TTIC 31190:
Natural Language Processing

Kevin Gimpel

Spring 2018

Lecture 14:

Syntax and Parsing

Project Proposal

- project proposal due today

Midterm

- midterm in one week
- you can bring notes
  - but we'll try to give you all formulas/definitions you'll need
- Monday: review for midterm, including sample questions

Roadmap

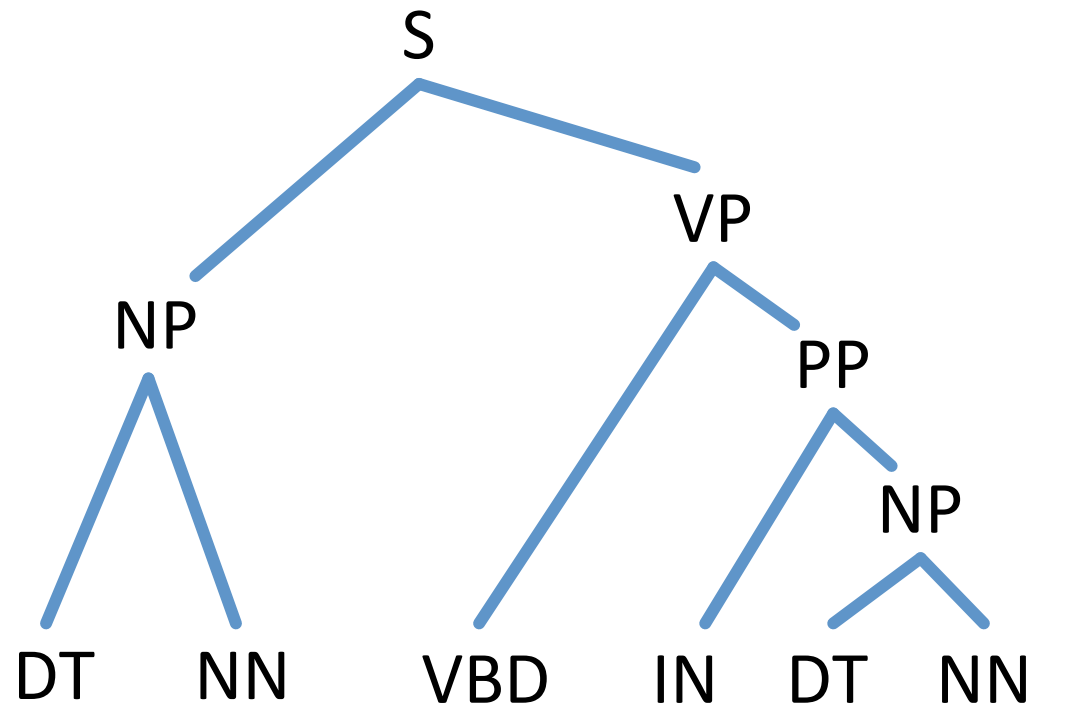
- words, morphology, lexical semantics
- text classification
- language modeling
- word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

What is Syntax?

- rules, principles, processes that govern sentence structure of a language

Constituent Parse (Bracketing/Tree)

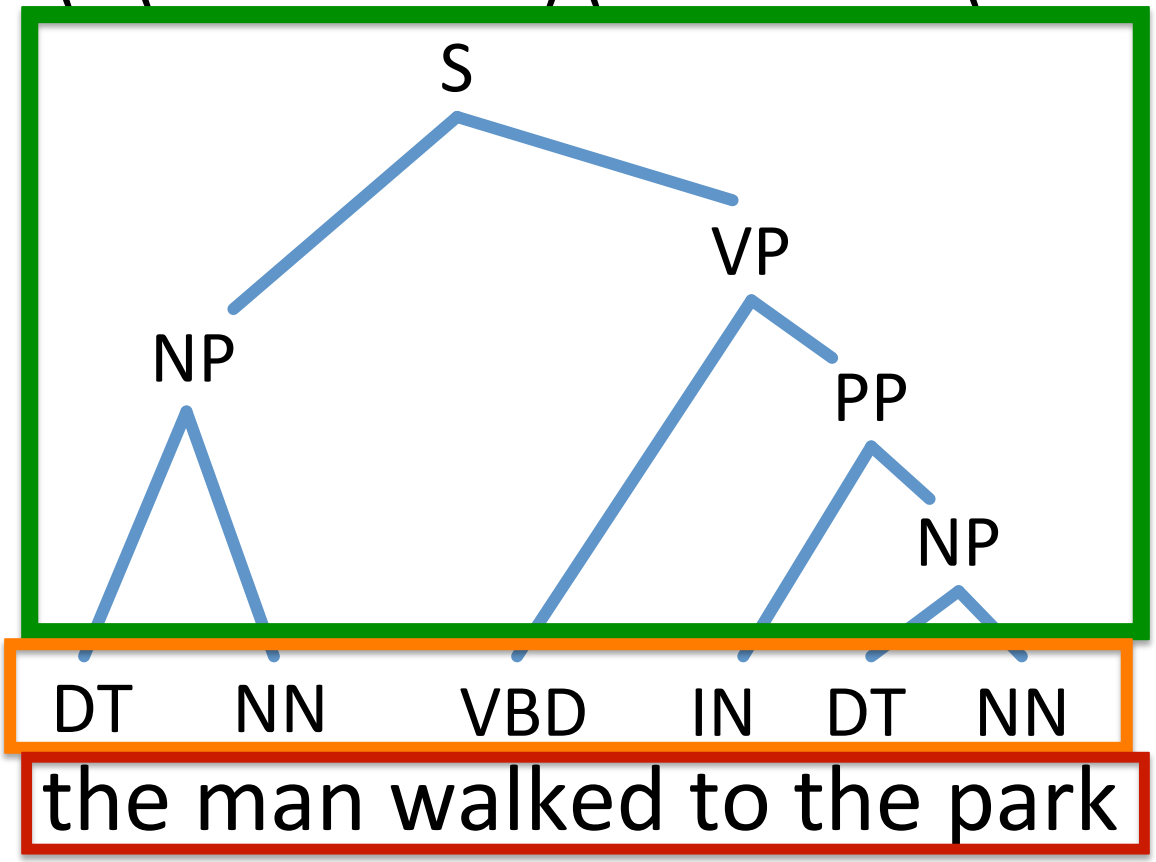
(S (NP the man) (VP walked (PP to (NP the park))))



Key:
S = sentence
NP = noun phrase
VP = verb phrase
PP = prepositional phrase
DT = determiner
NN = noun
VBD = verb (past tense)
IN = preposition

Constituent Parse

(S (NP the man) (VP walked (PP to (NP the park))))

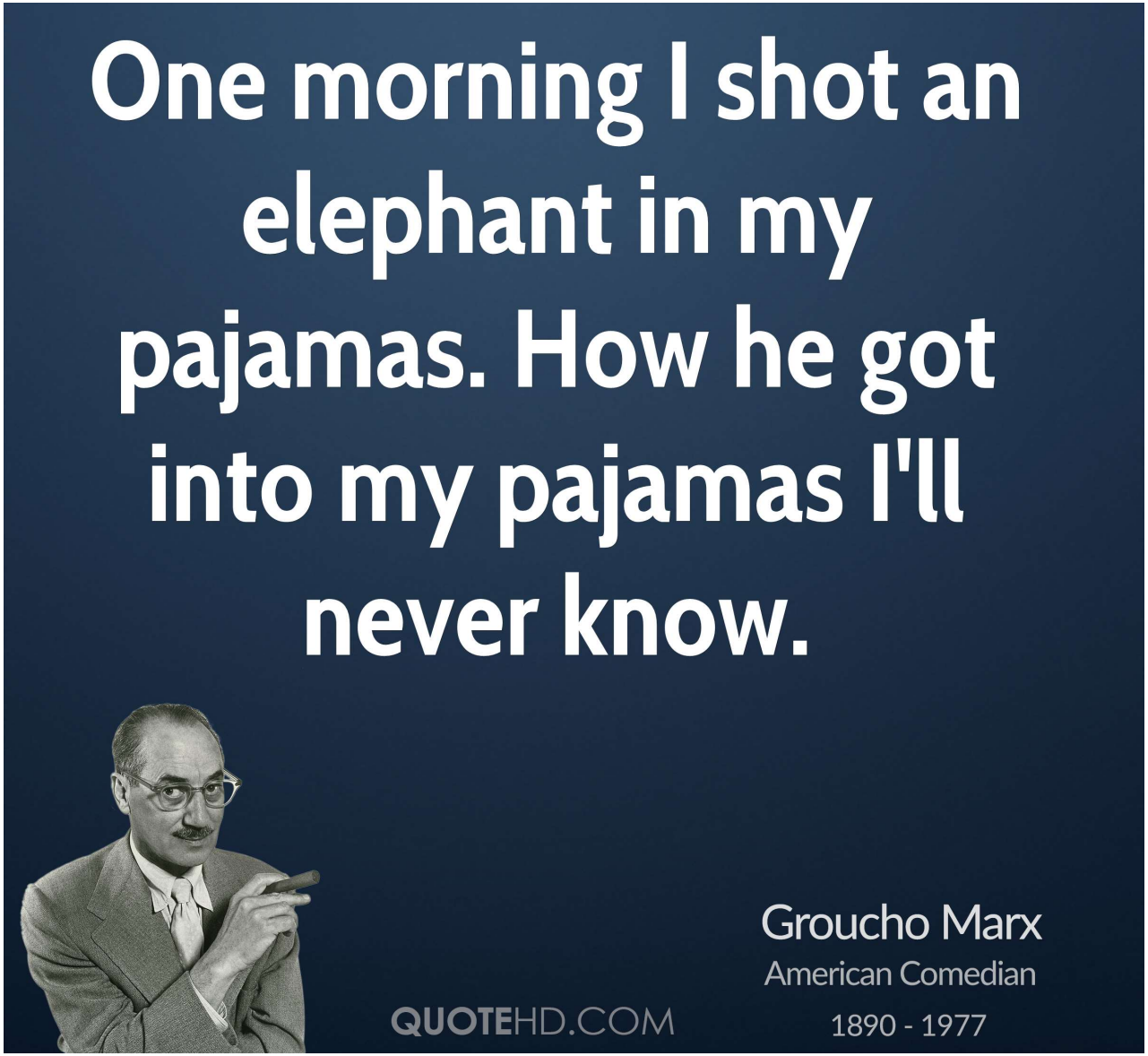


nonterminals

preterminals

terminals

Attachment Ambiguity



NLP Task: Constituent Parsing

- given a sentence, output its constituent parse
- widely-studied task with a rich history
- most based on the Penn Treebank (Marcus et al.), developed at Penn in early 1990s

