Module 3

Ambiguities in Syntax Analysis – Issues with CFG based Parsing – Shallow Parsing – Conditional Random Fields – Dependency Grammar – Dependency Parsing – Neural Network Dependency Parsing

Structure of Sentences

- Basic units – words (the – determiner, cat – Noun, cuddly – adjective)
- Words combine to produce – phrases (the cuddly cat – Noun Phrase, by the door – Prepositional Phrase(preposition[Noun Phrase])
- Phrases combine together – Bigger phrases (the cuddly cat by the door)
- Dependency Structure – Shows which words depend on (modify, attach to, are arguments of) which other words.

Why need Structure?

- Humans communicate complex ideas by composing words together
- Listeners work out what modifies[attaches to] what ?
- A model needs to understand sentence structure in order to be able to interpret language correctly

Syntactic Parsing

- Basic techniques for grammar-driven natural language parsing.
- Analyzing a string of words (typically a sentence) to determine its structural description according to a formal grammar.
- Parsing is the process of taking a string and a grammar and returning parse tree(s) for that string.
- Syntactic Parsing - Automatic methods of finding the syntactic structure for a sentence –
  - Symbolic methods: A phrase grammar or another description of the structure of language is required. (The chart parser).
  - Statistical methods: A text corpus with syntactic structures is needed (a treebank - A corpus in which every sentence is annotated with a parse tree)

Challenges in Syntactic Parsing

Challenges in Syntactic Parsing

- **Extreme structural ambiguity**
- **Long-distance dependencies**
- **Parser Failure**

Challenges in Syntactic Parsing

- **Long-distance dependencies** as a result of Strong Generative Capacity, in English wh-questions:
  - Who did you sell the car to?
  - Who do you think that you sold the car to?
  - Who do you think that he suspects that you sold the car to?
- **Extreme structural ambiguity** of natural language.

  Put the block in the box on the table
  - Put the block [in the box on the table]
  - Put [the block in the box] on the table
  - The process of just returning all the possible analyses would lead to a combinatorial explosion.
  - How the most appropriate analysis can be selected (**disambiguation**).
- **Parser Failure**
  - Failure to produce a parsing result is due to an error in the input or to the lack of coverage of the grammar

Syntactic / Structural Ambiguities

Structural ambiguity occurs when the grammar can assign more than one parse to a sentence.

- Prepositional Phrase(PP) Attachment Ambiguity
- Verb Phrase (VP) Attachment Ambiguity
- Coordination Scope Ambiguity
- Adjectival / Adverbial modifier Ambiguity

Syntactic Ambiguities

- Prepositional Phrase Attachment Ambiguity

  San Jose cops kill man with a knife

  - **[San Jose cops kill] [man with a knife]**
  - [San Jose cops kill man] [with a knife]

  He lifted the beetle with the red cap

  - **He lifted the [beetle with the red cap]**
  - He [lifted the beetle] with the red cap

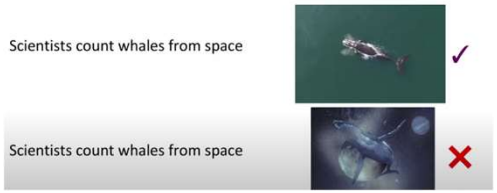


FIGURE: beetle with red cap

FIGURE: beetle with red cap

Syntactic Ambiguities

Verb Phrase (VP) Attachment Ambiguity

- Mutilated body washes up on Rio Beach to be used for Olympics beach volleyball
- Mutilated body washes up on Rio Beach [to be used for Olympics beach volleyball]

Syntactic Ambiguities

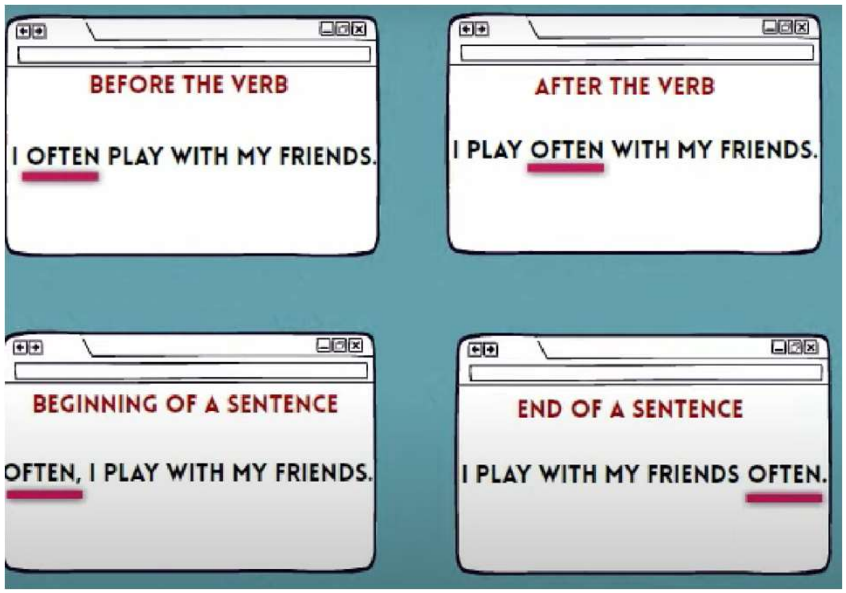
- Coordination Scope Ambiguity
  - [Shuttle Veteran] and [longtime NASA executive Fred Gregory] appointed to board.
  - [Shuttle Veteran and longtime NASA executive] Fred Gregory appointed to board.

Syntactic Ambiguities

- Adjectival / Adverbial modifier Ambiguity

  Students get first hand job experience

  - Students get [first hand][job experience]
  - Students get first [hand job] experience



Syntactic Ambiguities (Structural Ambiguity)

- The presence of two or more possible meanings within a single sentence or sequence of words.
- Examples
  - The professor said on Monday he would give an exam
  - The burglar threatened the student with the knife
  - Visiting relatives can be boring

Grammar Formalisms

- Context Free Grammar
- Mildly Context-Sensitive Grammars
- Constraint-Based Formalisms
- Immediate Dominance/Linear Precedence
- Head Grammars
- Lexicalized Grammars
- Dependency Grammars
- Type-Theoretical Grammars

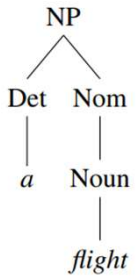
Context Free Grammar (CFG)

- Widely used formal system for modeling constituent structure in natural language is the context-free grammar, or CFG.
- Also referred to as Phrase Structured Grammar
- A context-free grammar consists of a set of rules or productions and the set of lexicons
  - Rules - The ways that symbols of the language can be grouped and ordered together
  - Lexicons – Words and symbols (Terminals and Non Terminals)
    - Terminals – The symbols that correspond to words(the, flight)
    - Non Terminals – The symbols that express abstractions over these terminals

Note : In each context-free rule, the item to the right of the arrow (→) is an ordered list of one or more terminals and non-terminals; to the left of the arrow is a single non-terminal symbol expressing some cluster or generalization

CFGs

- Sequence of rule expansions to generate set of words – Derivation
- Derivation is represented by parse tree
- A parse tree for “a flight”



- In the parse tree shown,
  - node NP **dominates** all nodes in the tree (Det, Nom, Noun, a, flight).
  - NP **immediately dominates** the nodes Det and Nom
- Language defined by CFG - The set of strings that are derivable from the designated **start symbol.**

Example - Grammar L_o

Noun →	*flights* | *flight* | *breeze* | *trip* | *morning*
Verb →	*is* | *prefer* | *like* | *need* | *want* | *fly* | *do*
Adjective →	*cheapest* | *non-stop* | *first* | *latest* | *other* | *direct*
Pronoun →	*me* | *I* | *you* | *it*
Proper-Noun →	*Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American*
Determiner →	*the* | *a* | *an* | *this* | *these* | *that*
Preposition →	*from* | *to* | *on* | *near* | *in*
Conjunction →	*and* | *or* | *but*

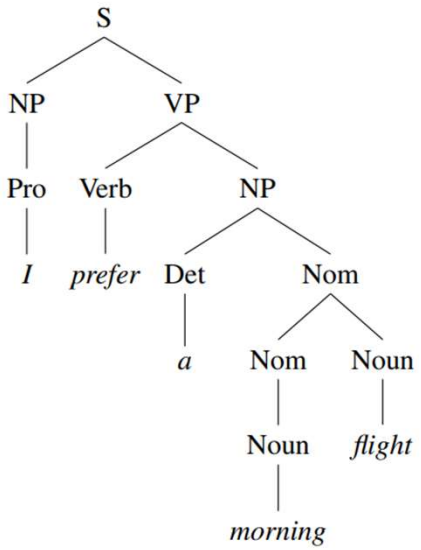
Figure : Sample Lexicon

Example - Grammar L_o

Grammar Rules			Examples
S	$\rightarrow$	*NP VP*	I + want a morning flight
NP	$\rightarrow$	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	$\rightarrow$	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	$\rightarrow$	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	$\rightarrow$	*Preposition NP*	from + Los Angeles

