Parsing

Lizhen Qu

Overview of the NLP Lectures

- Introduction to natural language processing (NLP).
- Regular expressions, sentence splitting, tokenization, part-of-speech tagging.

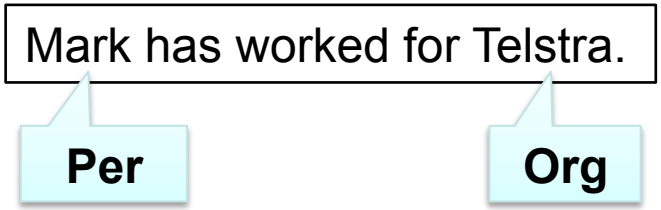
- Language models.

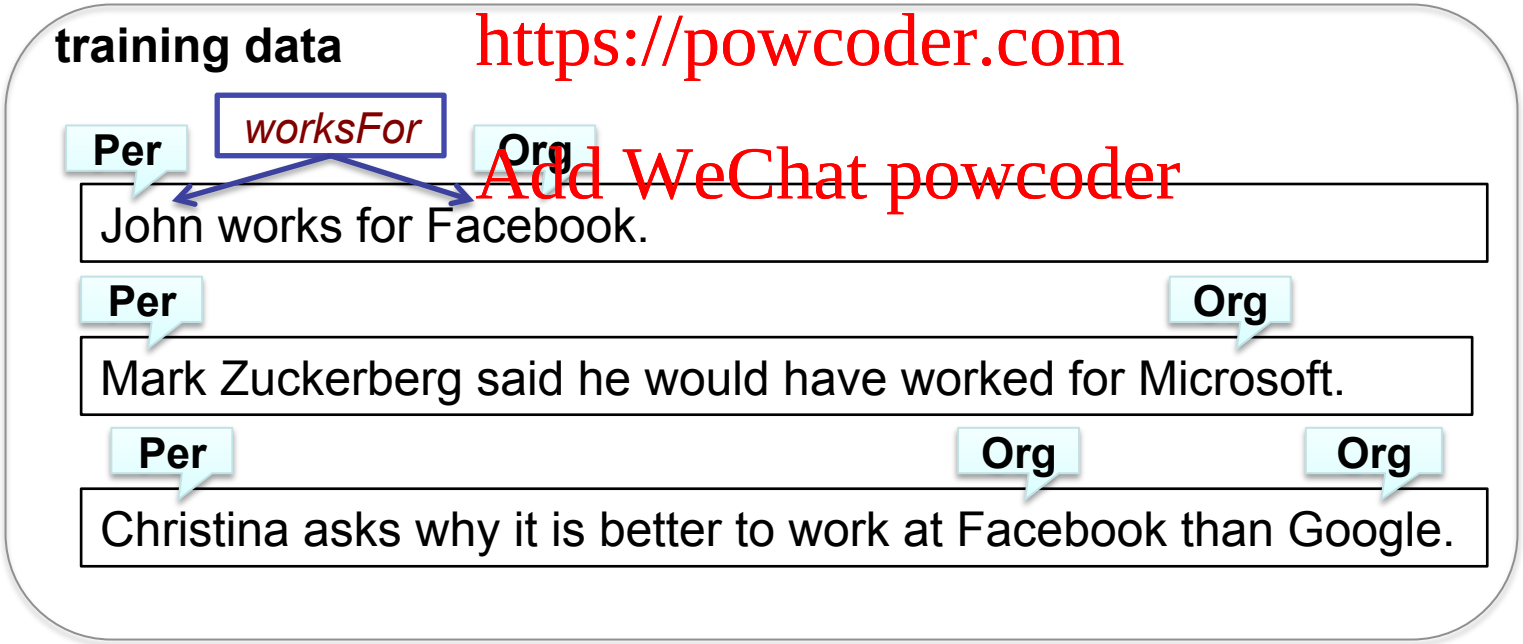
- Vector semantics.

- Parsing.
  - Dependency parsing.
- Semantics.

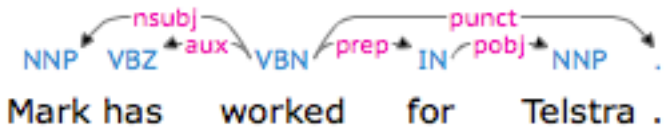
Relation Extraction

- Find *worksFor*(*entity_a, entity_b*) relation from text.

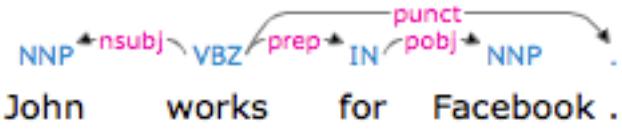




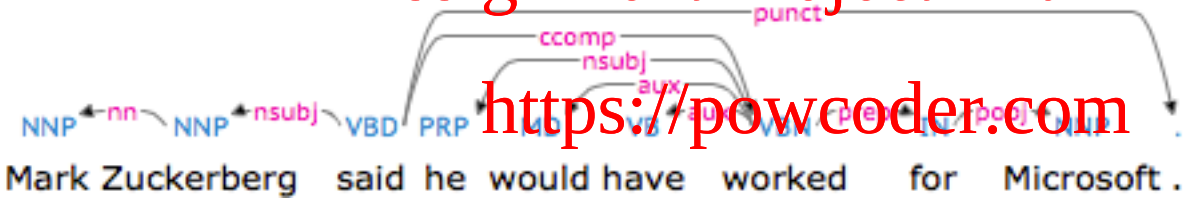
Use Shortest Paths between Entity Mentions



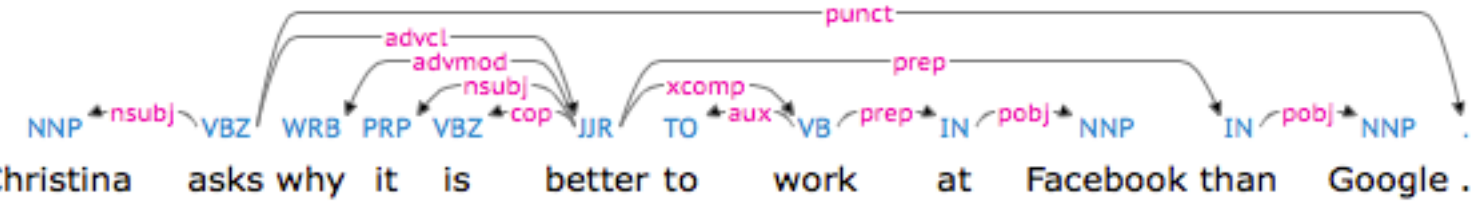
(Mark, Telstra) <-nsubj-prep->pobj->



(John, Facebook) <-nsubj-prep->pobj->



(Zuckerberg, Microsoft) <-nsubj-ccomp->prep->pobj->



(Christina, Google) <-nsubj-advcl->prep->pobj->

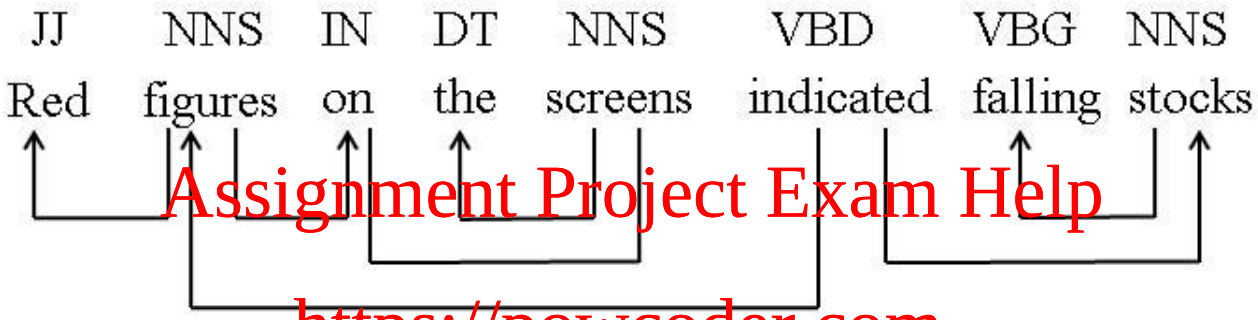
Dependency Grammar

- Syntactic structure consists of lexical items, linked by binary asymmetric relations called dependencies.

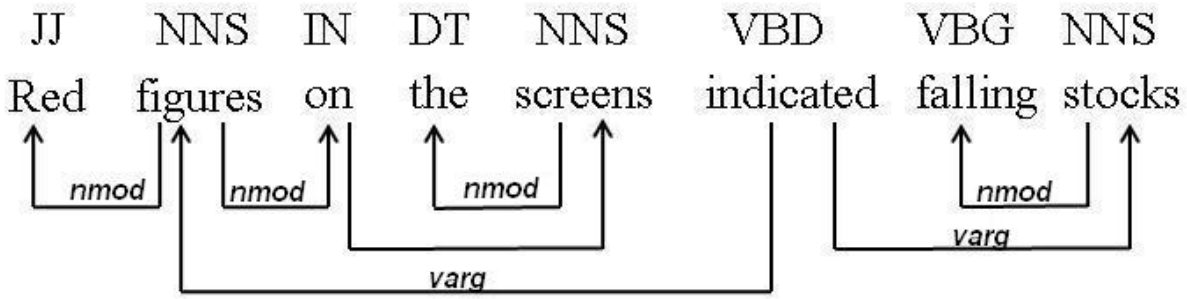
- head → dependent
  - head (governor) : grammatically most important.
  - dependent (modifier): modifier, object, or complement.

Dependency Trees

- Without labels.



- With labels.