- Treebank = “corpus of annotated parse trees”

Context-Free Grammar (CFG)

- has “rewrite rules” to rewrite nonterminals as terminals or other nonterminals

  S → NP VP

  “S goes to NP VP”

  NP → DT NN

  VP → VBD PP

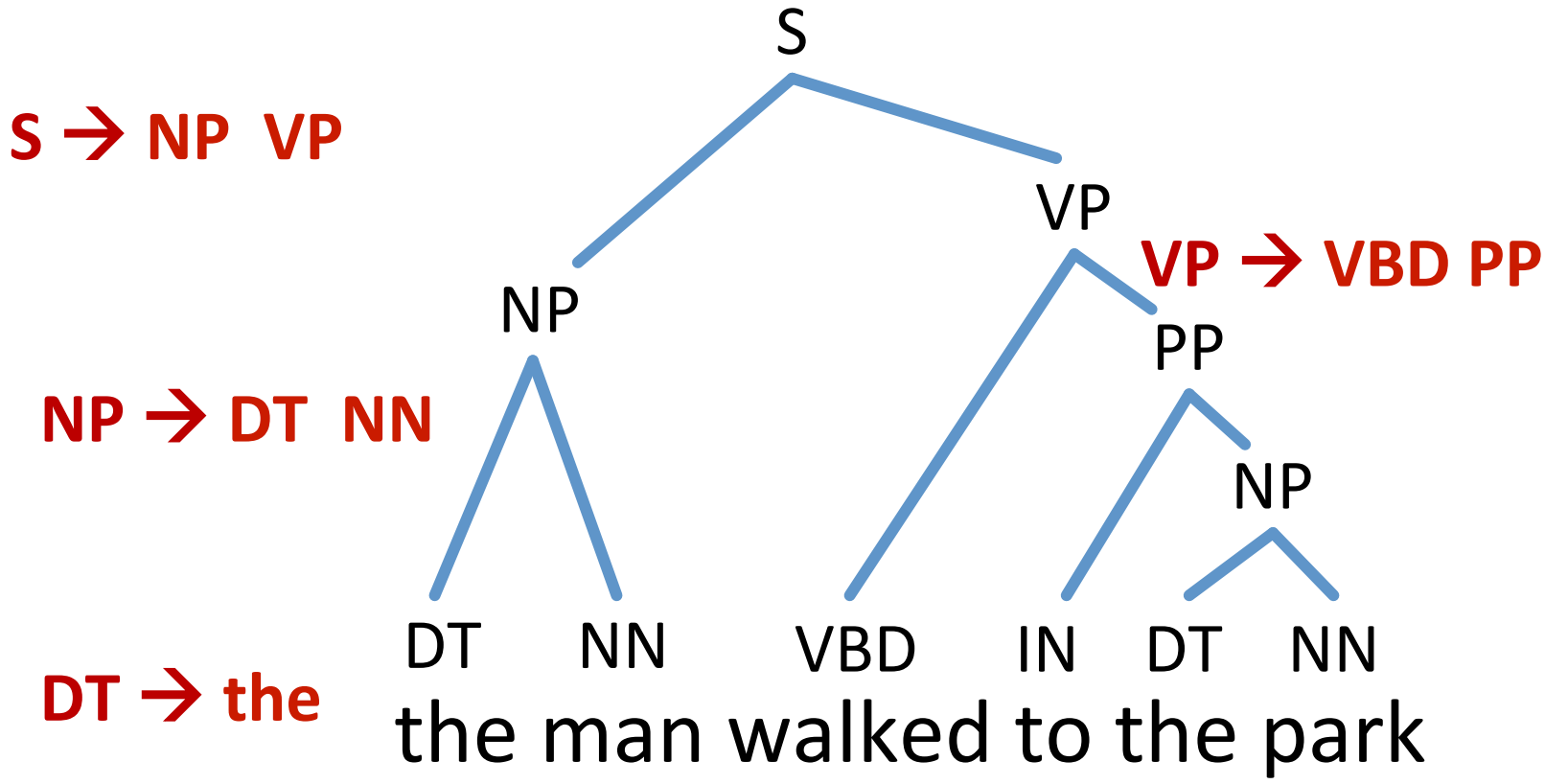
  PP → IN NP

  NN → man

  DT → the

Context-Free Grammar (CFG)

- sequence of rewrites corresponds to a bracketing (induces a hierarchical tree structure)



Why “context-free”?

- a rule to rewrite NP does not depend on the context of NP
- that is, the left-hand side of a rule is only a single non-terminal (without any other context)

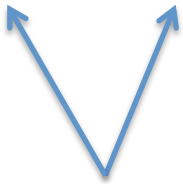
Probabilistic Context-Free Grammar (PCFG)

- assign probabilities to rewrite rules:

NP → DT NN 0.5

NP → NNS 0.3

NP → NP PP 0.2



same nonterminal can be on both left and right sides

Probabilistic Context-Free Grammar (PCFG)

- assign probabilities to rewrite rules:

  NP → DT NN 0.5

  NP → NNS 0.3

  NP → NP PP 0.2

  probabilities must sum to one for each left-hand side nonterminal

Probabilistic Context-Free Grammar (PCFG)

- assign probabilities to rewrite rules:

NP → DT NN	0.5
NP → NNS	0.3
NP → NP PP	0.2
NN → man	0.01
NN → park	0.0004
NN → walk	0.002
NN → ….	

given a treebank, we can estimate these probabilities using maximum likelihood estimation ("count and normalize")

just like n-gram language models and HMMs for POS tagging

Probabilistic Context-Free Grammar (PCFG)

- for each nonterminal, a PCFG has a probability distribution over possible right-hand side sequences
- so, a PCFG assigns probabilities to:
  - bracketings of sentences
  - sequences of rewrite operations (**derivations**) that eventually terminate in terminals
  - hierarchical tree structures that ground out in sequences of terminals
- these are different ways of saying the same thing

Constituent Parsing

- evaluation: `evalb` score
  - first compute precision and recall (at the level of constituents)
  - then compute F1 (harmonic mean of precision and recall)

Precision, Recall, F1

- **precision**:
  - what fraction of the things I found are good?

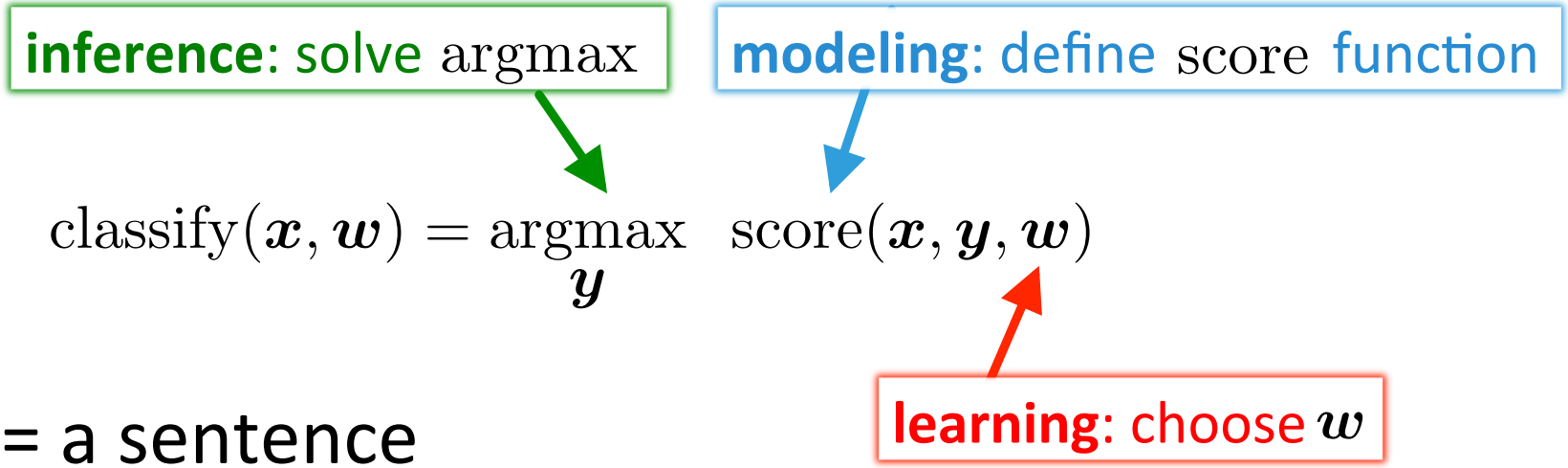
$$P = \frac{|F \cap G|}{|F|}$$

- **recall**:
  - what fraction of good things did I find?

