Dependency Parsing

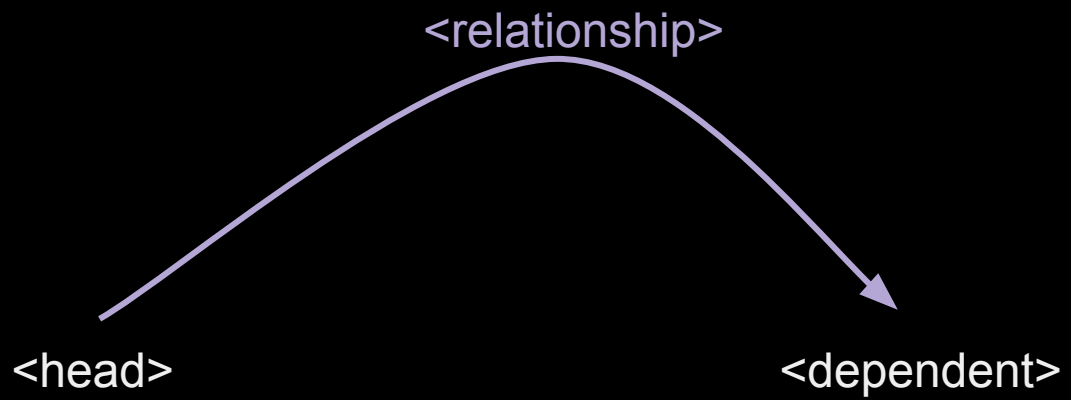
CSE538 - Spring 2024

Topics

Dependency Parsing

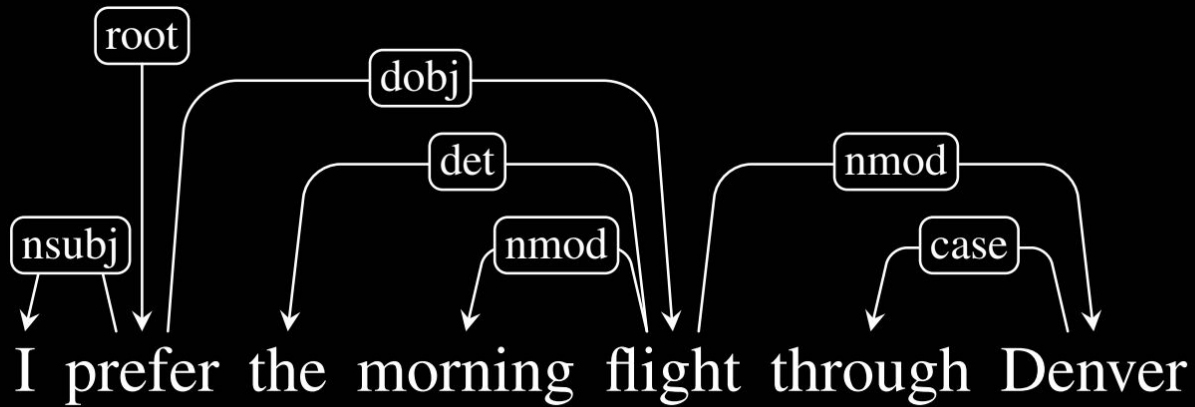
Transition-based (Shift-Reduce algorithm)

Dependency Parsing



dependency -- binary asymmetrical relation between tokens

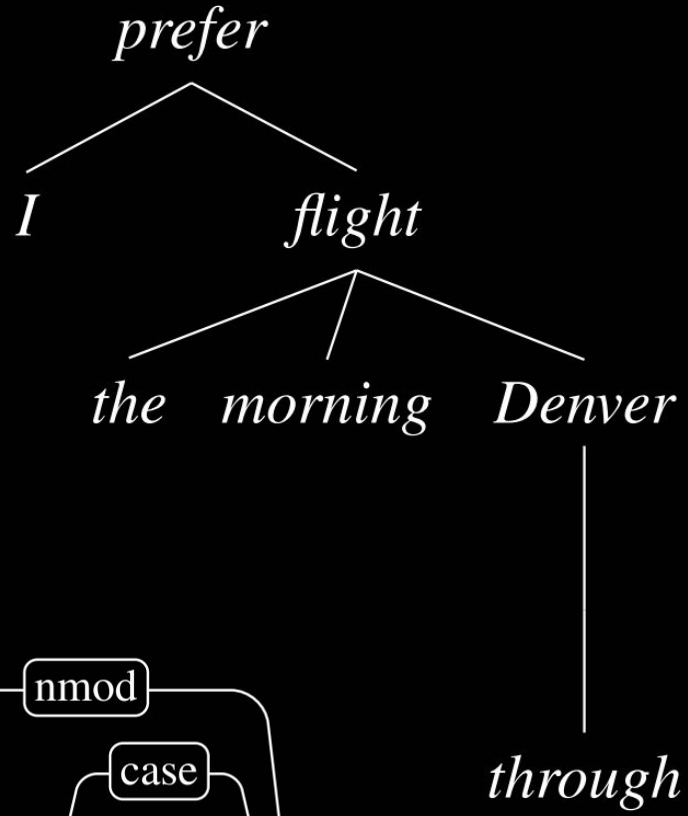
Dependency Parsing

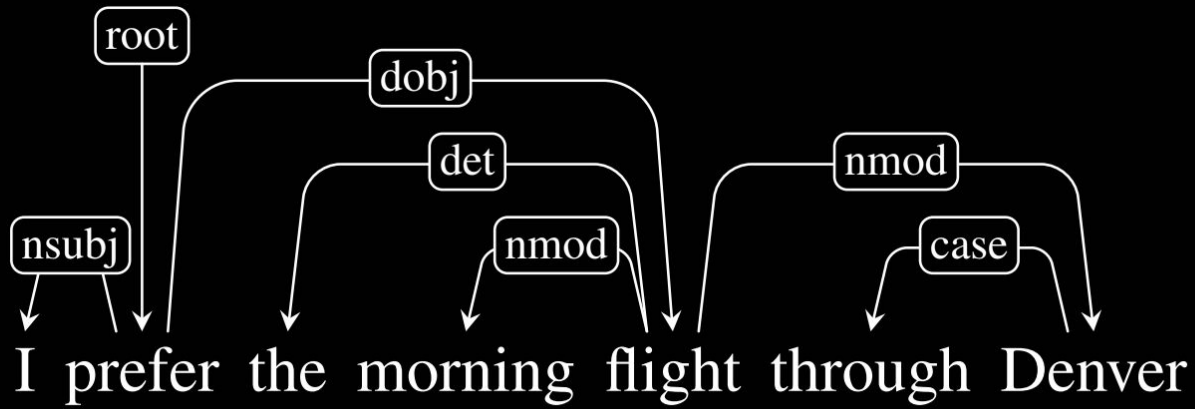


(13.1)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing





(13.1)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing

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

Figure 13.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing

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 13.3 Examples of core Universal Dependency relations.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing

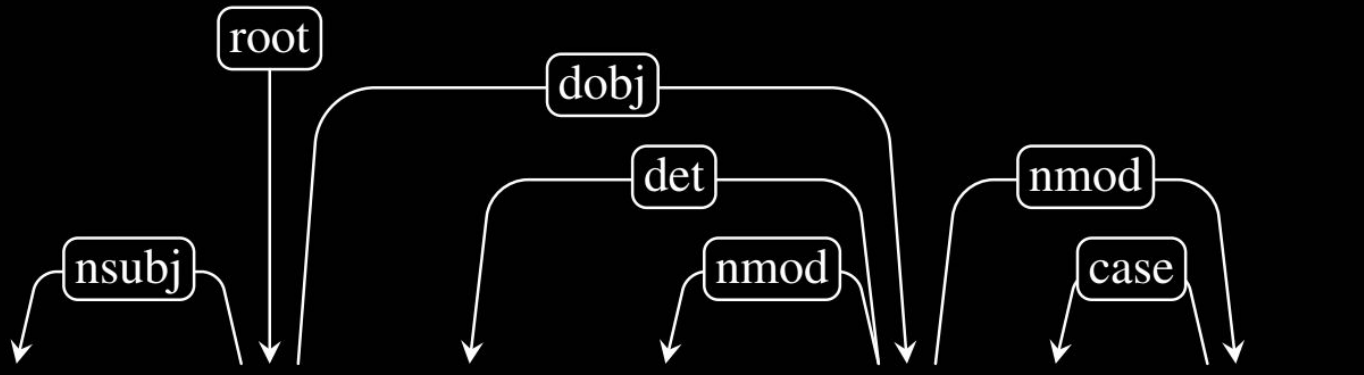
Verbal Predicate -- like a function, takes arguments: “United” and “the flight” in this case.

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 13.3 Examples of core Universal Dependency relations.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates

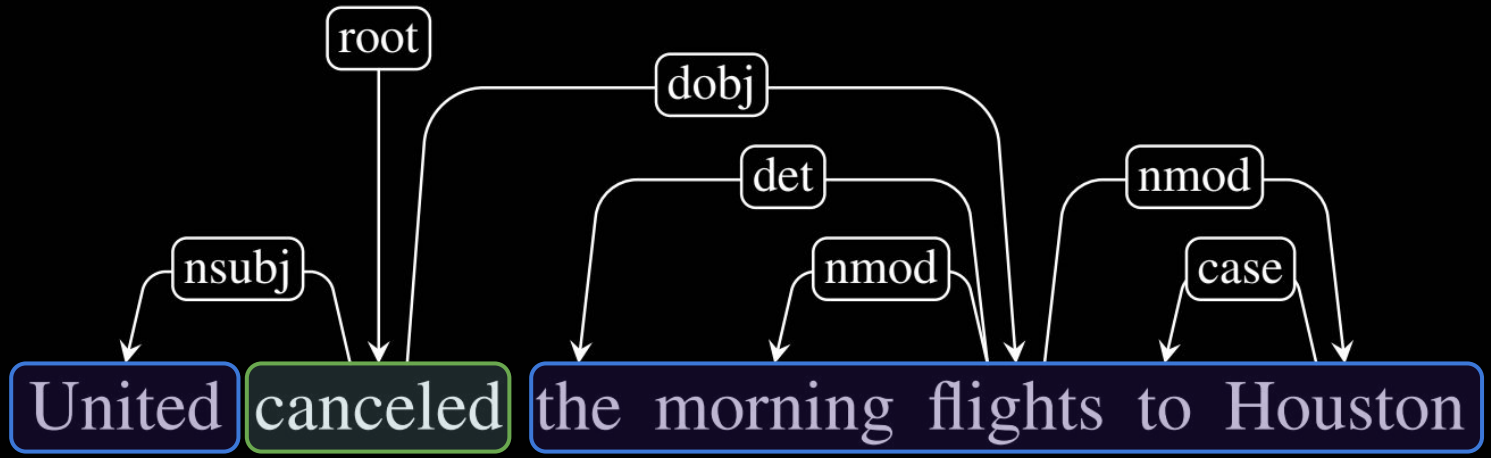


(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates

cancel(“United”, “the morning flights to Houston”)