Dependency Parsing

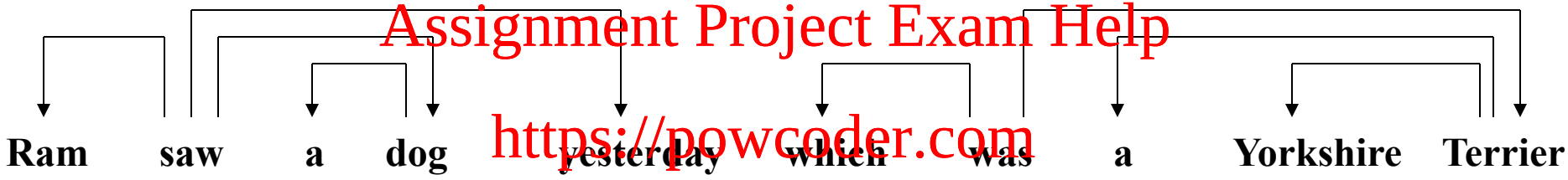
- Formal definition for unlabeled dependency trees:

Dependency graph $D = (V, E)$ where

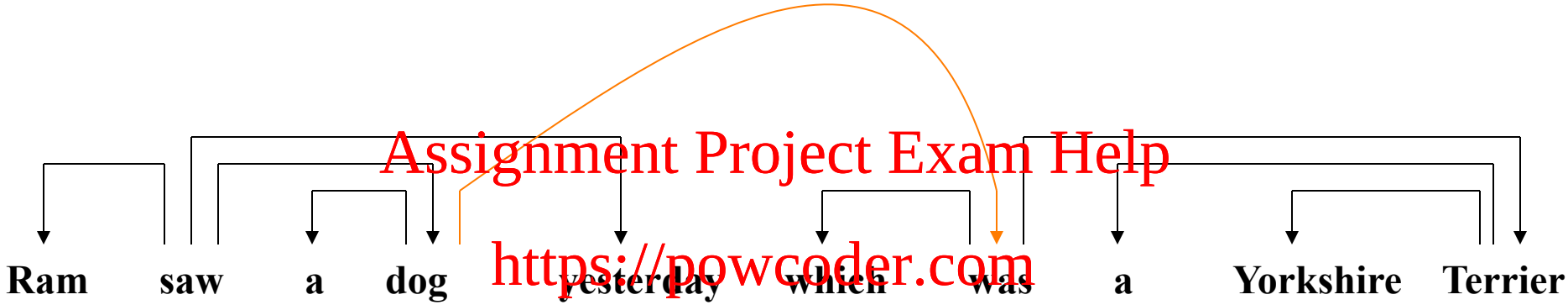
- V is the set of nodes (words in a input sentence)
- E is the set of arcs indicating grammatical relations.
- $v_i \rightarrow v_j$ or $(v_i, v_j) \in E$ denotes an arc from head v_i to dependent v_j.

- Dependency parsing: task of mapping an input string to a dependency graph satisfying certain conditions.

Projective Dependency Tree



Non-Projective Dependency Tree



Crossing lines!

English has very few non-projective cases.

Well-Formedness

- A dependency graph is well-formed iff
  - Single head: Each word has only one head.
  - Acyclic: The graph should be acyclic.
  - Connected: There is a path between any pairs of nodes.
  - Projective: iif an edge from word A to word B implies that there exists a directed path in the graph from A to every word between A and B in the sentences.

Parsing Algorithms

- Graph-based parsing
  - CYK, Eisner, McDonald

- Transition-based parsing
  - Covington, Yamada and Matsumuto, Nivre etc.

Nivre's Algorithm (Arc-eager) [3]

- Transition-based.
- Parser configuration $< S, I, A >$:
  - S is the stack. $S[0]$ is the topmost node.
  - I is the list of remaining input words. $I[0]$ is the leftmost word.
  - A is the set of current dependencies (arcs) for the dependency graph.
- INPUT: a word sequence $\mathbf{v} = v_1|...|v_n$, a set of rules R.

Parser Transitions

Left-Arc (LA) $< v_i|S, v_j|I, A > \Rightarrow < S, v_j|I, A \cup \{(v_j, v_i)\}$ $v_i \leftarrow v_j \in R$, $\nexists v_k (v_k, v_i) \in A$

single head

Right-Arc (RA) $< v_i|S, v_j|I, A > \Rightarrow < v_j|v_i|S, I, A \cup \{(v_i, v_j)\}$ $v_i \rightarrow v_j \in R$, $\nexists v_k (v_k, v_j) \in A$

Reduce (R) $< v_i|S, I, A > \Rightarrow < S, I, A >$ $\exists (v_k, v_i) \in A$

Shift (S) $< S, v_j|I, A > \Rightarrow < v_j|S, I, A >$

Parsing Details

- Slight modifications:
  - Each dependency graph has an artificial root in order to form a tree.
  - Parsing starts with an initial configuration $< [ROOT], \mathbf{n}, \emptyset >$ and terminates when it reaches $< [ROOT], \mathbf{nil}, A >$ through a sequence of transitions.

- Nondeterministic transitions?

  - Priority ordering of transitions.

    $\mathbf{LA} > \mathbf{RA} >$
    if $S[0]$ can be a transitive head of $I[0]$, then **Shift**, otherwise **Reduce**.

  - Guided parsing.

