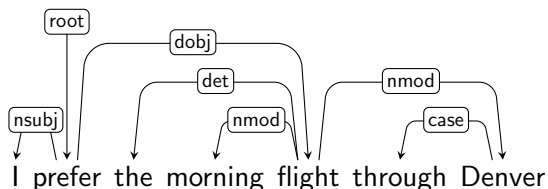


# Dependency Grammars: Avoiding Constituents

- ▶ Traditional way of thinking
  - ▶ Goes back to Panini (Pāṇini circa 350BC)
  - ▶ Modern form: Lucien Tesnière, 1950s
- ▶ *Typed dependency structure*: Captures grammatical relations directly between words



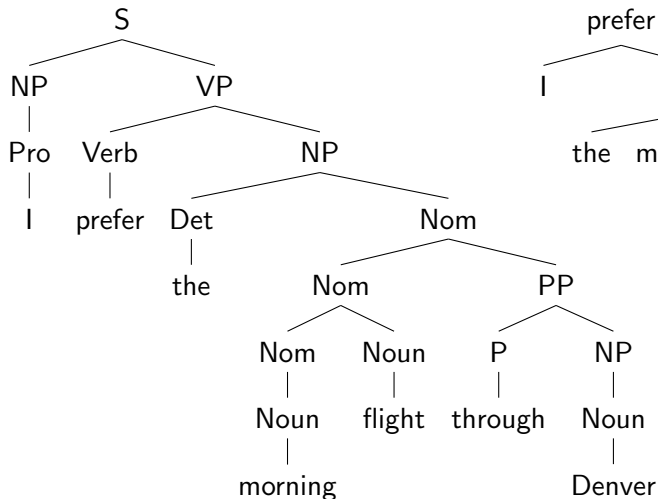
- ▶ Well-suited for languages that have free word order

# Free Word Order Languages

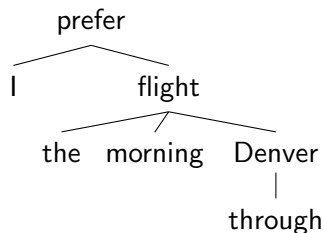
- ▶ Convey information about types through richer morphemes
- ▶ CFGs focus on structure and word order
  - ▶ Lead to large grammars to handle allowed orders
  - ▶ Produce large structures
  - ▶ Relationships between words that relevant for understanding the meaning can be several edges away in a parse tree
- ▶ Dependency representations
  - ▶ Can express the elements of the structure essential for meaning
  - ▶ Bring forth the head word for each phrase and the relations in reference to the head word

# Constituency versus (Untyped) Dependency Parses

Constituency parse:



Untyped dependency parse:



# Constituency versus (Untyped) Dependency Parses

What are some tradeoffs?

# Constituency versus (Untyped) Dependency Parses

What are some tradeoffs?

- ▶ Constituency parses
  - ▶ Preserve word order
  - ▶ More information on structure
- ▶ Dependency parses
  - ▶ Lose word order
  - ▶ More *functional*: parent “applies” on children

## Case and Thematic Roles

- ▶ Case (more syntactic): A grammatical relation with respect to a verb
- ▶ Thematic Role (more semantic): An “argument” assigned by a verb
- ▶ Essential to understanding the meaning of a sentence
- ▶ Panini’s *karaka*
- ▶ Latin has cases indicated by *declensions*
- ▶ Fillmore’s case grammar ~ 1960s
- ▶ Example thematic roles
  - ▶ Agent: intentional doer
  - ▶ Experiencer: one who undergoes a (change in a) state of being
  - ▶ Theme or Patient: receiver of an action
  - ▶ Instrument
  - ▶ Goal or Telos: where the action takes us
  - ▶ Location: where the action occurs
  - ▶ Source: from where
  - ▶ Benefactive: from whom
  - ▶ Cause or point of departure

## Excerpted from Churchill's Memoir

Churchill was told to memorize this table about tables (first two columns)

Mensa	a table	Nominative	The table is solid
Mensa	O table	Vocative	Fold up, table!
Mensam	a table	Accusative	Scratched the table
Mensae	of a table	Genitive	The top of the table
Mensae	to or for a table	Dative	Give the table a wash
Mensa	by, with, or from a table	Ablative	Fell off the table

“Mensa, O table, is the vocative case,” he replied

*“But why O table?” I persisted in genuine curiosity*

“O table – you would use that in addressing a table, in invoking a table”

And then seeing he was not carrying me with him,

“You would use it in speaking to a table”

*“But I never do,” I blurted out in honest amazement*

“If you are impertinent, you will be punished, and punished, let me tell you, very severely,” was his conclusive rejoinder

When would someone address a table?