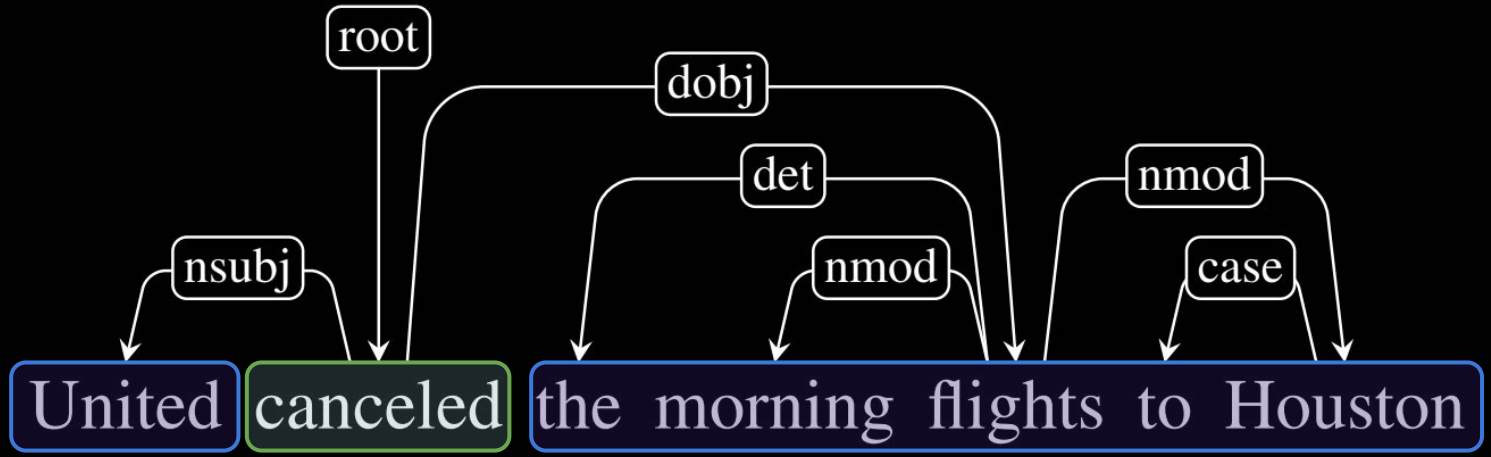


(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates

to_call_off(“United”, “the morning flights to Houston”)

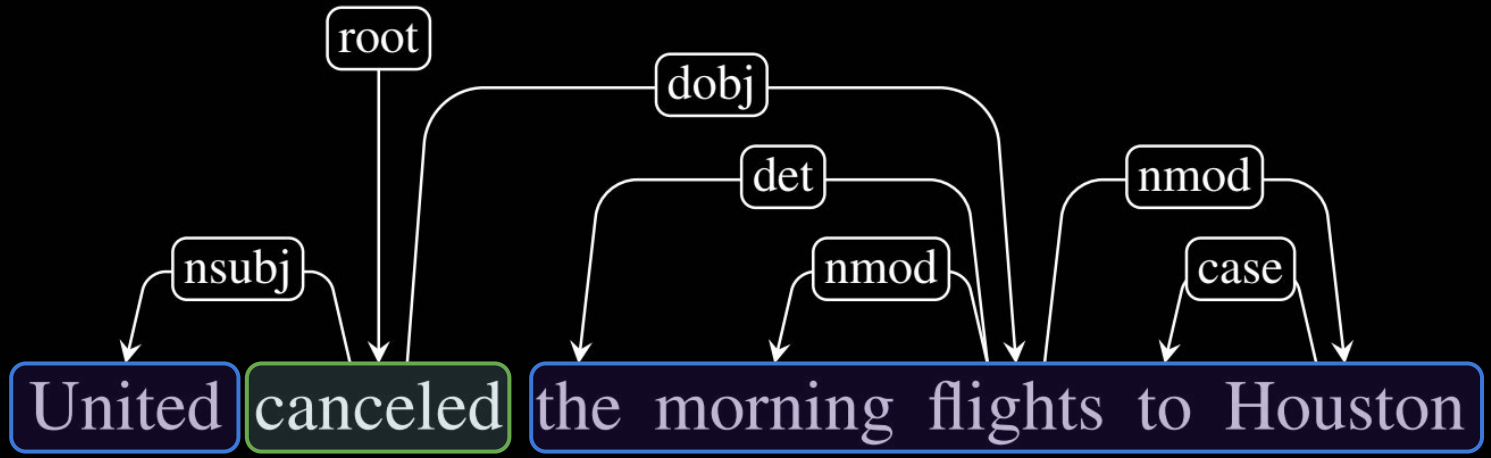


(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates
Semantic Roles

to_call_off(agent="United", event="the morning flights to Houston")

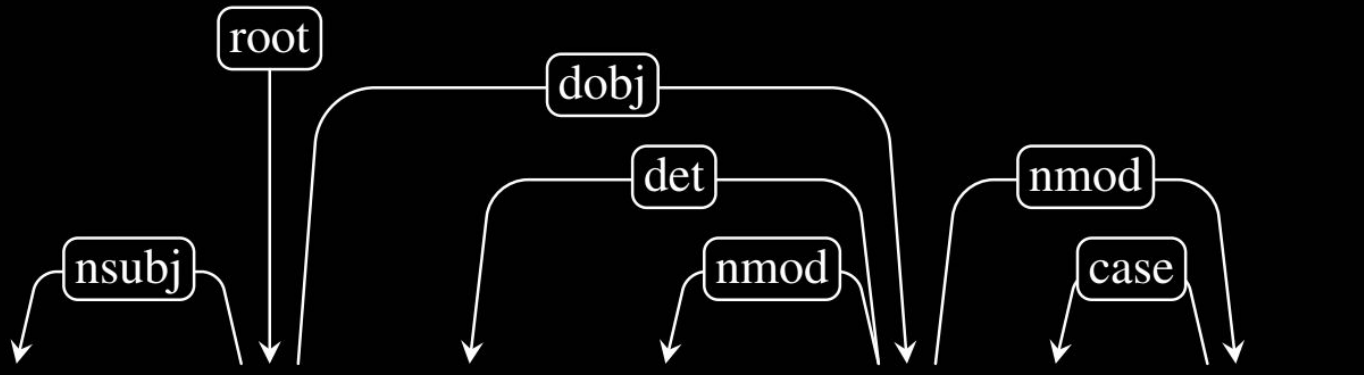


(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V2, A2), …] (vertices and arcs)
Restrictions:
?