Grammatical Rules for the Example

$$Noun \rightarrow Adj$$

$$ROOT \rightarrow Verb$$

$$Noun \rightarrow Det$$

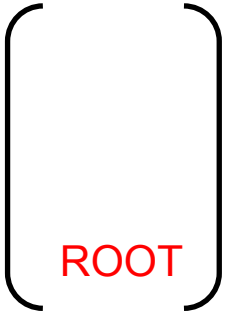
$$Verb \rightarrow Prep$$

$$Verb \rightarrow Noun$$

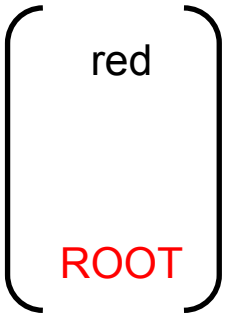
$$figure \rightarrow on$$

$$on \rightarrow screen$$

Example



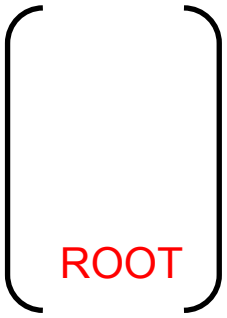
Example



ROOT Red figures on the screen indicated falling stocks I

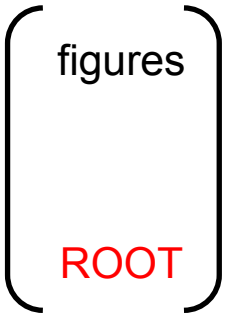
Shift

Example



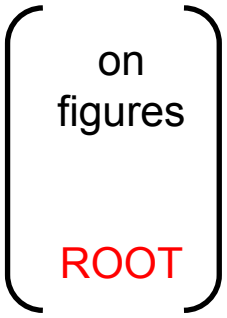
Left-arc

Example



Shift

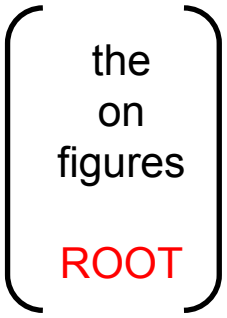
Example

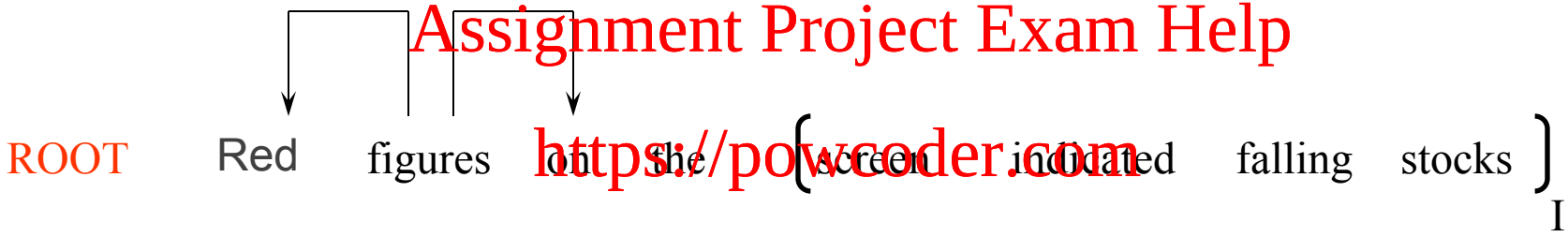


ROOT Red figures on the screen indicated falling stocks

Right-arc

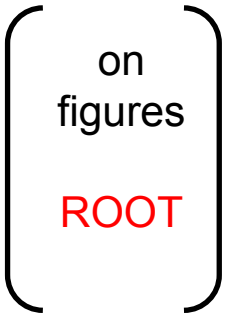
Example

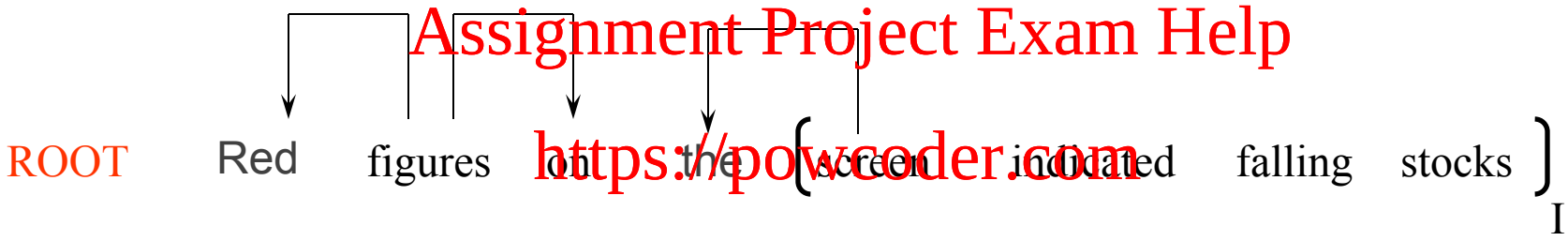




Shift

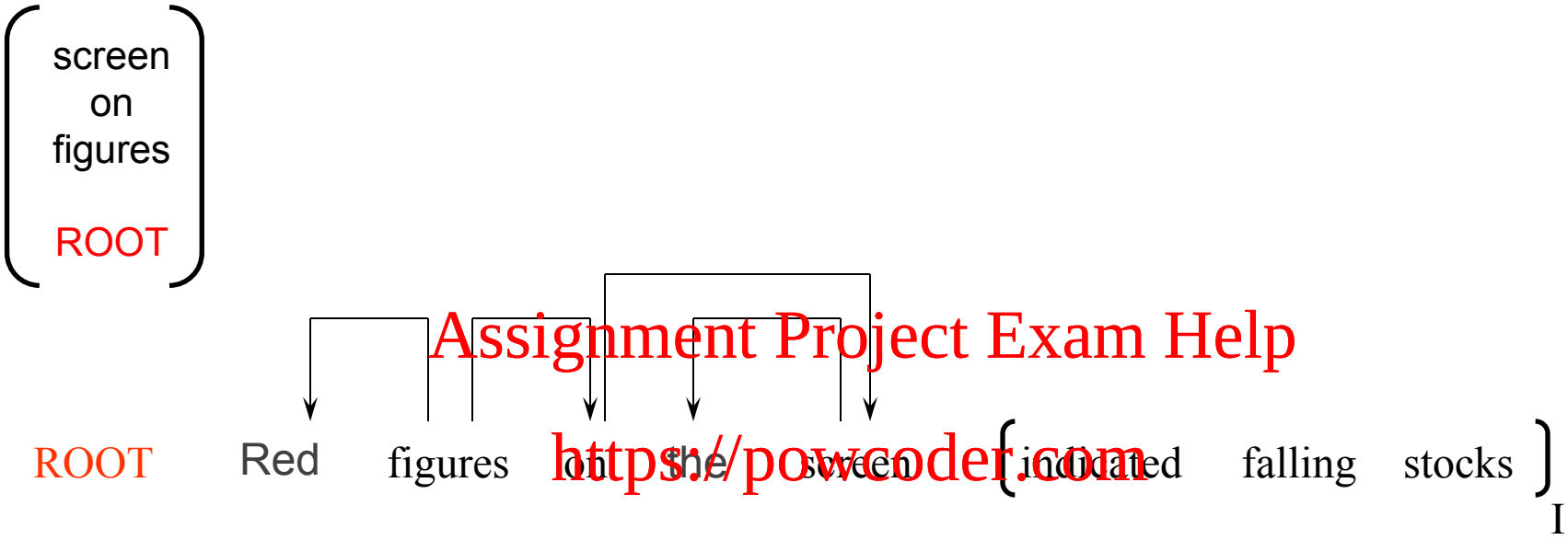
Example





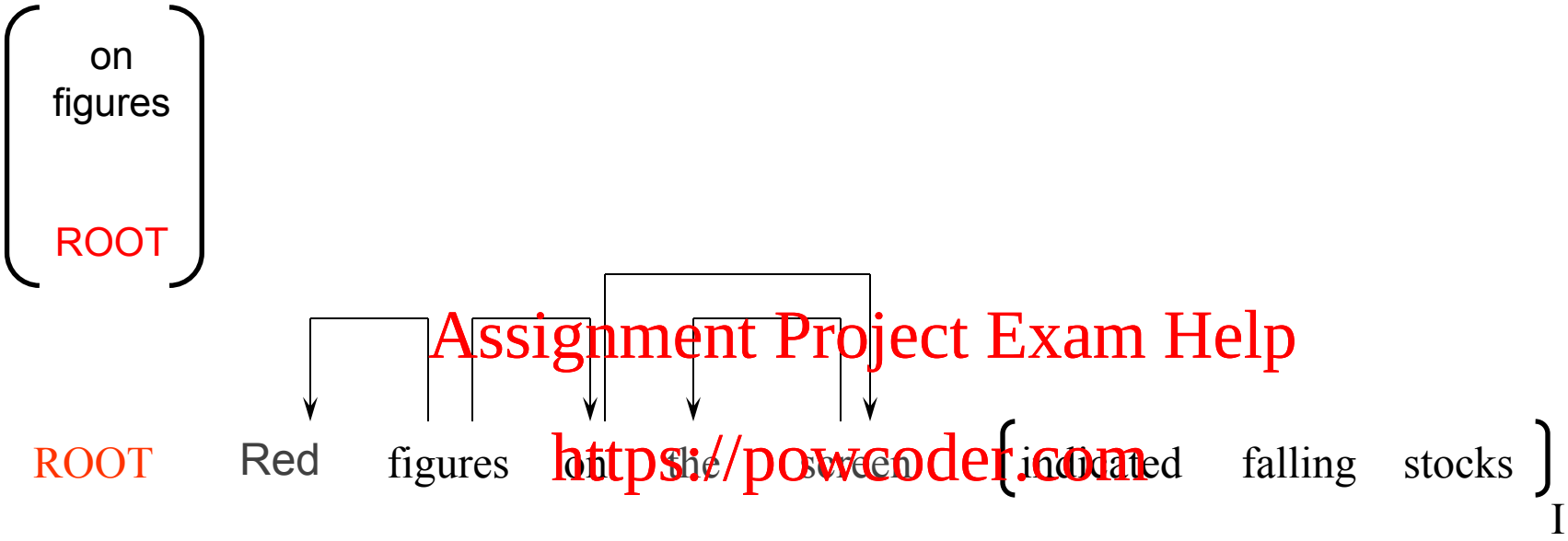
Left-arc

Example



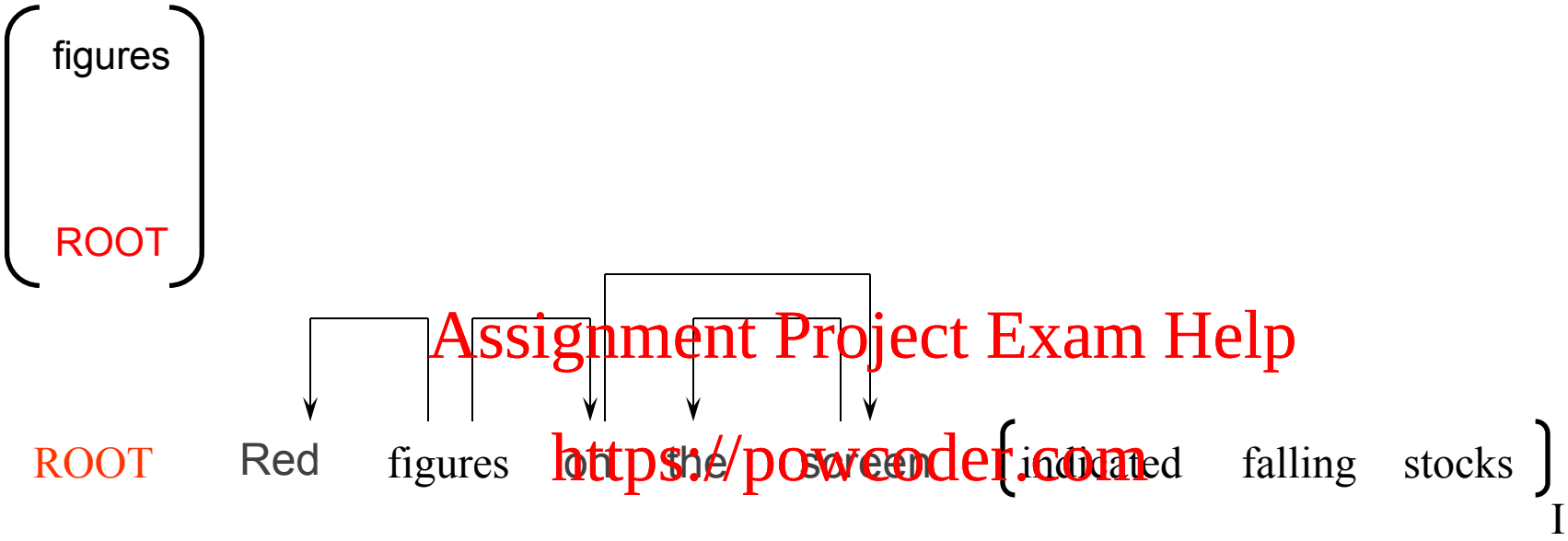
Right-arc

Example



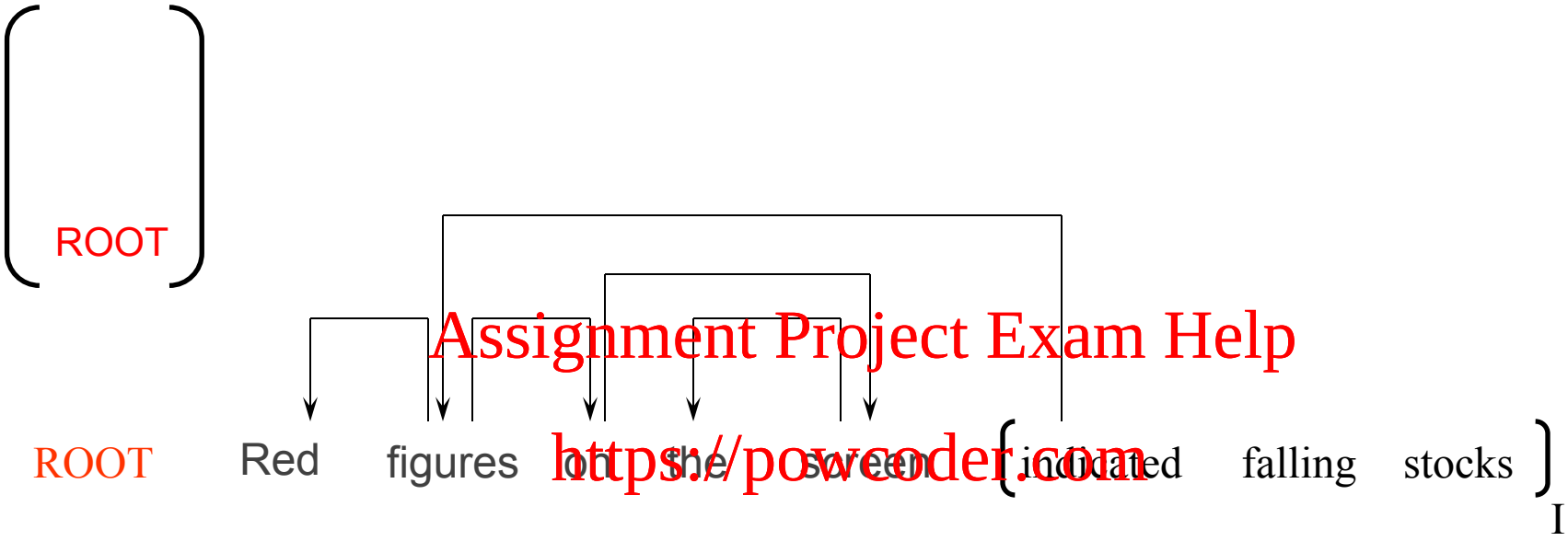
Reduce

Example



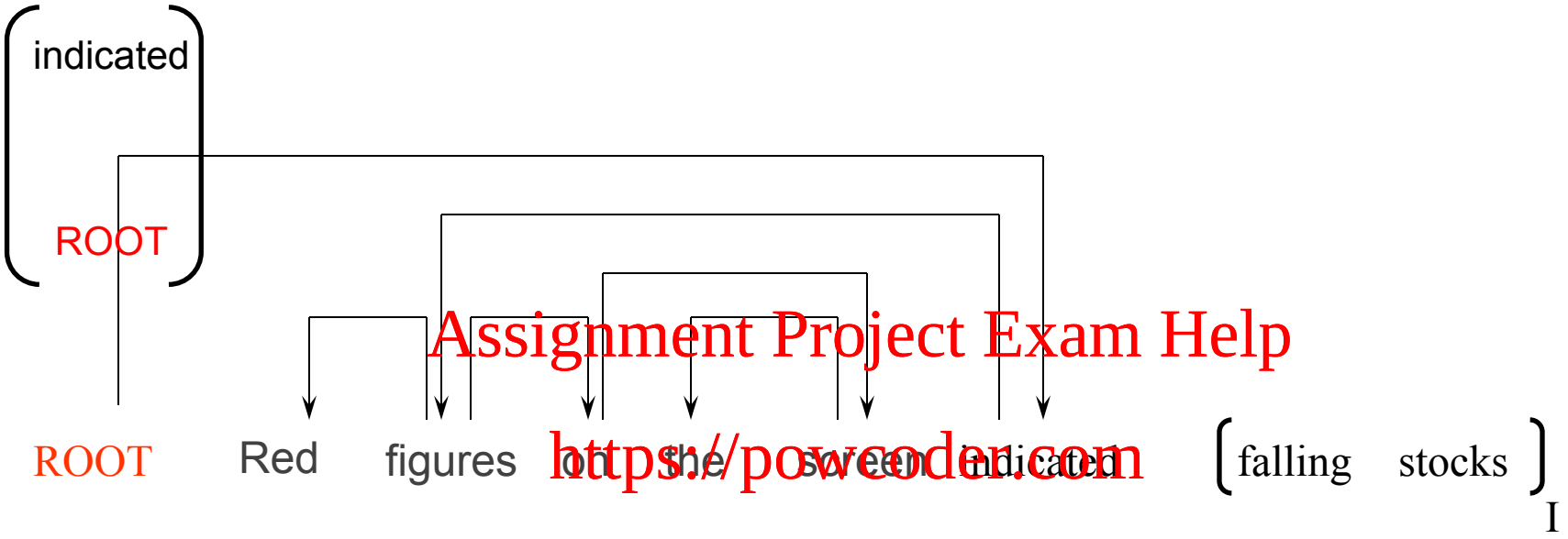
Reduce

Example



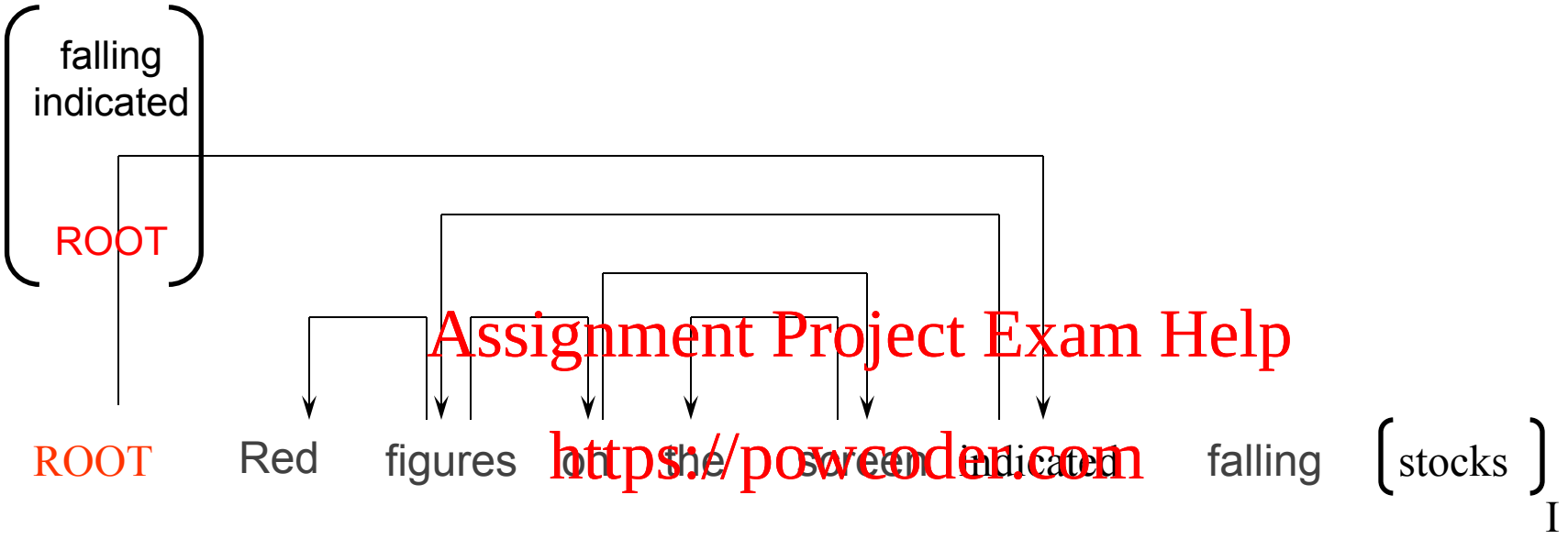
Left-arc

Example



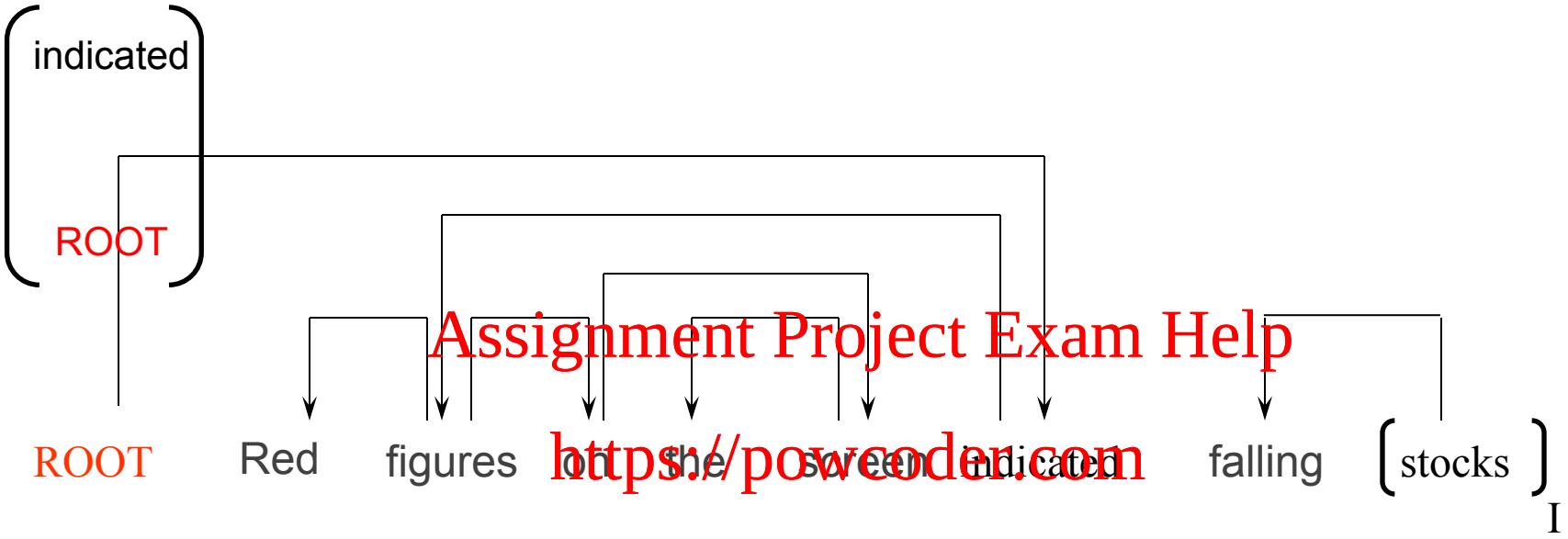