Figure - Production Rules

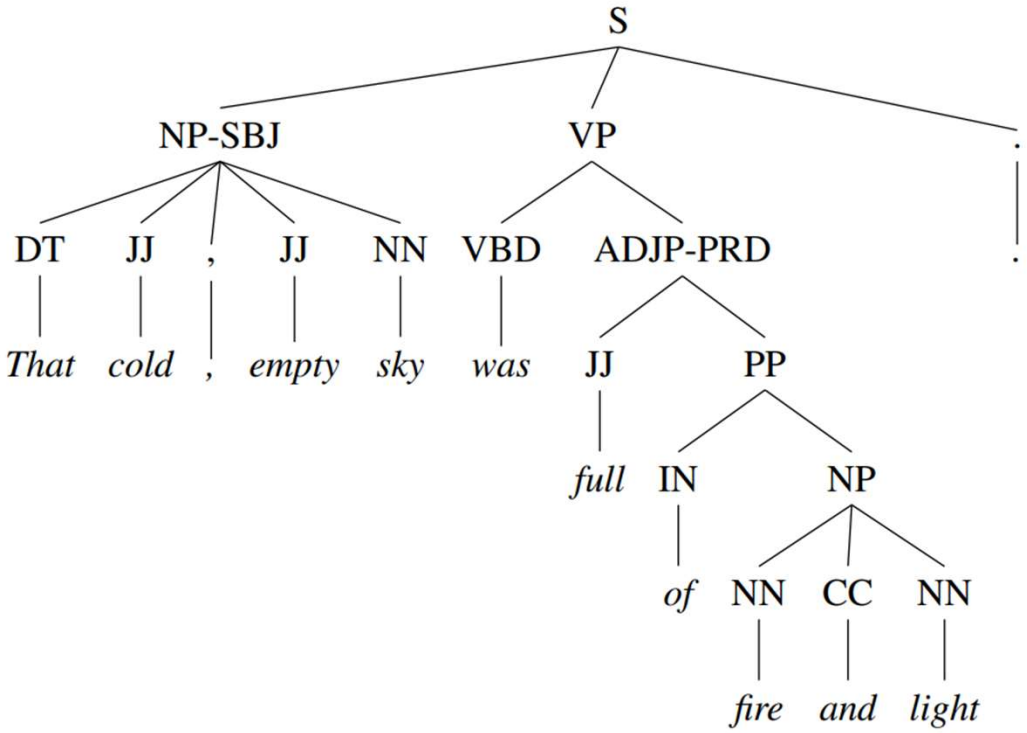
Example - Grammar L_0



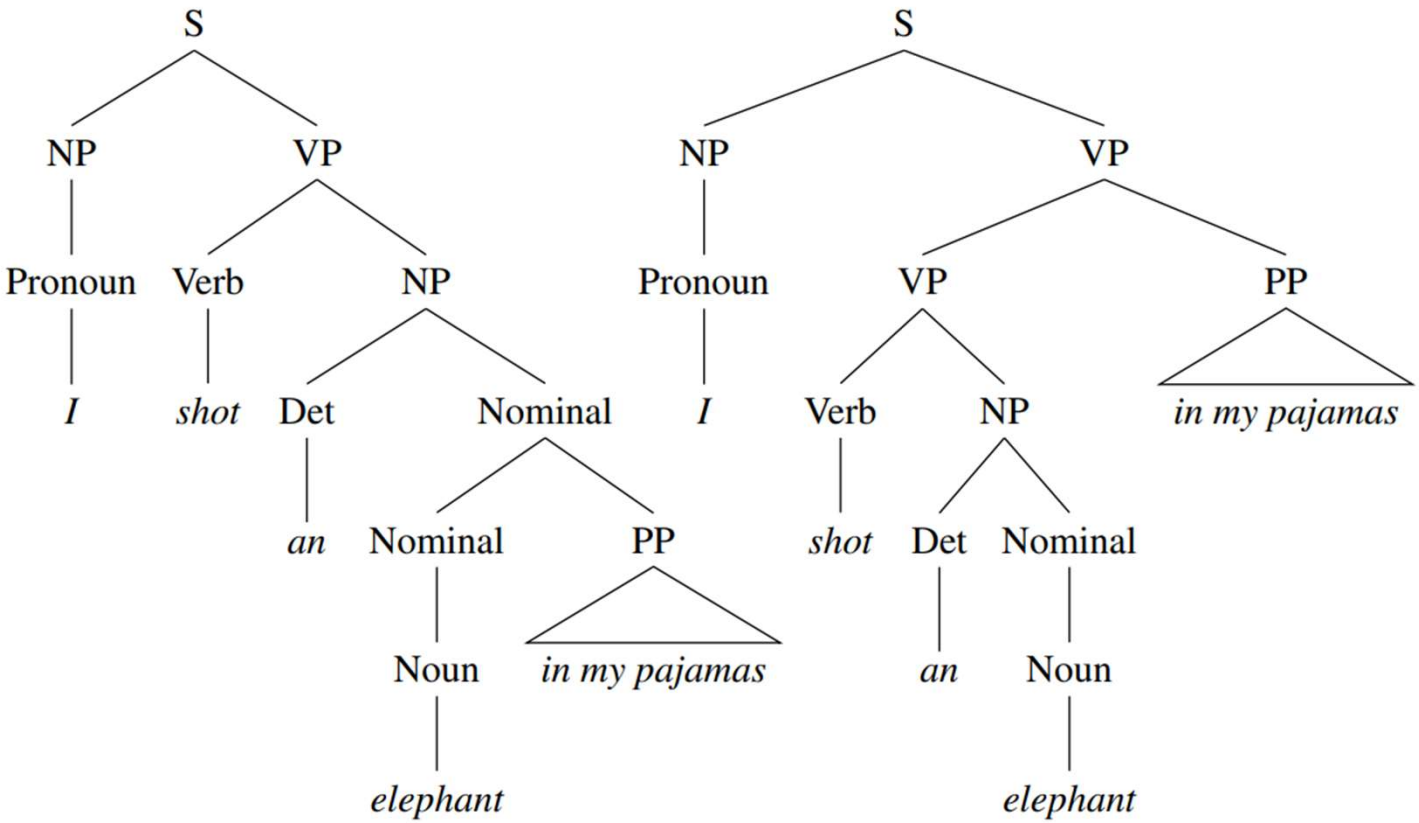
- The parse tree for “I prefer a morning flight” according to grammar L_0.

Parse Tree and Bracketed Notation

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```



Structural Ambiguity



Grammar Equivalence and Normal Form

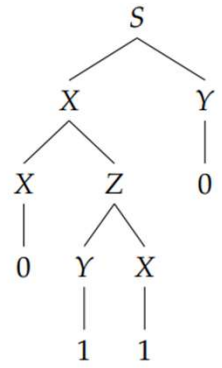
- Strongly Equivalent Grammars
  - Two grammars are strongly equivalent if they generate the same set of strings and if they assign the same phrase structure to each sentence.
- Weakly Equivalent Grammars
  - Two grammars are weakly equivalent if they generate the same set of strings but if they assign different phrase structure to each sentence.

Chomsky Normal Form(CNF)

- A CFG is said to be in its CNF if every rule of G has one of the following three forms

1. $X \rightarrow YZ$, for variables X, Y, and Z, and where neither Y nor Z is the start variable,
2. $X \rightarrow a$, for a variable X and a symbol a, or
3. $S \rightarrow \varepsilon$, for S the start variable.

- Advantages – Simple parse tree



Conversion from CFG to CNF

- Consider a CFG that generates balanced parantheses ,

$$S \to (S)S \mid \varepsilon$$

1. Add a new start variable S_0 along with the rule $S_0 \to S$.
2. Introduce a new variable X_a for each symbol $a \in \Sigma$.
3. Split up rules of the form $X \to Y_1 \cdots Y_m$, whenever $m \geq 3$, using auxiliary variables in a straightforward way.

   In particular, $X \to Y_1 \cdots Y_m$ can be broken up as

$$X \to Y_1 Z_2$$
$$Z_2 \to Y_2 Z_3$$

4. Eliminate ε-rules of the form $X \to \varepsilon$ and "repair the damage."
5. Eliminate unit rules, which are rules of the form $X \to Y$.