$$R = \frac{|F \cap G|}{|G|}$$

- **F1 score**:
  - harmonic mean of precision and recall

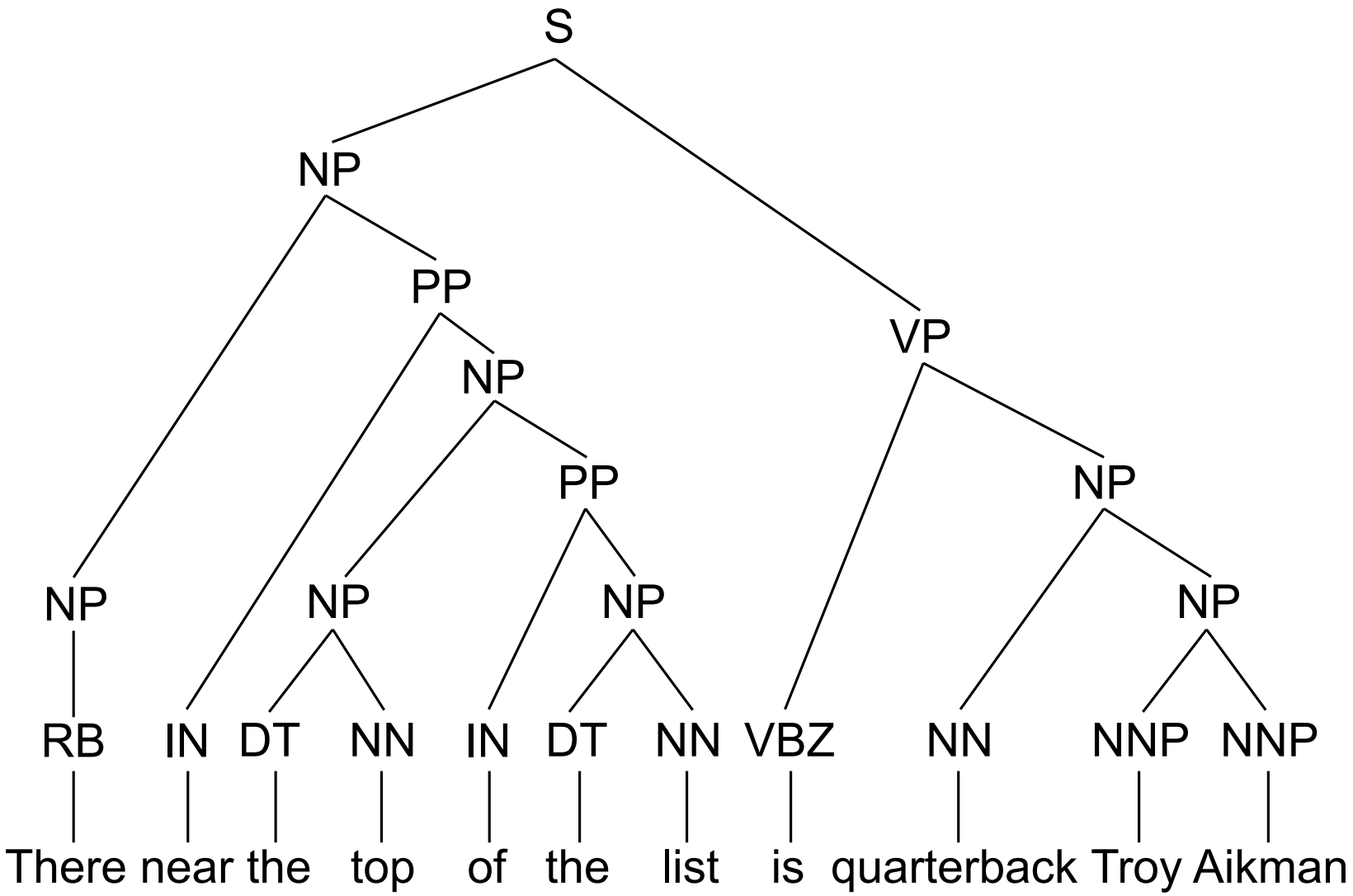
Modeling, Inference, Learning



- ***x*** = a sentence
- ***y*** = a constituent parse
- inference requires iterating over all possible constituent parses!
- this can be done using dynamic programming but is still expensive (cubic in the sentence length)

Inference in PCFGs

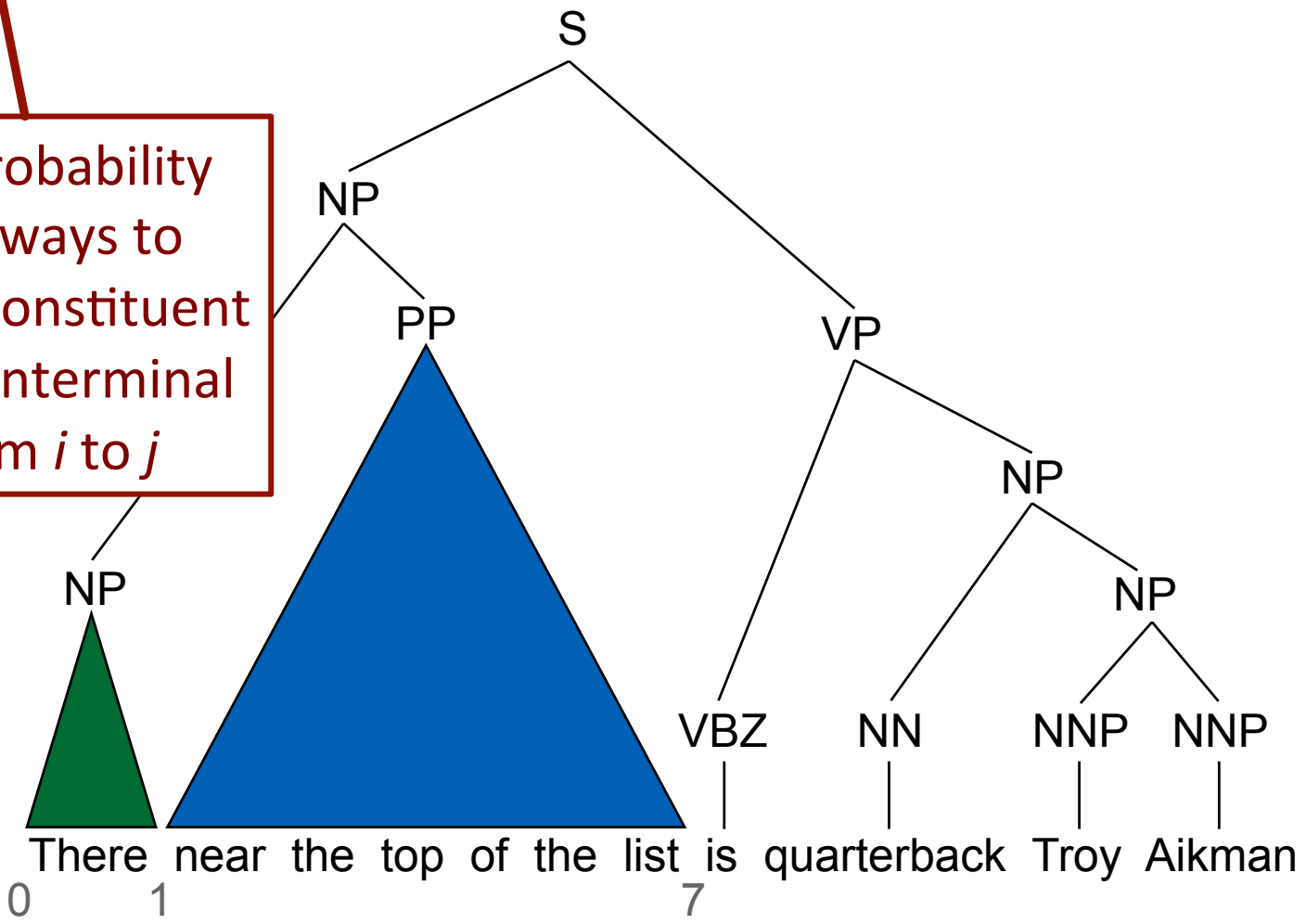
- to find max-probability tree for a sentence, use dynamic programming: **CKY algorithm**
- to find the best way to build a tree covering words *i* to *j*:
  - consider all possible "split points" *k* between *i* and *j*
  - for each split point *k*, consider all possible nonterminals for the two smaller trees created by that split


S
NP
PP
VP
NP
PP
NP
NP
NP
NP
NP
NP
RB
IN
DT
NN
IN
DT
NN
VBZ
NN
NNP
NNP
There
near
the
top
of
the
list
is
quarterback
Troy
Aikman

CKY Algorithm

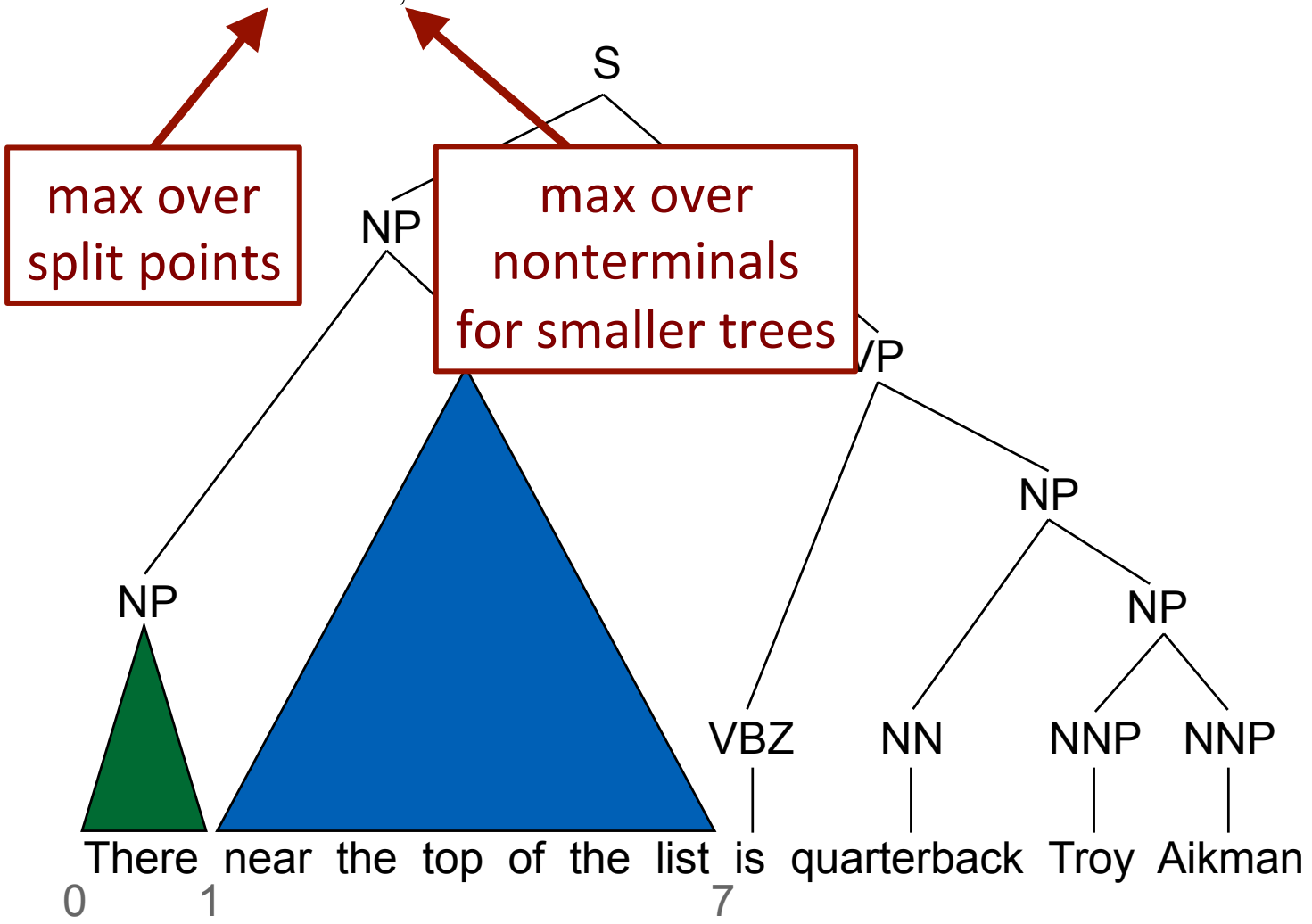
$$C(Z, i, j) = \max_{k} \max_{A,B} \left(C(A, i, k)\, C(B, k, j)\, \text{score}(\langle Z \rightarrow A\, B \rangle) \right)$$