Right-arc

Example



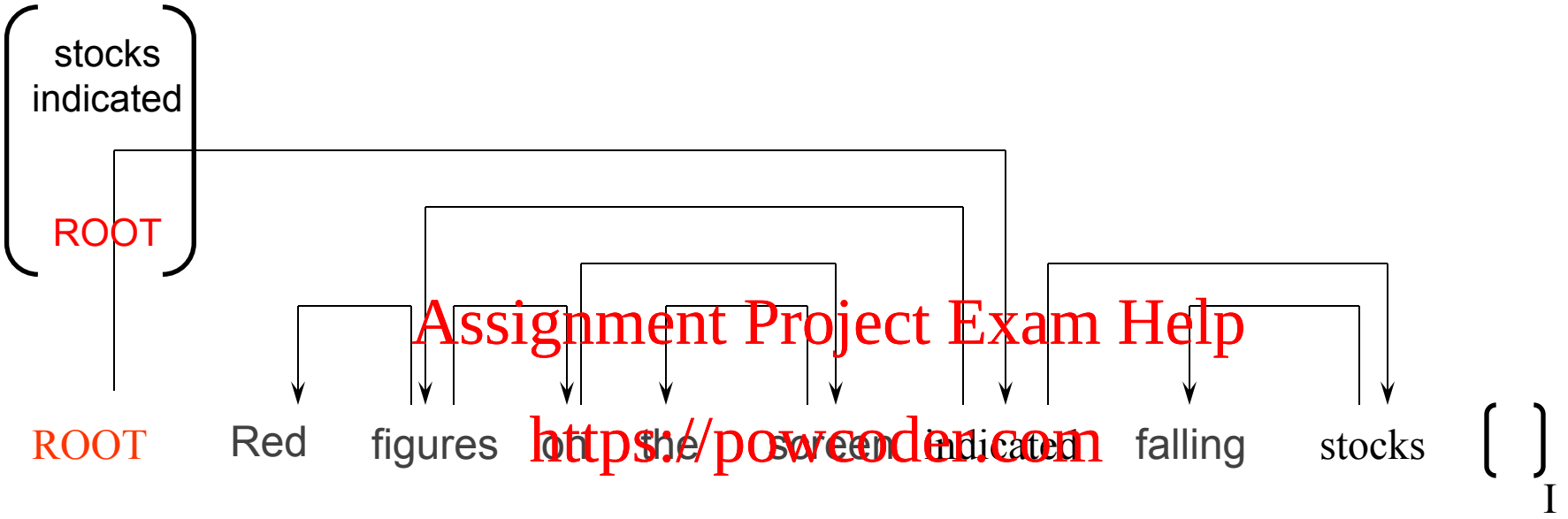
Shift

Example



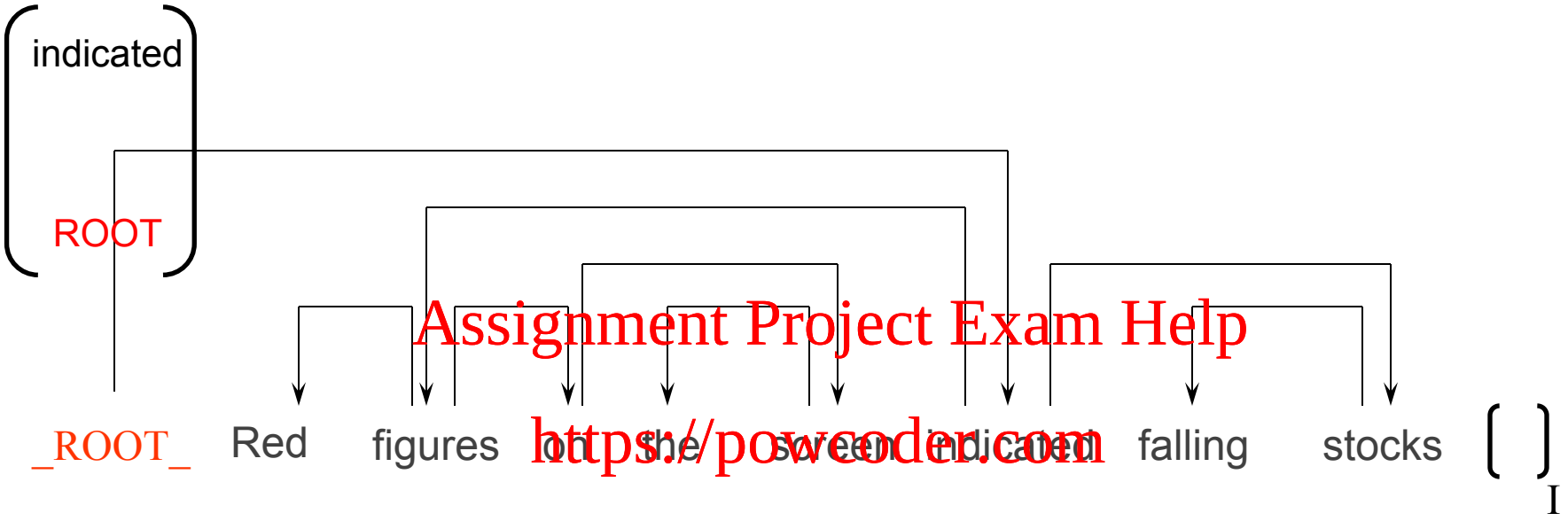
Left-arc

Example



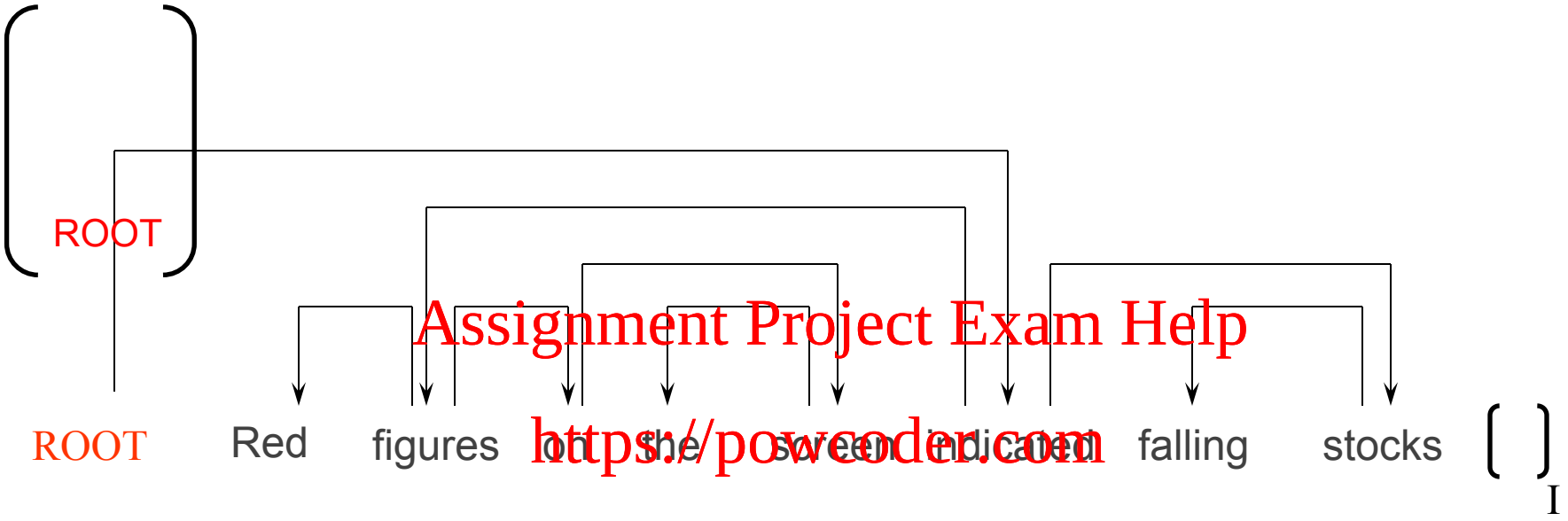
Right-arc

Example



Reduce

Example



Reduce

Configurations of the Example

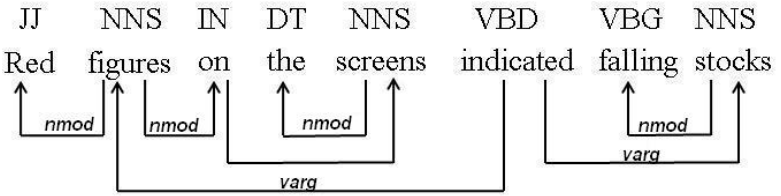
<ROOT, Red figures on the screen indicated falling stocks, ∅>
S <Red ROOT, figures on the screen indicated falling stocks,∅ >
LA <ROOT, figures on the screen indicated falling stocks, {(figures, Red)}>
S <figures ROOT, on the screen indicated falling stocks, {(figures, Red)}>
RA <on figures ROOT, the screen indicated falling stocks, {(figures, Red), (figures, on)}>
S <the on figures ROOT, screen indicated falling stocks, {(figures, Red), (figures, on)}>
LA <on figures ROOT, screen indicated falling stocks, {(figures, Red), (figures, on), (screen the)}>
RA <screen on figures ROOT, indicated falling stocks, {(figures, Red), (figures, on), (screen, the), (on, screen)}>
R <on figures ROOT, indicated falling stocks, {(figures, Red), (figures, on), (screen, the), (on, screen)})
R <figures ROOT, indicated falling stocks, {(figures, Red), (figures, on), (screen, the), (on, screen)}>