(13.2)

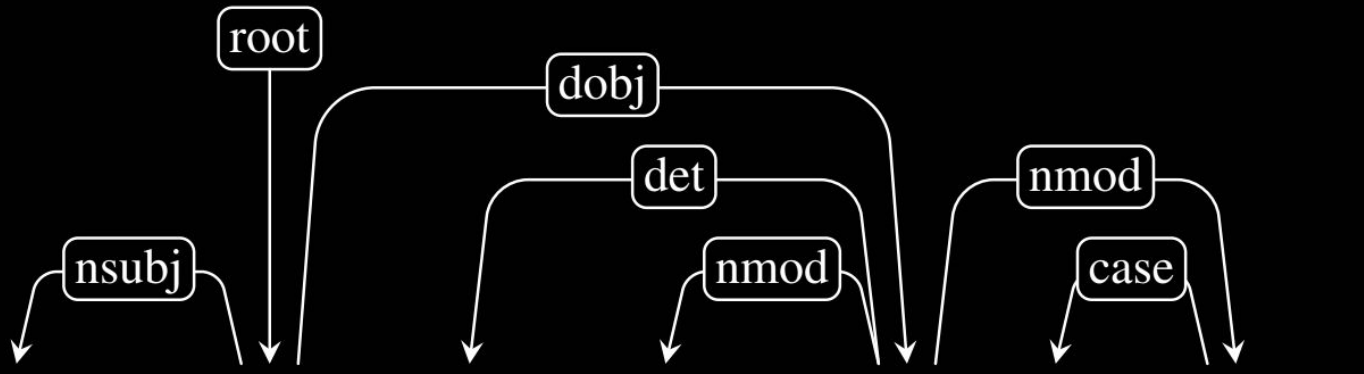
(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V2, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

Inspired by “Shift-reduce parsing” -- process one word at a time, using a stack to keep some sort of memory.

Elements:

- *S*: stack, initialized with “ROOT”
- *B*: input buffer, initialized with tokens (w1, w2, ….) of sentence
- *A:* set of dependency arcs, initialized empty
- *T:* Actions, given *wi* (next token in stack)

Transition-based Dependency Parsing

Inspired by “Shift-reduce parsing” -- process one word at a time, using a stack to keep some sort of memory.

Elements:

- *S*: stack, initialized with “ROOT”
- *B*: input buffer, initialized with tokens (w1, w2, ….) of sentence
- *A:* set of dependency arcs, initialized empty
- *T:* Actions, given *wi* (next token in stack)
  - *shift*(*B,S*): move w from *B* to *S*
  - *left-arc(S,A):* make top of stack **head** of next item: add to A; remove dependent from stack
  - *right-arc(S,A):* make top of stack **dependent** of next item: add to A; remove dep from stack

Using discriminative classifiers (i.e. logistic regression) to make decisions.

Transition-based Dependency Parsing

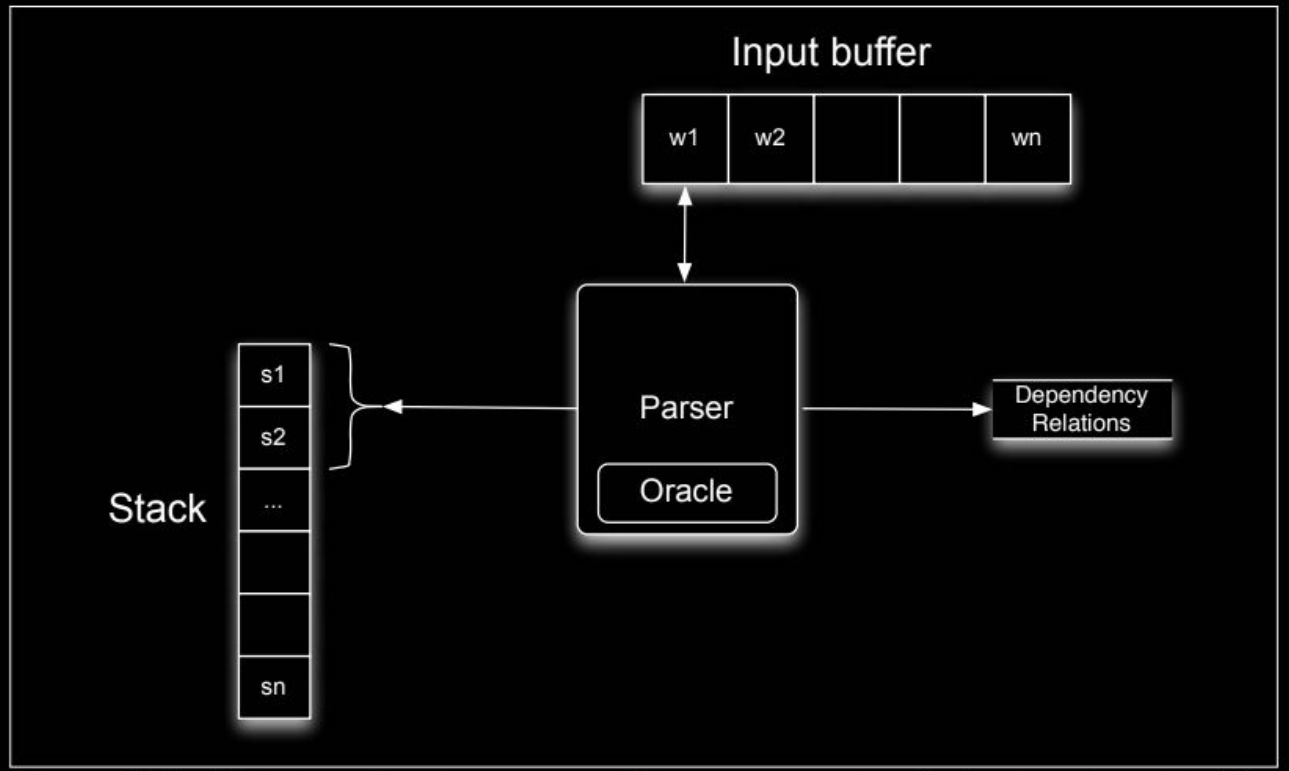


Figure 13.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

```
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
```