Steps for Conversion

- Step 1

$$S_0 \to S$$
$$S \to (S)S \mid \varepsilon$$

- Step 2

$$S_0 \to S$$
$$S \to LSRS \mid \varepsilon$$
$$L \to ($$
$$R \to)$$

- Step 3

$$S_0 \to S$$
$$S \to LZ_2 \mid \varepsilon$$
$$Z_2 \to SZ_3$$
$$Z_3 \to RS$$
$$L \to ($$
$$R \to)$$

- Step 4

$$S_0 \to S \mid \varepsilon$$
$$S \to LZ_2$$
$$Z_2 \to SZ_3 \mid Z_3$$
$$Z_3 \to RS \mid R$$
$$L \to ($$
$$R \to)$$

- Step 5

$$S_0 \to LZ_2 \mid \varepsilon$$
$$S \to LZ_2$$
$$Z_2 \to SZ_3 \mid RS \mid)$$
$$Z_3 \to RS \mid)$$
$$L \to ($$
$$R \to)$$

CFG based Parsing - CKY Parsing: A Dynamic Programming Approach

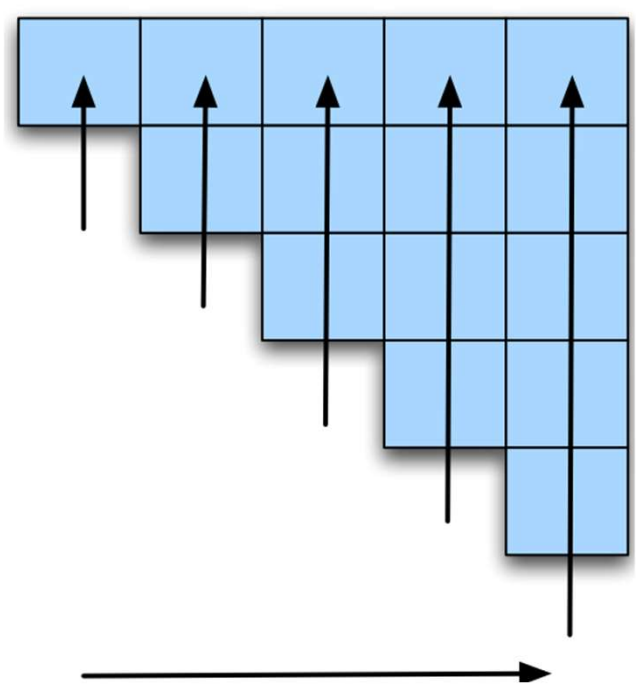
- Dynamic programming based parsing
- Cocke - Kasami Younger (CKY)
- A two-dimensional matrix is used to encode the structure of an entire tree.
- Bottom up Construction
- Sentence of length “n” → upper triangular portion of the matrix
- Each cell [i, j] in this matrix contains the set of non-terminals that represent all the constituents that span positions i through j of the input.

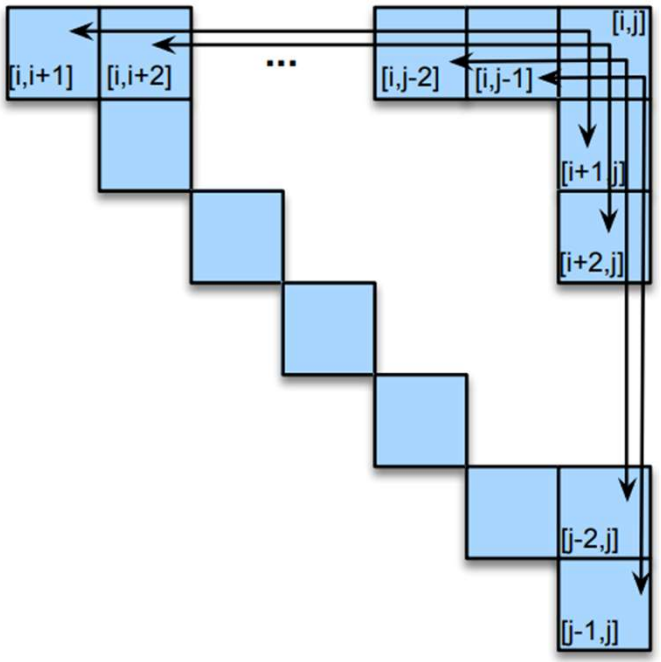
CKY Parsing .....

- Fence Posts
  - $_0$Book $_1$ that $_2$ flight $_3$
- The cell that represents the entire input resides in position [0,n] in the matrix
- Entries in the table -> Non Terminals
- For each constituent represented by an entry [i, j] , there must be a position in the input, k, where it can be split into two parts such that $i < k < j$ such that the first constituent [i, k] must lie to the left of entry [i, j] somewhere along row i, and the second entry [k, j] must lie beneath it, along column j

CKY Parsing

- Bottom up fashion of filling the cells





Let's look at a simple example before we explain the general case.

Grammar Rules in CNF

NP	→	Det Nom
Nom	→	*book* | *orange* | AP Nom
AP	→	*heavy* | *orange* | AdvD A
A	→	*heavy* | *orange*
Det	→	*a*
AdvD	→	*very*

(N.B. Converting to CNF sometimes breeds duplication!)

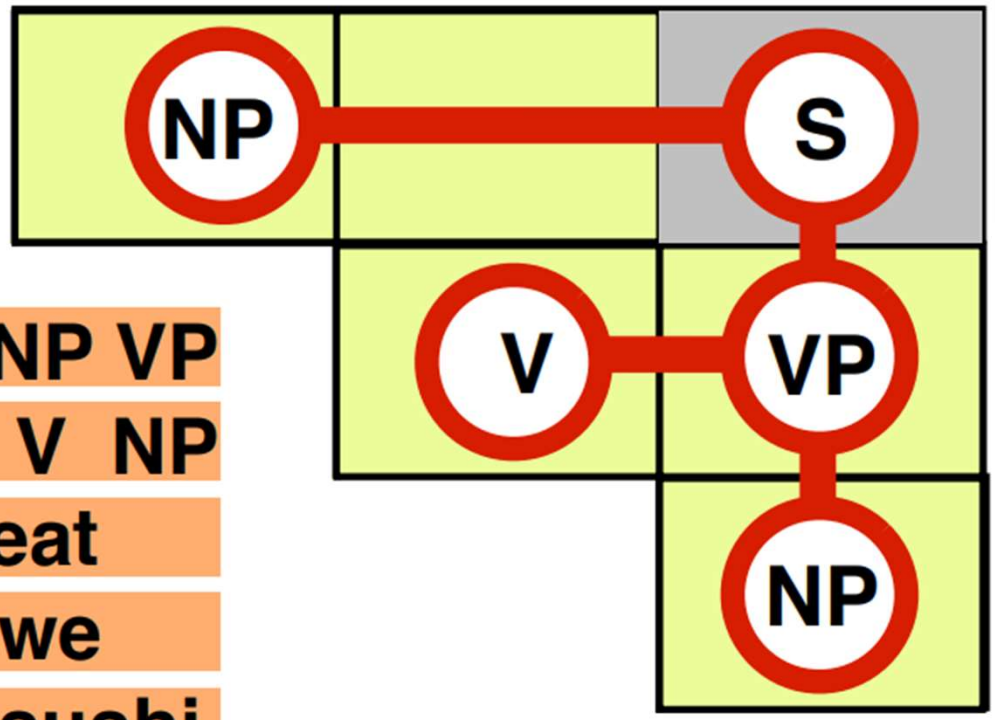
Now let's parse: *a very heavy orange book*

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det			NP	NP
1	very		AdvD	AP	Nom	Nom
2	heavy			A,AP	Nom	Nom
3	orange				Nom,A,AP	Nom
4	book					Nom

The CKY parsing algorithm

To recover the parse tree, each entry needs **pairs** of backpointers.



S → NP VP
VP → V NP
V → eat
NP → we
NP → sushi

We eat sushi

The CKY parsing algorithm

Consider the grammar Construct the CKY Parser for "we eat sushi with tuna"

S → NP VP
VP → V NP
VP → VP PP
V → eat
NP → NP PP
NP → we
NP → sushi
NP → tuna
PP → P NP
P → with

How do you count the number of parse trees for a sentence ?

