Welcome back to CIS 530!

PLEASE TYPE YOUR QUESTIONS IN THE CHAT

IF YOUR INTERNET IS TOO SLOW TO SEE THE VIDEO, YOU CAN FIND THE SLIDES ON THE CLASS WEBSITE

New course policies

1. I’m granting everyone 10 extra late days. You can now use up to 3 late days per HW, quiz or project milestone.
2. I’m offering a HW option for the term project component of the grade. You can do 4 extra HW assignments instead of a project.
3. I’m allowing everyone to drop their lowest scoring quiz.
4. Everyone can drop their lowest scoring homework. (You can’t drop project milestones).
5. You can opt to do the course pass/fail. 50% and above is passing.

Homework Option

We are creating a set of 4 additional weekly homework assignments. They will have the same deadlines as the project milestones. You may do the homework assignments individually or in pairs.

HW9: Classifying Depression due – requires special data access

HW10: Neural Machine Translation

HW11: BERT

HW12: Perspectives Detection

You can do the homework individually or in pairs.

HW will be graded based on leaderboard and reports (autograders may not be available).

Project Option

The project is a team exercise, with teams of 4-6. Your project will be a self-designed multi-week team-based effort. Milestones:

1. Submit a formal project definition and a literature review. (due 4/8)
2. Collect your data, write an evaluation script and a baseline. (4/15)
3. Implement a published baseline. Prepare a draft of your final project presentation. (4/22)
4. Finish all your extensions to the public baseline, and submit your final report. (4/29)

You need to declare whether you intend to do the project or homework option by this Wednesday using the [Google form linked on Piazza].

http://computational-linguistics-class.org/term-project.html

Office hours

Office hours are going to be held via Zoom. TAs host a host a Zoom group meeting and post the link on Piazza.

We will use the chat to manage the queue. Just like you would write your name on the whiteboard in an in-person meeting. You should write this info to add yourself to the queue:

1. Your name
2. A short version of your question
3. Whether it should be discussed publicly and privately (code help)

For private questions, the TA will add you to a breakout room. For public ones, we'll discuss them as a group so you can hear the answers to other students' questions.

Schedule http://computational-linguistics-class.org/lectures.html#now

Mon, Mar 23, 2020	Wrap-up of Constituency Parsing / Dependency Parsing [Zoom link]	Jurafsky and Martin, Chapter 14 "Statistical Constituency Parsing" Jurafsky and Martin, Chapter 15 "Dependency Parsing" Dragomir Radev, Dependency Parsing Dragomir Radev, Statistical Parsing
Wed, Mar 25, 2020	Logical Representations of Sentence Meaning [Zoom link]	Jurafsky and Martin, Chapter 16 "Logical Representations of Sentence Meaning
Wed, Mar 25, 2020	HW7 "Named Entity Recognition" due	
Mon, Mar 30, 2020	Information Extraction [Zoom link]	Jurafsky and Martin, Chapter 18 "Information Extraction"
Mon, Mar 30, 2020	Quiz due (covers Chapters 12-14)	Jurafsky and Martin, Chapter 12 "Constituency Grammars" Jurafsky and Martin, Chapter 13 "Constituency Parsing" Jurafsky and Martin, Chapter 14 "Statistical Constituency Parsing"
Wed, Mar 25, 2020	Deadline to decide on term project versus weekly homework option. Please specify your preference by filling out this form.	
Fri, Mar 27, 2020	Deadline to complete the IRB training for HW9, if you're doing the HW option. Please follow the instruction for obtaining data on the HW9 page.	
Wed, Apr 1, 2020	Semantic Role Labeling [Zoom link]	Jurafsky and Martin, Chapter 20 "Semantic Role Labeling"
Wed, Apr 1, 2020	HW8 "Learning Hypernyms" due	
Fri, Apr 3, 2020	Withdraw Deadline	

Reminders

HOMEWORK 7 DUE DATE IS DUE BY MIDNIGHT ON 3/25. HW8 WILL BE DUE 4/1.

WASH YOUR HANDS

TAKE CARE OF YOURSELF. MENTAL HEALTH IS IMPORTANT TOO.

Review: Constituency Parsing

JURAFSKY AND MARTIN CHAPTERS 12-14

Formal Definition of a PCFG

A probabilistic context-free grammar *G* is defined by four parameters:

N is a set of **non-terminal symbols** (or variables)

- In NLP, we often use the Penn Treebank tag set

Σ is set of **terminal symbols**

- These are the words (also sometimes called the leaf nodes of the parse tree)

R is a set of production rules, each of the form $A \rightarrow \beta$ [probability]

- $S \rightarrow NP\ VP$ [0.8]
- $S \rightarrow Aux\ NP\ VP$ [0.15]
- $S \rightarrow VP$ [0.05]

S is the start symbol (a non-terminal)

Treebanks as grammar

Treebanks == data

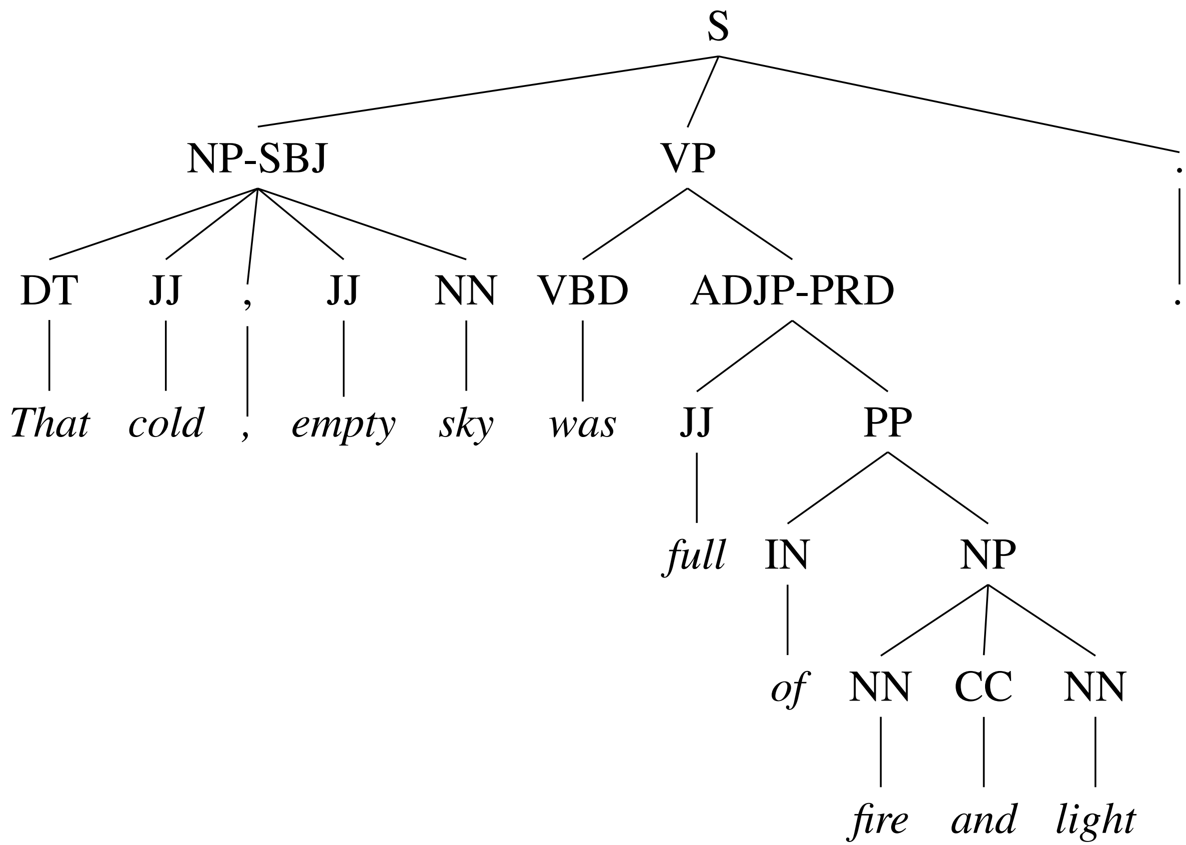
Mitch Marcus

Initially, building a treebank might seem like it would be a lot slower and less useful than building a grammar.

However, a treebank gives us many things

- Reusability of the labor
  - Many parsers, POS taggers, etc.
  - Valuable resource for linguistics
- Broad coverage
- Frequencies and distributional information
- A way to evaluate systems

[Marcus et al. 1993, *Computational Linguistics*]



Extracted rules		
S → NP VP .	DT → That	JJ → full
NP → DT JJ , JJ NN	JJ → cold	IN → of
VP → VBD ADJP	, → ,	NN → fire
ADJP → JJ PP	JJ → empty	CC → and
PP → IN NP	NN → sky	NN → light
NP → NN CC NN	VBD → was	

Rules with counts

40717 PP → IN NP
33803 S → NP-SBJ VP
22513 NP-SBJ → -NONE-
21877 NP → NP PP
20740 NP → DT NN
14153 S → NP-SBJ VP .
12922 VP → TO VP
11881 PP-LOC → IN NP
11467 NP-SBJ → PRP
11378 NP → -NONE-
11291 NP → NN
...
989 VP → VBG S
985 NP-SBJ → NN
983 PP-MNR → IN NP
983 NP-SBJ → DT
969 VP → VBN VP

100 VP → VBD PP-PRD
100 PRN → : NP :
100 NP → DT JJS
100 NP-CLR → NN
99 NP-SBJ-1 → DT NNP
98 VP → VBN NP PP-DIR
98 VP → VBD PP-TMP
98 PP-TMP → VBG NP
97 VP → VBD ADVP-TMP VP
...
10 WHNP-1 → WRB JJ
10 VP → VP CC VP PP-TMP
10 VP → VP CC VP ADVP-MNR
10 VP → VBZ S , SBAR-ADV
10 VP → VBZ S ADVP-TMP

Compute Probabilities using MLE.

CKY Algorithm

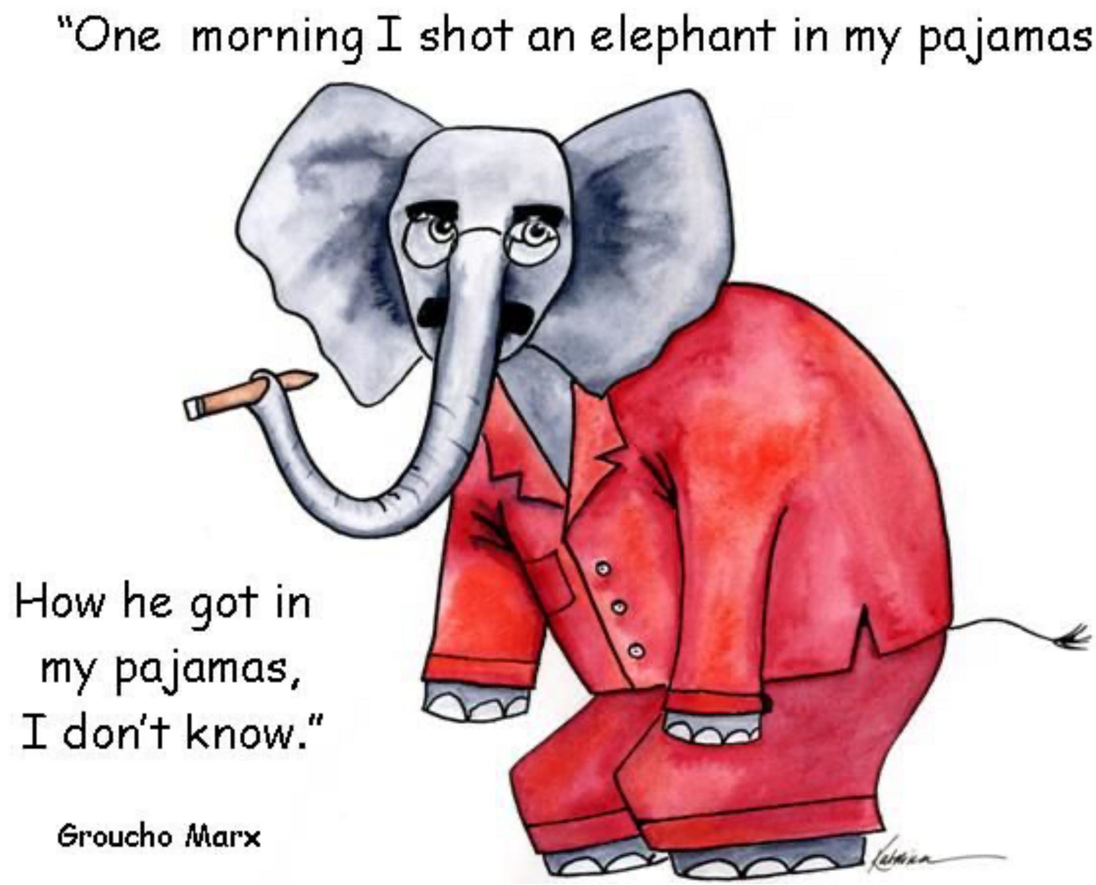
function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  for all $\{A \mid A \rightarrow words[j] \in grammar\}$
    $table[j-1, j] \leftarrow table[j-1, j] \cup A$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      for all $\{A \mid A \rightarrow BC \in grammar$ **and** $B \in table[i,k]$ **and** $C \in table[k,j]\}$
        $table[i,j] \leftarrow table[i,j] \cup A$