LA <ROOT, indicated falling stocks, {(figures, Red), (figures, on), (screen, the), (on, screen), (indicated, figures)}>
RA <indicated ROOT, falling stocks, {(figures, Red), (figures, on), (screen, the), (on, screen), (indicated, figures), (ROOT, indicated)}>
S <falling indicated ROOT, stocks, {(figures, Red), (figures, on), (screen, the), (on, screen), (indicated, figures), (ROOT, indicated)}>
LA <indicated ROOT, stocks, {(figures, Red), (figures, on), (screen, the), (on, screen), (indicated, figures), (ROOT, indicated), (stocks, falling)}>
RA <stocks indicated ROOT, **nil**, {(figures, Red), (figures, on), (screen, the), (on, screen), (indicated, figures), (ROOT, indicated), (stocks, falling), (indicated, stocks)}>
R <indicated ROOT, **nil**, {(figures, Red), (figures, on), (screen, the), (on, screen), (indicated, figures), (ROOT, indicated), (stocks, falling), (indicated, stocks)}>
R <ROOT, **nil**, {(figures, Red), (figures, on), (screen, the), (on, screen), (indicated, figures), (ROOT, indicated), (stocks, falling), (indicated, stocks)}>

Properties of Nivre's Algorithm

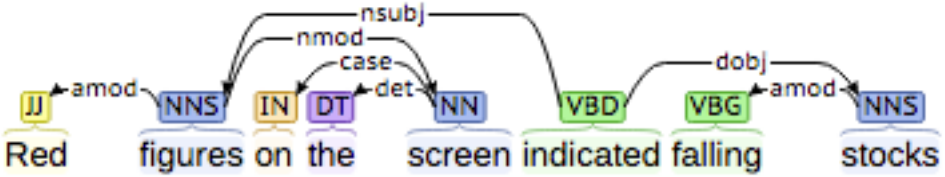
- O(n) : Linear time complexity.

- Full dependency graphs are well-formed.