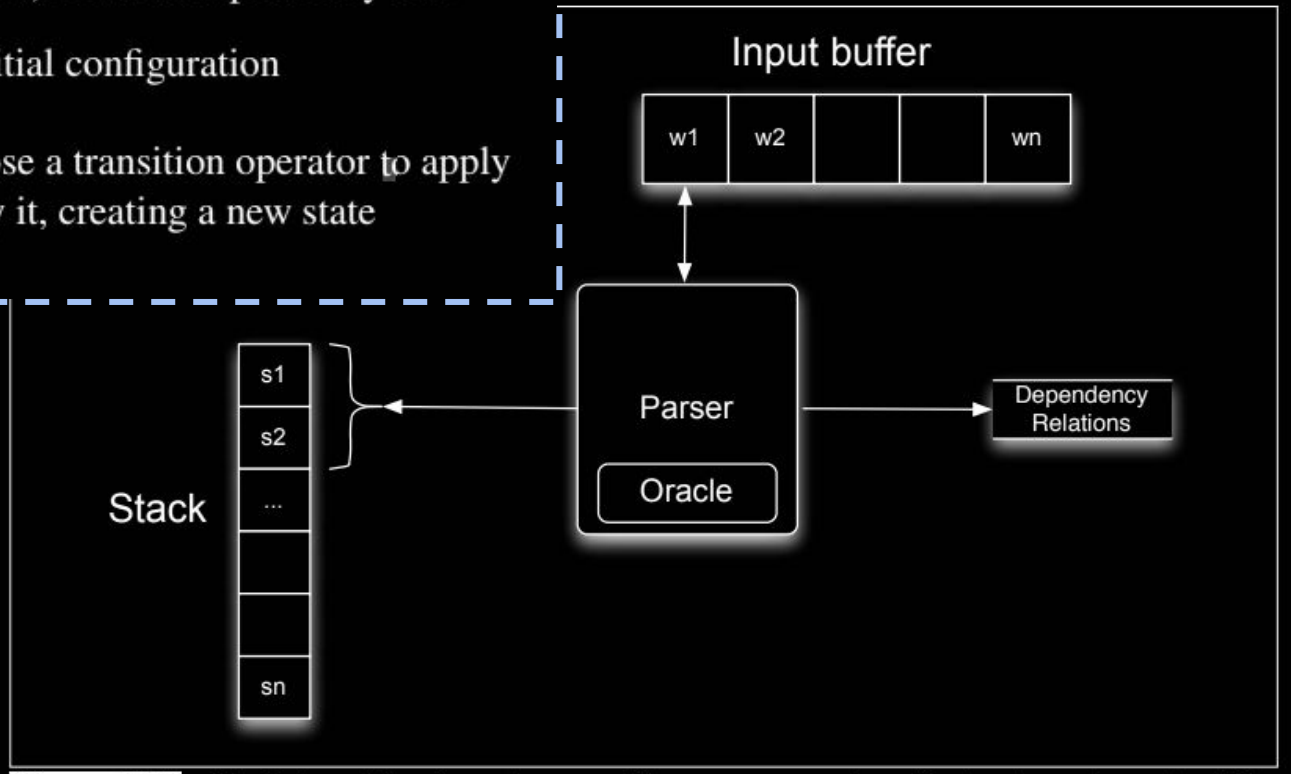


Figure 13.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

```
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
```

(13.5) Book me the morning flight

Let's consider the state of the configuration at Step 2, after the word *me* has been pushed onto the stack.

Stack	Word List	Relations
[root, book, me]	[the, morning, flight]	

The correct operator to apply here is RIGHTARC which assigns *book* as the head of *me* and pops *me* from the stack resulting in the following configuration.

Stack	Word List	Relations
[root, book]	[the, morning, flight]	(book → me)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	

shift(*B,S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	

shift(*B,S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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)

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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	

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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)

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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 13.7 Trace of a transition-based parse.

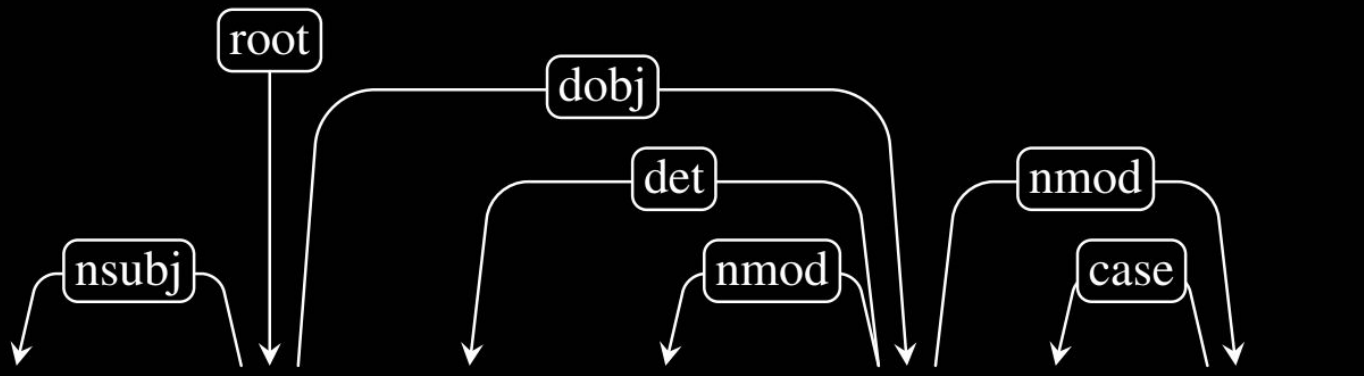
(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

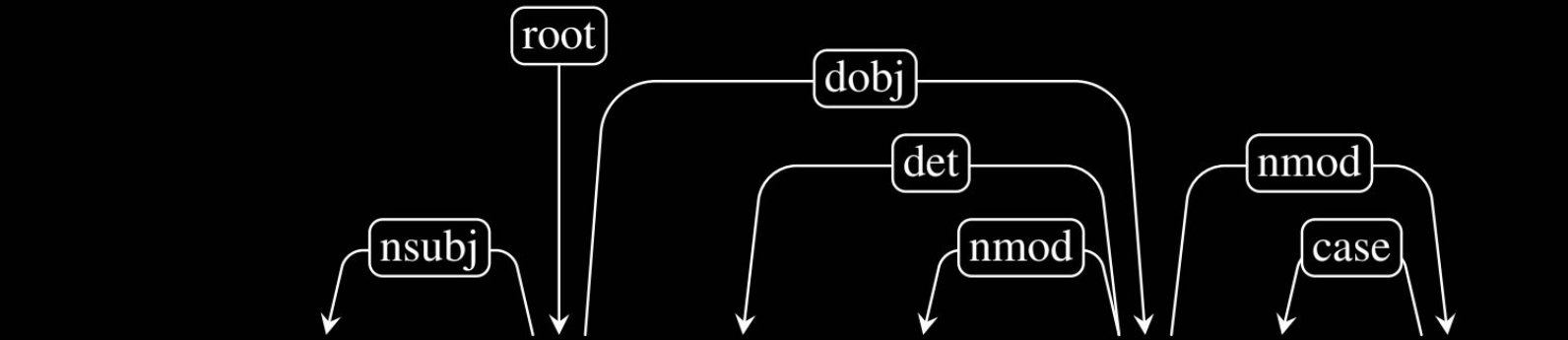
Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