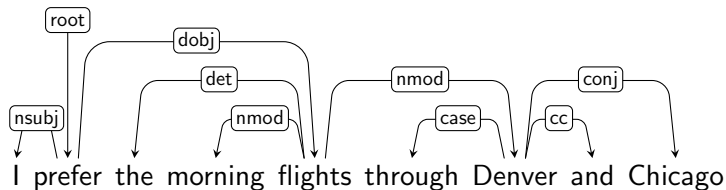
# Universal Dependencies Project

Joakim Nivre and others

- ▶ Identify relations that are
  - ▶ Linguistically justified
  - ▶ Occur in multiple languages
  - ▶ Potentially usable for NLP
- ▶ Clausal relations
  - ▶ Capture syntactic roles with respect to a verb
- ▶ Modifier relations
  - ▶ How a word modifies its head
- ▶ Coordinating conjunctions
  - ▶ An arbitrary or corpus-specific choice as to head and dependent
  - ▶ An EMT and a police officer revived the victim: EMT or officer as head?



# Exercise: Clausal, Modifier, or Coordinating Relation



# Head versus Dependent

Kübler, McDonald, and Nivre 2009

Criteria for identifying a head H and a dependent D in a linguistic “construction” (e.g., constituent) C

- ▶ H determines the syntactic category of C and can often replace C
  - ▶ This would be an *endocentric* construction
- ▶ H determines the semantic category of C; D gives semantic specification
- ▶ H is mandatory; D may be optional
- ▶ H selects D and determines whether D is mandatory or optional
  - ▶ Optional (here, an adjective): Dan likes sugared water
  - ▶ Mandatory (here, a determiner): Ayaan ate a/the/one/Ian's pear
- ▶ The form of D depends on H (agreement or government)
  - ▶ He (\*him) helped Maya versus Suma helped him (\*he)
  - ▶ Where are the bananas (\*banana)
- ▶ The linear position of D is specified in relation to H (before in English)

# Endocentric versus Exocentric

- ▶ Endocentric
  - ▶ Support substitution of an entire construct by its head
  - ▶ Typically, head-modifier relations
  - ▶ Adjective, adverb, nominal modifier, ...
- ▶ Exocentric
  - ▶ Do not support substitution of an entire construct by its head
  - ▶ Typically, head-complement relations
  - ▶ Subject, object, copula, ...
    - ▶ NB: Copula is a linking word rooted in *be*

The marker is green

# Determining Head-Dependent Relations can be Tricky

Joakim Nivre's example

I can see that they rely on this and that .

- ▶ Complex verb groups
  - ▶ Auxiliary and main verb “can see”
- ▶ Subordinate clauses
  - ▶ Complementizer and verb “see that ... ”
- ▶ Coordination
  - ▶ Coordination and conjuncts “this and that”
- ▶ Prepositional phrases
  - ▶ Preposition and nominal “on (this and that)”
- ▶ Punctuation
  - ▶ Link to the verb “can see . . . .”

# Important Dependency Relations (Head to Dependent)

De Marneffe, Dozat, Silveira, Haverinen, Ginter, Nivre, Manning

## Functional categories used as edge labels

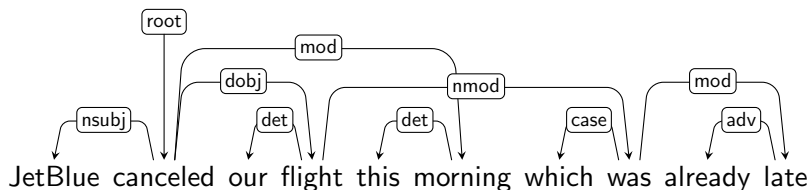
<i>Clausal Argument</i>	<i>Description</i>	<i>Example</i>
NSUBJ	Nominal subject	<u>Ian</u> ate a cake
DOBJ	Direct object $\approx$ accusative	Bhavana gave Amitha a <u>cake</u>
IOBJ	Indirect object $\approx$ dative	Bhavana gave <u>Amitha</u> a cake
CCOMP	Clausal complement	I know the cake <u>contains</u> sugar
XCOMP	Open clausal complement	Arvind learned to <u>bake</u> a cake
<i>Nominal Modifier</i>	<i>Description</i>	
NMOD	Nominal modifier	<u>cake</u> platter
AMOD	Adjectival modifier	<u>fluffy</u> cake
NUMMOD	Numeric modifier	<u>three</u> main ingredients
APPOS	Appositional modifier	Sam, the <u>baker</u> , brought cake
DET	Determiner	<u>Kyle's</u> cake
CASE	Prepositions, postpositions, and other case markers	The icing <u>on</u> the cake
<i>Other</i>	<i>Description</i>	
CONJ	Conjunct	Luke likes cake and <u>syrup</u>
CC	Coordinating conjunction	Luke likes cake <u>and</u> syrup