max probability of all ways to build a constituent with nonterminal *Z* from *i* to *j*



CKY Algorithm

$$C(Z, i, j) = \max_{k} \max_{A,B} \left(C(A, i, k)\, C(B, k, j)\, \text{score}(\langle Z \rightarrow A\, B \rangle) \right)$$



- detail: CKY requires the PCFG to be in **Chomsky Normal Form (CNF)**
- basically: every rule has either 2 nonterminals or 1 terminal on the right-hand side

How well does a PCFG work?

- a PCFG learned from the Penn Treebank with maximum likelihood estimation (count & normalize) gets about 73% F1 score
- state-of-the-art parsers are around 92%

How well does a PCFG work?

- a PCFG learned from the Penn Treebank with maximum likelihood estimation (count & normalize) gets about 73% F1 score
- state-of-the-art parsers are around 92%
- but, simple modifications can improve the PCFG a lot!
  - smoothing
  - tree transformations (selective flattening)
  - "parent annotation"

Parent Annotation

$VP \rightarrow V \ NP \ PP$

$VP^{S} \rightarrow V \ NP^{VP} \ PP^{VP}$

adds more information, but also fragments counts, making parameter estimates noisier (since we're just using MLE)

Johnson (1998)

PCFG Models of Linguistic Tree Representations

Mark Johnson*
Brown University

The kinds of tree representations used in a treebank corpus can have a dramatic effect on performance of a parser based on the PCFG estimated from that corpus, causing the estimated likelihood of a tree to differ substantially from its frequency in the training corpus. This paper points out that the Penn II treebank representations are of the kind predicted to have such an effect, and describes a simple node relabeling transformation that improves a treebank PCFG-based parser's average precision and recall by around 8%, or approximately half of the performance difference between a simple PCFG model and the best broad-coverage parsers available today. This performance variation comes about because any PCFG, and hence the corpus of trees from which the PCFG is induced, embodies independence assumptions about the distribution of words and phrases. The particular independence assumptions implicit in a tree representation can be studied theoretically and investigated empirically by means of a tree transformation/detransformation process.

Johnson (1998)

	22	22 Id	Id	NP-VP	N′-V′	Flatten	Parent
Number of rules		2,269	14,962	14,297	14,697	22,652	22,773
Precision	1	0.772	0.735	0.730	0.735	0.745	0.800
Recall	1	0.728	0.697	0.705	0.701	0.723	0.792
NP attachments	279	0	67	330	69	154	217
VP attachments	299	424	384	0	503	392	351
NP* attachments	339	3	67	399	69	161	223
VP* attachments	412	668	662	150	643	509	462

How well does a PCFG work?

- PCFG learned from the Penn Treebank with MLE gets about 73% F1 score
- state-of-the-art parsers are around 92%
- simple modifications can improve PCFGs:
  - smoothing
  - tree transformations (selective flattening)
  - parent annotation

How well does a PCFG work?

- PCFG learned from the Penn Treebank with MLE gets about 73% F1 score
- state-of-the-art parsers are around 92%
- simple modifications can improve PCFGs:
  - smoothing
  - tree transformations (selective flattening)
  - parent annotation
  - **lexicalization**

Collins (1997)

Three Generative, Lexicalised Models for Statistical Parsing

Michael Collins*
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA, 19104, U.S.A.
mcollins@gradient.cis.upenn.edu

Abstract

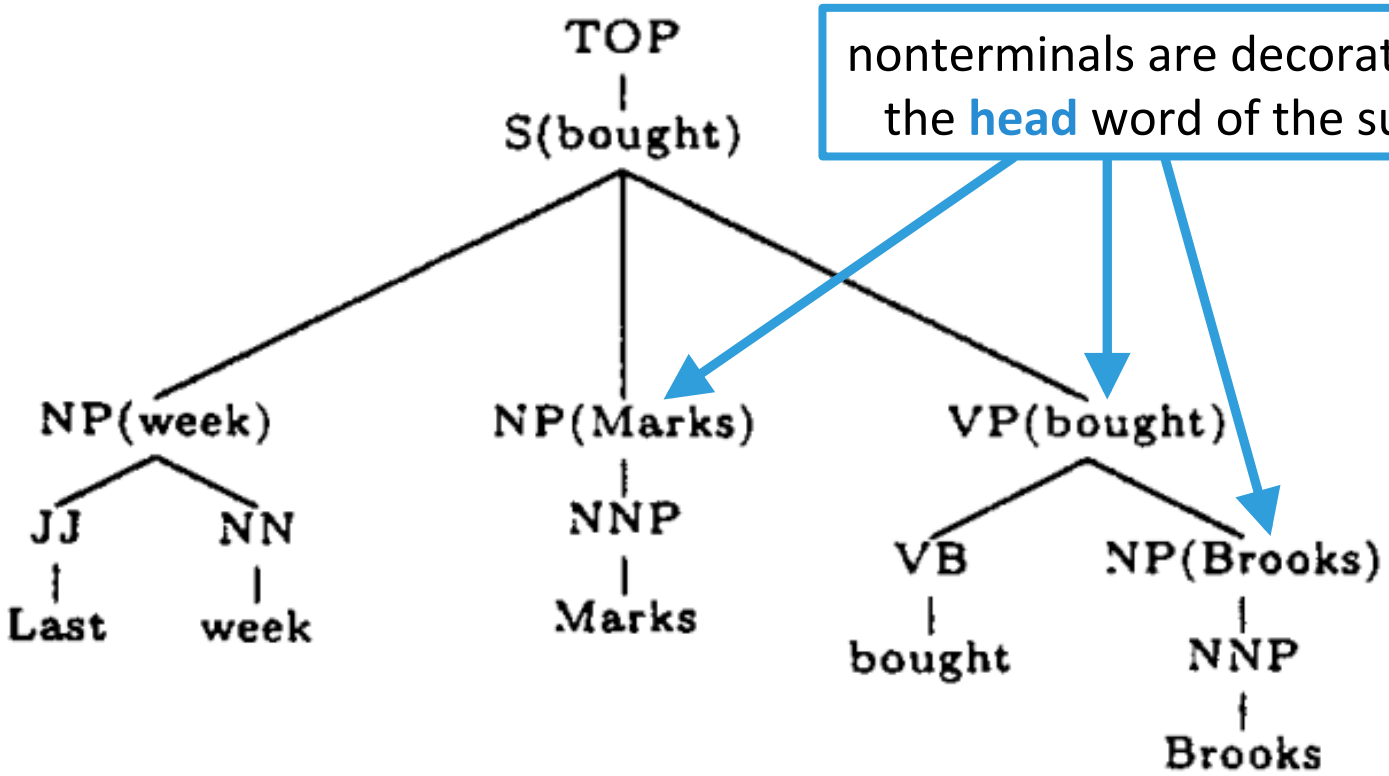
In this paper we first propose a new statistical parsing model, which is a generative model of lexicalised context-free grammar. We then extend the model to include a probabilistic treatment of both subcategorisation and wh-movement. Results on Wall Street Journal text show that the parser performs at 88.1/87.5% constituent precision/recall, an average improvement of 2.3% over (Collins 96).

1 Introduction

Generative models of syntax have been central in linguistics since they were introduced in (Chomis derived from the analysis given in Generalized Phrase Structure Grammar (Gazdar et al. 95). The work makes two advances over previous models: First, Model 1 performs significantly better than (Collins 96), and Models 2 and 3 give further improvements — our final results are 88.1/87.5% constituent precision/recall, an average improvement of 2.3% over (Collins 96). Second, the parsers in (Collins 96) and (Magerman 95; Jelinek et al. 94) produce trees without information about wh-movement or subcategorisation. Most NLP applications will need this information to extract predicate-argument structure from parse trees.

In the remainder of this paper we describe the 3 models in section 2, discuss practical issues in section 3, give results in section 4, and give conclusions in section 5.