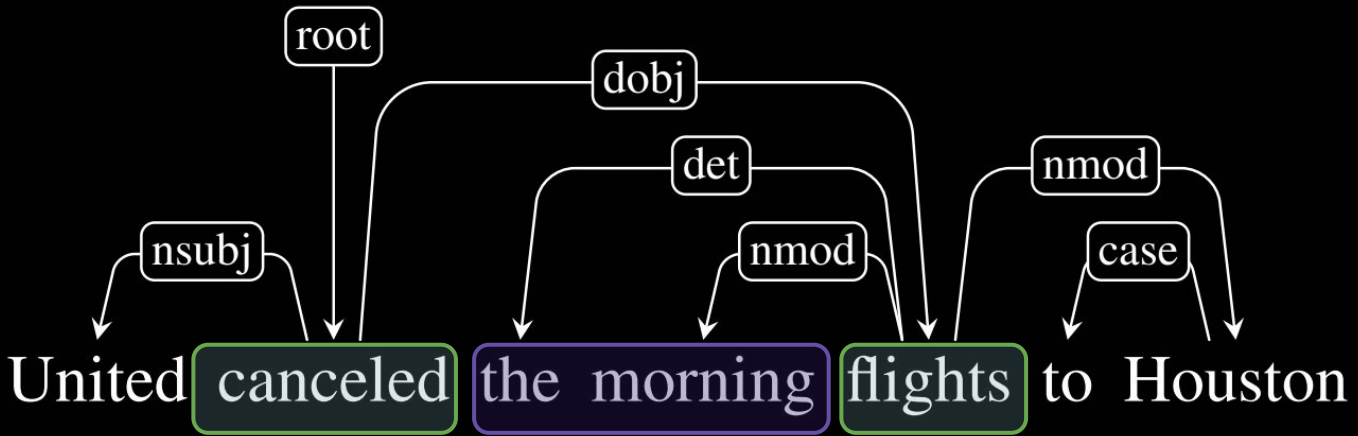
Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

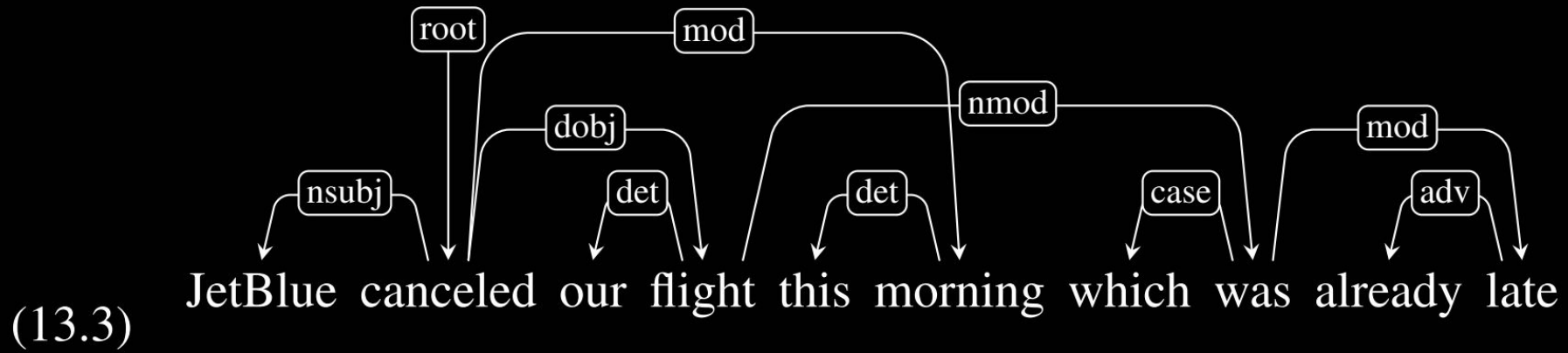
Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word



(13.3)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

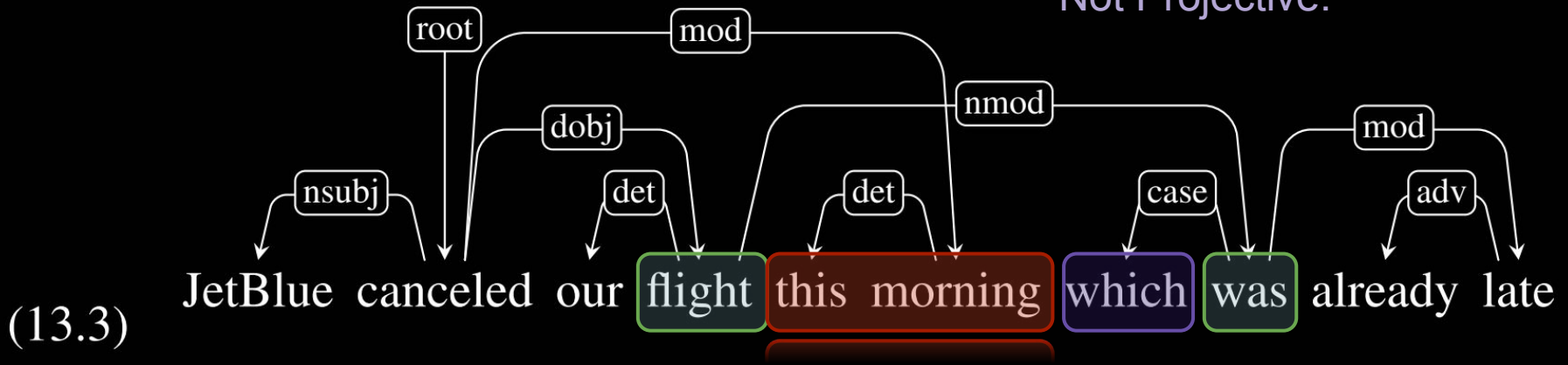
A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word.

Not Projective:



(13.3)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word.