# Formal Properties of Dependencies

- ▶ A dependency graph is a tree
  - ▶ Single designated ROOT
  - ▶ Each vertex except the root depends on exactly one vertex
  - ▶ Thus, a unique path from ROOT to each vertex
- ▶ Projectivity
  - ▶ Dependencies don't cross with respect to word order
  - ▶ Any vertices that lie between a head and dependent pair descend from that head
- ▶ Dependency trees generated from CFGs are projective
- ▶ Projectivity is not suitable for free word order languages

## Example Violating Projectivity

Projectivity is often too restrictive an assumption

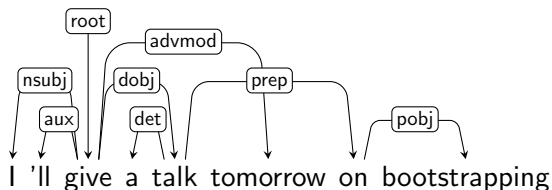


Projectivity fails for free word order languages

## Example Violating Projectivity

Projectivity is often too restrictive an assumption

- ▶ This is Manning's example with dependency types added
- ▶ Notice that, unlike modern approaches, it
  - ▶ Uses older dependency relations: `PREP` versus `CASE`
  - ▶ Treats *on* as the head of *on bootstrapping*





# Dependency Treebanks

Set of sentences along with a reference dependency tree for each

- ▶ Create from scratch by hand
  - ▶ Annotation guidelines in the Universal Dependencies project, for example
- ▶ Convert constituency parses to dependency structures
- ▶ For any constituent
  - ▶ Identify its head child and nonhead children
  - ▶ Make the head of each nonhead child depend on the head of the constituent's head child
- ▶ Information the original trees lack is omitted from the dependency structure either

# Example: Convert Constituent Parse to Dependency Structure

Book the flight through Houston

- ▶ Build a constituent parse
- ▶ Convert to dependency structure

# Case Study: Bootstrapping a Domain-Specific Sentiment Lexicon

A *segment* is part of one or more sentences that expresses a single sentiment

- ▶ Generate a dependency tree for each segment
- ▶ Remove all relations except some selected types (next page)
- ▶ Apply heuristics to add or modify relations, e.g., to handle negation
- ▶ Associate candidate dependency triples with sentiment (review ratings)
- ▶ Select sufficiently frequent triples that associate with one sentiment (positive, neutral, negative)

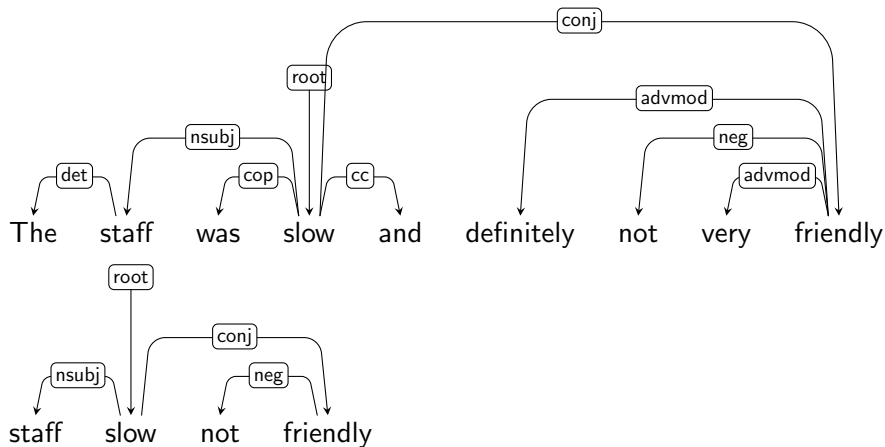
(Work with Zhe Zhang)