Exercise: CKY parser

$$
\begin{array}{lcl}
\mathbf{S} & \rightarrow & \mathbf{NP\ VP} \\
\mathbf{NP} & \rightarrow & \mathbf{NP\ PP} \\
\mathbf{NP} & \rightarrow & \mathbf{Noun} \\
\mathbf{VP} & \rightarrow & \mathbf{VP\ PP} \\
\mathbf{VP} & \rightarrow & \mathbf{Verb\ NP}
\end{array}
$$

I eat sushi with chopsticks

Exercise: CKY parser

I eat sushi with chopsticks with you

S	→	NP	VP
NP	→	NP	PP
NP	→	sushi	
NP	→	I	
NP	→	chopsticks	
NP	→	you	
VP	→	VP	PP
VP	→	Verb	NP
Verb	→	eat	
PP	→	Prep	NP
Prep	→	with	

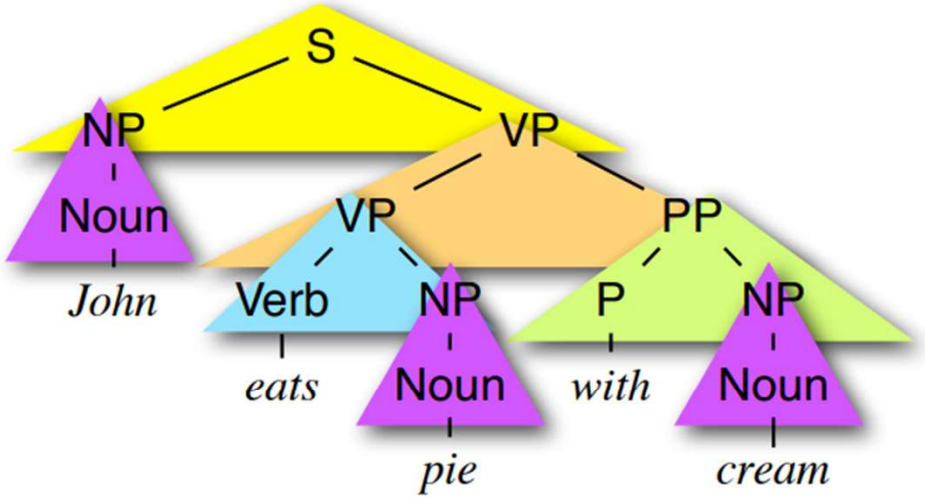
Probabilistic Context-Free Grammars

For every nonterminal X, define a probability distribution $P(X \rightarrow \alpha \mid X)$ over all rules with the same LHS symbol X:

S	$\rightarrow$	NP VP	0.8
S	$\rightarrow$	S conj S	0.2
NP	$\rightarrow$	Noun	0.2
NP	$\rightarrow$	Det Noun	0.4
NP	$\rightarrow$	NP PP	0.2
NP	$\rightarrow$	NP conj NP	0.2
VP	$\rightarrow$	Verb	0.4
VP	$\rightarrow$	Verb NP	0.3
VP	$\rightarrow$	Verb NP NP	0.1
VP	$\rightarrow$	VP PP	0.2
PP	$\rightarrow$	P NP	1.0

Computing P(τ) with a PCFG

The probability of a tree τ is the product of the probabilities of all its rules:



S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

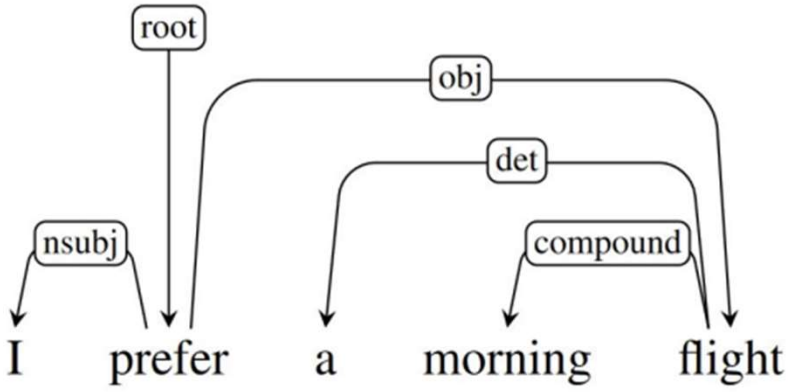
$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$

$= \mathbf{0.00384}$

Dependency Grammar

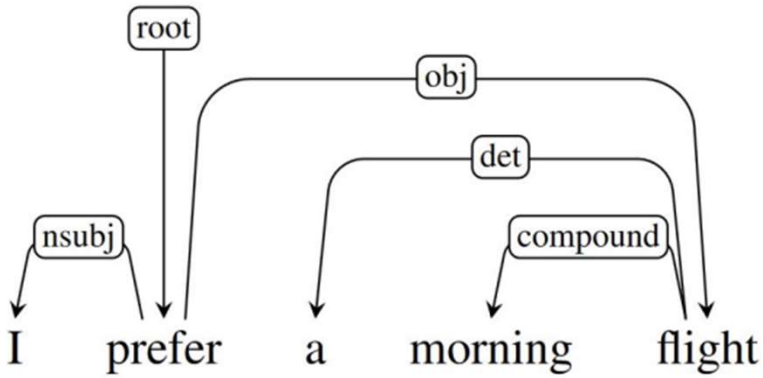
So what is dependency grammar?

Directed binary grammatical relations between the words
This direct encoding of the relationship between the predicates and their arguments (e.g., *prefer* takes *I* and *flight* as its arguments) is one of the reasons dependency grammar is more popular than constituency grammar in NLP!



What is Dependency Grammar ?

- **Arcs** go from **heads** to **dependents**
- **Exactly 1 incoming edge** for all tokens! (but can have many **outgoing** ones)
- **Root** is the head of the entire structure
- Especially useful for languages with **free word order** (constituency grammar is less suited to those)

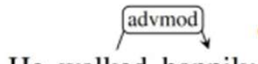


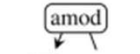
Dependency Relations

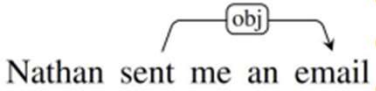
- Depends on which particular framework you use
  - E.g., Universal Dependencies (UD; de Marneffe et al., 2021), Stanford Dependencies
  - We'll stick to UD in this lecture
- Different set of relation labels
- Different definition of "heads" (function heads, content heads)
  - UD is based on content heads!

Dependency relations

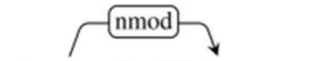
- advmod: adverb modifier
- amod: adjective modifier
- aux: auxiliary
- case: case marker
  (think of this as object of preposition)
- det: determiner
- iobj: indirect object
- nmod: noun modifier
- nmod:poss: possessive modifier
- nsubj: subject
- nummod: number modifier
- obj: direct object
- obl: oblique case ("prepositionally marked nominals functioning adverbially")
- root: root

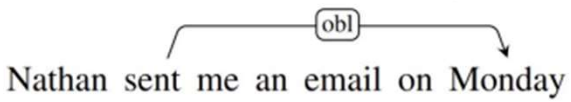






- My brother has a degree in biology.
- My brother has a degree in biology.
- My brother has a degree in biology.
- He walked happily.
- There is a big cat.
- I will study for the exam.
- I will study for the exam.
- I will study for the exam.
- I have two books.
- Nathan sent me an email on Monday.
- Nathan sent me an email on Monday.
- Nathan sent me an email on Monday.



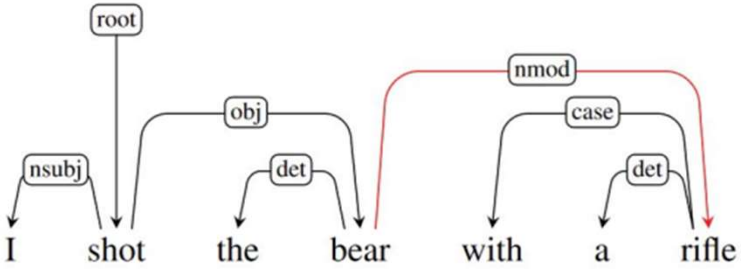


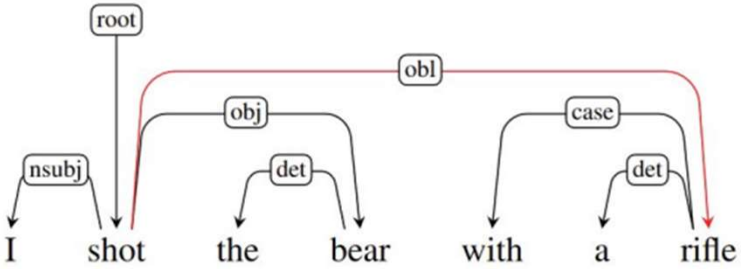
Criteria for Dependency

D is likely a dependent of H in construction C:

- H determines syntactic category of C and often replace C
- H gives semantic specification of C; D specifies H
- H is obligatory; D may be optional
- H selects D and determines whether D is obligatory
- The form of D depends on H(agreement or government)
- The linear position of D is specified with reference to H

Structural ambiguity (again!)