Not Projective:

Why do we care? Dependency trees from Context-Free Grammars are guaranteed to be projective; Thus, transition based techniques are certain to have errors occasionally on non-projective dependency graphs.

(From SLP 3rd ed., Jurafsky and Martin 2018)

From Syntax to Semantics

- We've already seen words have many meanings.
  - Context is key

- Verbs can been seen as functions (predicates) that take arguments.
  - **Syntactic** arguments fulfill **semantic** roles

- Words have implicit syntactic relationships
  with each other in given sentences.
  - Dependency Parsing: each word has one head
  - Easily constructed through 3 actions of shift-reduce parsing.

Takeaway: There is an interplay between word meaning and sentence structure!

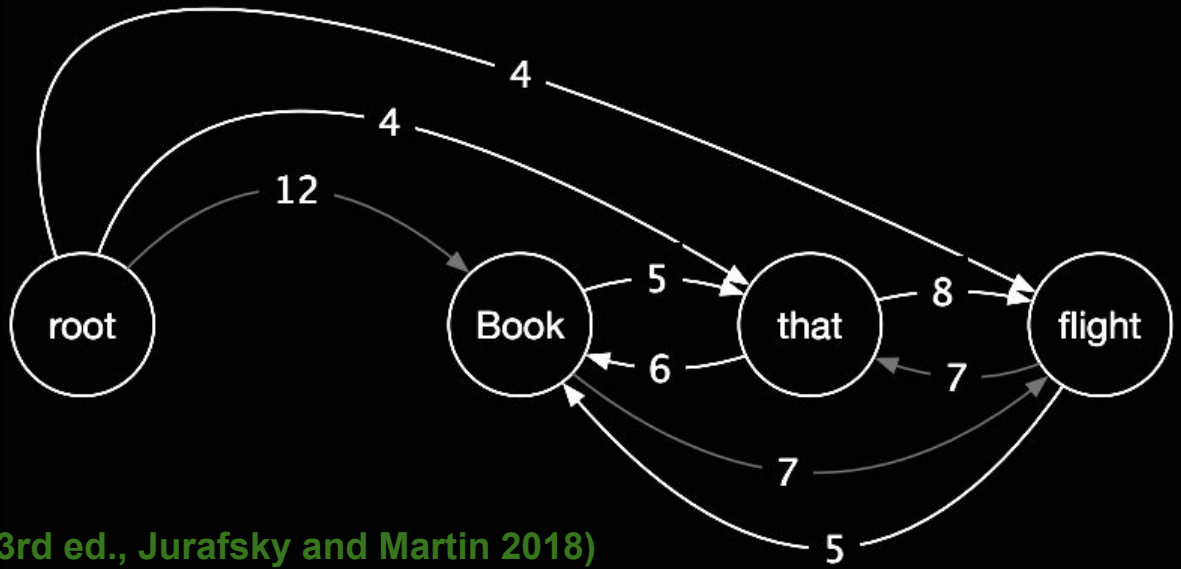
Graph-based Approaches

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Idea: Search through all possible trees and pick best.



(From SLP 3rd ed., Jurafsky and Martin 2018)

Graph-based Approaches

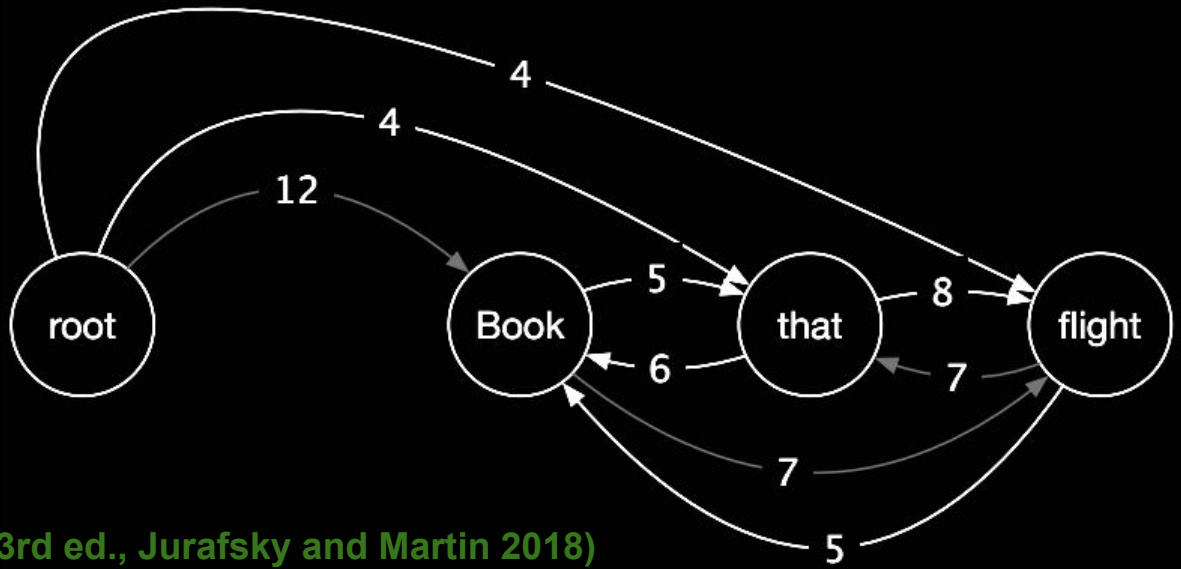
A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Idea: Search through all possible trees and pick best.

General approach: For each word, pick the most likely head. Then check if still a fully-connected tree, and adjust.