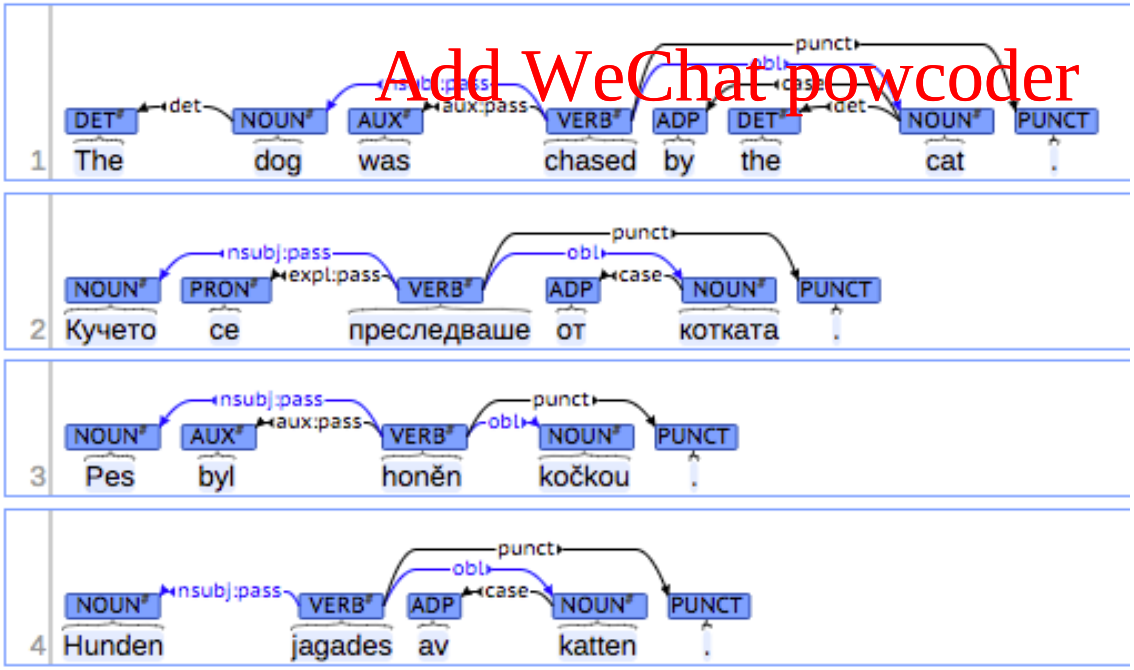
Dependency Corpora



- CoNLL dependencies.
  - http://www.aclweb.org/anthology/D07-1096



- Stanford typed dependencies.
  - http://nlp.stanford.edu/software/dependencies_manual.pdf
- Universal dependencies.
  - http://universaldependencies.org/overview/syntax.html



Guided Parsing [6]

- Train a classifier to predict parse transitions!
  - $f : configuration \rightarrow \{LA, RA, R, S\} \times (A \cup \{\text{nil}\})$
  - A is a set of typed dependencies (arcs).

- Feature space:

TOP	The token on top of the stack
TOP.POS	The part-of-speech of TOP
TOP.DEP	The dependency type of TOP (if any)
TOP.LEFT	The dependency type of TOP's leftmost dependent (if any)
TOP.RIGHT	The dependency type of TOP's rightmost dependent (if any)
NEXT	The next input token
NEXT.POS	The part-of-speech of NEXT
NEXT.LEFT	The dependency type of NEXT's leftmost dependent (if any)
LOOK.POS	The part-of-speech of the next plus one input token

Arc Standard

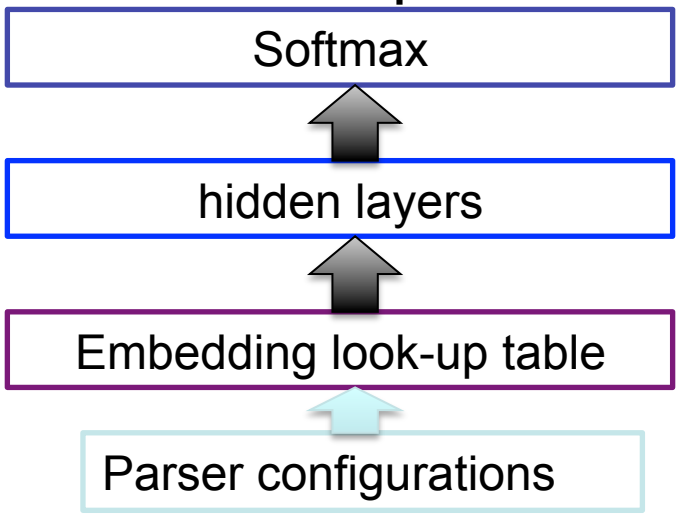
- Three parse actions.

$$\textbf{Left-Arc (LA)} \qquad < v_i|v_j|S, I, A > \Rightarrow < v_i|S, I, A \cup \{(v_i, v_j)\}$$

$$\textbf{Right-Arc (RA)} \qquad < v_i|v_j|S, I, A > \Rightarrow < v_j|S, I, A \cup \{(v_j, v_i)\}$$

$$\textbf{Shift (S)} \qquad < S, v_j|I, A > \Rightarrow < v_j|S, I, A >$$

- Neural networks for action prediction [9].



Off-the-Shelf Dependency Parsers

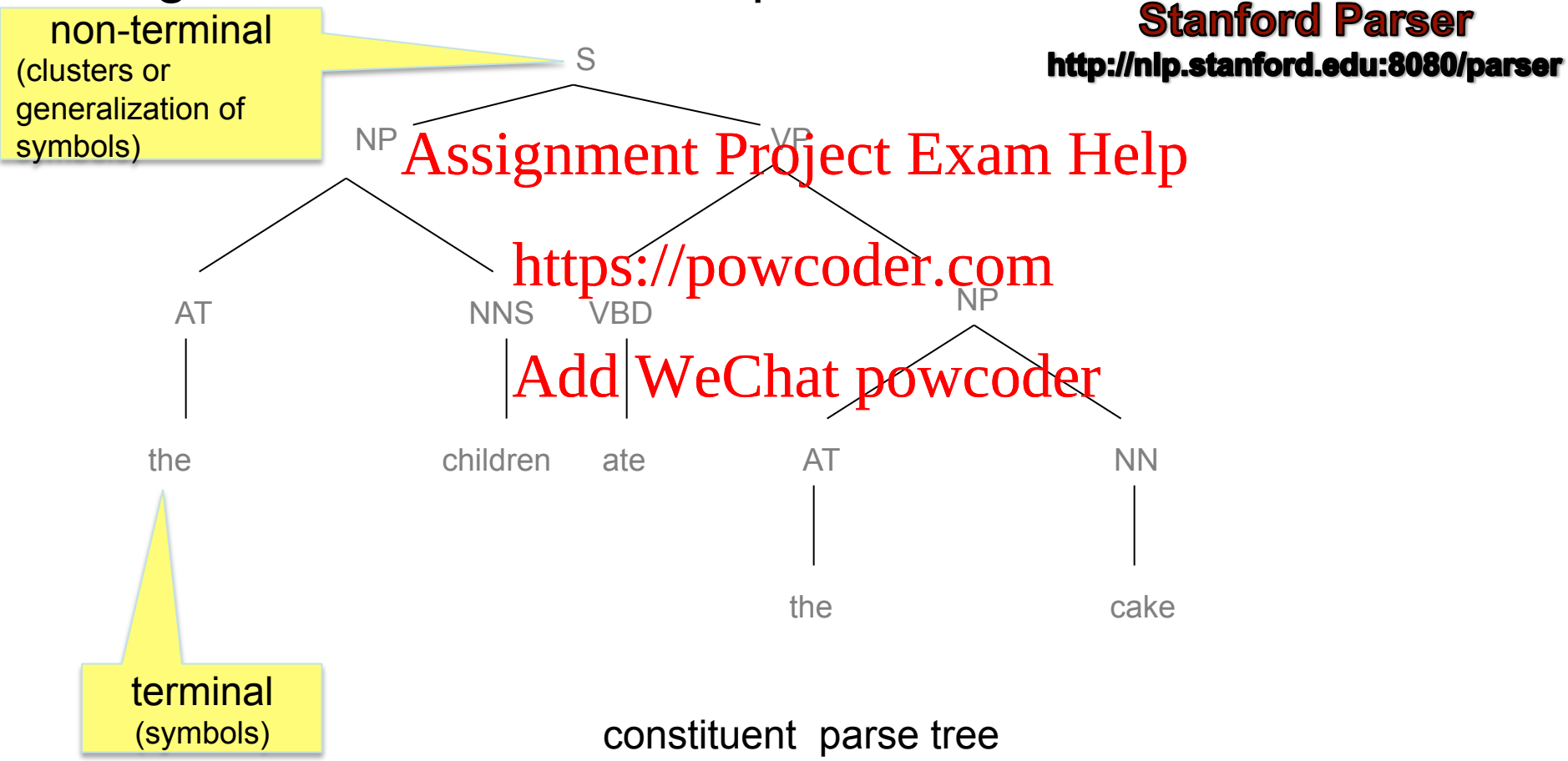
- MaltParser (http://www.maltparser.org/)
- SyntaxNet (https://github.com/tensorflow/models/tree/master/research/syntaxnet)
- Stanford parser (http://nlp.stanford.edu/software/lex-parser.shtml)
- TurboParser (http://www.cs.cmu.edu/~ark/TurboParser/)

Overview of the NLP Lectures

- Introduction to natural language processing (NLP).
- Regular expressions, sentence splitting, tokenization, part-of-speech tagging.
- Language models.
- Vector semantics.
- Parsing.
  - Dependency parsing.
  - Constituency parsing.
- Compositional semantics and NLP applications.

Constituency Parsing

- Deeper understanding of word groups and their grammatical relationships.



constituent parse tree

Constituency

- Constituent: a word or a group of words that behaves as a single unit.

- Why do these words group together?
  - Appear in similar syntactic environments.

three parties from Sydney *arrive* …
Drunk driver *fled* …
they *sit* …

from *arrive* …
the *fled* …
as *sit* …

  - Preposed or postposed construction.

On August 30th, I'd like to fly from Canberra to Sydney.
I'd like to fly on August 30th from Canberra to Sydney.
I'd like to fly from Canberra to Sydney on August 30th.