Images from: https://www.cs.utexas.edu/~dnp/frege/subsection-178.html
https://www.istockphoto.com/vector/hunter-shoots-a-bear-gm930143498-255034331

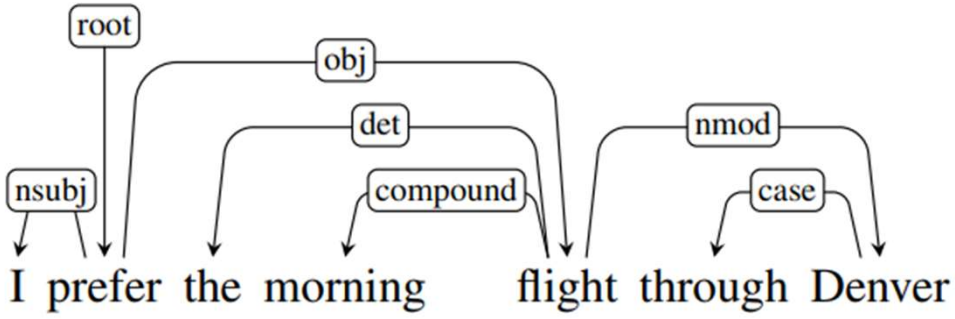
Criteria for Dependency - Conditions

- Intuitions
  - Syntactic structure is complete
  - Syntactic structure is hierarchical(Acyclic)
  - Every word has at most one syntactic head(Single Head)
- Connectedness is enforced by adding special root node

Dependency Parsing

Heads

- The head is the word in the phrase that is grammatically the most important.
- Heads are passed up the parse tree; thus, each non-terminal in a parse tree is annotated with a single word, which is its lexical head.
- The syntactic structure of a sentence is described solely in terms of directed binary grammatical relations between the words, as in the following dependency parse:



Note : Syntactic constituents can be associated with a lexical head; N is the head of an NP, V is the head of a VP

Dependency parsing is the task of analyzing the syntactic dependency structure of a given input sentence S.

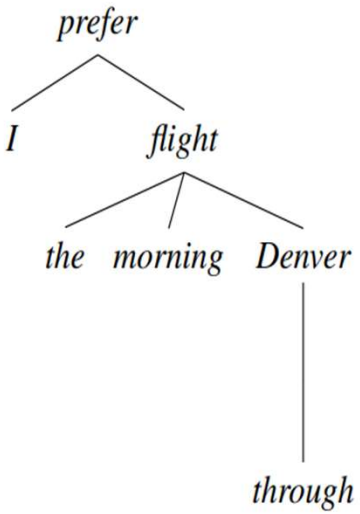
The output of a dependency parser is a dependency tree where the words of the input sentence are connected by typed dependency relations.

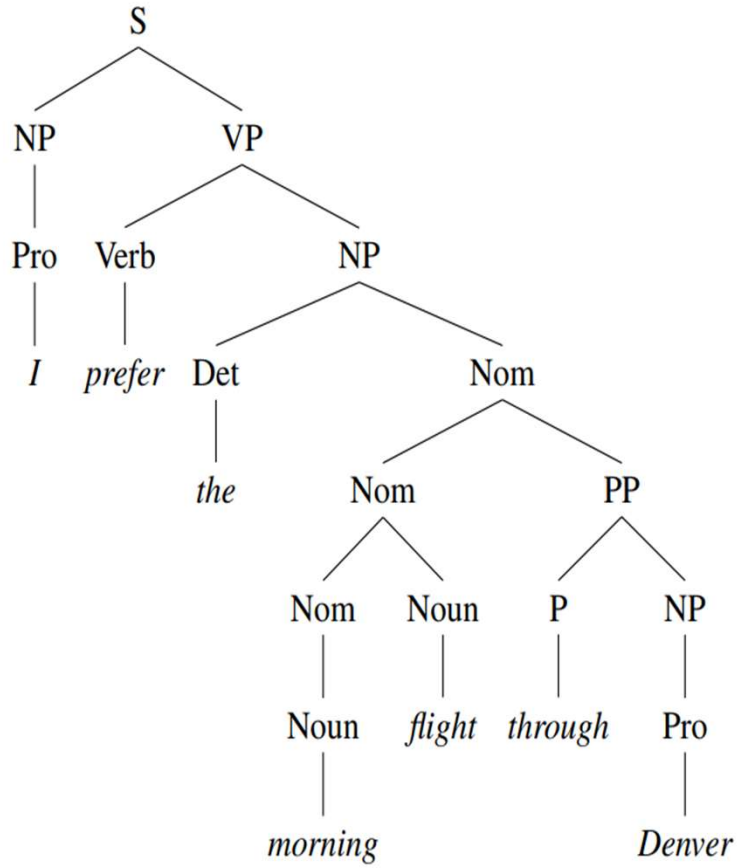
Formally, the dependency parsing problem asks to create a mapping from the input sentence with words $S = w_0 w_1 ... w_n$ (where w_0 is the ROOT) to its dependency tree graph G.

Subproblems in Dependency Parsing

- Learning: Given a training set D of sentences annotated with dependency graphs, induce a parsing model M that can be used to parse new sentences.

- Parsing: Given a parsing model M and a sentence S, derive the optimal dependency graph D for S according to M.



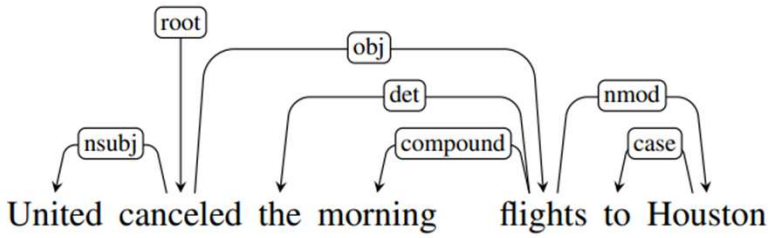


Grammatical Relations

- Head and dependent
- Grammatical Functions

Clausal Relations

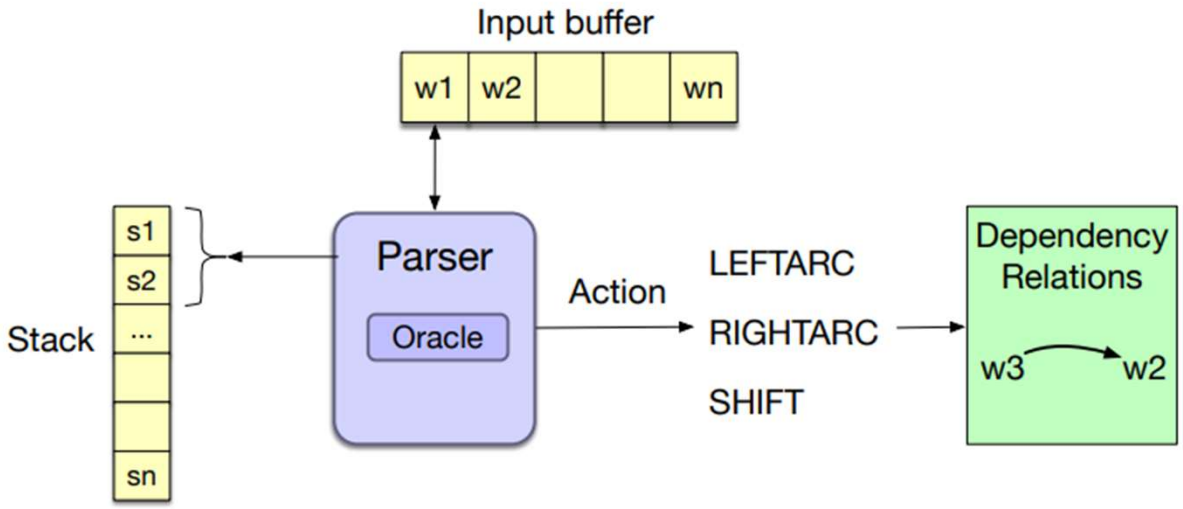
Clausal Argument Relations	Description
NSUBJ	Nominal subject
OBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other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



- Here the clausal relations NSUBJ and DOBJ identify the subject and direct object of the predicate cancel, while the NMOD, DET, and CASE relations denote modifiers of the nouns flights and Houston

