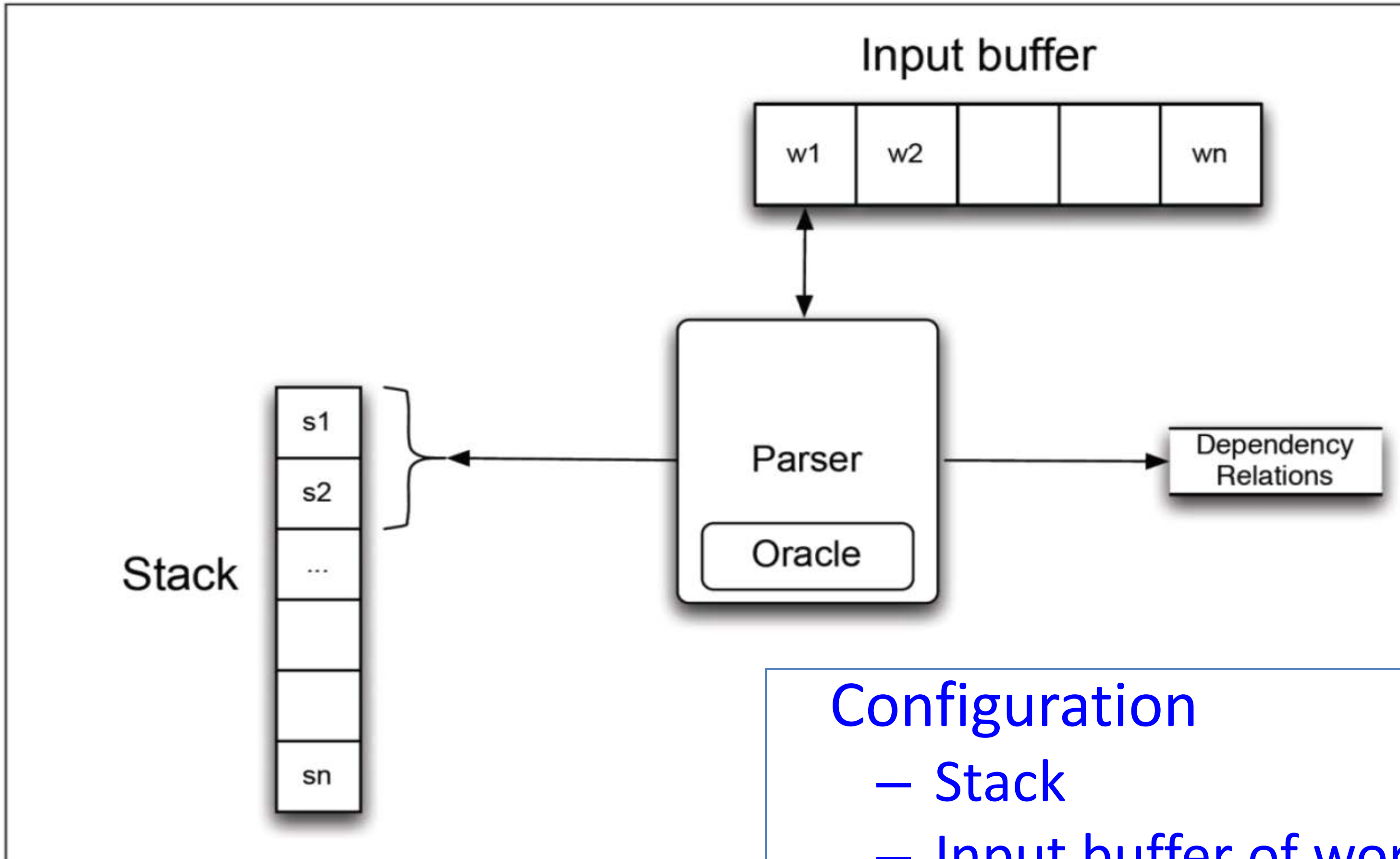


Transition-Based Parsing

- A state consists of three elements
  - A stack representing partially processed words
  - A buffer/list containing the remaining words to be processed
  - A set/arcs containing the relations discovered so far

- Makes arc decisions about entries in the top of the stack and buffer.
- Keeps shifting words from the buffer until all words are consumed.