CKY Demo at http://lxmls.it.pt/2015/cky.html

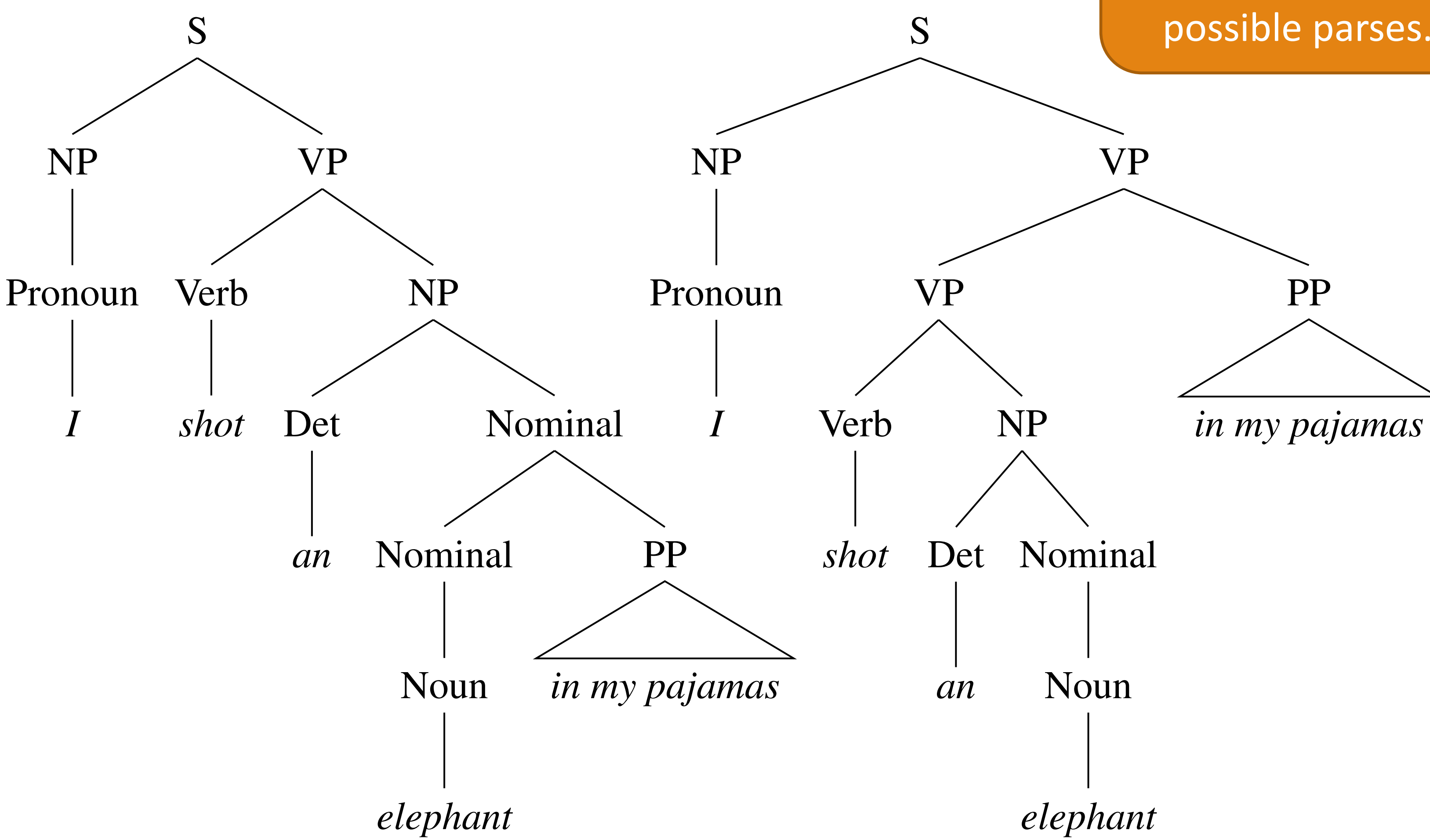
Ambiguity

Ambiguity can arise because of words with **multiple senses or POS tags**. Many kinds of ambiguity are also structural.



Attachment Ambiguity

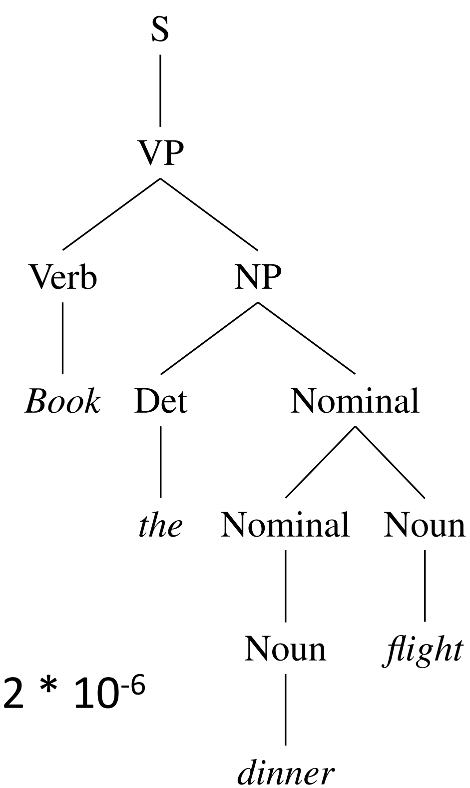
Probabilities give us a way of choosing between possible parses.



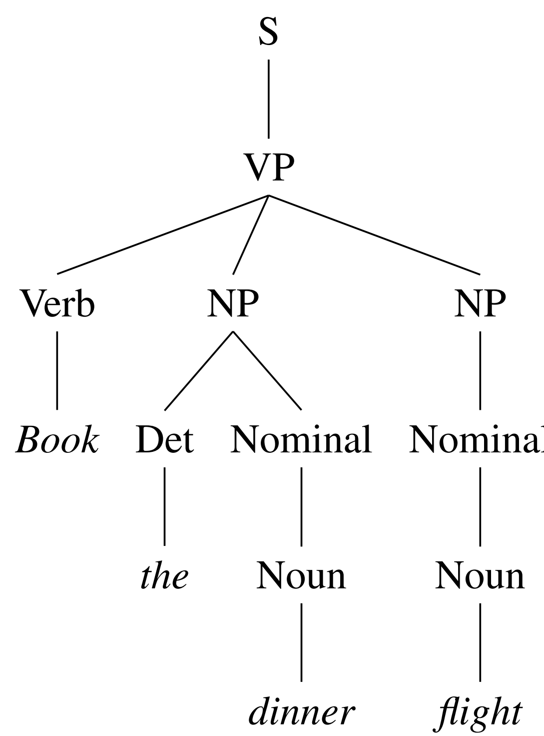
Finding best parse

Pick the parse with the highest probability.

$$\hat{T}(S) = \underset{T\,s.t.\,S=\text{yield}(T)}{\text{argmax}} P(T|S) = P(T,S) = \prod_{i=1}^{n} P(RHS_i|LHS_i)$$