Context-Free Grammars (CFGs)

- A context free grammar consists of
  - a set of context-free rules, each of which expresses the ways that symbols of the language can be grouped and ordered together.

$$\text{NP} \rightarrow \text{Det Nominal}$$

$$\text{Nominal} \rightarrow \text{Noun}$$

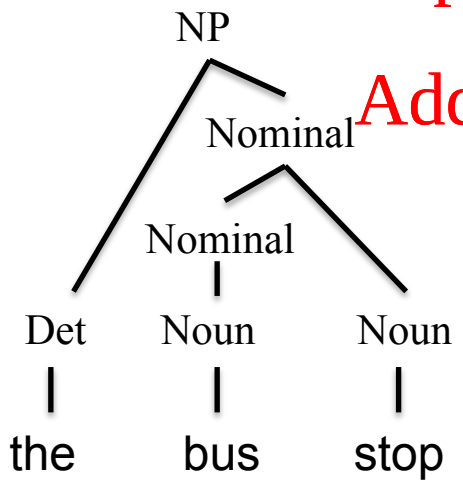
$$\text{Nominal} \rightarrow \text{Noun} \mid \text{Nominal Noun}$$

  - a lexicon of words and symbols.

bus
stop
the
.
a
…

Derivations

- The sequence of rule expansions is called a derivation of the string of words.
  - parse tree.
  - bracketed notation.



[*NP* [*Det* the][*Nom* [*Nom* [*Noun* bus]][*Noun* stop]]

A Toy Example

Noun → bus
Noun → stop
Det → the | a | an

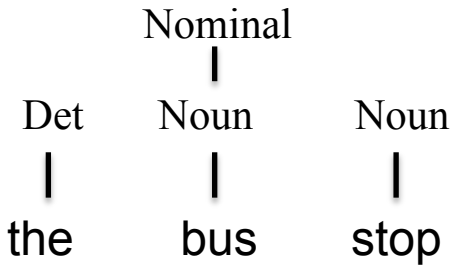
Nominal → Noun
NP → Det Nominal
Nominal → Noun | Nominal Noun

the bus stop

A Toy Example

Noun → bus
Noun → stop
Det → the | a | an

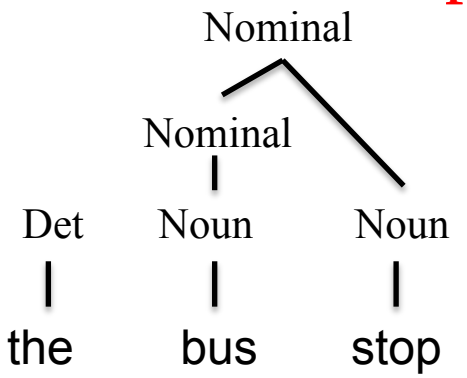
Nominal → Noun
NP → Det Nominal
Nominal → Noun | Nominal Noun



A Toy Example

Noun → bus
Noun → stop
Det → the | a | an

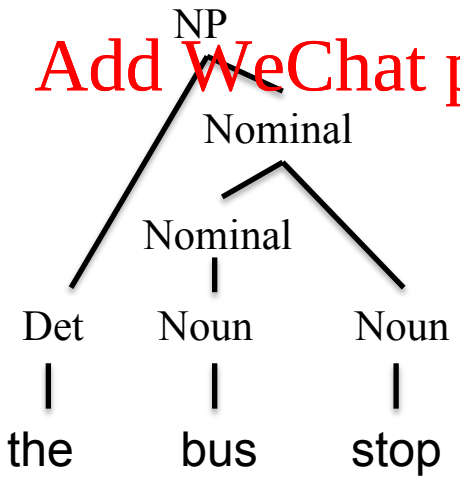
Nominal → Noun
NP → Det Nominal
Nominal → Noun | Nominal Noun



A Toy Example

Noun → bus
Noun → stop
Det → the | a | an

Nominal → Noun
NP → Det Nominal
Nominal → Noun | Nominal Noun

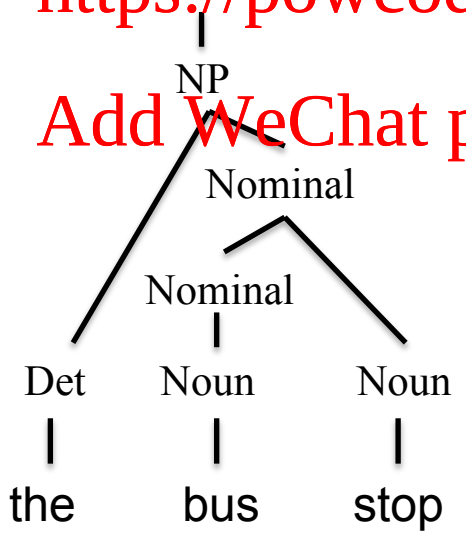


Formal Definition of CFG

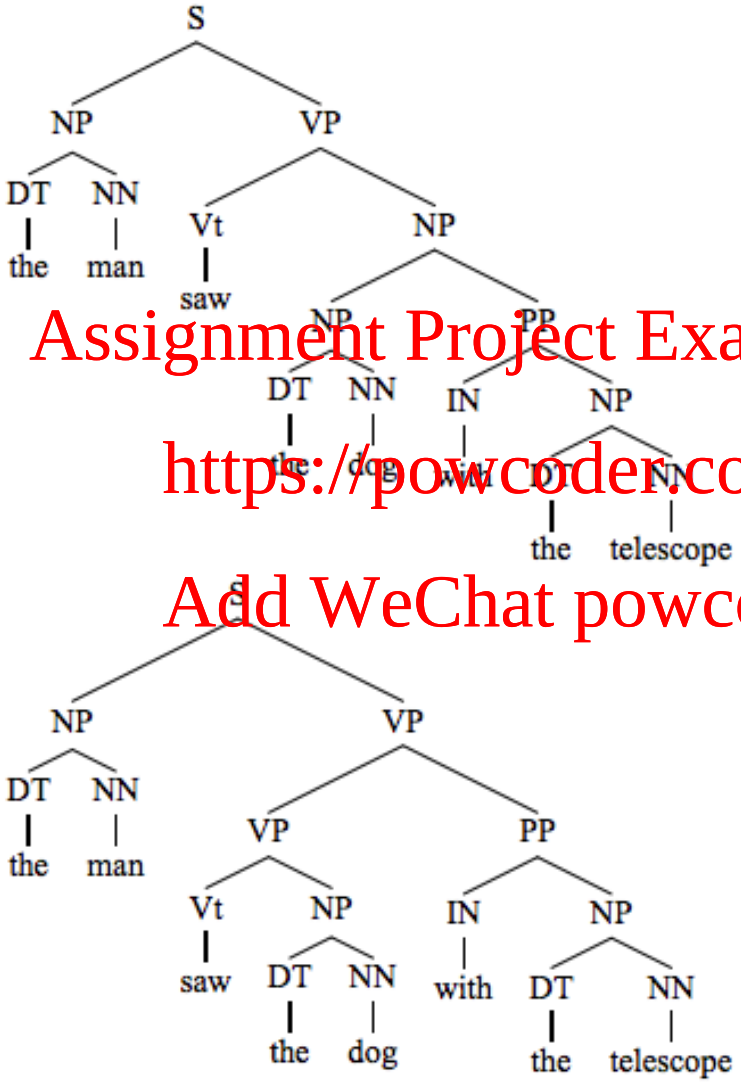
- A context-free grammar $G = (N, \Sigma, R, S)$.
  - N is a set of non-terminals.
  - Σ is a set of terminal symbols, $N \cap \Sigma = \emptyset$.
  - R is a set of rules (productions), each of the form $A \rightarrow B$, where A is a non-terminal, B is a string of symbols from the infinite set of strings $\{\Sigma \cup N\}^*$.
  - S is a designated start symbol.

A Toy Example

Noun → bus
Noun → stop
Det → the | a | an
S → NP
Nominal → Noun
NP → Det Nominal
Nominal → Noun | Nominal Noun



Ambiguity of Parsing



Probabilistic context-free grammar (PCFG)

- A parameter to each grammar rule [3].

S	⇒	NP	VP	1.0
VP	⇒	Vi		0.4
VP	⇒	Vt	NP	0.4
VP	⇒	VP	PP	0.2
NP	⇒	DT	NN	0.3
NP	⇒	NP	PP	0.7
PP	⇒	P	NP	1.0

Vi	⇒	sleeps	1.0
Vt	⇒	saw	1.0
NN	⇒	man	0.7
NN	⇒	woman	0.2
NN	⇒	telescope	0.1
DT	⇒	the	1.0
IN	⇒	with	0.5
IN	⇒	in	0.5

$$p_G(t) = \prod_{i=1}^{n} q(\alpha \rightarrow \beta)$$

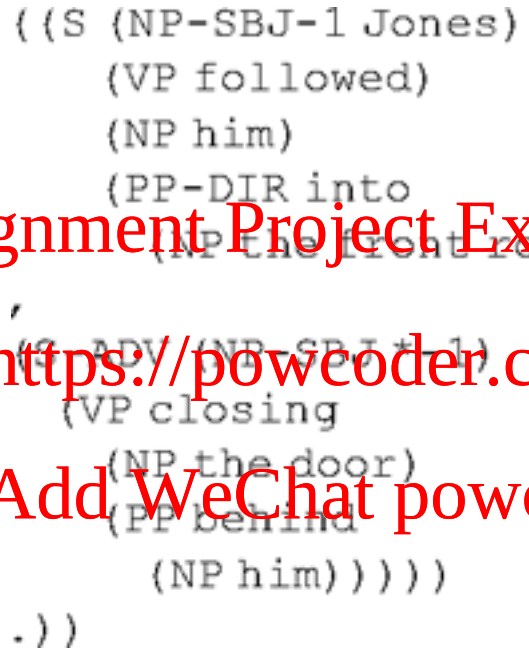
rule parameter

$$\arg\max_{t \in T_G} p_G(t)$$

find the most likely parse tree. T is set of all possible trees.