Transition-based dependency parsing



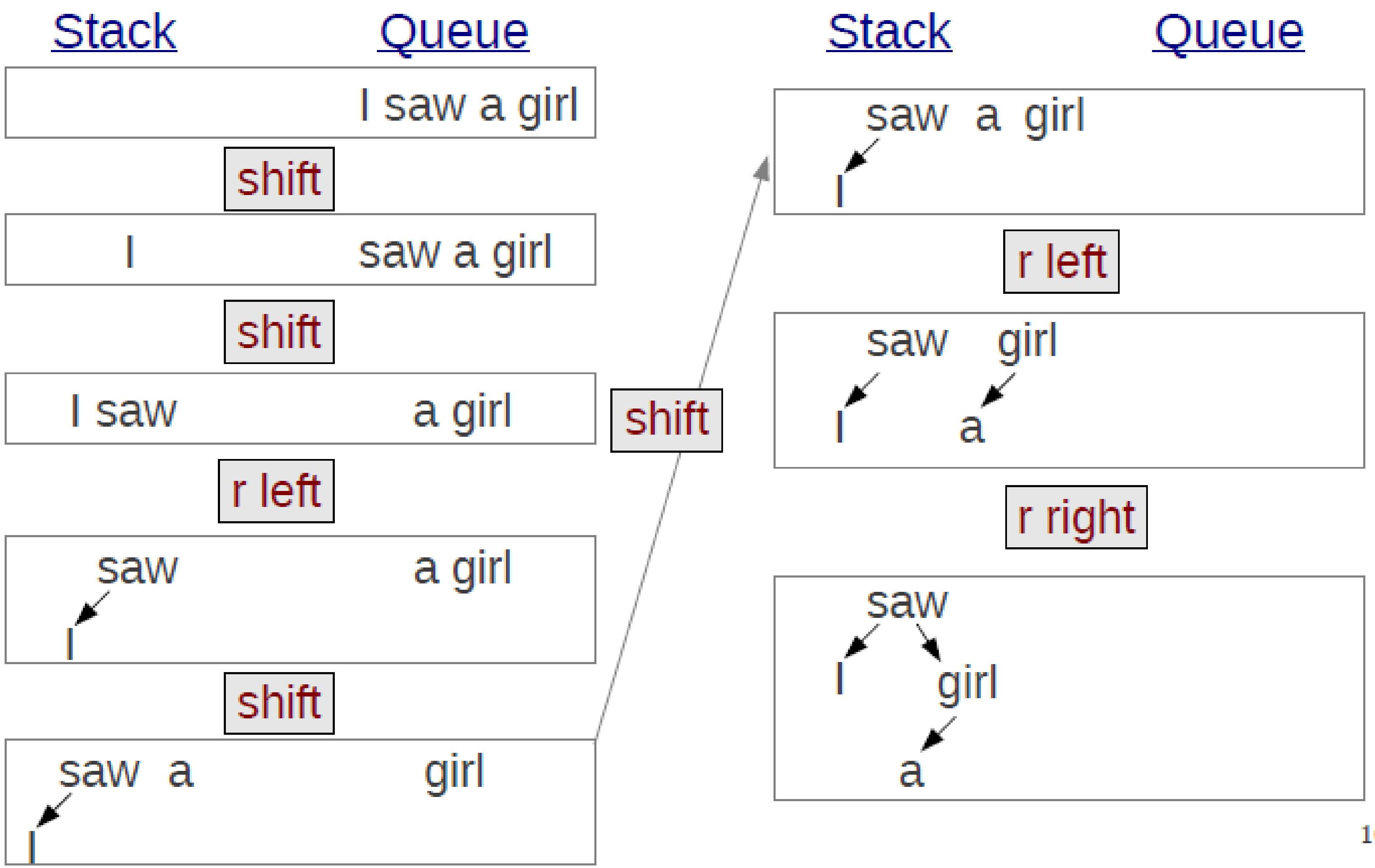
Configuration
- Stack
- Input buffer of words
- Set of dependency relations