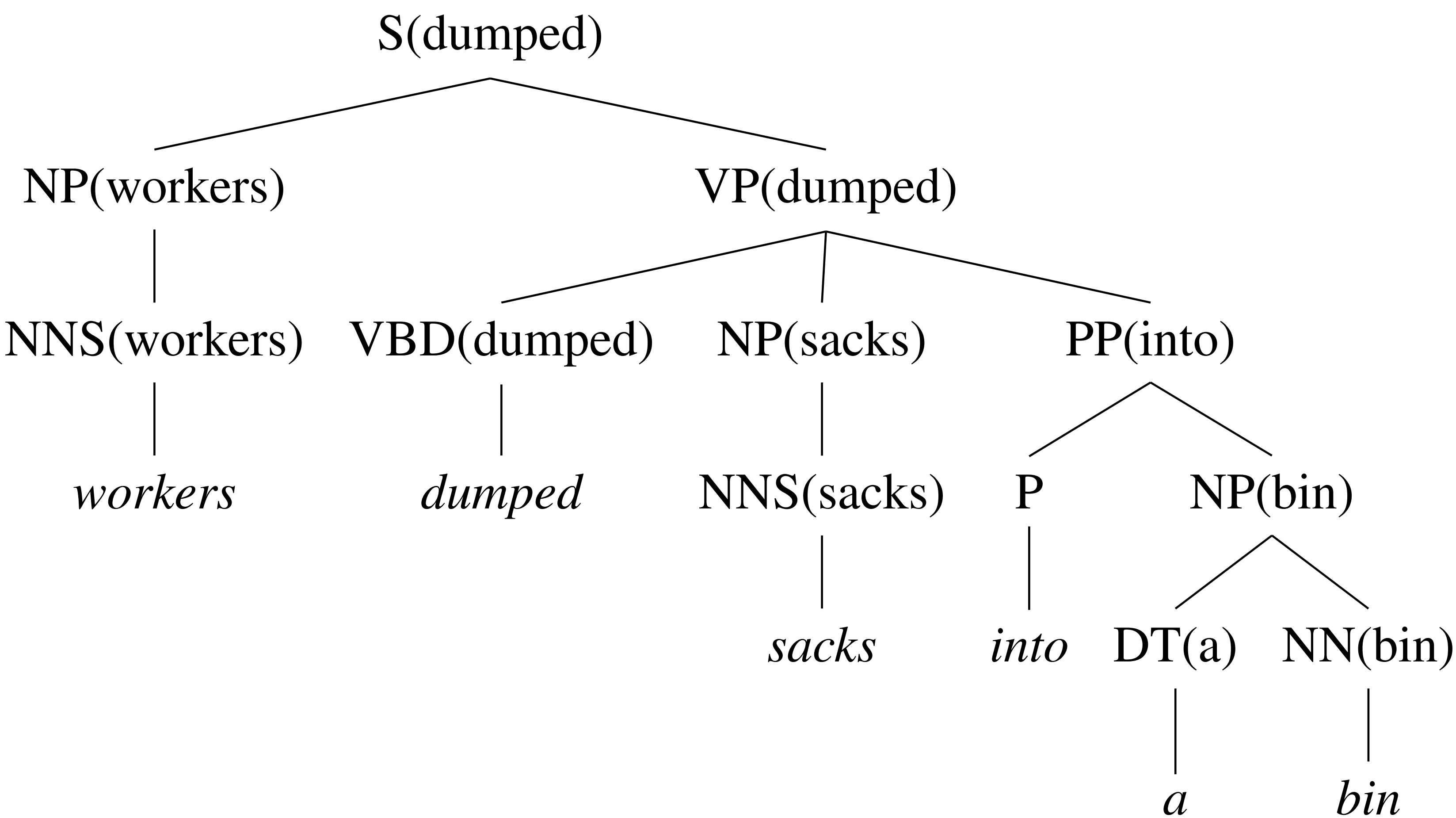


P(T,S) = 2.2 * 10^{-6}



P(T,S) = 6.1 * 10^{-7}

Constituents have heads

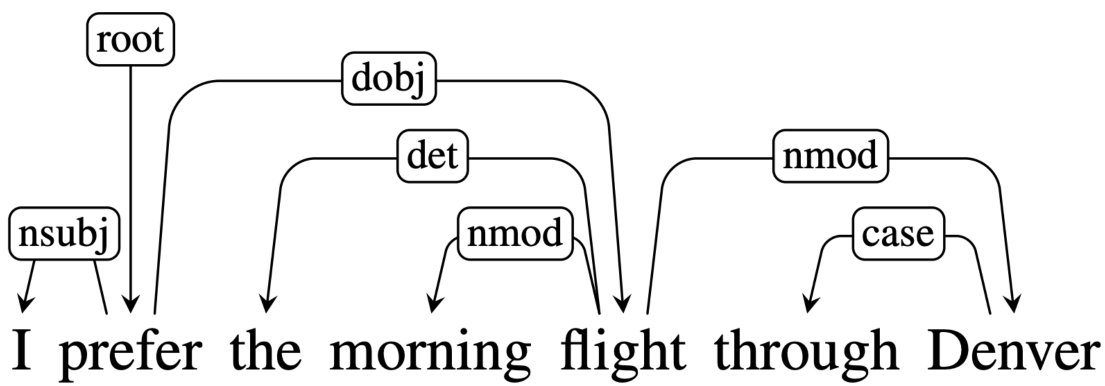


Dependency Parsing

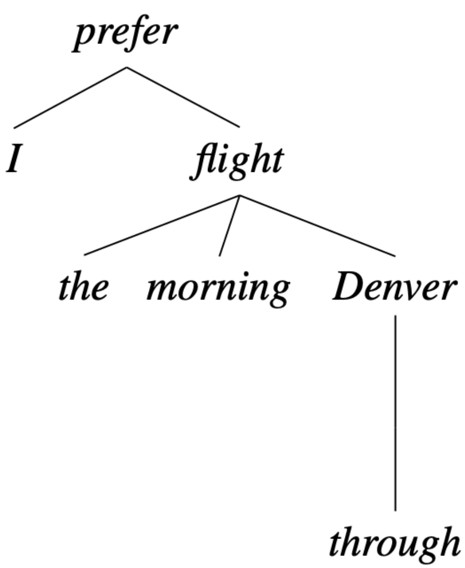
JURAFSKY AND MARTIN CHAPTER 15

Dependency Grammars

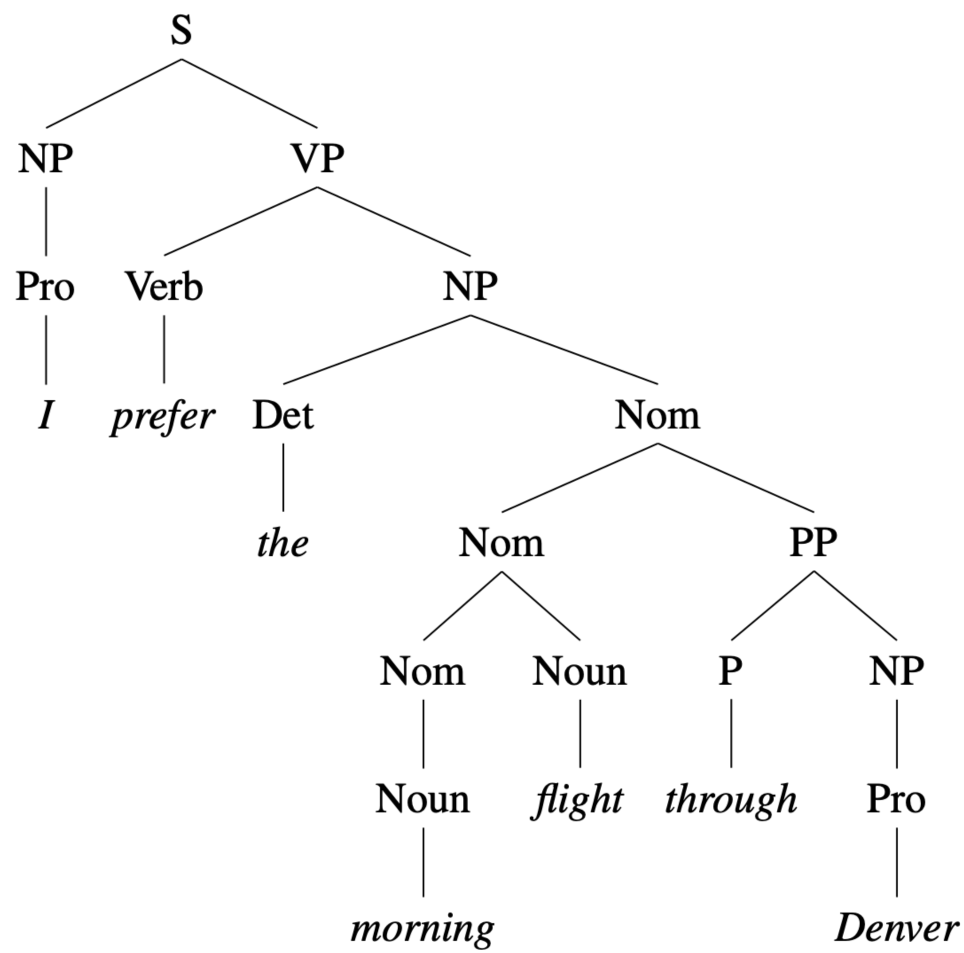
Dependency grammars depict the syntactic structure of sentences solely in terms of the **words in a sentence** and an **associated set of directed head-dependent grammatical relations** that hold among these words.



Dependency – based



Constituent– based



Advantages of dependencies

- Dependencies don’t have nodes corresponding to phrasal constituents. Instead they **directly encode information** that is often buried in phrase structure parses.
- Dependency grammars are better able deal with languages that have a relatively **free word order.**
- Dependency relations **approximate semantic relationships** between words and arguments, which is useful for many applications
  - coreference resolution
  - question answering
  - information extraction.

Dependency Formalism

The dependency structures are directed graphs.

$$\mathbf{G = (V, A)}$$

where **V** is a **set of vertices** and **A** is a set of **ordered pairs of vertices** (or **directed arcs**). Each arc points from the **head** to a **dependent**

$$\textbf{Head} \longrightarrow \textbf{Dependent}$$

Directed arcs can also be **labeled** with the **grammatical relation** that holds between the head and a dependent**.**

Dependency Trees

Other common constraints are that dependency structure must be **connected**, have a **designated root node,** and be acyclic or planar. These result in a **rooted tree** called **a dependency tree**.

A dependency tree is a digraph where:

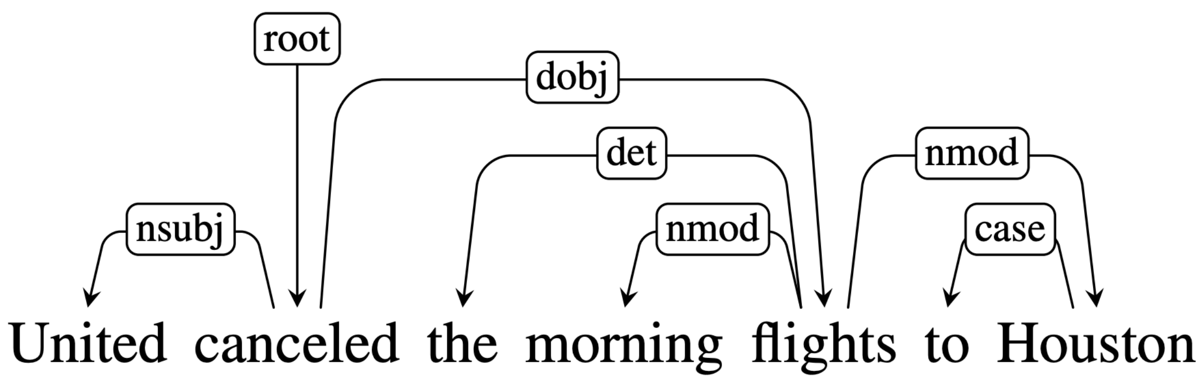
1. There is a **single designated root node** that has no incoming arcs
2. Each vertex has **exactly one incoming arc** (except the root node)
3. There is a **unique path** from the root node to each vertex in V

This mean that each word in the sentence has exactly one head.

Head → **Dependent**

Dependency Relations

In addition having directed arcs point from the head to the dependent, arc can be labeled with the **type of grammatical function** involved between the words



- `nsubj` and `dobj` identify the subject and direct object of the verb *cancelled*
- `nmod`, `det` and `case` relations denote modifiers of the nouns *flights* and *Houston.*

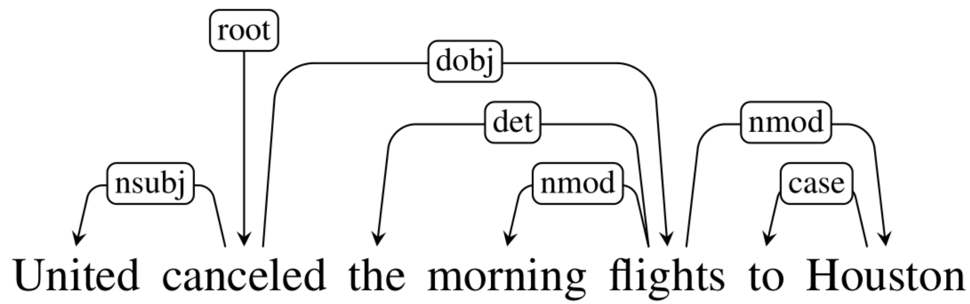
Dependency Relations

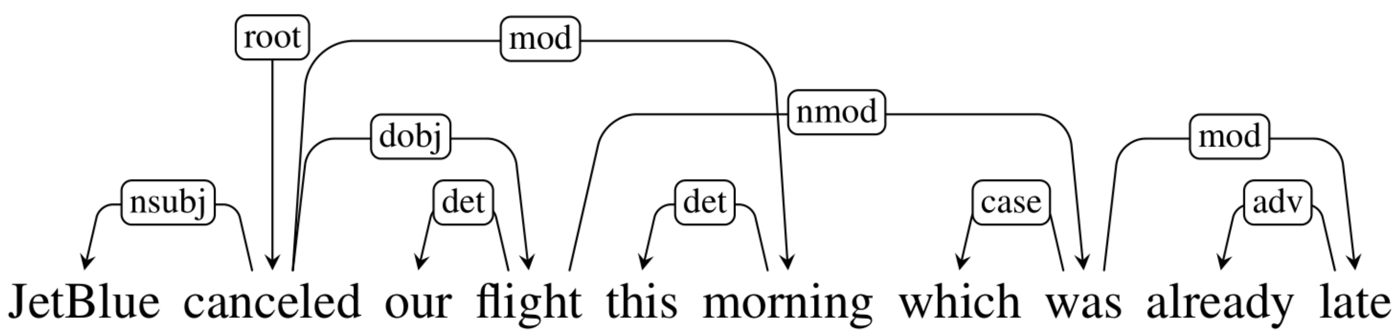
Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET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

Dependency Relations

Relation	Examples with head and dependent
NSUBJ	**United canceled** the flight.
DOBJ	United **diverted** the **flight** to Reno.
IOBJ	We **booked her** the flight to Miami.
NMOD	We **took** the **morning flight.**
AMOD	Book the **cheapest flight.**
NUMMOD	JetBlue canceled **1000 flights.**
APPOS	**United**, a **unit** of UAL, matched the fares.
DET	**The flight** was canceled.
CONJ	We **flew** to Denver and **drove** to Steamboat.
CC	We flew to Denver **and drove** to Steamboat.
CASE	Book the flight **through Houston.**