Lexicalized PCFGs




```
TOP                 ->   S(bought)
S(bought)           ->   NP(week)      NP(Marks)      VP(bought)
NP(week)            ->   JJ(Last)      NN(week)
NP(Marks)           ->   NNP(Marks)
VP(bought)          ->   VB(bought)    NP(Brooks)
NP(Brooks)          ->   NNP(Brooks)
```

Lexicalization

- this adds a lot more rules!
- many more parameters to estimate → smoothing becomes much more important
  - e.g., right-hand side of rule might be factored into several steps
- but it's worth it because head words are really useful for constituent parsing

Results (Collins, 1997)

MODEL	≤ 40 Words (2245 sentences)				
	LR	LP	CBs	0 CBs	≤ 2 CBs
(Magerman 95)	84.6%	84.9%	1.26	56.6%	81.4%
(Collins 96)	85.8%	86.3%	1.14	59.9%	83.6%
Model 1	87.4%	88.1%	0.96	65.7%	86.3%
Model 2	88.1%	88.6%	0.91	66.5%	86.9%
Model 3	88.1%	88.6%	0.91	66.4%	86.9%

Head Rules

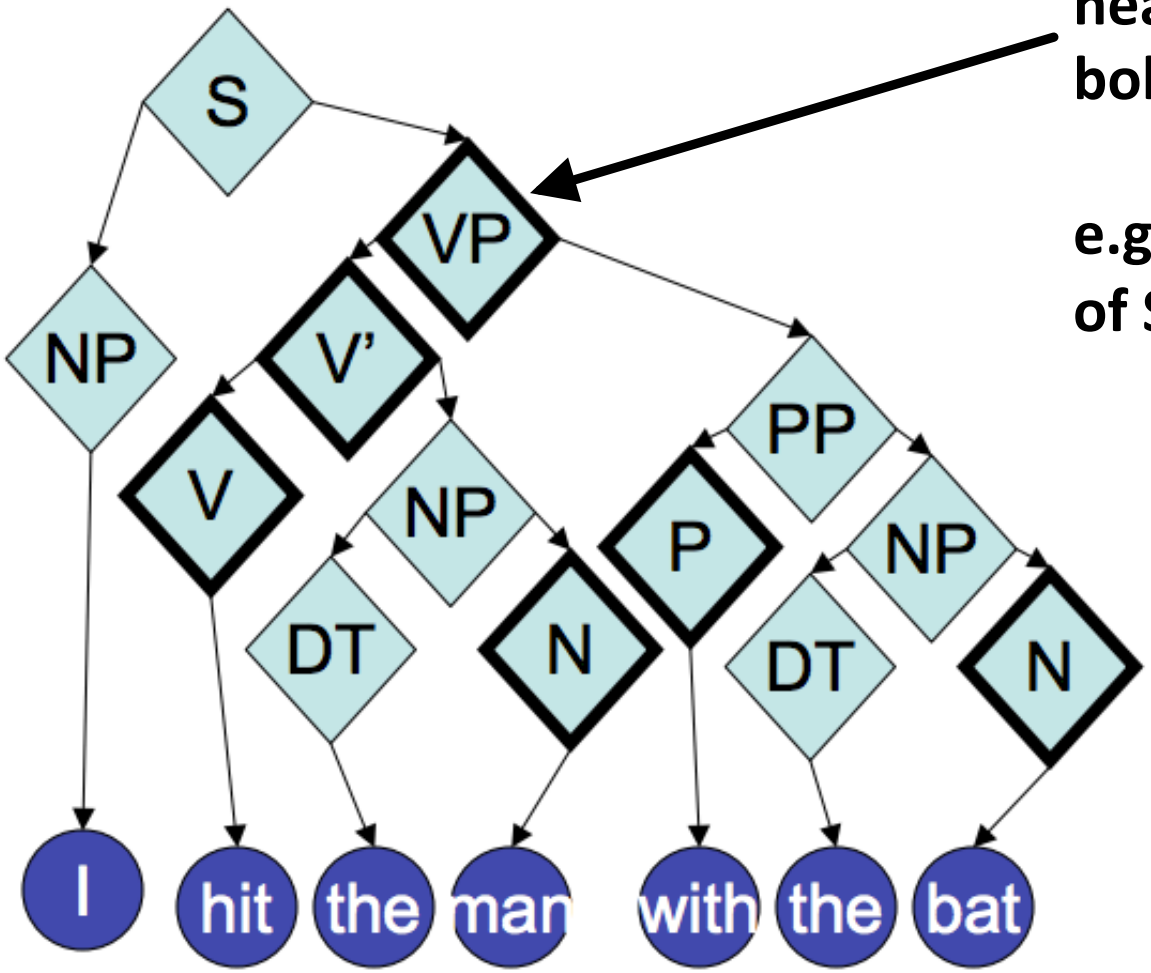
- how are heads decided?
- initially, researchers used deterministic head rules (Magerman/Collins)
- for a PCFG rule A → B_1 … B_N, these head rules say which of B_1 … B_N is the head of the rule
- examples:

  S → NP <u>VP</u>

  VP → <u>VBD</u> NP PP

  NP → DT JJ <u>NN</u>

Head Annotation

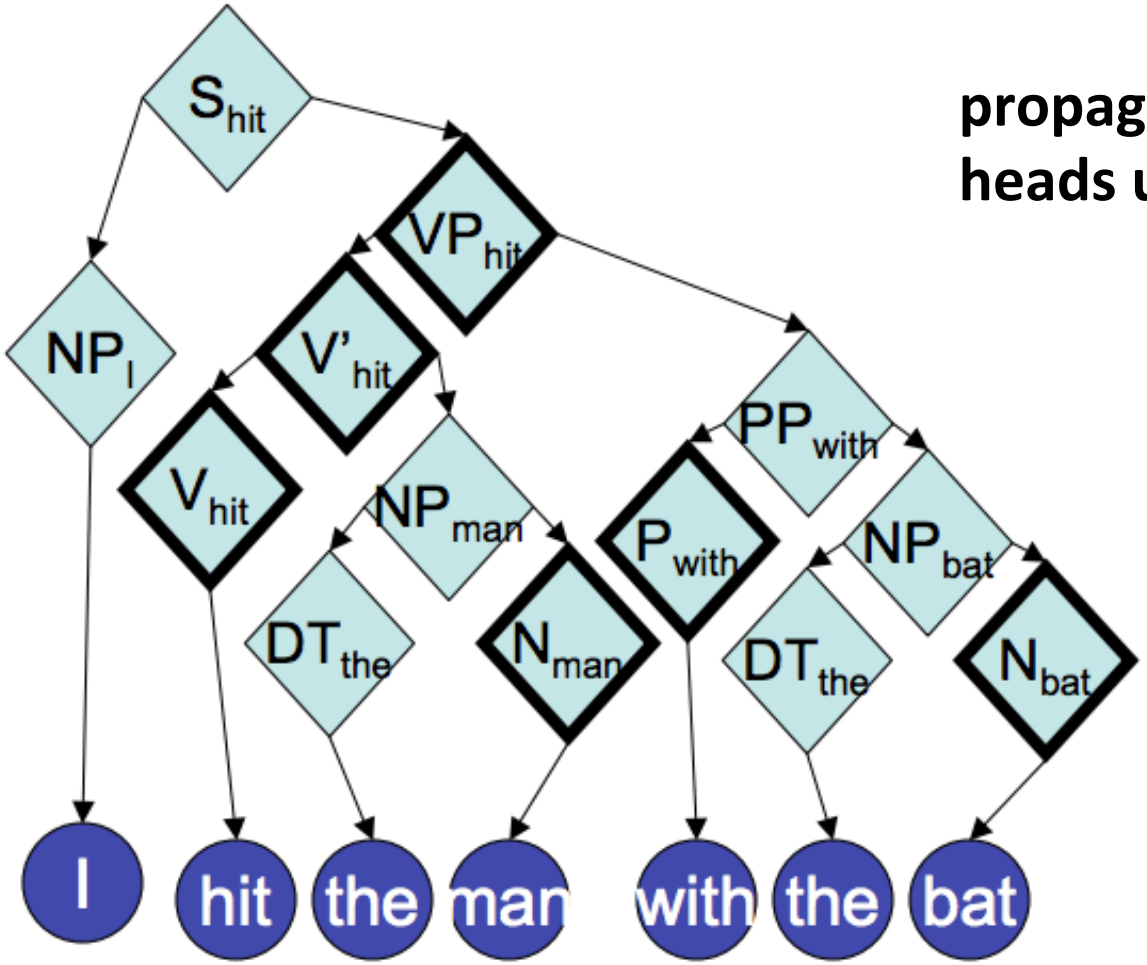


heads have bold outline

e.g., VP is head of S -> NP VP

from Noah Smith

Lexical Head Annotation

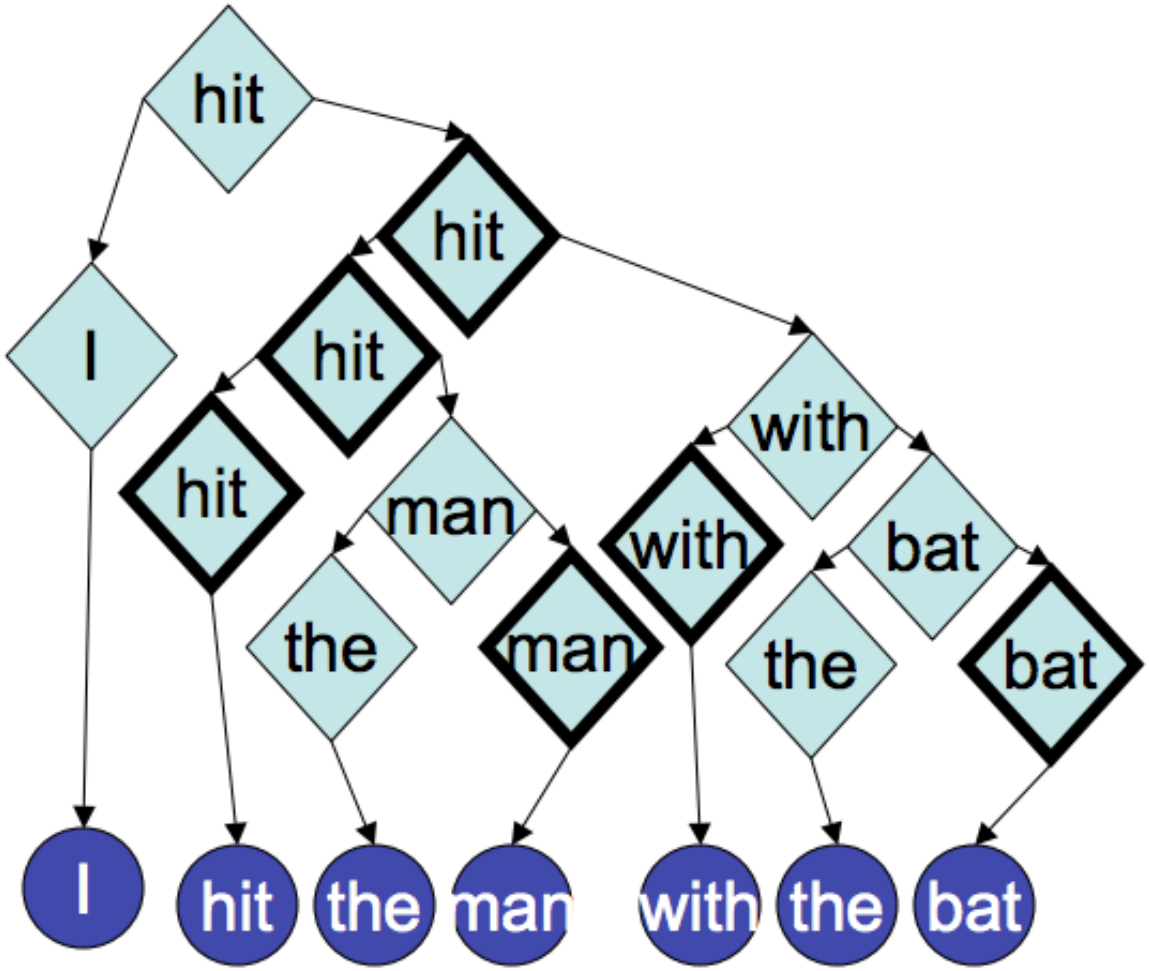


propagate lexical heads up the tree

from Noah Smith

Lexical Head Annotation → Dependencies

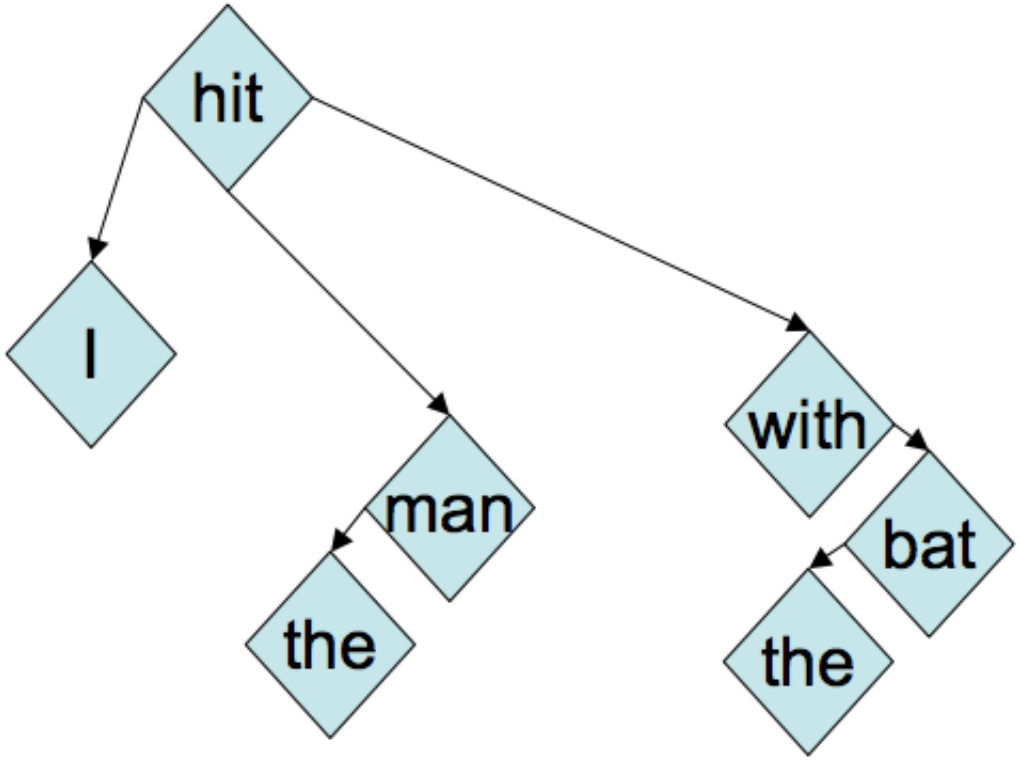
remove nonlexical parts:



from Noah Smith

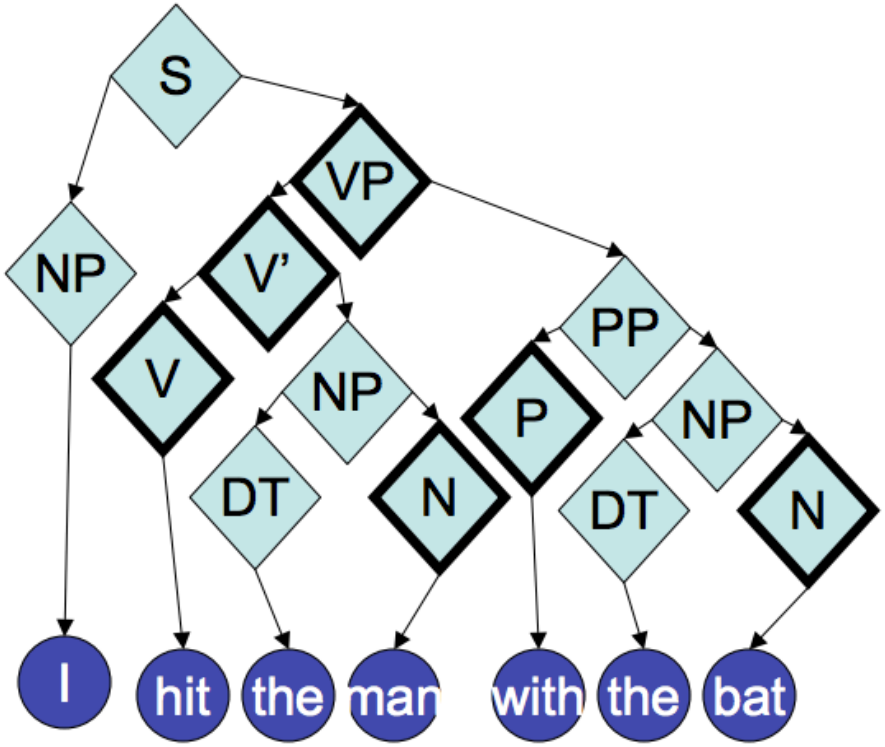
Dependencies

merge redundant nodes:

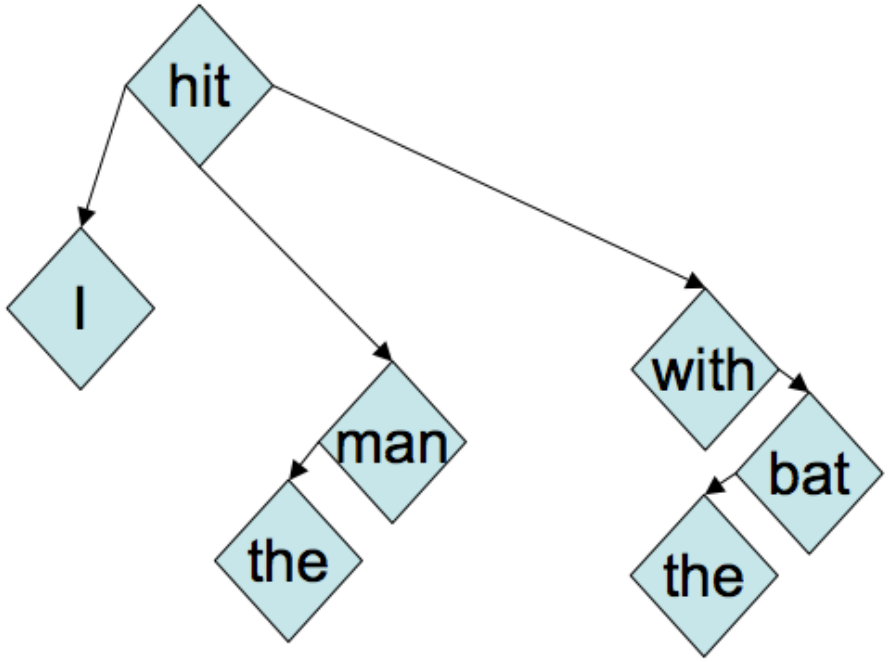


from Noah Smith

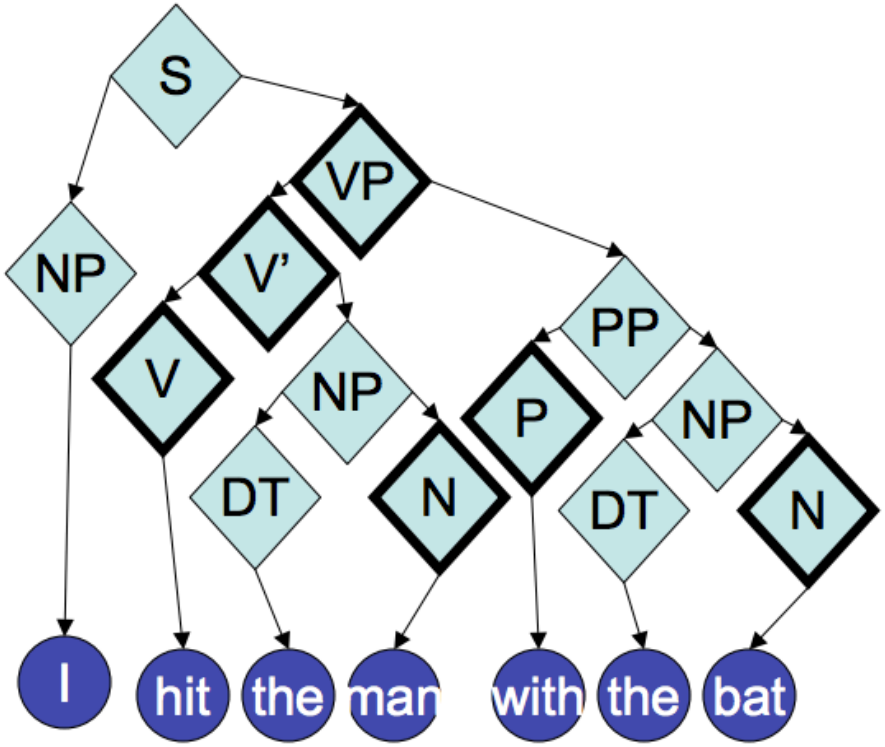
constituent parse:



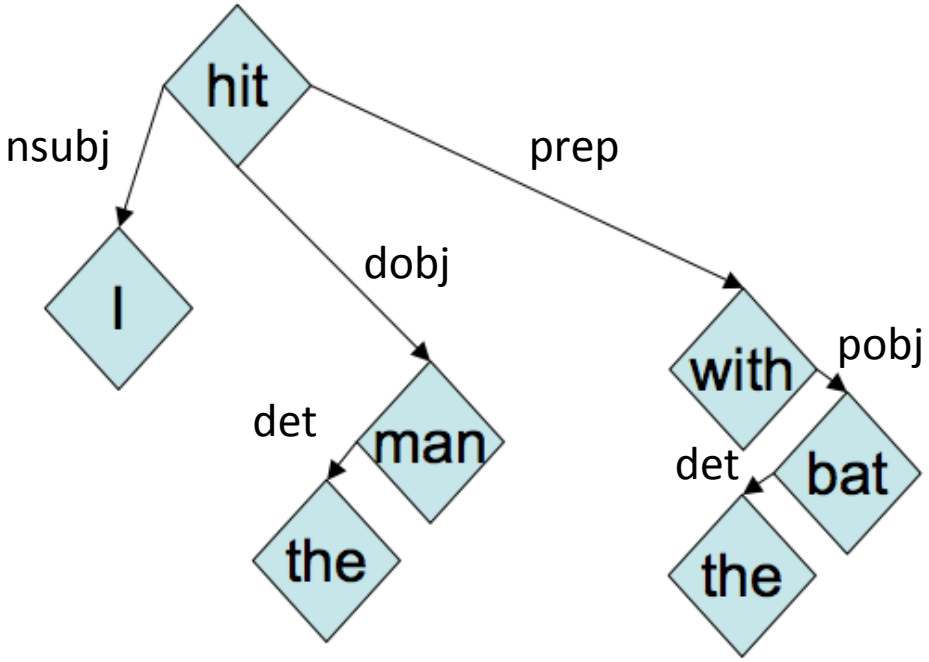
dependency parse:



constituent parse:

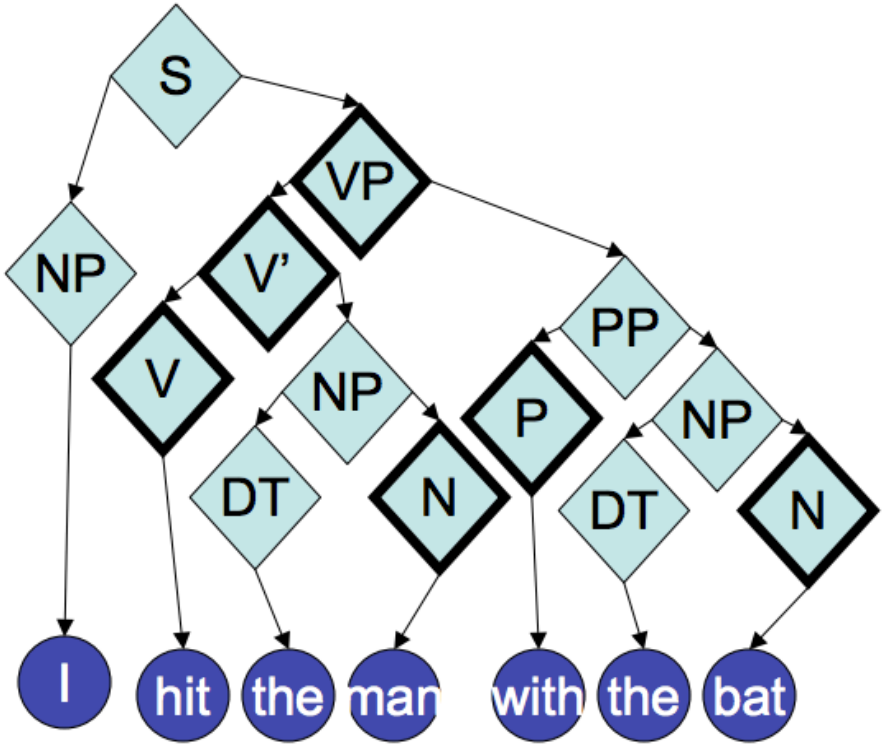


labeled dependency parse:

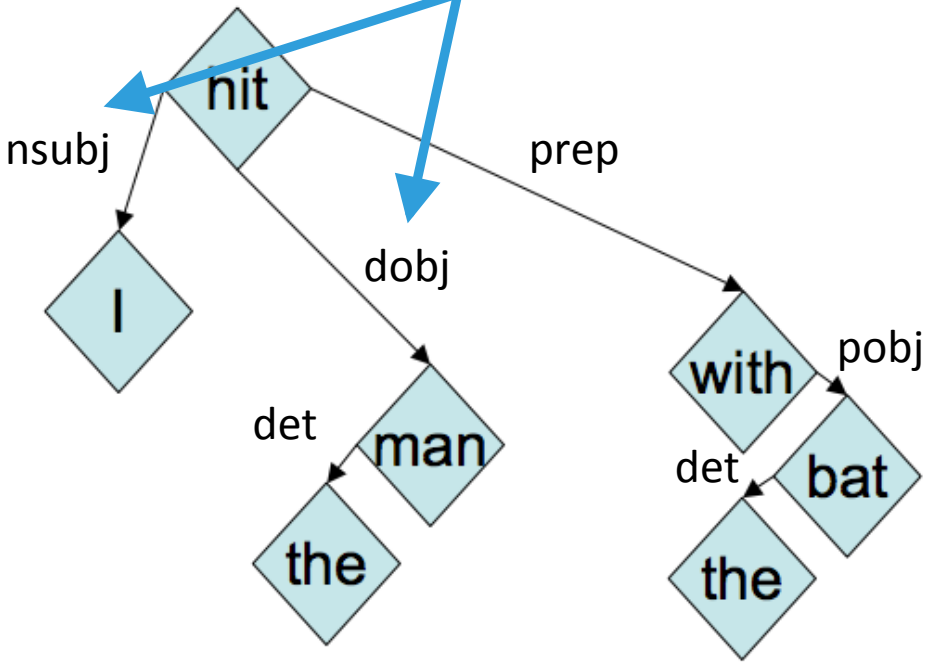


nsubj = “nominal subject”
dobj = “direct object”
prep = “preposition modifier”
pobj = “object of preposition”
det = “determiner”

constituent parse:

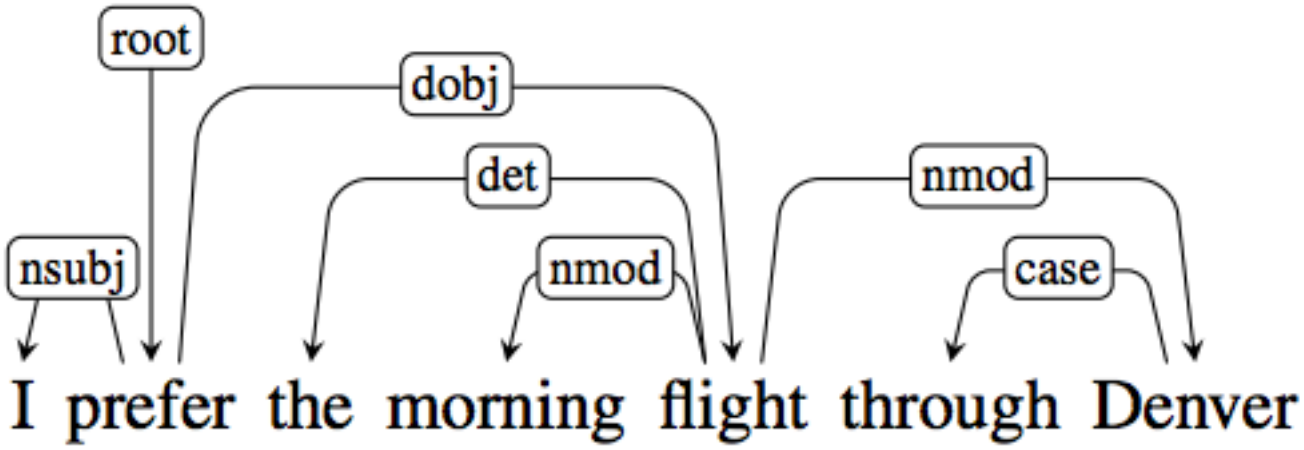


captures some semantic relationships



nsubj = “nominal subject”
dobj = “direct object”
prep = “preposition modifier”
pobj = “object of preposition”
det = “determiner”