## Selected Dependency Relations

- ▶ Adjectival modifier: *amod*
  - ▶ e.g., “Great hotel, friendly helpful staff.”
  - ▶  $\hookrightarrow$  *amod* (*hotel*, *Great*)
- ▶ Adjectival complement: *acomp*
  - ▶ e.g., “Pool looked nice especially at night.”
  - ▶  $\hookrightarrow$  *acomp* (*looked*, *nice*)
- ▶ Nominal subject: *nsubj*
  - ▶ e.g., “The hotel and staff were perfect.”
  - ▶  $\hookrightarrow$  *nsubj* (*perfect*, *hotel*)
- ▶ Negation modifier: *neg* (no, not, nothing, ...)
- ▶ Conjunction: *conj\_and*
- ▶ Preposition: *prep\_with*
- ▶ Root: *root*

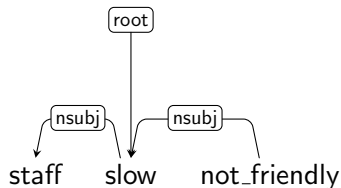
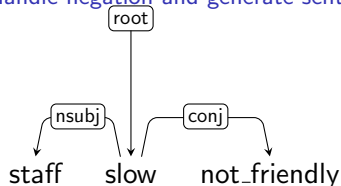
# Sentiment Lexicon: 1

Build dependency parse and discard relations except those given above



## Sentiment Lexicon: 2

Handle negation and generate sentiment triples



- ▶ New not\_friendly node
- ▶ The last step is not a dependency tree; also the relationship is nsubj
- ▶ Extracted triples: {root\_adj(ROOT, slow), nsubj(slow, staff), nsubj(not\_friendly, staff)}

# Heuristics for Producing Sentiment Triples

Function	Condition	Replace or Assert
Handle Negation	$neg(w_H, w_D)$	$w_H \leftarrow w_D + \_ + w_H$
Build Relationships ( <i>conj_and</i> and <i>amod</i> )	$amod(w_H, w_i)$ $conj\_and(w_i, w_j)$	$amod(w_H, w_i)$ $amod(w_H, w_j)$
Build Relationships ( <i>conj_and</i> and <i>acom</i> )	$acom(w_H, w_i)$ $conj\_and(w_i, w_j)$	$acom(w_H, w_i)$ $acom(w_H, w_j)$
Build Relationships ( <i>conj_and</i> and <i>nsubj</i> )	$nsubj(w_i, w_D)$ $conj\_and(w_i, w_j)$	$nsubj(w_i, w_D)$ $nsubj(w_j, w_D)$

Example:  $neg(\text{friendly}, \text{not})$  maps to  $\text{not\_friendly}$

# Transition-Based Dependency Parsing

Based on shift-reduce (stack-based) parsing for CFGs

- ▶ Configuration
  - ▶ Input words and cursor indicating how far read, initially at beginning
  - ▶ State of a stack, initially a ROOT node
  - ▶ Output dependency tree
- ▶ Shift: move token from input to stack (working memory)
- ▶ Reduce: assert a head-dependent relation involving the top token and another token from the stack
  - ▶ Either of them could be the head
- ▶ Transitions between configurations
  - ▶ Shift
  - ▶ Reduce
- ▶ Terminal configuration
  - ▶ Input processed in its entirety
  - ▶ Empty stack: nothing dangling
  - ▶ Dependency tree: as constructed—thus rooted at ROOT



# Arc Standard Parser: Greedy Approach but Works Well

Terminal state: ROOT is at the top of the stack

- ▶ Transition: Left arc
  - ▶ Prerequisite: Two or more elements are on the stack
  - ▶ Prerequisite: ROOT is not the second word since ROOT cannot be a dependent of anything
  - ▶ Assert: word at stack top as head of the next word
  - ▶ Remove the lower word from the stack
- ▶ Transition: Right arc
  - ▶ Prerequisite: Two or more elements are on the stack
  - ▶ Assert: word at stack top as dependent of the next word
  - ▶ Remove the upper word from the stack
- ▶ Transition: Shift
  - ▶ Remove word from input
  - ▶ Push that word on top of the stack

Need an *oracle*, a way to choose the dependency relation asserted in the Left Arc and Right Arc transitions

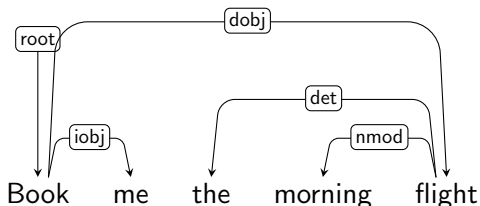
# Arc Standard Parser: Exercise

Book me the morning flight

- ▶ What is an edge in a dependency parse?
- ▶ Which elements are reduced?
- ▶ Which of these becomes the head?