Projective vs Non-projective



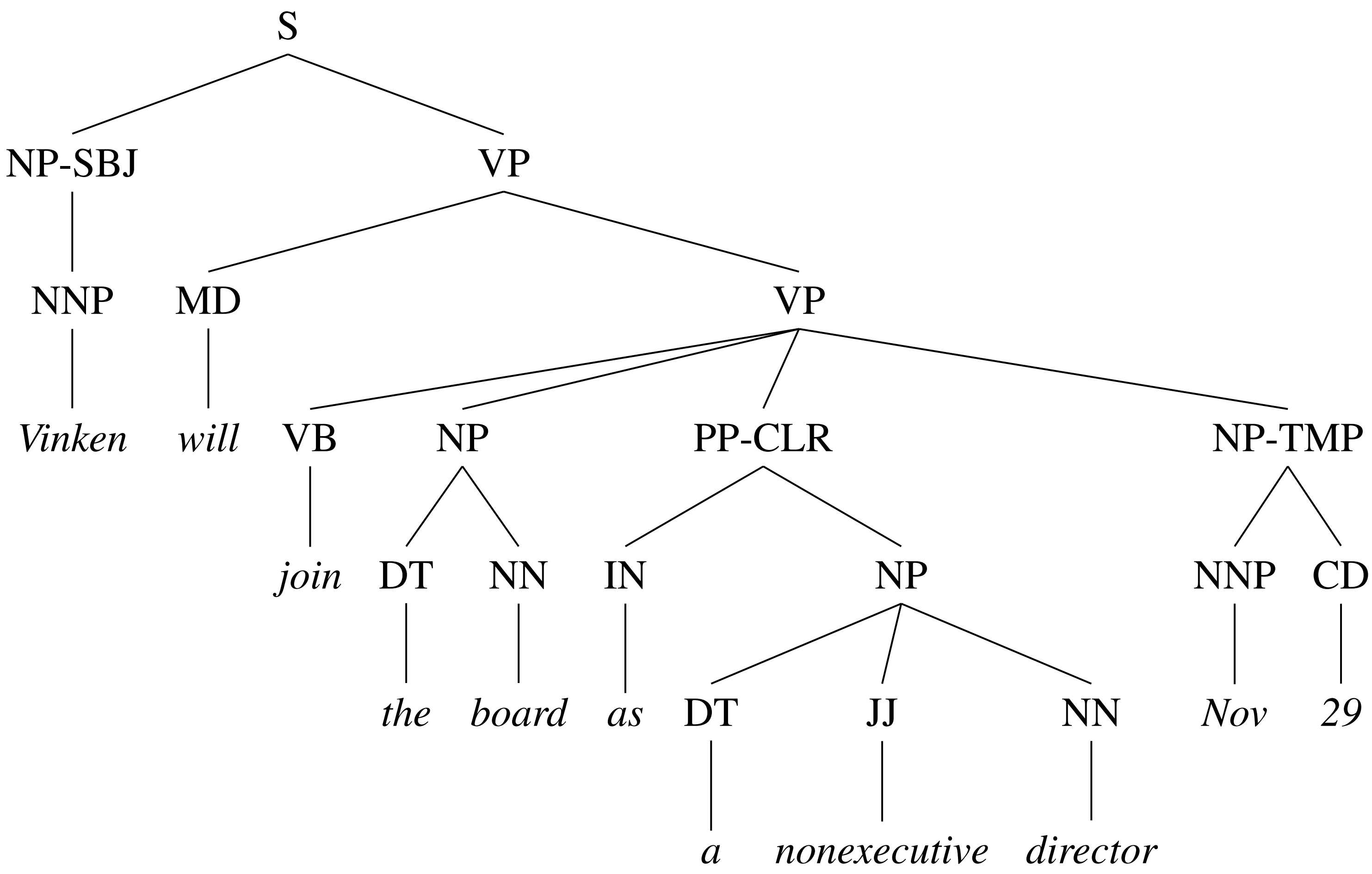


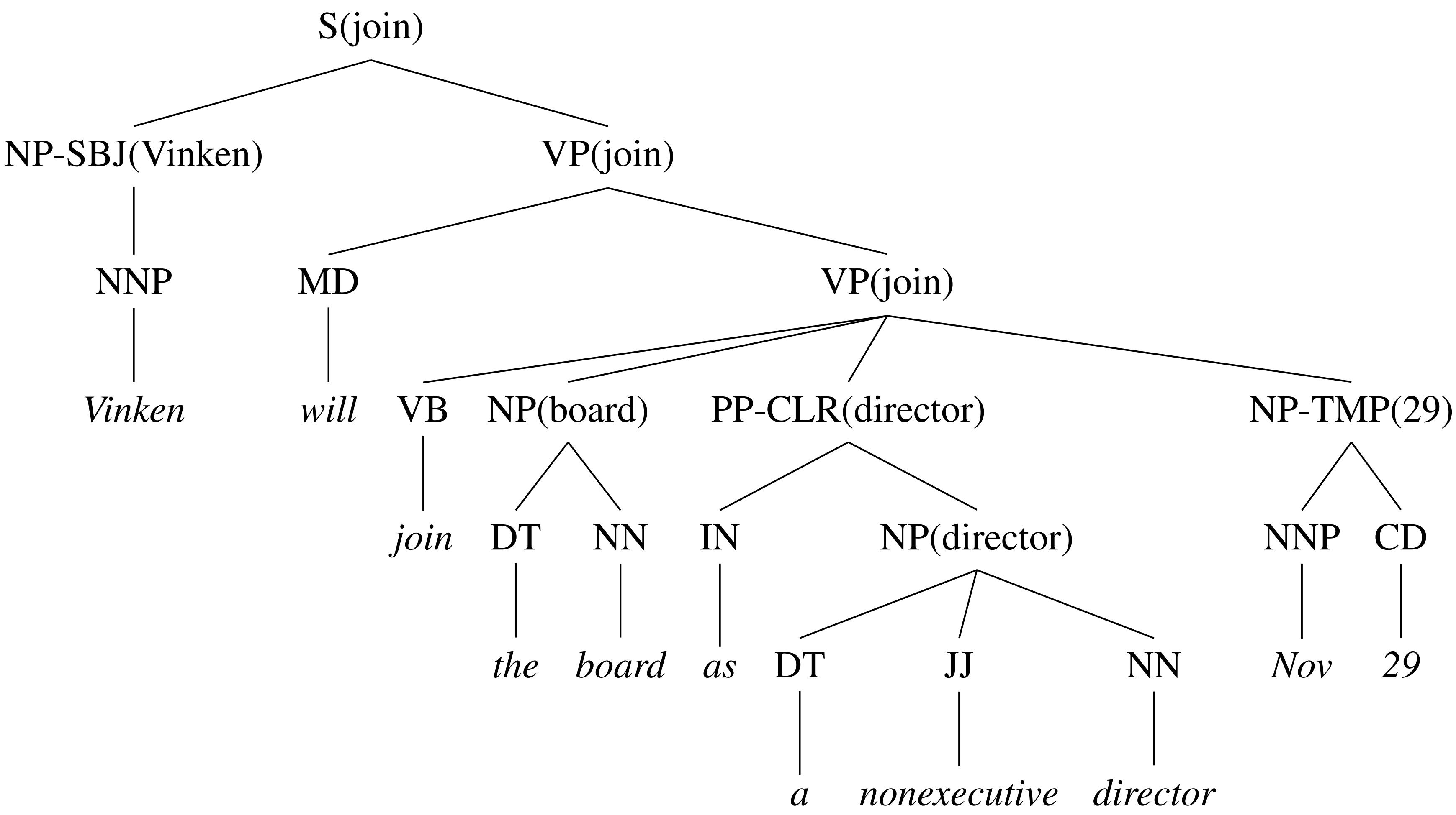
Dependency Treebanks

Dependency Treebanks are typically created by the following methods:

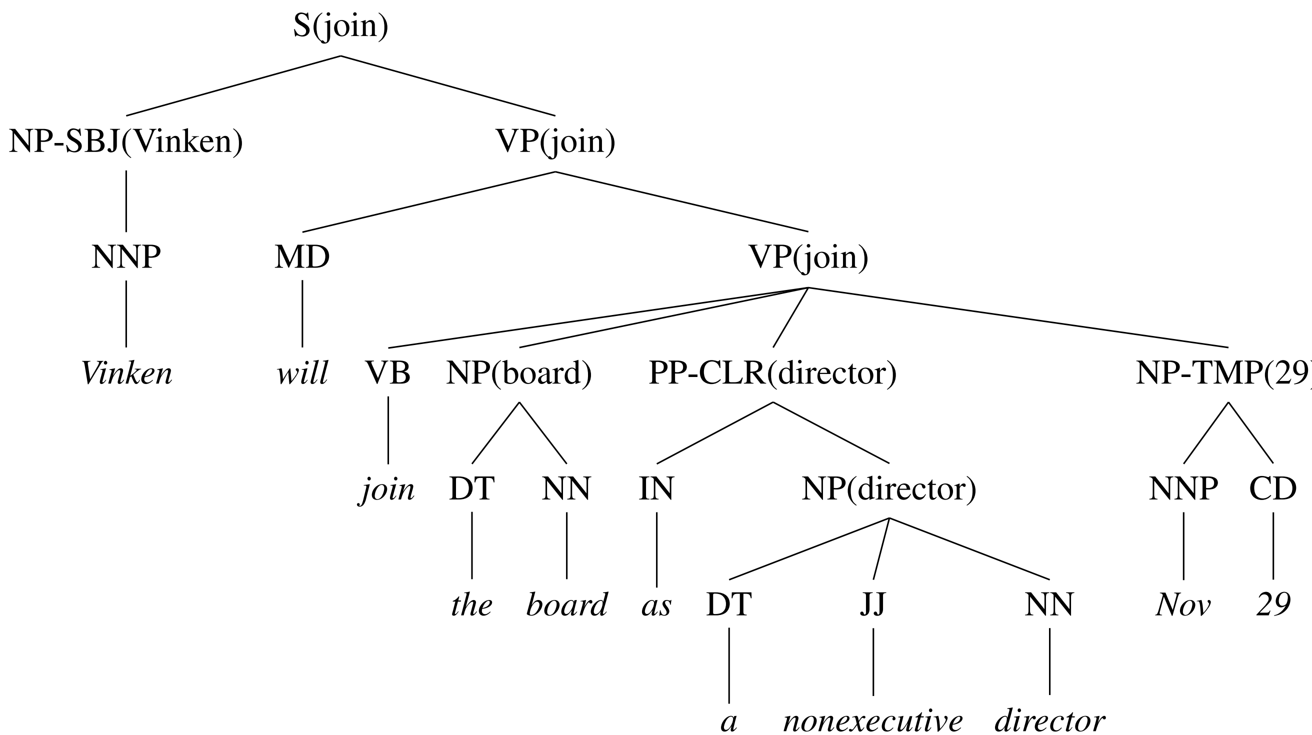
1. Having human annotators build dependency structures **directly**
2. Using an **automatic parser** and then employing human annotators to correct the output
3. Automatically **transforming phrase-structure treebanks** into dependency structure treebanks

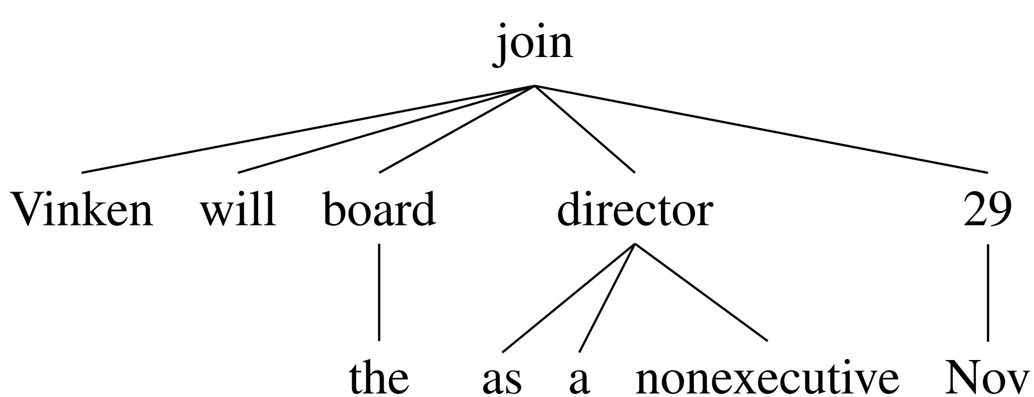
Directly annotated dependency treebanks have been often created for **morphologically rich languages** such as Czech (Prague Dependency Treebank), Hindi and Finnish.