Transition-Based Dependency Parsing

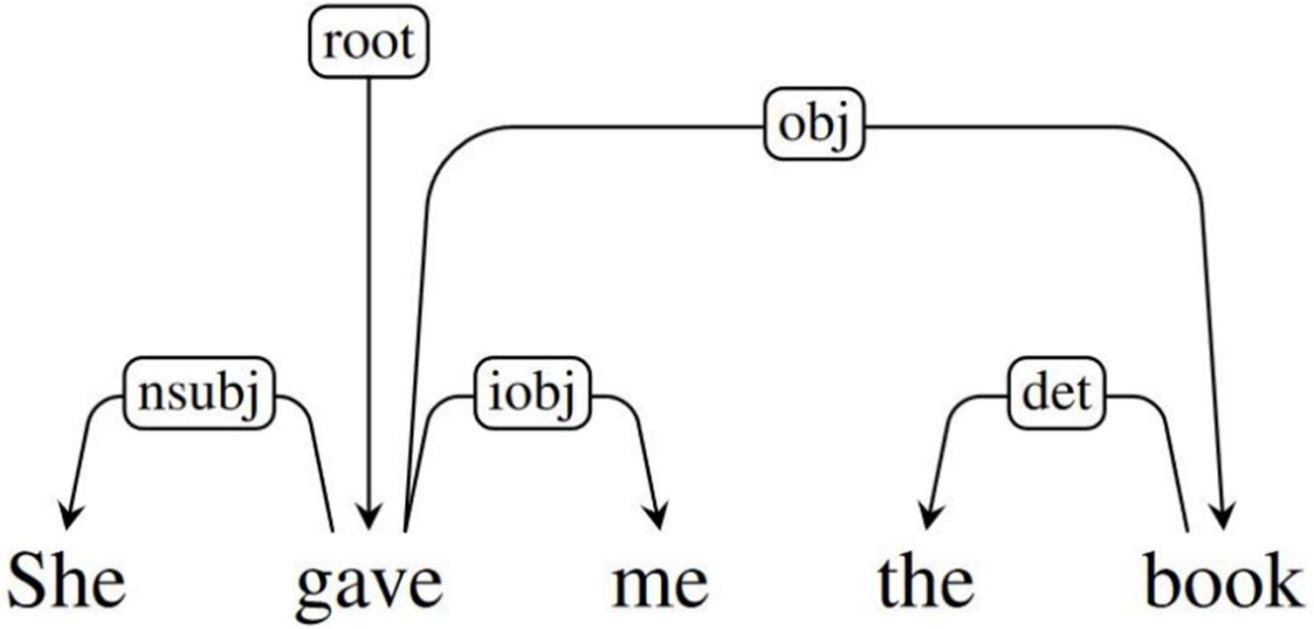
- Based on shift-reduce parsing
- The parser walks through the sentence left-to-right, successively shifting items from the buffer onto the stack.



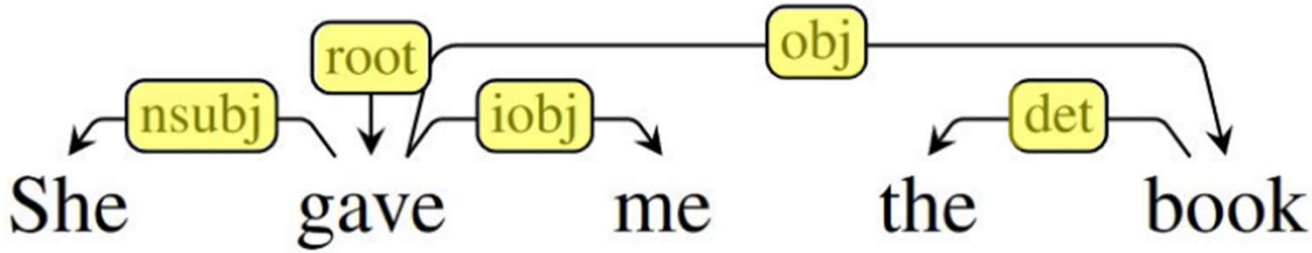
Transition-Based Dependency Parsing

- At each time point we examine the top two elements on the stack, and the oracle makes a decision about what transition to apply to build the parse.
- Three transition operators
  - LEFTARC - Assert a head-dependent relation between the word at the top of the stack and the second word; remove the second word from the stack.
  - RIGHTARC - Assert a head-dependent relation between the second word on the stack and the word at the top; remove the top word from the stack.
  - SHIFT - Remove the word from the front of the input buffer and push it onto the stack.

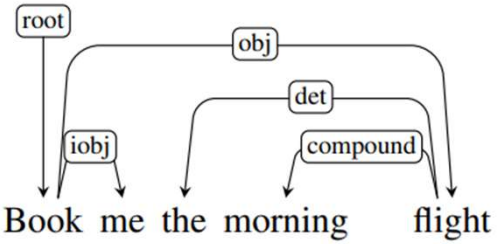
Transition based Parsing



Step	Stack	Input Buffer	Action	Relation Added
0	[root]	[She, gave, me, the, book]	S	
1	[root, She]	[gave, me, the, book]	S	
2	[root, She, gave]	[me, the, book]	LA	(She ← gave)
3	[root, gave]	[me, the, book]	S	
4	[root, gave, me]	[the, book]	RA	(gave → me)
5	[root, gave]	[the, book]	S	
6	[root, gave, the]	[book]	S	
7	[root, gave, the, book]	[]	LA	(the ← book)
8	[root, gave, book]	[]	RA	(gave → book)
9	[root, gave]	[]	RA	(root → gave)
10	[root]	[]	Done	



Transition-Based Dependency Parsing



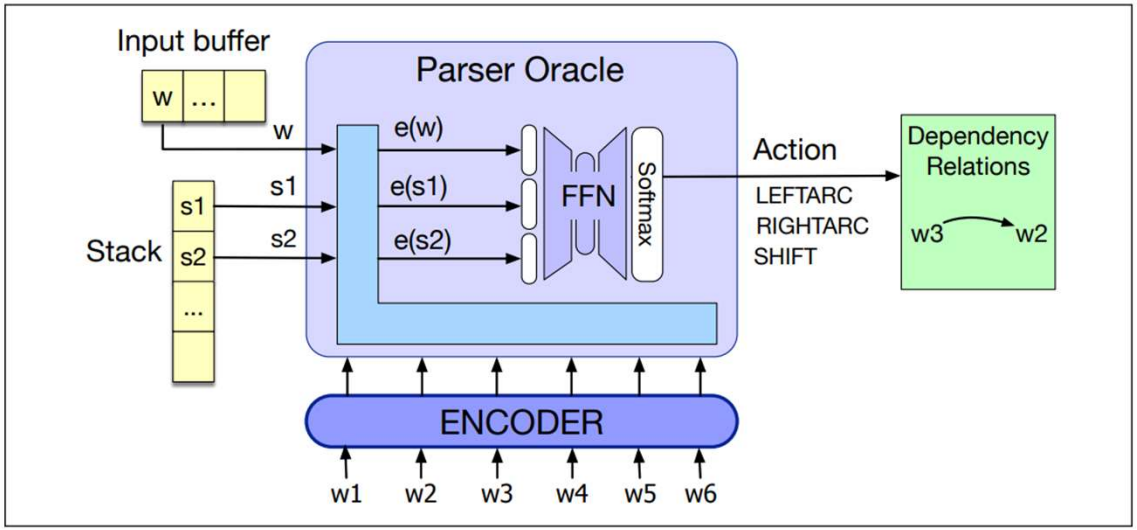
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, 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	

Neural Dependency Parsing

- Greedy transition based parsers
- Demonstrated comparable performance and significantly better efficiency than traditional feature-based discriminative dependency parsers.
- Employs the arc-standard system for transitions.
- The aim of the model is to predict a transition sequence from some initial configuration c to a terminal configuration, in which the dependency parse tree is encoded.
- As the model is greedy, it attempts to correctly predict one transition $T \in$ {shift, Left-Arc , Right-Arc} at a time, based on features extracted from the current configuration $c = (\sigma, \beta, A)$,where
  - σ is the stack, β the buffer, and A the set of dependency arcs for a given sentence

Neural Dependency Parsing

- The parser takes the top 2 words on the stack and the first word of the buffer
- Represents them by their encodings (from running the whole sentence through the encoder)
- Concatenates the embeddings and passes through a SoftMax to choose a parser action (transition).



- Deep contextualized word embeddings form the encoder allow parsers to pack information about global sentence.
- This global structure information is available for Parser Oracle to embed into local feature representations.

Advantages

- Deep contextualized word representations are more effective at reducing errors in transition based parsing than in graph-based parsing.
- The differential error reduction should be especially visible on phenomena such as:
  - Longer dependencies
  - Dependencies closer to the root
  - Certain parts of speech
  - Certain dependency relations
  - Longer sentences

Feature Selection

Depending on the desired complexity of the model, there is flexibility in defining the input to the neural network. The features for a given sentence S generally include some subset of:

1. S_{word}: Vector representations for some of the words in S (and their dependents) at the top of the stack σ and buffer β.