Learning PCFG from Treebanks

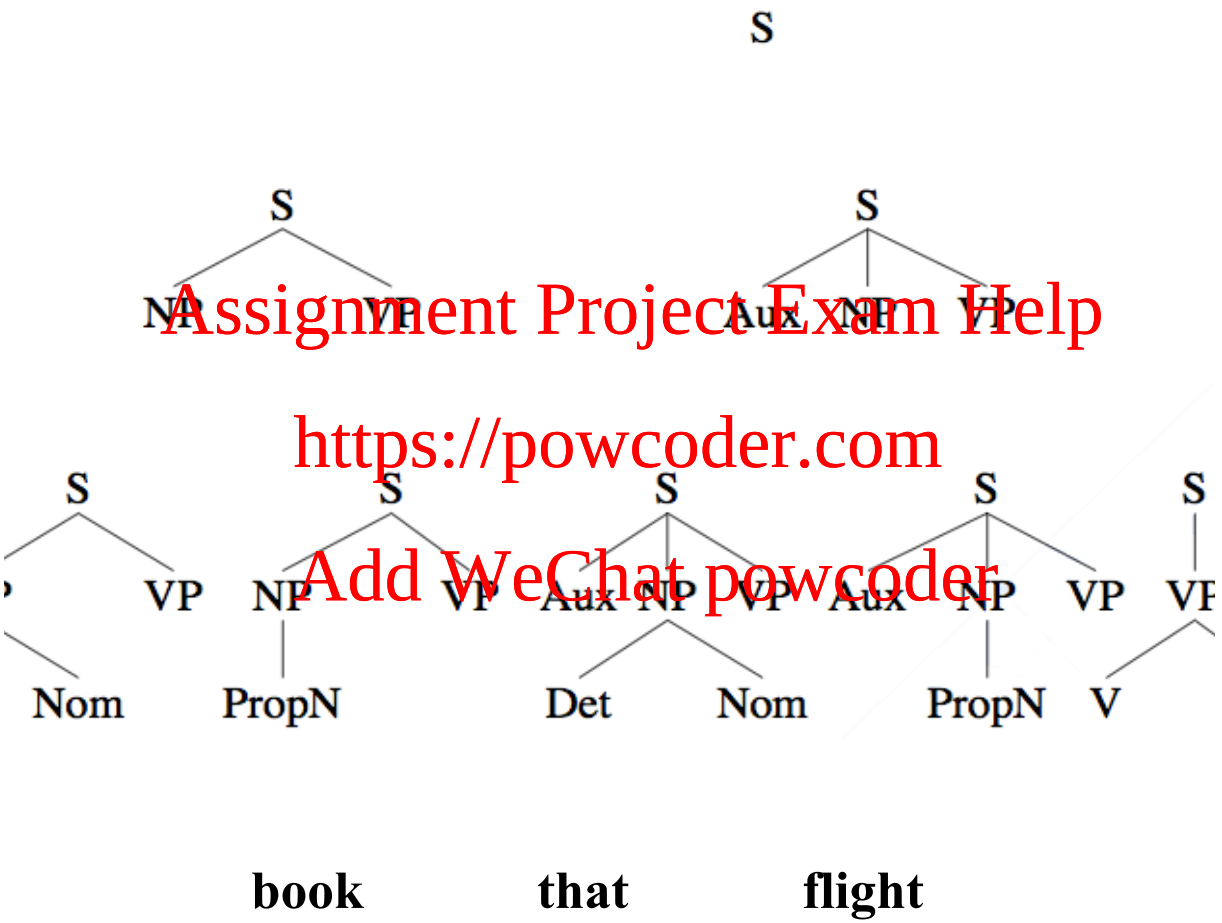
- Penn treebank and English Web treebank.



Maximum-Likelihood estimation: $q^*(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$

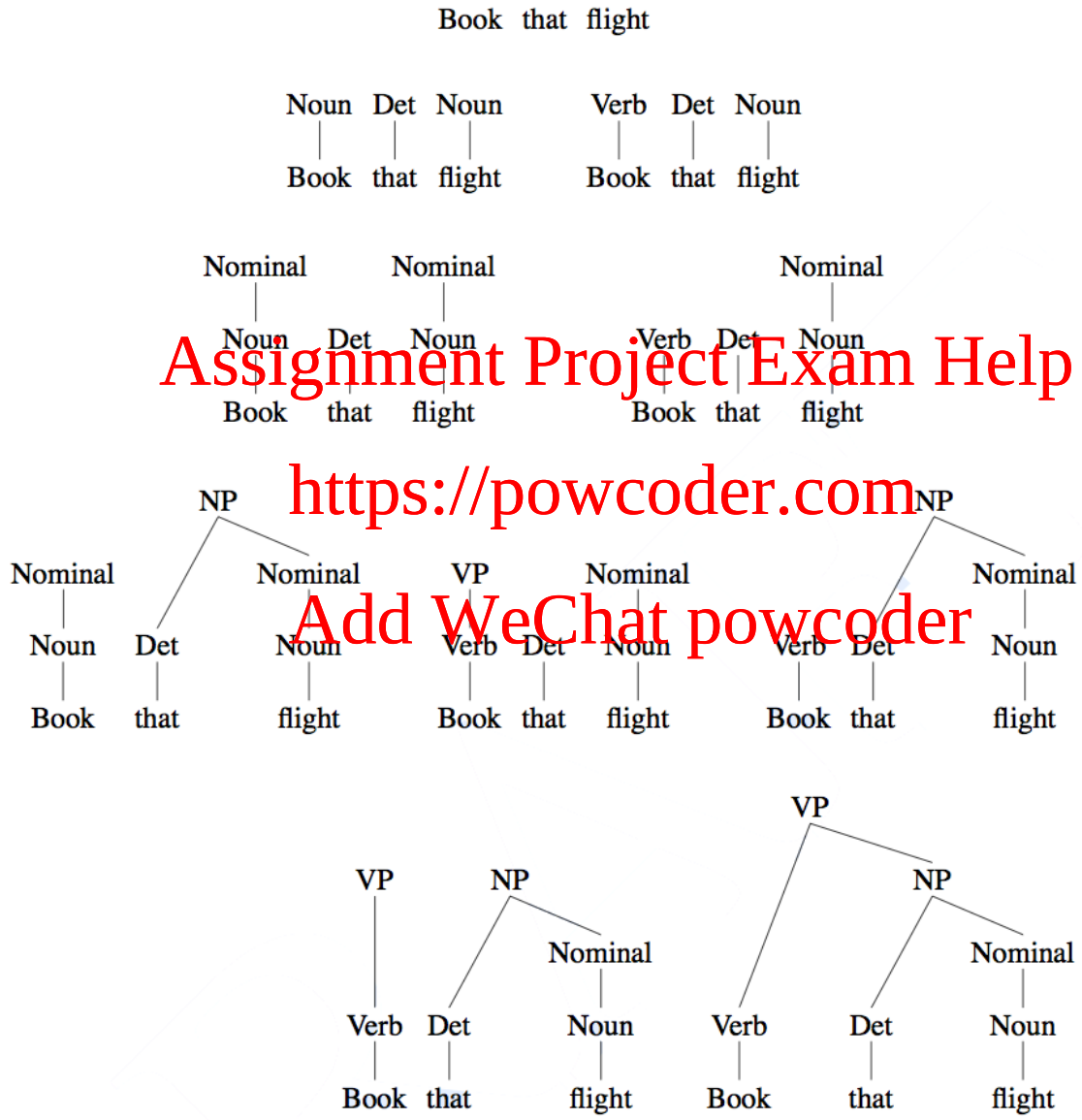
Top Down Parsing

$$\arg\max_{t \in T_G} p_G(t)$$



Bottom Up Parsing

$$\arg\max_{t \in T_G} p_G(t)$$



Grammar Equivalence

- Two grammars are equivalent if they generate the same language (set of strings) .
- Chomsky Normal Form (CNF).
  - Allow only two types of rules. The right-hand side of each rule either has two non-terminals or one terminal,

except $S \to \varepsilon$

$$A \to B\ a\ D$$

$$C \to \varepsilon$$

$$E \to A$$

unit production

where $A, B, C, D, E \in N$ and $a \in \Sigma$

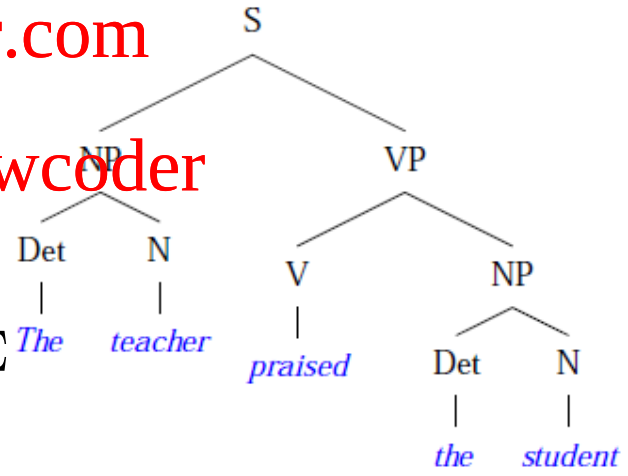
Grammar Equivalence

- Two grammars are equivalent if they generate the same language (set of strings) .
- Chomsky Normal Form (CNF).
  - Allow only two types of rules. The right-hand side of each rule either has two non-terminals or one terminal,

except $S \to \varepsilon$

$$G \to E\ D$$
$$F \to B\ G$$
$$E \to a$$

where $B, D, E, F, G \in N$ and $a \in \Sigma$



  - Every context-free grammar can be transformed into an equivalent one in CNF.

Dependency Structures vs. Phrase Structures

- Dependency structures explicitly represent
  - Head-dependent relations (directed arcs).
  - Functional categories (arc labels).
  - predicate-argument structure.
- Dependency structure independent of word order.
  - Suitable for free word order languages, such as Indian languages.
- Phrase structures explicitly represent
  - Phrases (non-terminal nodes).
  - Structural categories (non-terminal labels).
  - Fragments are directly interpretable.

Available Constituency Parsers

- Stanford parser.
  - http://nlp.stanford.edu/software/srparser.shtml
- Charniak-Johnson parser.
  - http://web.science.mq.edu.au/~mjohnson/Software.htm
- Charniak parser.
  - ftp://ftp.cs.brown.edu/pub/nlparser/

References

- [1] Speech and Language Processing (Chapter 12 & 13)
- [2] Lectures Notes on CFGs and CKY Parsing.
  - http://web.mit.edu/6.863/www/fall2012/lectures/lecture7-notes.pdf
- [3] http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf
- [4] Richard Socher, Christopher D. Manning, Andrew Y. Ng. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks.
- [5] An Efficient Algorithm for Projective Dependency Parsing (We modify it to allow an artificial tree root.)
- [6] Joakim Nivre, Johan Hall, Jens Nilsson. Memory-Based Dependency Parsing.
- [7] Exploiting Background Knowledge for Relation Extraction.
  - http://www.aclweb.org/anthology/C10-1018
- [8] https://web.stanford.edu/~jurafsky/slp3/14.pdf
- [9] https://cs.stanford.edu/~danqi/papers/emnlp2014.pdf