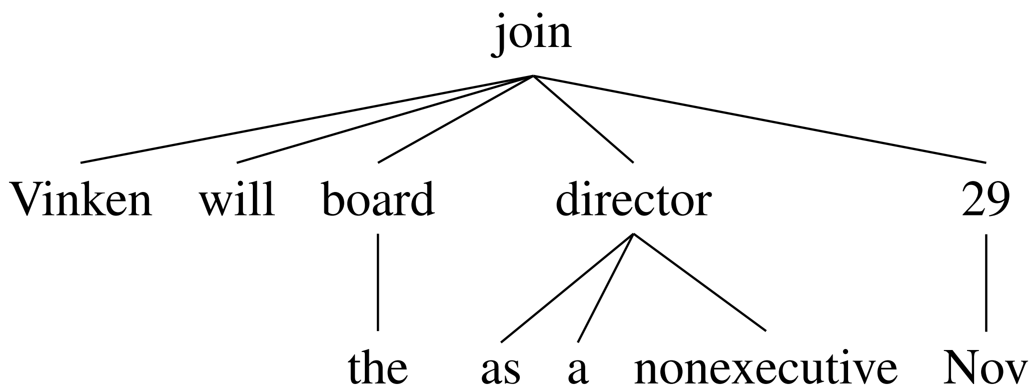

S
NP-SBJ
VP
NNP
MD
VP
Vinken
will
VB
NP
PP-CLR
NP-TMP
join
DT
NN
IN
NP
NNP
CD
the
board
as
DT
JJ
NN
Nov
29
a
nonexecutive
director

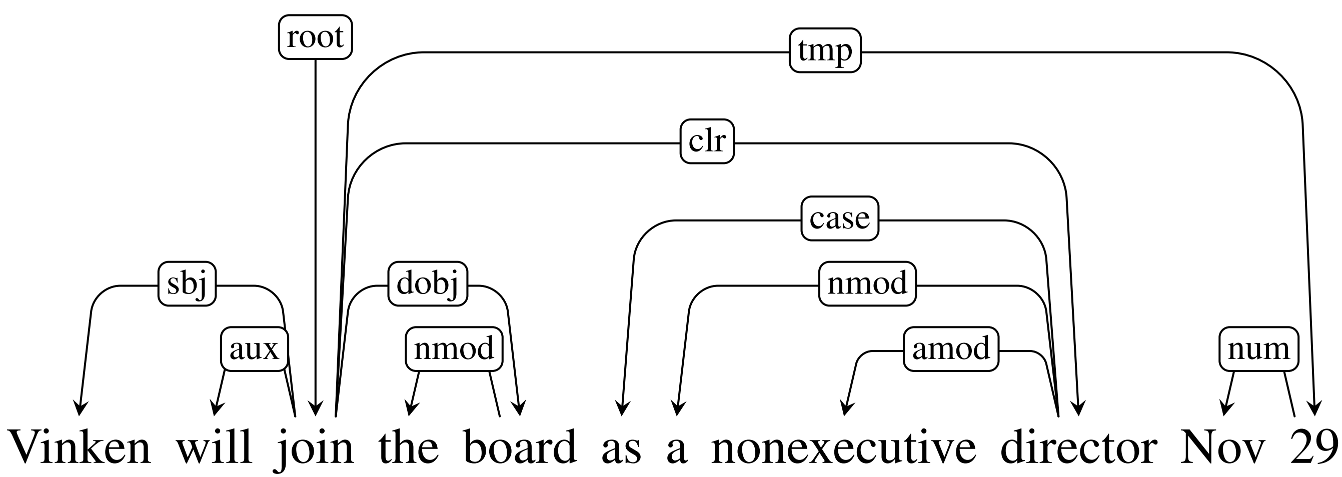


S(join)
NP-SBJ(Vinken)
VP(join)
NNP
MD
VP(join)
Vinken
will
VB
NP(board)
PP-CLR(director)
NP-TMP(29)
join
DT
NN
IN
NP(director)
NNP
CD
the
board
as
DT
JJ
NN
Nov
29
a
nonexecutive
director


S(join)
NP-SBJ(Vinken)
VP(join)
NNP
MD
VP(join)
Vinken
will
VB
NP(board)
PP-CLR(director)
NP-TMP(29)
join
DT
NN
IN
NP(director)
NNP
CD
the
board
as
DT
JJ
NN
Nov
29
a
nonexecutive
director


join
Vinken
will
board
director
29
the
as
a
nonexecutive
Nov


join
Vinken will board director 29
the as a nonexecutive Nov


root
tmp
clr
case
sbj
dobj
nmod
aux
nmod
amod
num
Vinken will join the board as a nonexecutive director Nov 29

Parsing Methods

There are two main approaches used in dependency parsers:

1. **Transition-Based**
2. **Graph-Based**

Transition-based approaches can only produce projective trees. Therefore any sentences with non-projective structures will contain errors.

In contrast, graph-based parsing approaches can handle non-projectivity but are more computationally expensive.

Transition-based Parsing

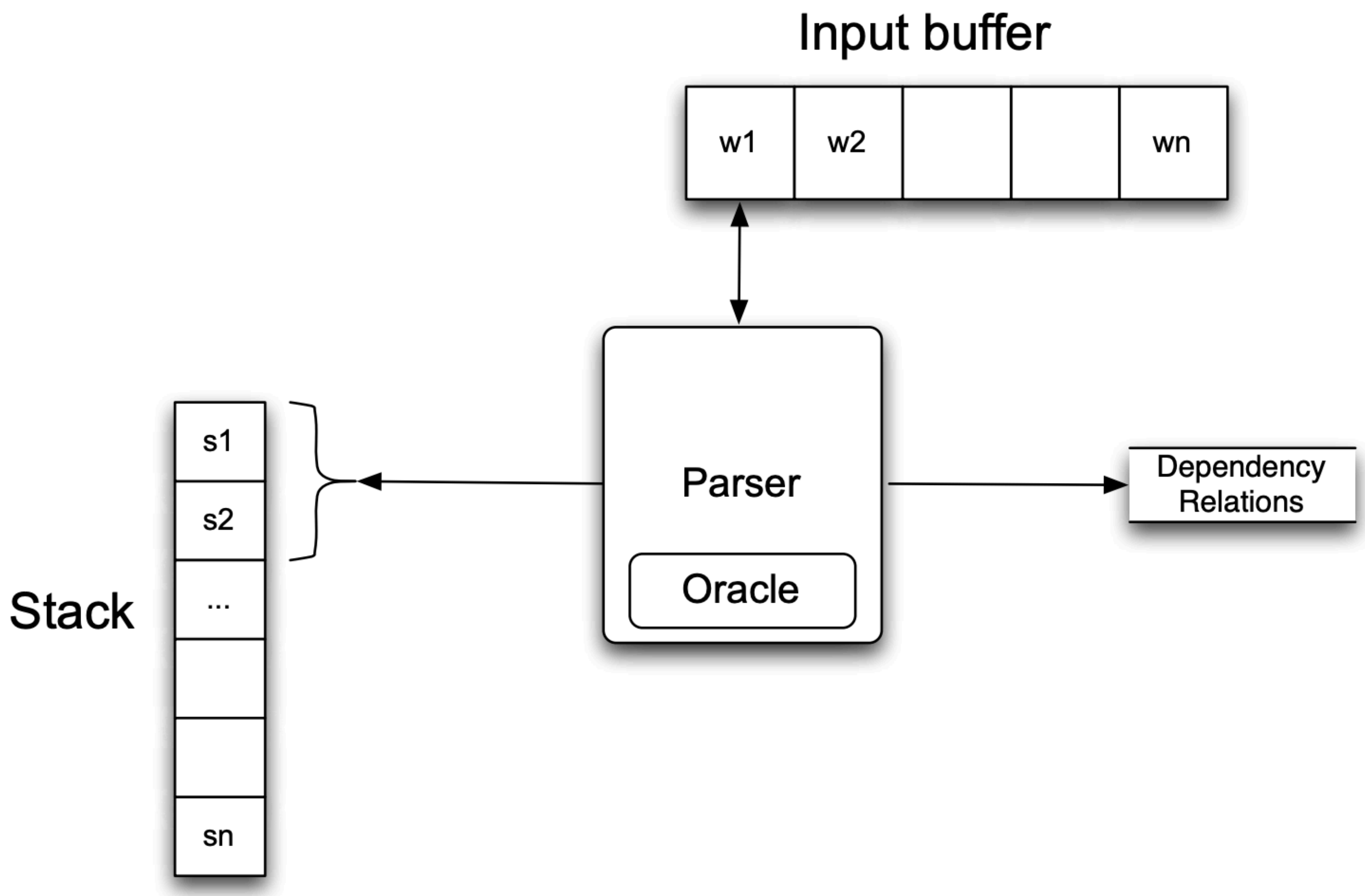
Transition-based parsing systems employ a **greedy stack-based** algorithm to create dependency structures.

A key element in transition-based parsing is the notion of a **configuration** which consists of a **stack**, an **input buffer of words** and a **set of relations representing the dependency tree.**

Parsing consists of a **sequence of "shift-reduce" transitions.** Once all the words have been moved off the stack, they have each and been assigned a head (and an appropriate relation).

The resulting configuration is a **dependency tree**.

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.

Transition-based Parser

Intuition: create a dependency tree by examining the words in a single pass over the input, moving from left to right:

- Assign the current word as the head of some previously seen word,
- Assign some previously seen word as the head of the current word,
- Or postpone doing anything with the current word, adding it to the stack so that it can be processed later.

Transition-based 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*

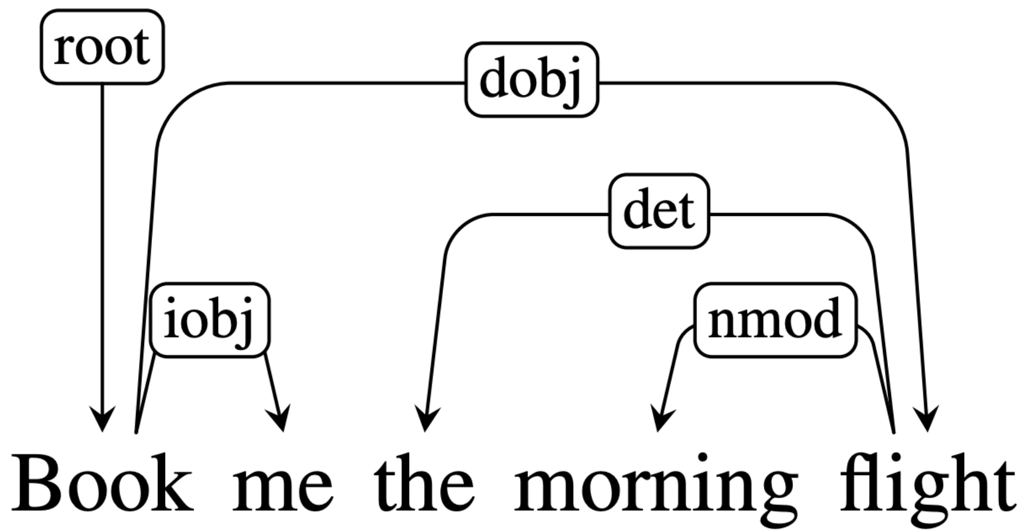
Complexity is **linear in the length of the sentence O(V)** since it is based on a single left to right pass through the words in the sentence → each word must be first shifted onto the stack and then reduced

Operators

There are three **transition operators** that will operate on the top two elements of the stack:

1. **LEFTARC**: Assert a head-dependent relation between the word at the top of the stack and the word directly beneath it; remove the lower word from the stack.
2. **RIGHTARC**: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
3. **SHIFT**: Remove the word from the front of the input buffer and push it onto the stack.

Worked example:



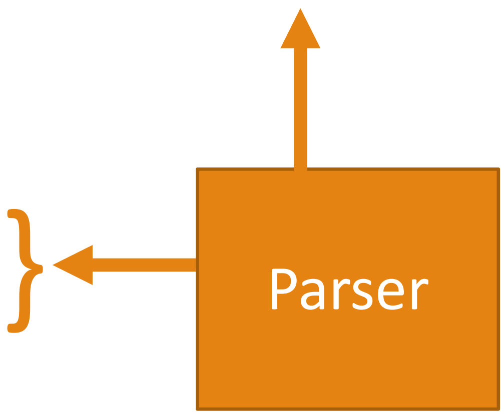
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book $\rightarrow$ me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning $\leftarrow$ flight)
7	[root, book, the, flight]	[]	LEFTARC	(the $\leftarrow$ flight)
8	[root, book, flight]	[]	RIGHTARC	(book $\rightarrow$ flight)
9	[root, book]	[]	RIGHTARC	(root $\rightarrow$ book)
10	[root]	[]	Done	

Figure 15.7 Trace of a transition-based parse.

input buffer:

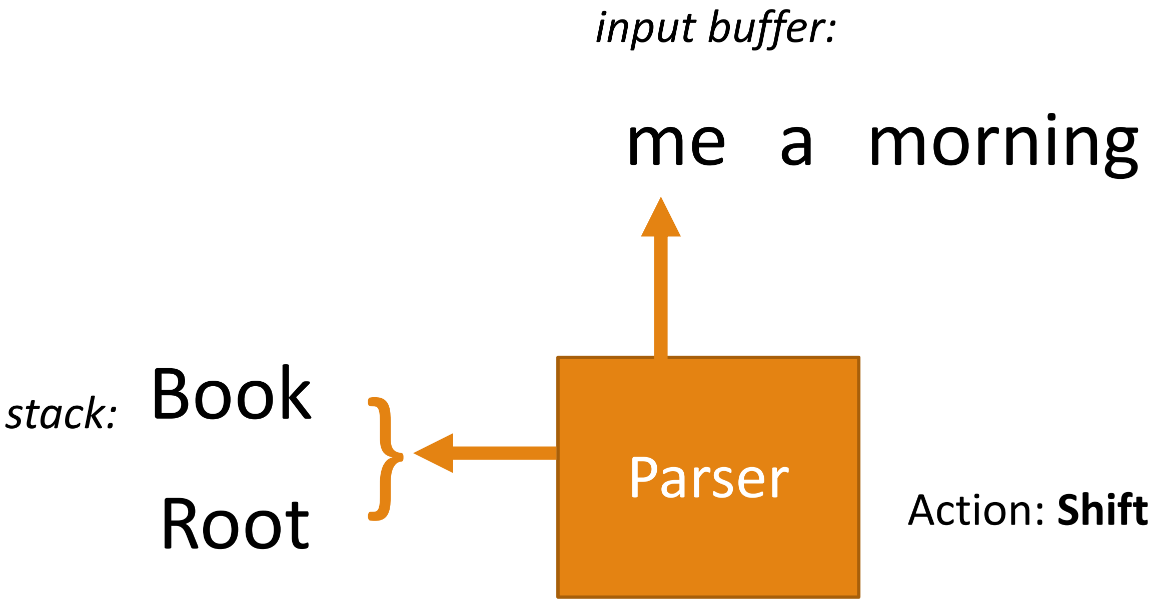
Book me a morning flight

stack: Root

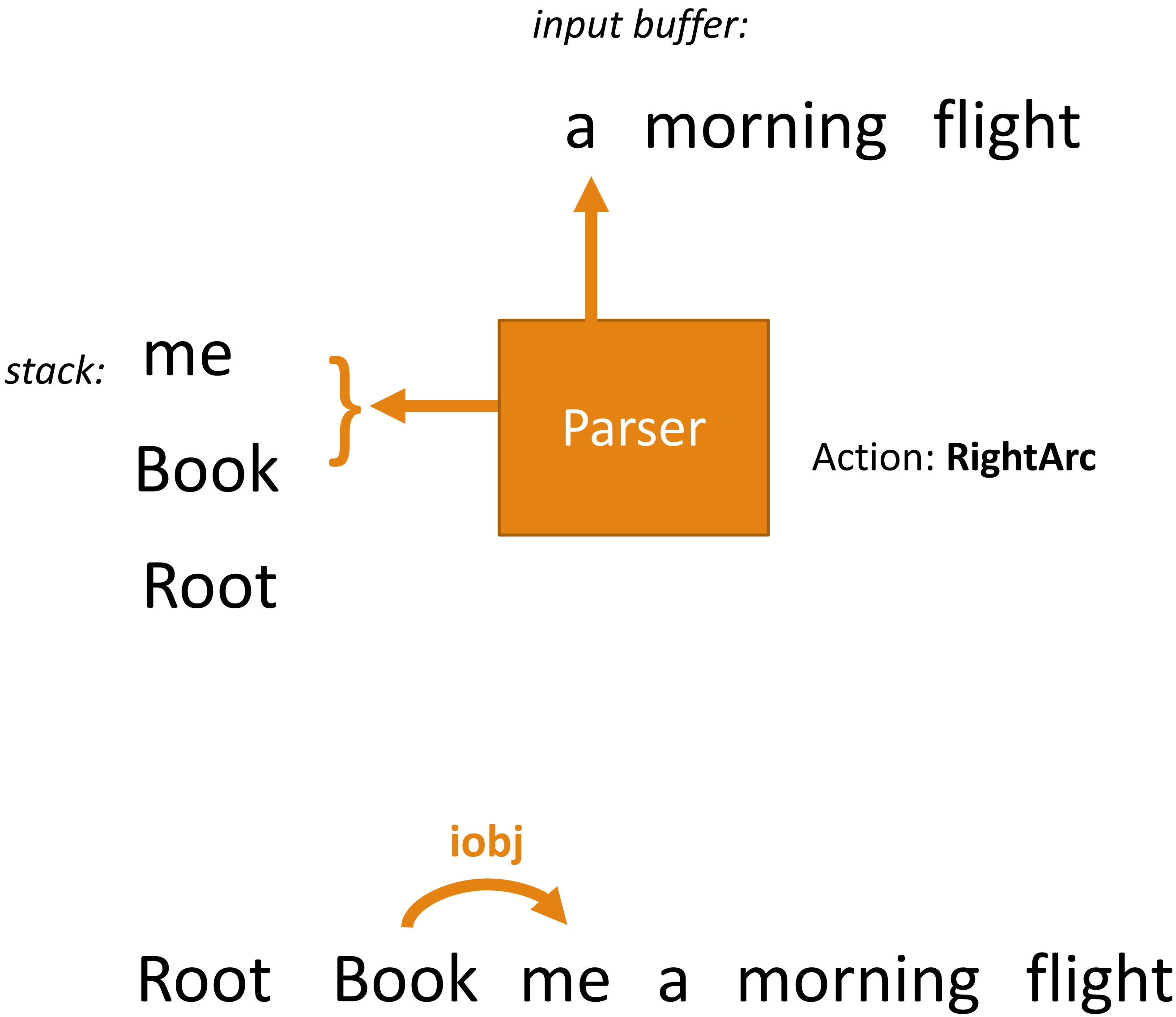


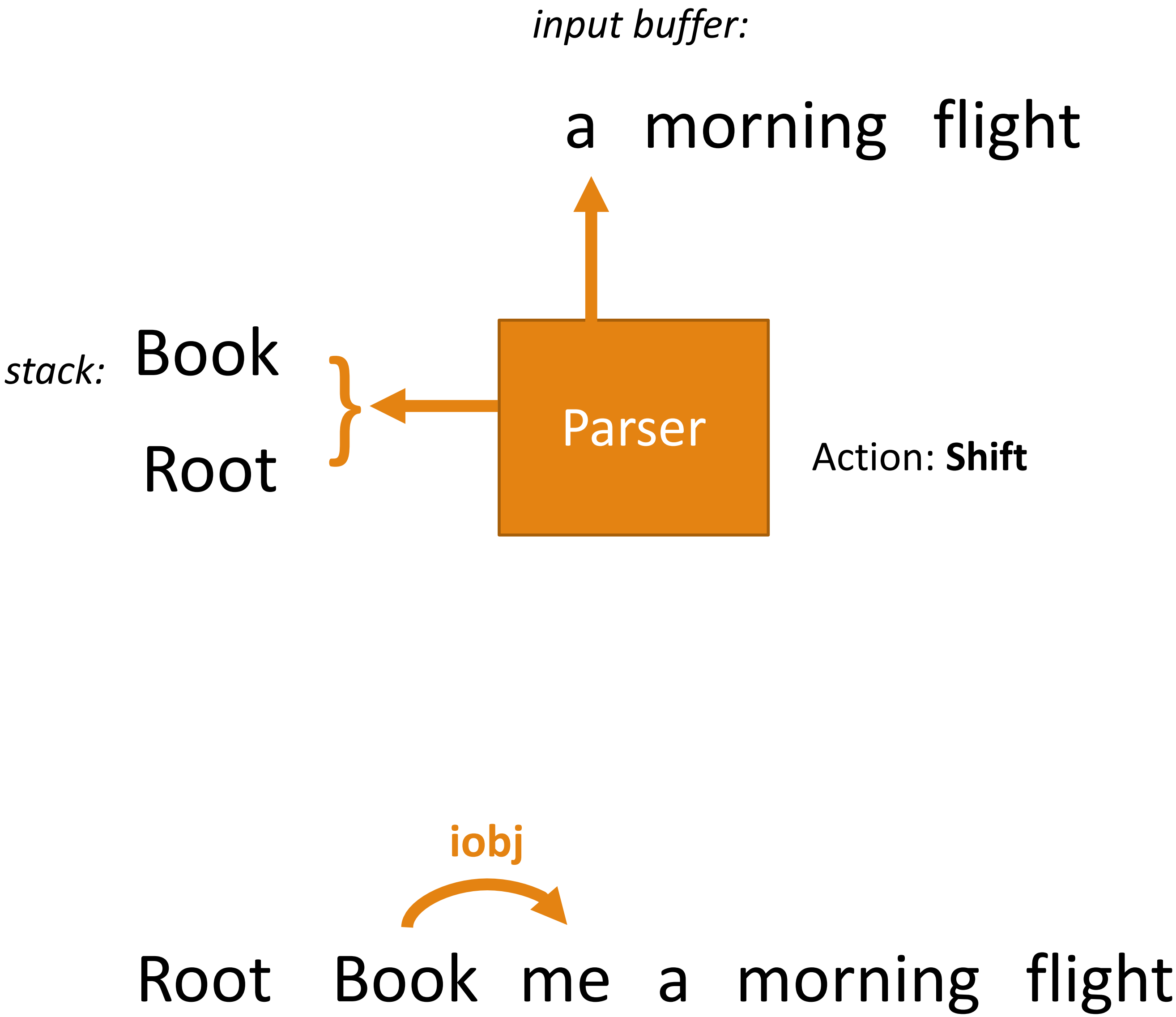
Action: **Shift**

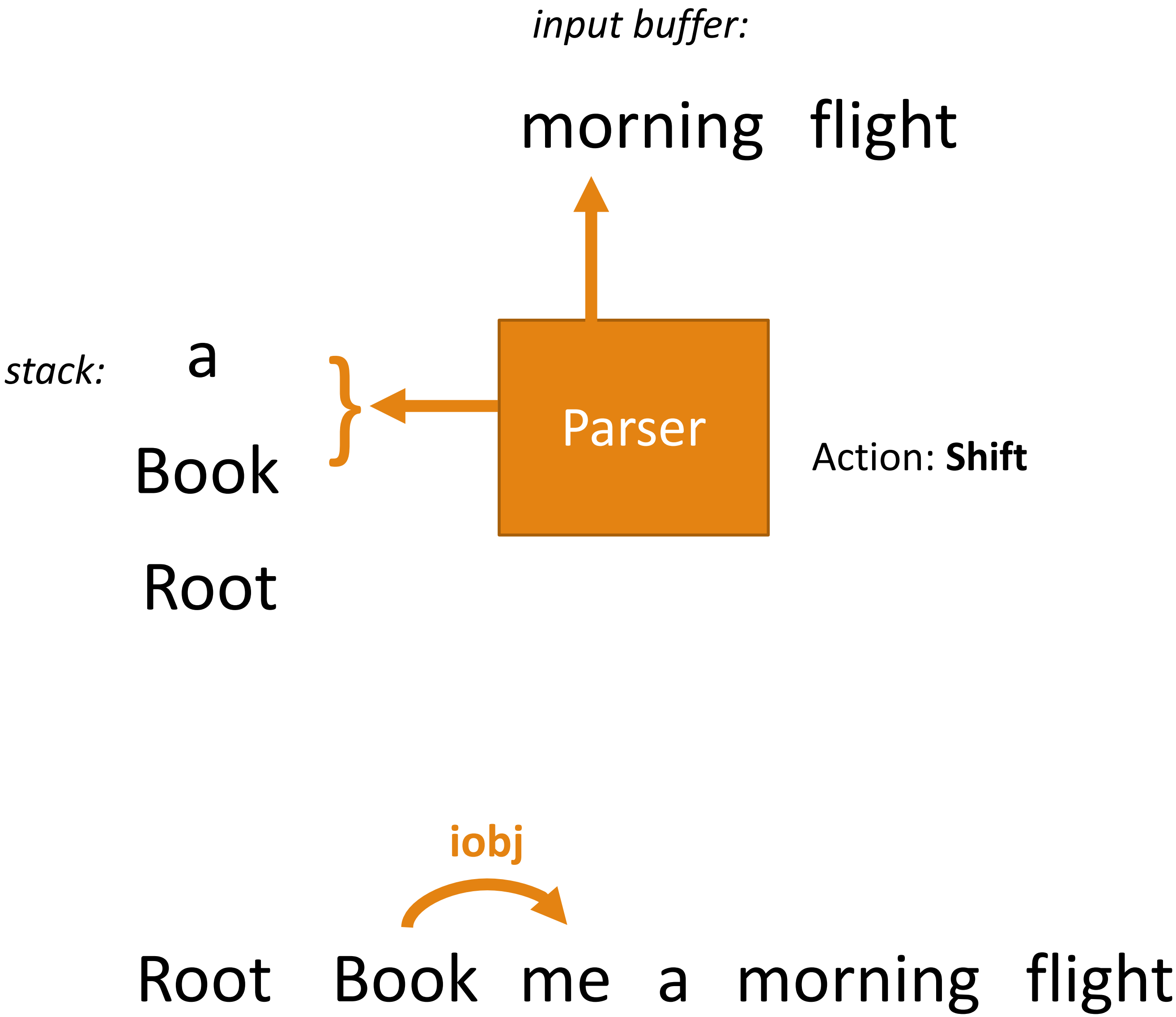
Root Book me a morning flight



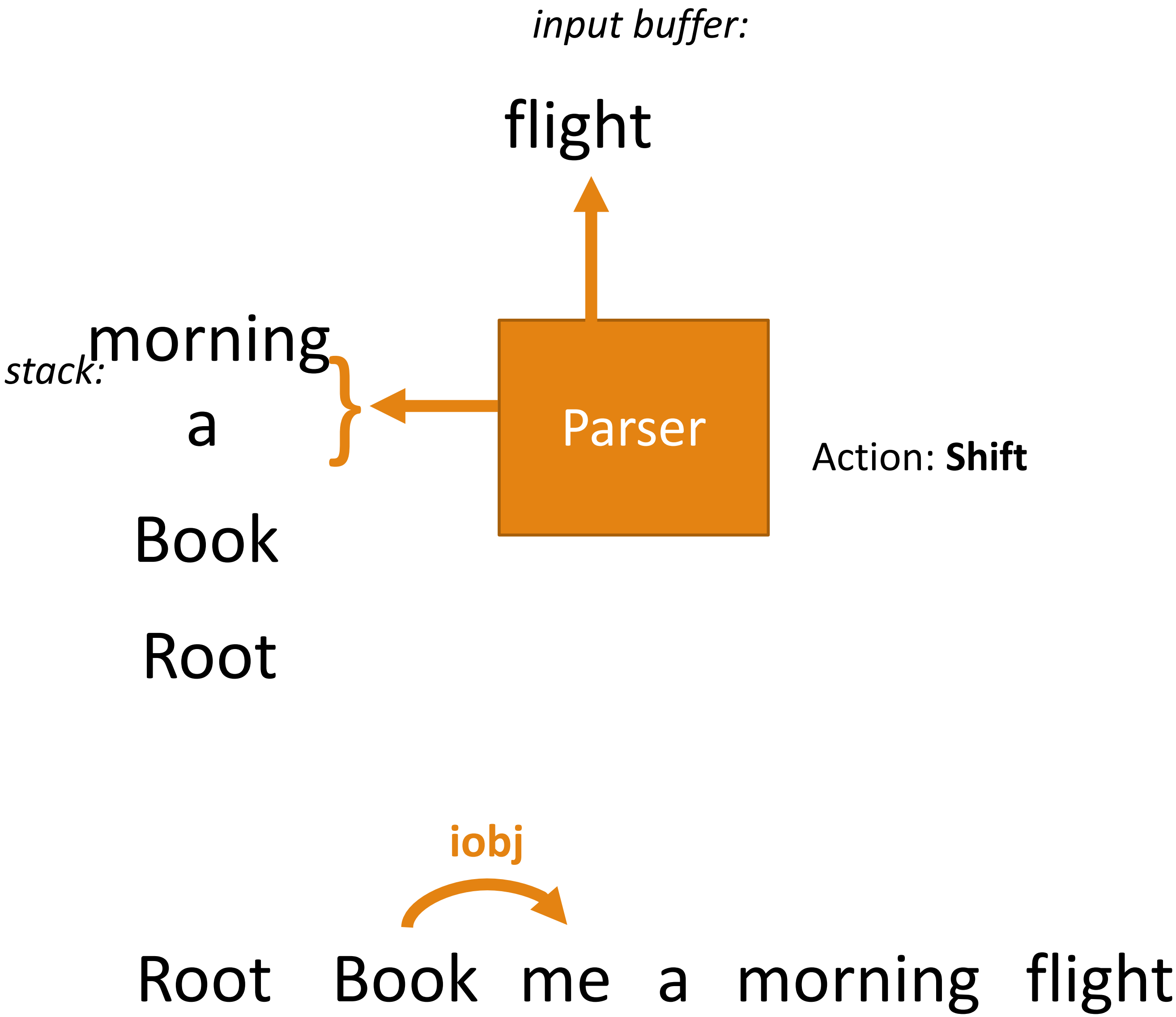
Root Book me a morning flight


input buffer:
a morning flight
stack:
me
Book
Root
Parser
Action: **RightArc**
iobj
Root Book me a morning flight


input buffer:
a morning flight
stack:
Book
Root
Parser
Action: **Shift**
iobj
Root Book me a morning flight



input buffer:
morning flight
stack:
a
Book
Root
Parser
Action: **Shift**
iobj
Root Book me a morning flight



input buffer:
flight
stack:
morning
a
Book
Root
Parser
Action: **Shift**
iobj
Root Book me a morning flight

input buffer:

stack: flight

morning

a

Book

Root

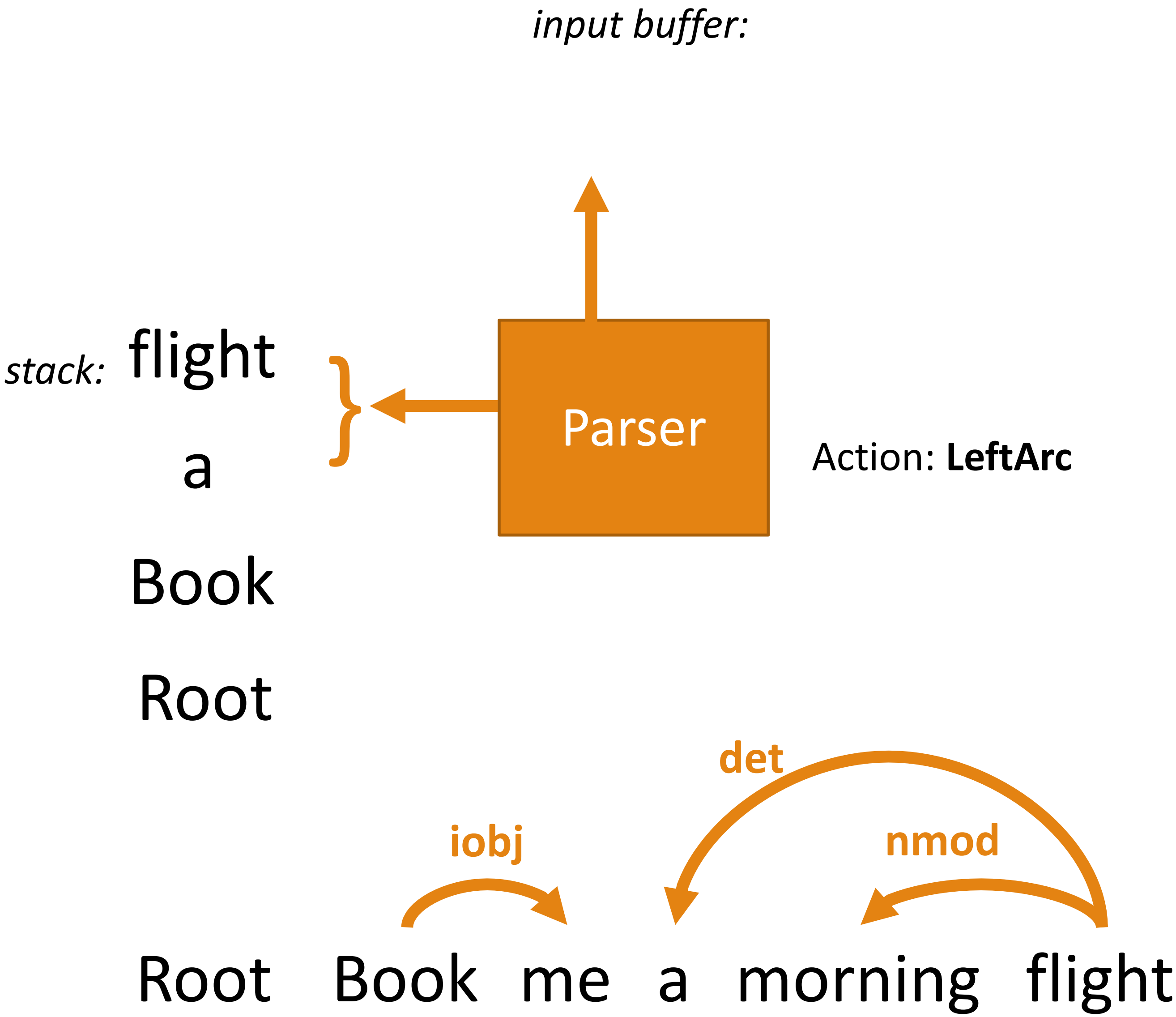
Parser

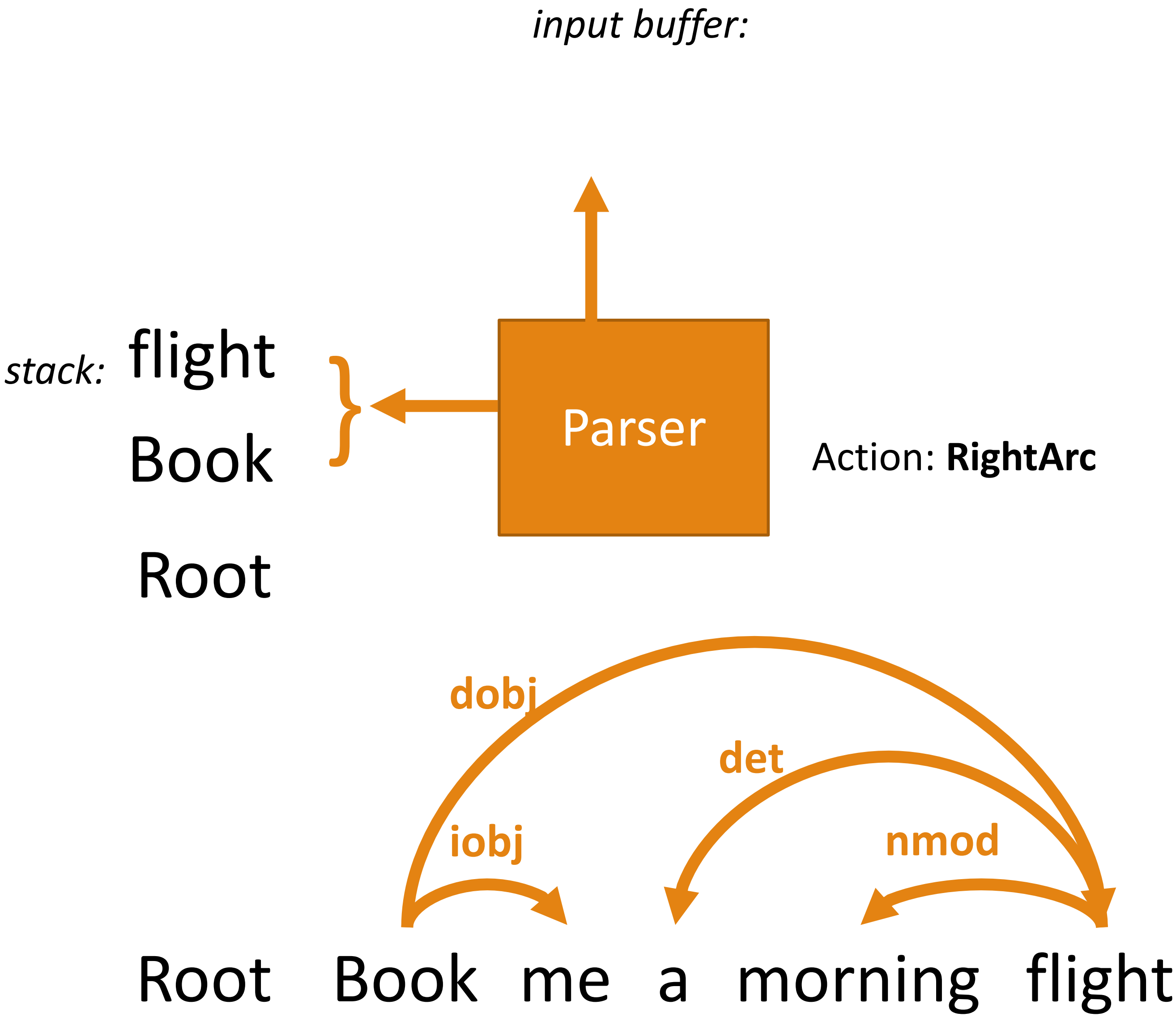
Action: **LeftArc**

iobj

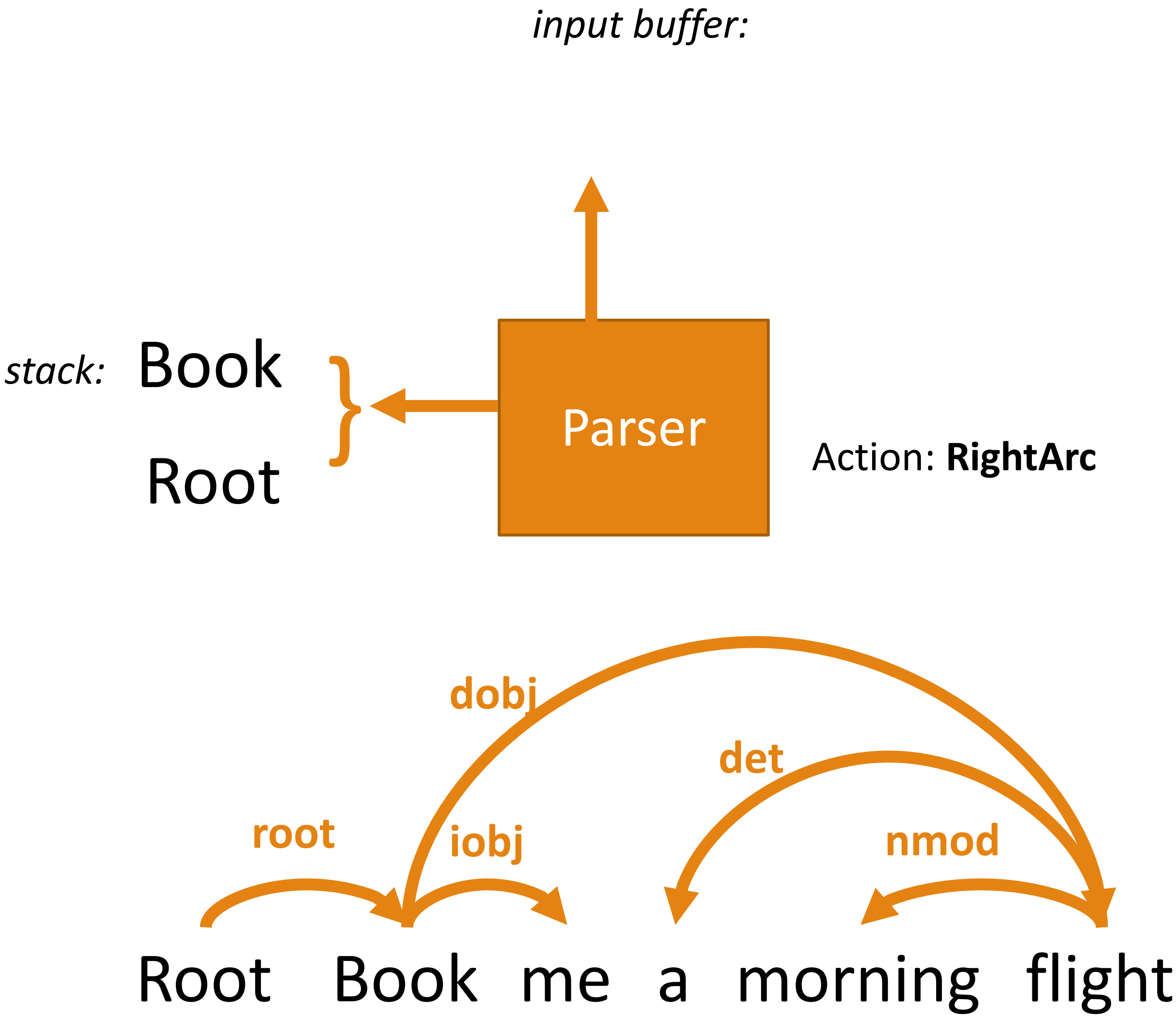
nmod

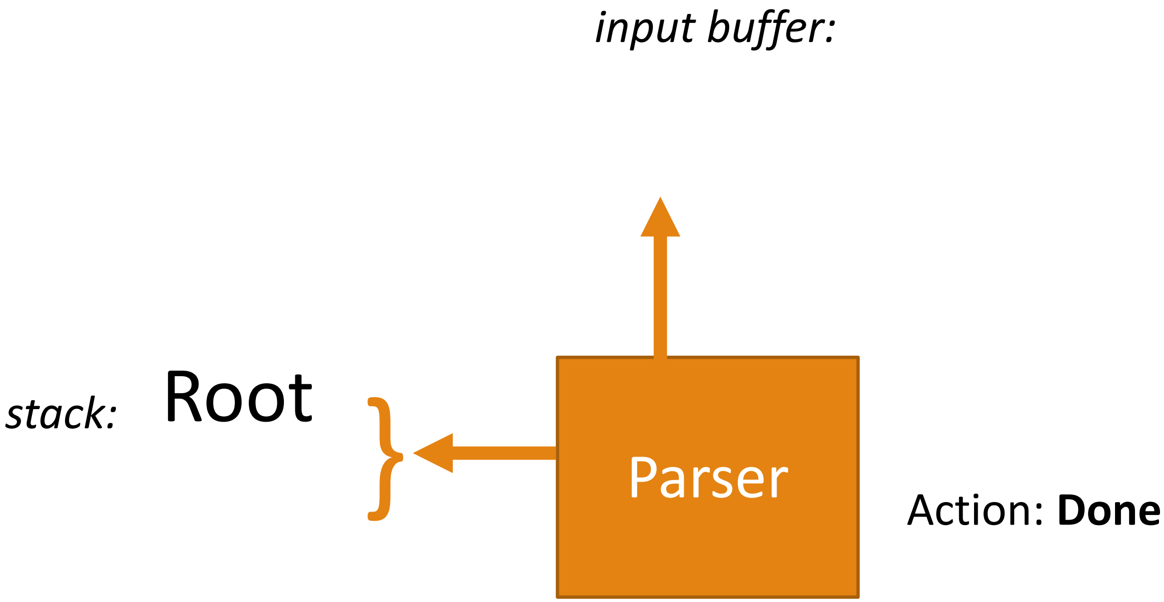
Root Book me a morning flight

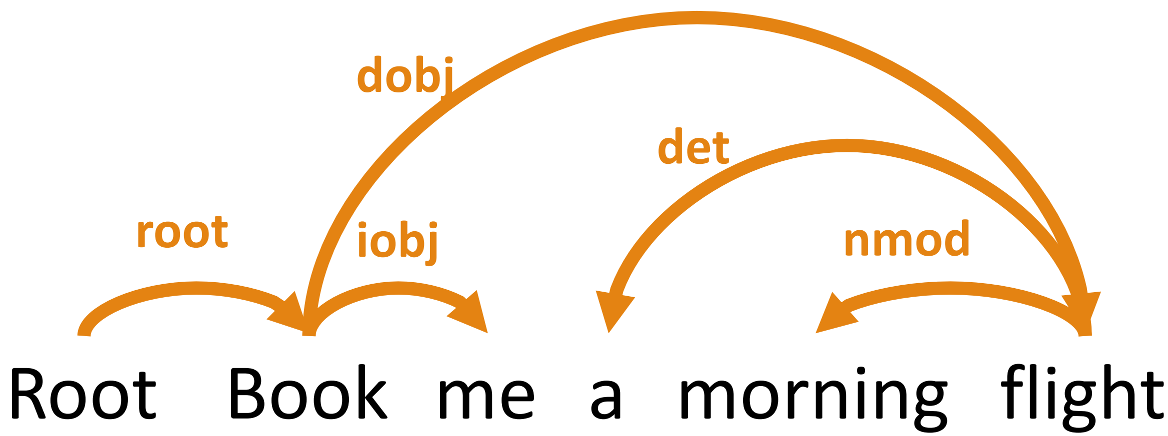

input buffer:
stack:
flight
a
Book
Root
Parser
Action: **LeftArc**
det
iobj
nmod
Root Book me a morning flight



input buffer:
stack:
flight
Book
Root
Parser
Action: **RightArc**
dobj
det
iobj
nmod
Root Book me a morning flight


input buffer:
stack:
Book
Root
Parser
Action: **RightArc**
dobj
det
root
iobj
nmod
Root Book me a morning flight


input buffer:
stack:
Root
Parser
Action: **Done**


dobj
det
root
iobj
nmod
Root Book me a morning flight

Creating the Oracle

SOTA transition-based systems use supervised machine learning methods to train classifiers that play the role of the **oracle,** which takes in as input a configuration and returns as output a transition operator.

Problem: What about the training data? To train the oracle, we need configurations paired with transition operators, which aren't provided by the Treebanks…

Solution: simulate the operation of the parser by running the algorithm and relying on a new training oracle to give correct transition operators for each successive operation.

Graph-based Parsing

Graph-based methods for creating dependency structures search through the space of possible dependency trees for a tree that maximizes some score function:

$$\hat{T}\,(S) = argmax_{t\,\in\,G_S}\, score\,(t,S)$$

where, the score for a tree is based on the scores of the edges that comprise the tree:

$$score\,(t,S) = \sum_{e\,\in t} score\,(e)$$

A common approach involves the use of **maximum spanning trees (MST)**