(From SLP 3rd ed., Jurafsky and Martin 2018)

Graph-based Approaches

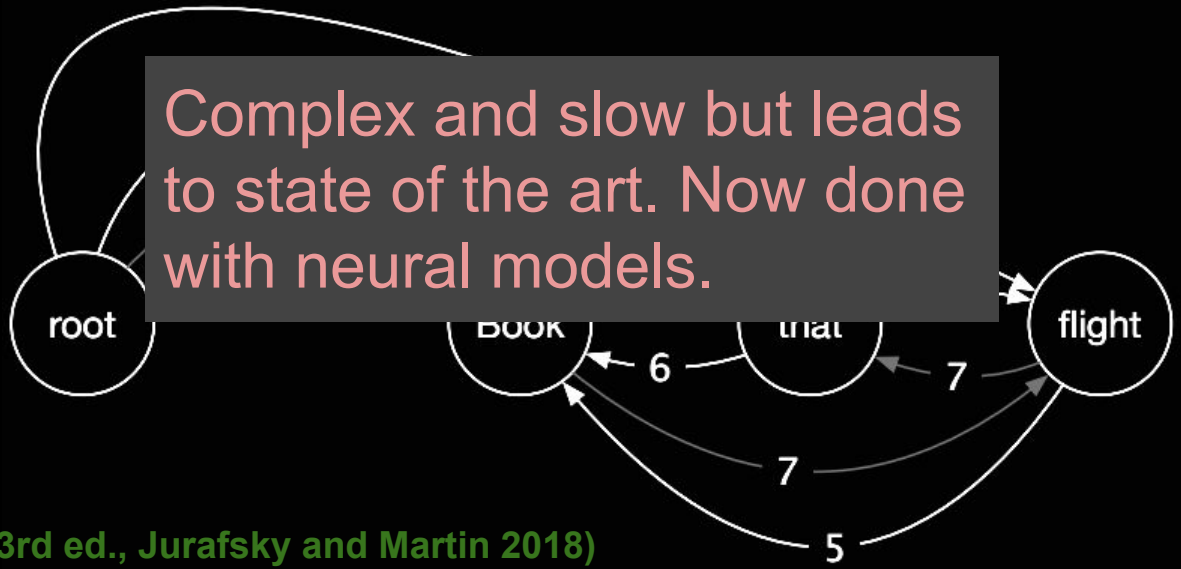
A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Idea: Search through all possible trees and pick best.

General approach: For each word, pick the most likely head. Then check if still a fully-connected tree, and adjust.

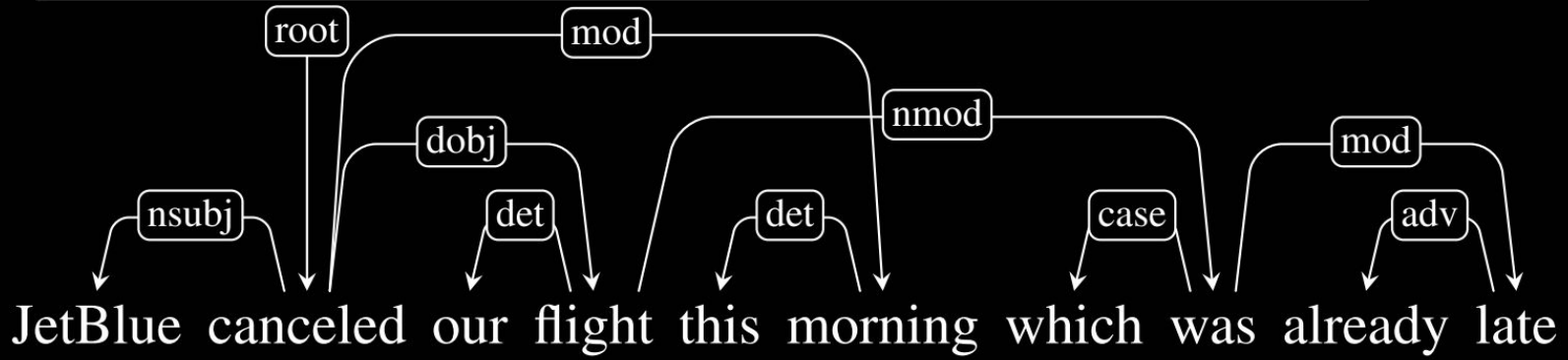


(From SLP 3rd ed., Jurafsky and Martin 2018)

Semantic Roles

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
CONTENT	The proposition or content of a propositional event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

(13.3)



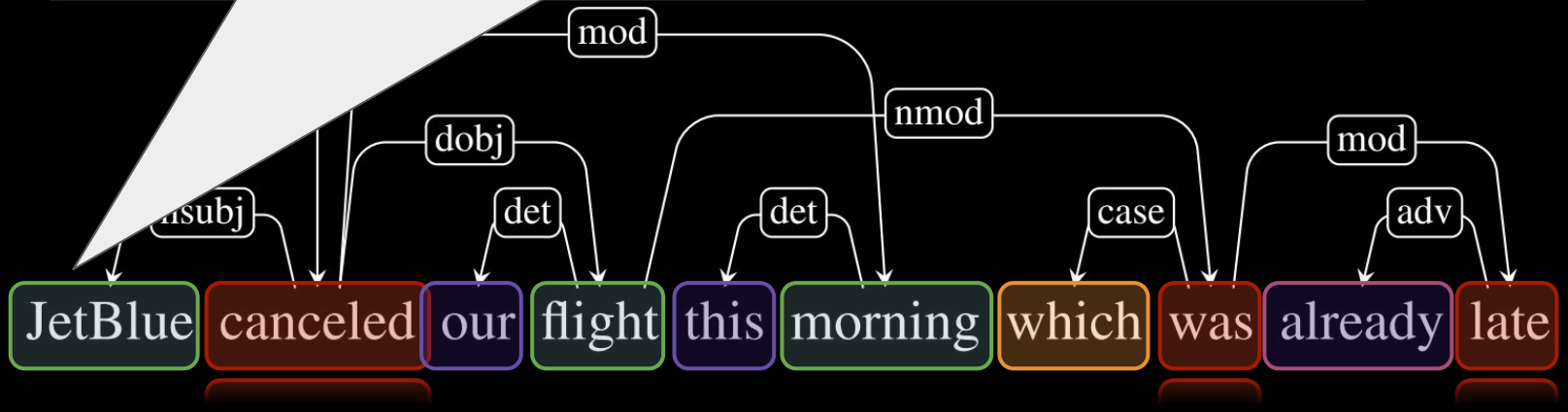
(From SLP 3rd ed., Jurafsky and Martin 2018)

Semantic Roles

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
GOAL	[illegible] of an object of a transfer event

Roles are restricted to nouns, but signalled through the verb and other parts of speech.

(13.3)



(From SLP 3rd ed., Jurafsky and Martin 2018)