Arc Standard Transition System

Defines 3 transition operators

- LEFT-ARC
  - create head-dependent rel. between word at top of stack and 2ndword (under top)
  - remove 2ndword from stack
- RIGHT-ARC:
  - Create head-dependent rel. between word on 2ndword 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

Shift Reduce Example



Example Transition Sequence

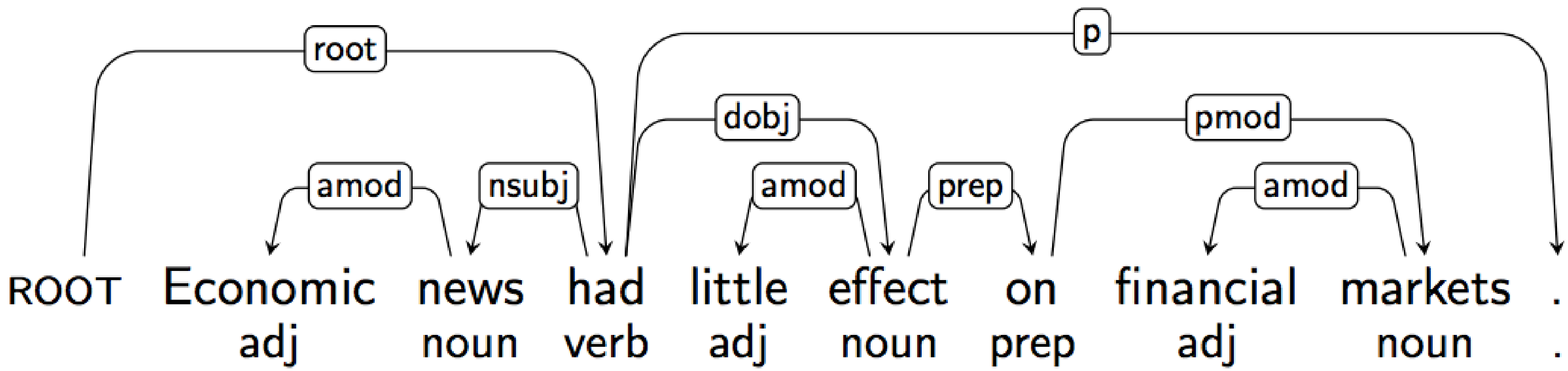
$[\text{ROOT}]_S \quad [\text{Economic, news, had, little, effect, on, financial, markets, .}]_B$

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	adj	noun	verb	adj	noun	prep	adj	noun	.

Assume we have some black-box that takes two words and magically gives you the dependency relation between them if one exists.

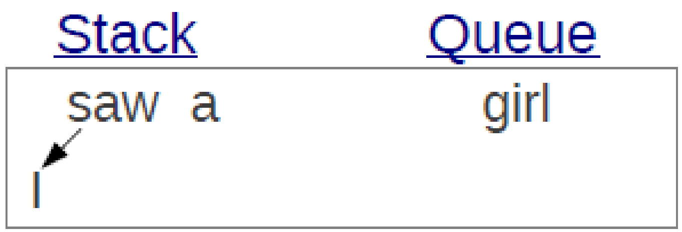
And on it goes until …

$[\text{ROOT, had, .}]_S \quad [\ ]_B$



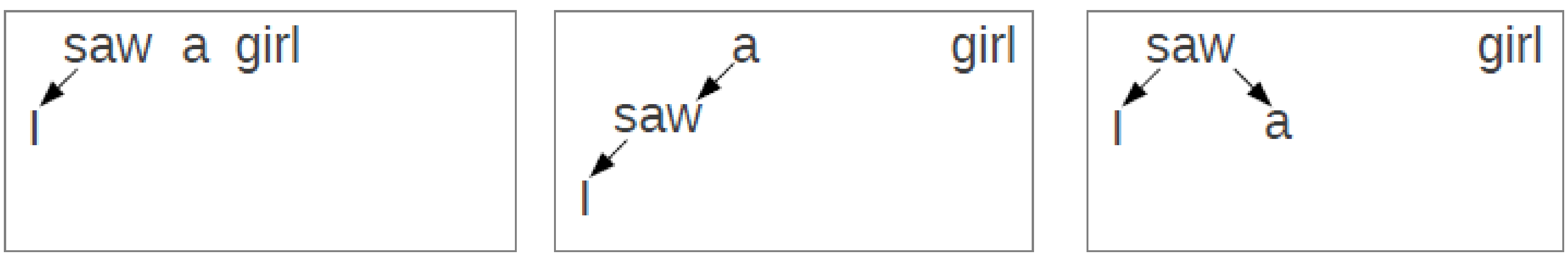
Classification for Shift-Reduce

- Given a state:



- Which action do we choose?

shift ? r left ? r right ?