```
function MaxSpanningTree(G=(V,E), root, score) returns spanning tree

    F ← []
    T' ← []
    score' ← []
    for each v ∈ V do
       bestInEdge ← argmax_{e=(u,v)∈ E} score[e]
       F ← F ∪ bestInEdge
       for each e=(u,v) ∈ E do
           score'[e] ← score[e] − score[bestInEdge]

       if T=(V,F) is a spanning tree then return it
       else
          C ← a cycle in F
          G' ← Contract(G, C)
          T' ← MaxSpanningTree(G', root, score')
          T ← Expand(T', C)
          return T

function Contract(G, C) returns contracted graph

function Expand(T, C) returns expanded graph
```

Figure 15.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

Training

While we can reduce the score of tree to a sum of the scores of the edges that comprise it, each edge score can also be reduced to a **weighted sum of features extracted from it.**

$$score\ (S, e) = \sum_{i=1}^{N} w_i f_i\ (S, e) = w\ \cdot f$$

Commonly used features include:

- **Wordforms, lemmas and POS of the headword and dependent**
- **Corresponding features of contexts before, after and between words**
- **Word embeddings**
- **Dependency relation type**
- **Direction of the relation (to the right or to the left)**
- **Distance from the head to the dependent**

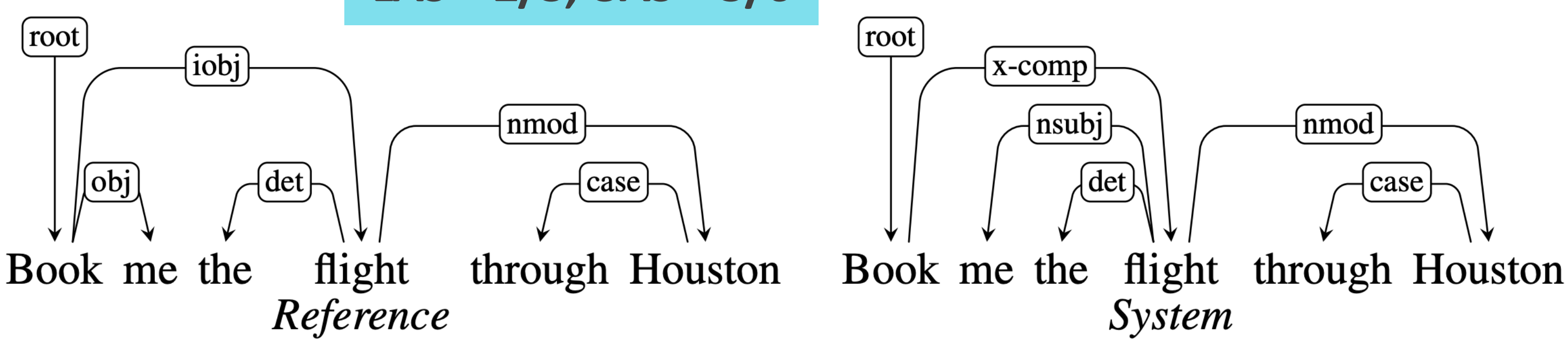
Evaluation

The common method for evaluating dependency parsers are **labeled attachment accuracy (LAS) and unlabeled attachment accuracy**

Labeled attachment refers to the proper assignment of a word to its head with the correct dependency relation.

Unlabeled attachment refers to the proper assignment of a word to its head ONLY (ignores dependency relation)

LAS = 2/3, UAS = 5/6



Next time: Logical Representations of Sentence Meaning