# Arc Standard Parser Example

Error in the book about iobj versus dobj



- ▶ Exercise: Let's work out an execution that produces this parse
- ▶ Reduction order
  - 1 Right arc: book  $\rightarrow$  me
  - 2 Left arc: morning  $\leftarrow$  flight
  - 3 Left arc: the  $\leftarrow$  flight
  - 4 Right arc: book  $\rightarrow$  flight
  - 5 Right arc: root  $\rightarrow$  book

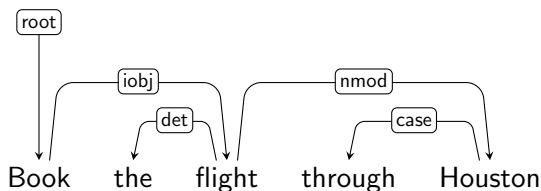
# Building a Training Set

Begin from a dependency treebank linking each sentence to a reference dependency parse

On parsing each sentence, for each configuration

- ▶ Choose Left Arc if
  - ▶ It produces a dependency relation present in the reference parse
- ▶ Choose Right Arc if
  - ▶ It produces a dependency relation present in the reference parse
  - ▶ All dependents of the word at the top in the reference parse have been handled
    - ▶ If there is an out-edge from the word in the tree, leave it alone
- ▶ Choose Shift otherwise

## Arc Standard Training Example



- ▶ Exercise: Let's work out an execution that learns from this parse
- ▶ Reductions are considered in this order:
  - 1 Left arc: root  $\leftarrow$  book: not present in reference parse
  - 2 Right arc: root  $\rightarrow$  book: would lose book prematurely, so No!
  - 3 Left arc: the  $\leftarrow$  flight
  - 4 Right arc: book  $\rightarrow$  flight: would lose flight prematurely, so No!
  - 5 Left arc: through  $\leftarrow$  Houston
  - 6 Right arc: flight  $\rightarrow$  Houston
  - 7 Right arc: book  $\rightarrow$  flight
  - 8 Right arc: root  $\rightarrow$  book (safe to do so after book's out-edges)

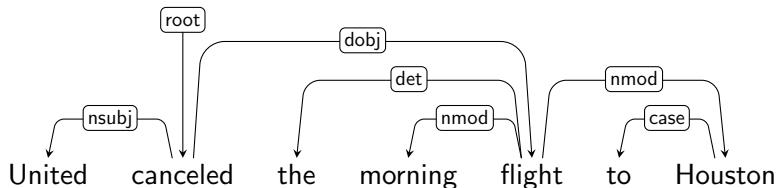
# What Training Set is Acquired from the Previous Example

Give a series of snapshots as the example develops

## Features Useful for Training a Dependency Parser

- ▶ Generally valuable features (as for POS tagging)
  - ▶ Word form
  - ▶ Lemmas
  - ▶ Parts of speech
- ▶ Language-specific morphosyntactic features, e.g., case marking
- ▶ Too many possible configurations and stack contents
  - ▶ Words near the top of the stack are more relevant
  - ▶ Relations between such words
  - ▶ Upcoming words in the input
- ▶ Feature templates pairing *location* and *property*
  - ▶ Locations: stack ( $s_i$ ), input buffer ( $b_j$ ), set of relations ( $r$ )
  - ▶ Properties of locations: word form ( $w$ ), lemma ( $l$ ), POS ( $t$ )
  - ▶ Example: feature of “word form at top of stack” is  $s_1.w$
  - ▶ Example: feature of “word form at top of stack and its POS” is  $s_1.wt$
- ▶ Composite templates concatenate two or more templates

# Example of Applying Feature Templates: 1





## Example of Applying Feature Templates: 2

When we have arrived at this configuration

Stack	Word buffer	Relations
root, canceled, flights	for Houston	canceled → United flights → morning flights → the

Compute the feature values

Feature	Transition
$s_1.w = \text{flights}$	Shift
$s_2.w = \text{canceled}$	Shift
$s_1.t = \text{NNS}$	Shift
$s_2.t = \text{VBD}$	Shift
$b_1.w = \text{to}$	Shift
$b_1.t = \text{TO}$	Shift
$s_1.wt = \text{flightsNNS}$	Shift
$s_1.t \circ s_2.t = \text{NNSVBD}$	Shift

# Evaluation

- ▶ Unlabeled attachment accuracy
  - ▶ Based on head assigned to each word
  - ▶ Ignores dependency relation
- ▶ Labeled attachment accuracy
  - ▶ Accounts for dependency relation