- Correct actions → correct tree

Classification for Shift-Reduce

- We have a weight vector for "shift" "reduce left" "reduce right"

$$w_s \ \ w_l \ \ w_r$$

- Calculate feature functions from the queue and stack

$$\phi(\text{queue, stack})$$

- Multiply the feature functions to get scores

$$s_s = w_s * \phi(\text{queue,stack})$$

- Take the highest score

$$s_s > s_l \&\& s_s > s_l > \text{sl} \ \rightarrow \text{do shift}$$

As a supervised classification task.

- Given the current state (i.e., stack, buffer and A) predict the next action.
- Can be viewed as a supervised learning problem.
  - Four way classification (if un-typed dependencies)
  - m-way classification, where m = 2 x number of types + 2
- Features
  - Compute features of the current configuration of the stack, buffer and A.
  - Word in stack, POS of word, Word in buffer and POS of Word in buffer.
  - Other features: Length of dependency arc
- Greedy classifier (no search involved)
  - At each stage ask the classifier to predict the next transition.
  - Select the best legal transition and apply it.
  - Works quite well, close to PCFG.
- Quite fast!
  - O(N) in length of sentence.

Three Problems

- To apply ML in situations like this we have three problems
  1. Discovering features that are useful indicators of what to do in any situation
     - Characteristics of the state we're in
  2. Acquiring the necessary training data
     - Treebanks
  3. Training a classifier

Features

1. Features are typically described along two dimensions in this style of parsing
   1. Position in the state (aka configuration)
      1. Position in the stack, position in the word list, location in the partial tree
   2. Attributes of particular locations or attributes of tuples of locations
      1. Part of speech of the top of the stack, POS of the third word in the remainder list, lemmas, last three letters
      2. Head word of a word, number of relations already attached to a word, does the word already have a SUBJ relation, etc.

Training

- Supervised ML methods require training material in the form of (input, output) pairs.
  - Our treebanks associate sentences with their corresponding trees
  - We need parser states paired with their corresponding correct operators
  - But we do know the correct trees for each sentence

Training

- We'll parse with our standard algorithm, asking an oracle which operator to use at any given time.
- The oracle has access to the correct tree for this sentence. At each stage it chooses using a case statement
  1. Left if the relation to be added is in the reference tree.
  2. Right if the resulting relation is in the correct tree AND if all the other outgoing relations associated with the word are already in the relation list.
  3. Otherwise shift

Evaluation

- If we have a test set from a treebank and if we represent parses as a list of relations

(booked, I) (booked, flight) (flight, a) (flight, morning)

- Unlabeled attachment score (UAS) is just what fraction of words were assigned the right head.
- Labeled attachment score (LAS) is what fraction of words were assigned the right head with the right relation.

States

- The start state looks like this

  [[root], [word list], ()]

- A valid final state looks like
  - [[root], [] (R)]
  - Where R is the set of relations that we've discovered. The [] represents the fact that all the words in the sentence are accounted for

Two Issues

1. We really want labeled relations
   – That is, we want things like subject, direct object, indirect object, etc. as relations
2. How did we know which operator (L, R, S) to invoke at each step along the way?
   – Since we're not backtracking, one wrong step and we won't get the tree we want
      • How do we even know what tree we want?

Grammatical Relations

- Well, to handle this we can just add new transitions
  - Essentially replace Left and Right with

    {Left, Right} X {all the relations of interest}
  - Note this isn't going to make problem 2 any easier to deal with
- Standard approaches have roughly 40 kinds of dependency relations

Example

- Start
  - [[root], [I booked a morning flight], ()]
- End
  - [[root],

    [],

    ((booked, I) (booked, flight) (flight, a) (flight, morning))]