2. S_{tag}: Part-of-Speech (POS) tags for some of the words in S. POS tags comprise a small, discrete set: $\mathcal{P} = \{NN, NNP, NNS, DT, JJ, ...\}$

3. S_{label}: The arc-labels for some of the words in S. The arc-labels comprise a small, discrete set, describing the dependency relation: $\mathcal{L} = \{amod, tmod, nsubj, csubj, dobj, ...\}$

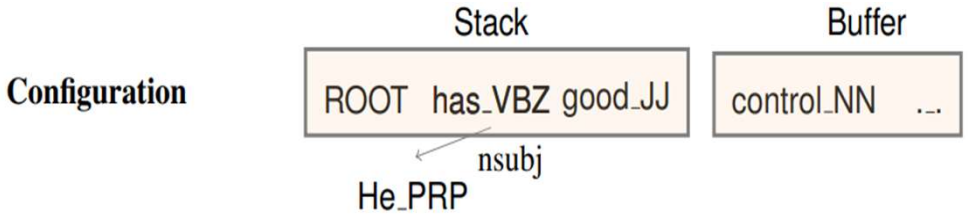
Feature Selection Example

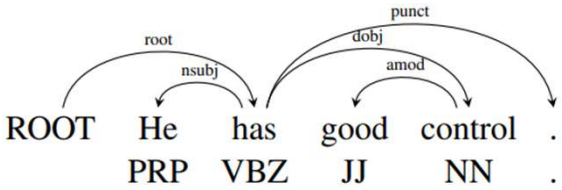
As an example, consider the following choices for S_{word}, S_{tag}, and S_{label}.

1. S_{word}: The top 3 words on the stack and buffer: $s_1, s_2, s_3, b_1, b_2, b_3$. The first and second leftmost / rightmost children of the top two words on the stack: $lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i)$, $i = 1, 2$. The leftmost of leftmost / rightmost of rightmost children of the top two words on the stack: $lc_1(lc_1(s_i)), rc_1(rc_1(s_i))$, $i = 1, 2$. In total S_{word} contains $n_w = 18$ elements.
2. S_{tag}: The corresponding POS tags for S_{tag} ($n_t = 18$).
3. S_{label}: The corresponding arc labels of words, excluding those 6 words on the stack/buffer ($n_l = 12$).

Extraction of Tags - Example

- For example, given the configuration





- $S^t = \{lc_1(s_2).t, s_2.t, rc_1(s_2).t, s_1.t\}$
- we will extract `PRP, VBZ, NULL, JJ` in order.

Graph based Dependency Parsing

- Graph-based approaches to dependency parsing search through the space of possible trees for a given sentence for a tree (or trees) that maximize some score.
- The overall score for a tree can be viewed as a function of the scores of the parts of the tree.
- Edge Factored Approach - score for a tree is based on the scores of the edges that comprise the tree

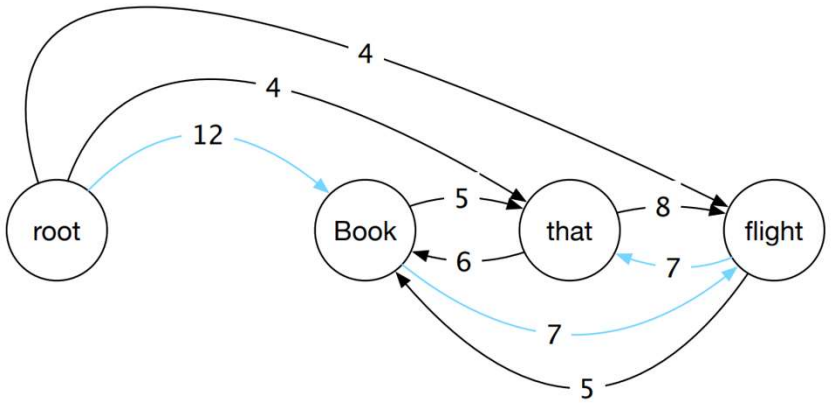
$$score(t,S) = \sum_{e \in t} score(e)$$

Graph based Dependency Parsing – Maximum Spanning Tree Algorithm

- Efficient greedy algorithm to search for optimal spanning trees in directed graphs
- Given an input sentence, it begins by constructing a fully-connected, weighted, directed graph where
  - The vertices are the input words
  - The directed edges represent all possible head-dependent assignments.
- An additional ROOT node is included with outgoing edges directed at all of the other vertices.

Graph based Dependency Parsing

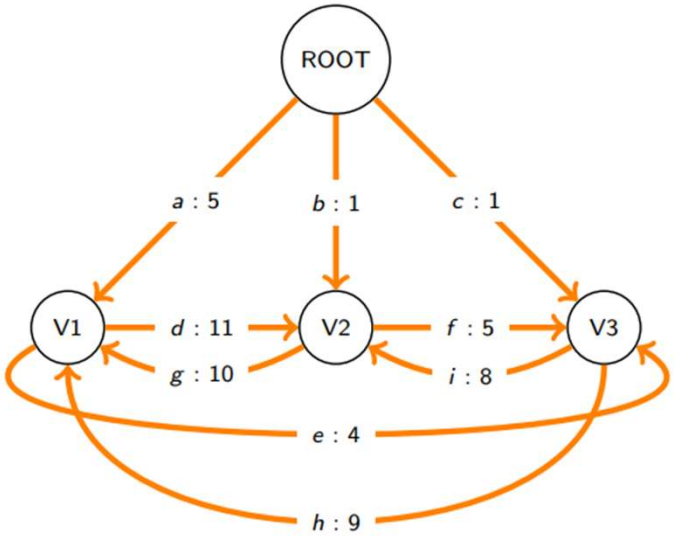
- The weights in the graph reflect the score for each possible head-dependent relation as provided by a model generated from training data.
- Given these weights, a maximum spanning tree of this graph emanating from the ROOT represents the preferred dependency parse for the sentence



Chu Liu and Edmonds Algorithm

- Clean up Phase - The cleanup phase begins by adjusting all the weights in the graph by subtracting the score of the maximum edge entering each vertex from the score of all the edges entering that vertex.
- Create a new graph and Rescoring - Having adjusted the weights, the algorithm creates a new graph by selecting a cycle and collapsing it into a single new node.
  - Edges that enter or leave the cycle are altered so that they now enter or leave the newly collapsed node.
  - Edges that do not touch the cycle are included and edges within the cycle are dropped.
  - Consists of two stages: Contracting and Expanding
- Note : All the nodes have ougoing edges to every other node except the root node. There is a outgoing edge from the root to every other node.

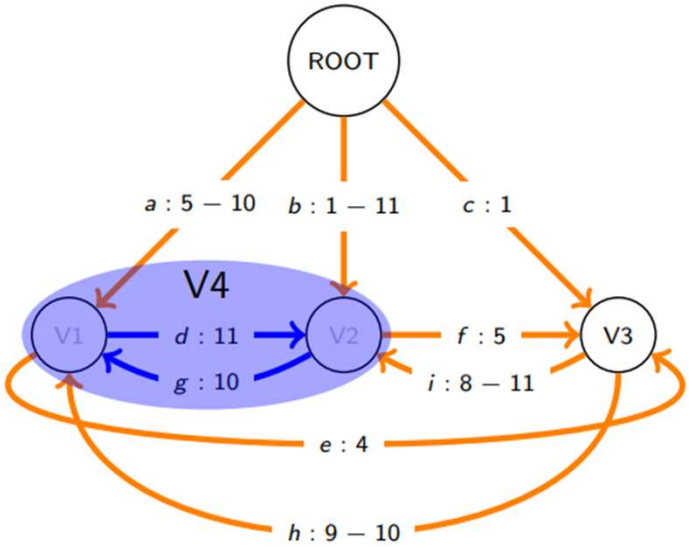
An Example - Contracting Stage



	bestInEdge
V1	
V2	
V3	

	kicksOut
a	
b	
c	
d	
e	
f	
g	
h	
i	

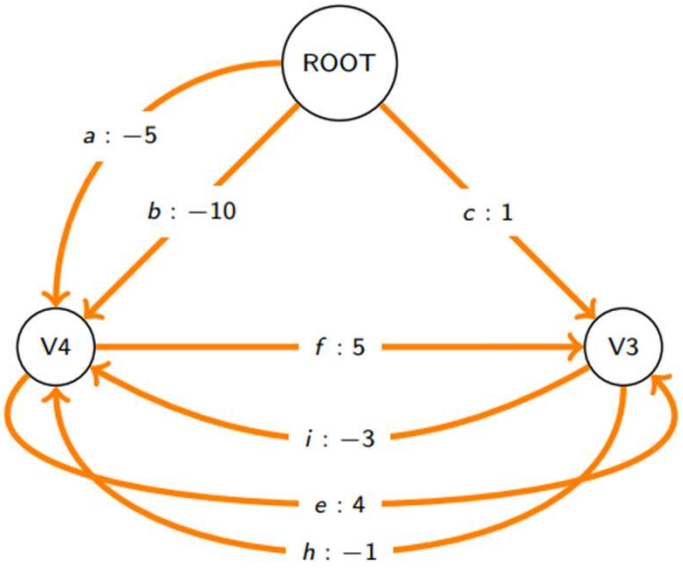
An Example - Contracting Stage



	bestInEdge
V1	g
V2	d
V3	

	kicksOut
a	g
b	d
c	
d	
e	
f	
g	
h	g
i	d

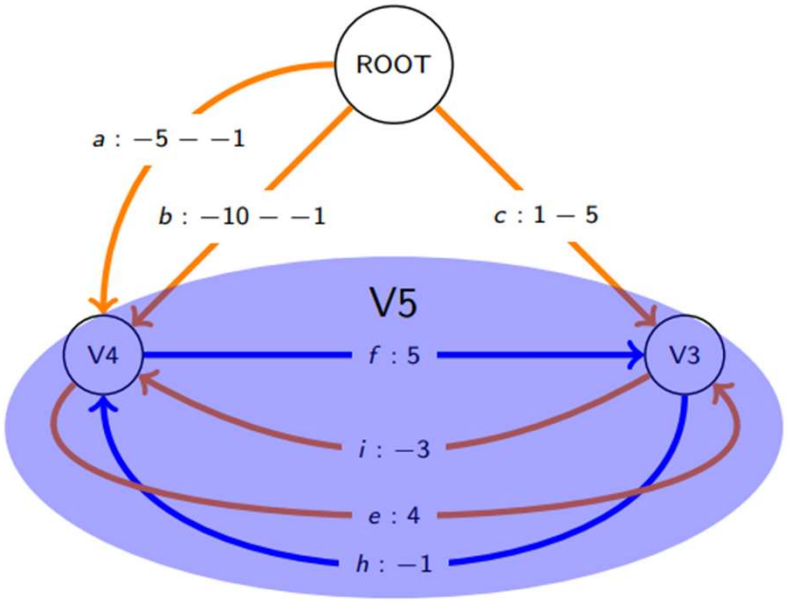
An Example - Contracting Stage



	bestInEdge
V1	g
V2	d
V3	
V4	

	kicksOut
a	g
b	d
c	
d	
e	
f	
g	
h	g
i	d

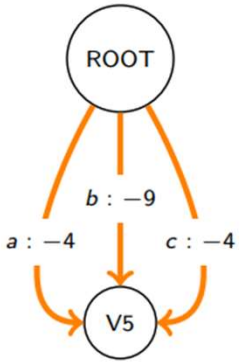
An Example - Contracting Stage



	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	

	kicksOut
a	g, h
b	d, h
c	f
d	
e	
f	
g	
h	g
i	d

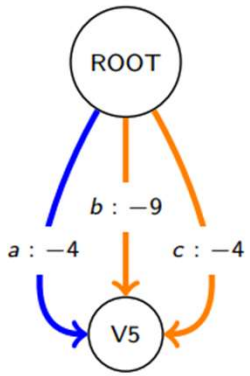
An Example - Contracting Stage



	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

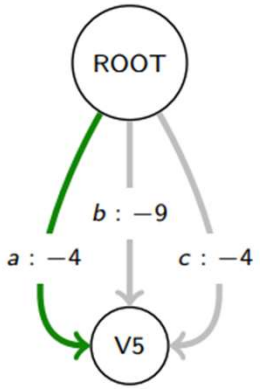
An Example - Contracting Stage



	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

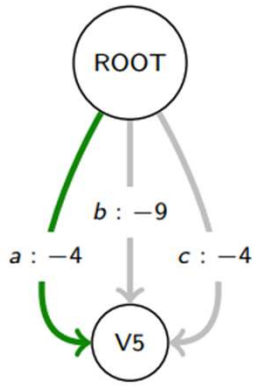
An Example - Expanding Stage



	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

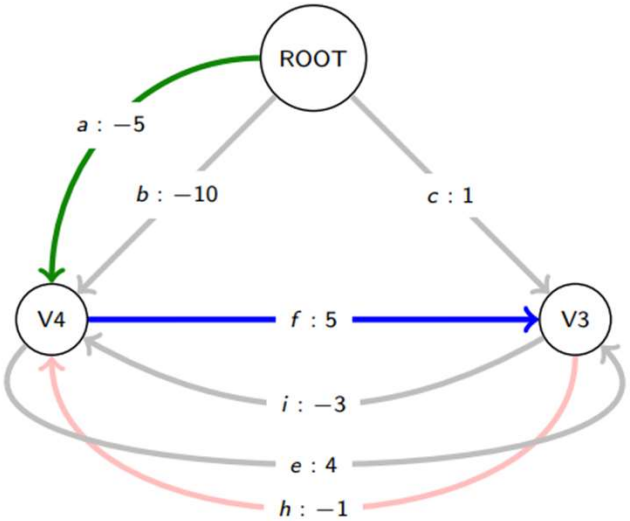
An Example - Expanding Stage



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

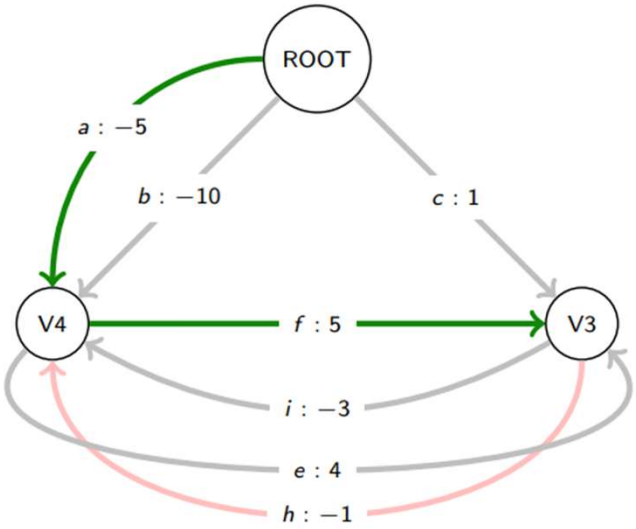
An Example - Expanding Stage



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

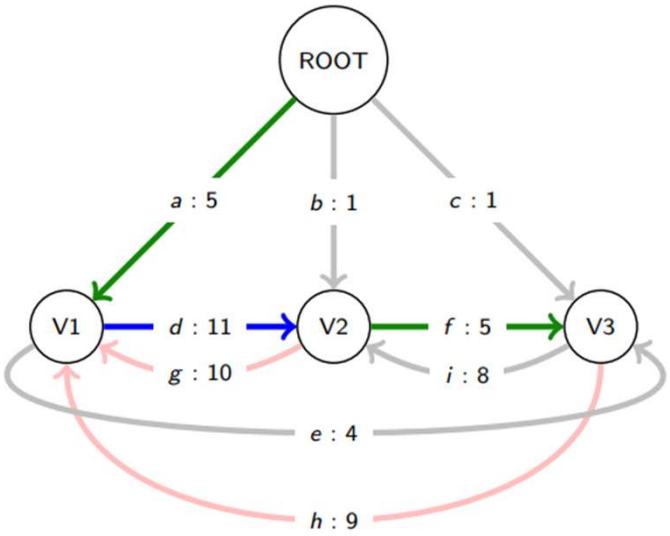
An Example - Expanding Stage



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

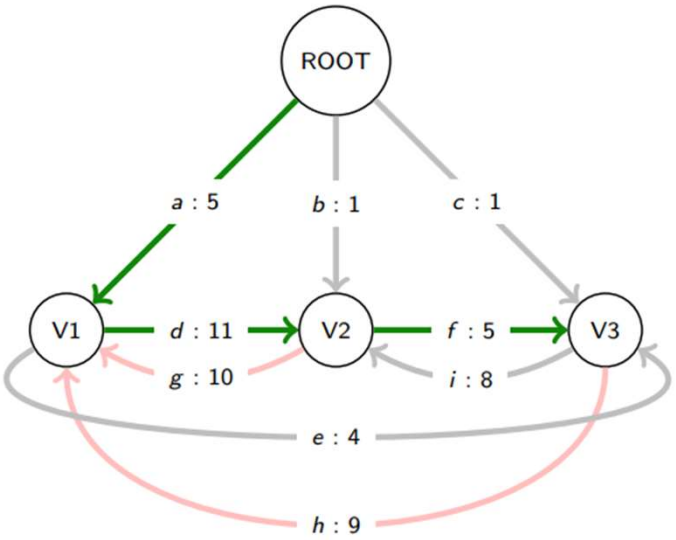
An Example - Expanding Stage



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

An Example - Expanding Stage



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

Apply Chu-Liu-Edmonds algorithm on the following graph.