A Typed Dependency Tree



J&M/SLP3

Some 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

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

J&M/SLP3

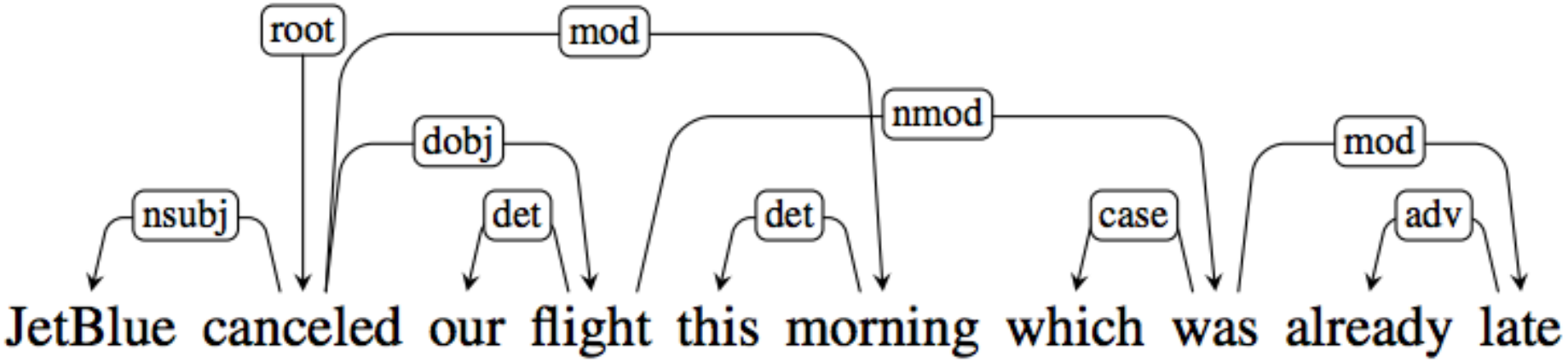
Some Dependency Relations

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 14.3 Examples of core Universal Dependency relations.

Crossing Dependencies = Nonprojective Tree

if dependencies cross ("nonprojective"), no longer corresponds to a CFG



J&M/SLP3

Projective vs. Nonprojective Dependencies

- English dependency treebanks are mostly projective
  - but when focusing more on semantic relationships, often becomes more nonprojective
- some (relatively) free word order languages, like Czech, are fairly nonprojective

Annotating Dependencies

- for many years, researchers build dependency parsers from deterministic head rules
- deterministic head rules are a blunt instrument
- would be better to directly annotate dependencies!
- there have been many annotation efforts with this goal

Universal Dependencies

Universal Dependencies (UD) is a project that is developing cross-linguistically consistent treebank annotation for many languages, with the goal of facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective.

This is illustrated in the following parallel examples from English, Bulgarian, Czech and Swedish, where the main grammatical relations involving a passive verb, a nominal subject and an oblique agent are the same, but where the concrete grammatical realization varies.

universaldependencies.org

Dependency Parsing

- several widely-used algorithms
- different guarantees but similar performance in practice
- graph-based:
  - dynamic programming (Eisner, 1997)
  - minimum spanning tree (McDonald et al., 2005)
- transition-based:
  - shift-reduce (Nivre, *inter alia*)

Transition-Based Dependency Parsing

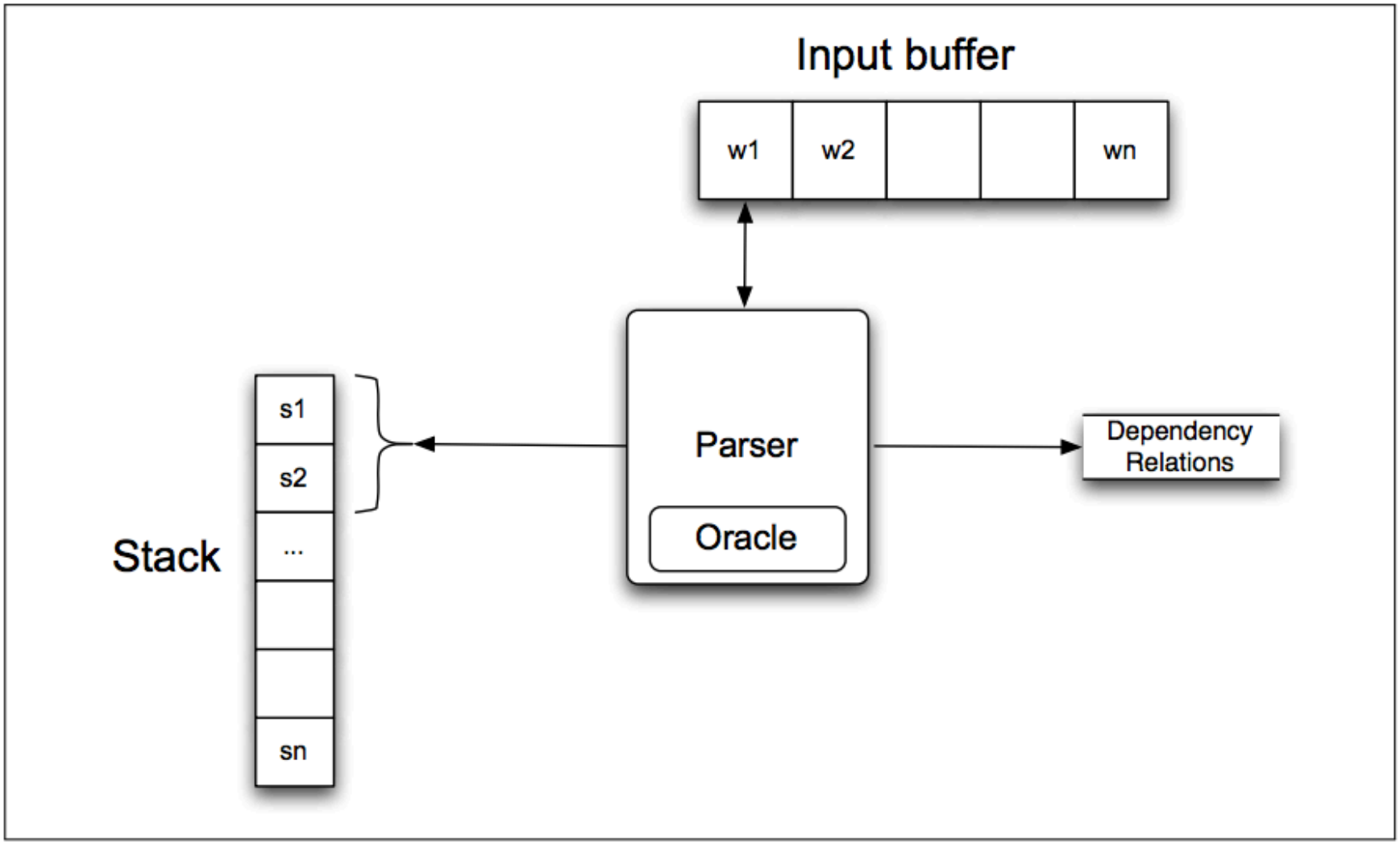


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

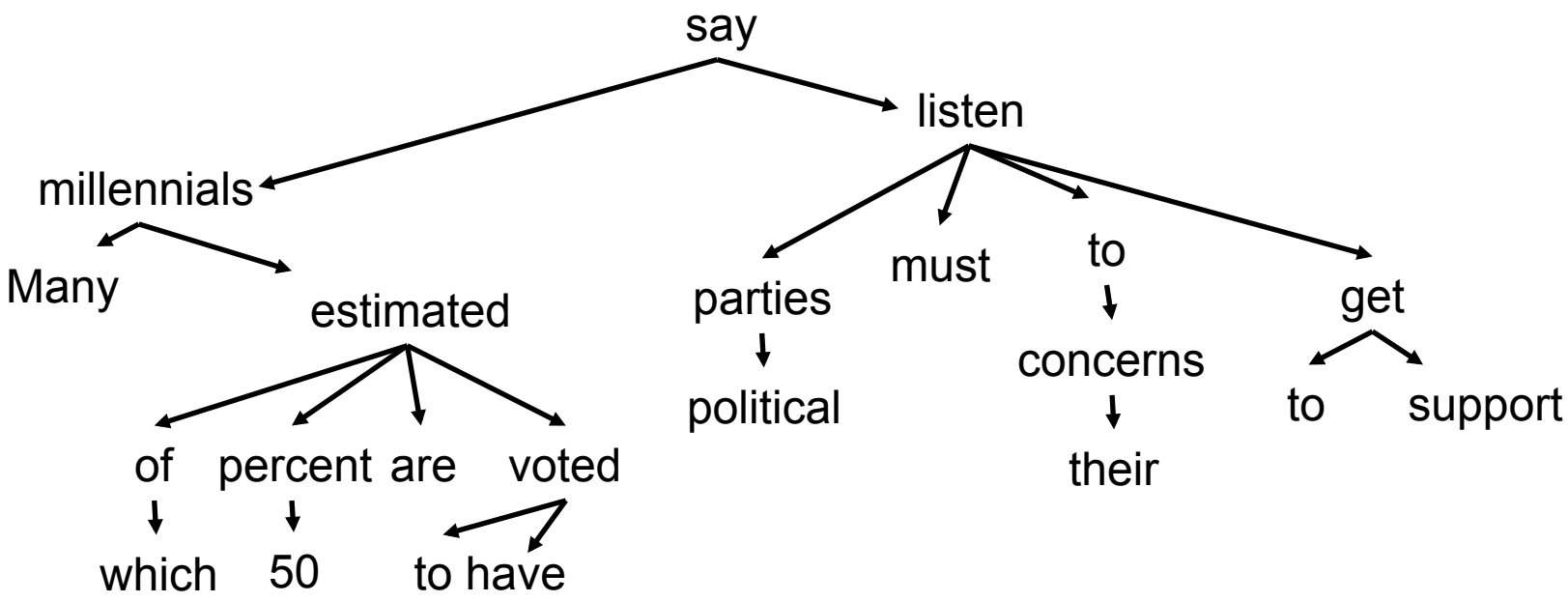
J&M/SLP3

Transition-Based Parsing

- there are many variations of greedy parsers that build parse structures as they process a sentence from left to right
  - "shift-reduce", "transition-based", etc.
- these form the backbone of many modern neural dependency (and constituency!) parsers
- we'll go through an example (thanks to Noah Smith for these slides)

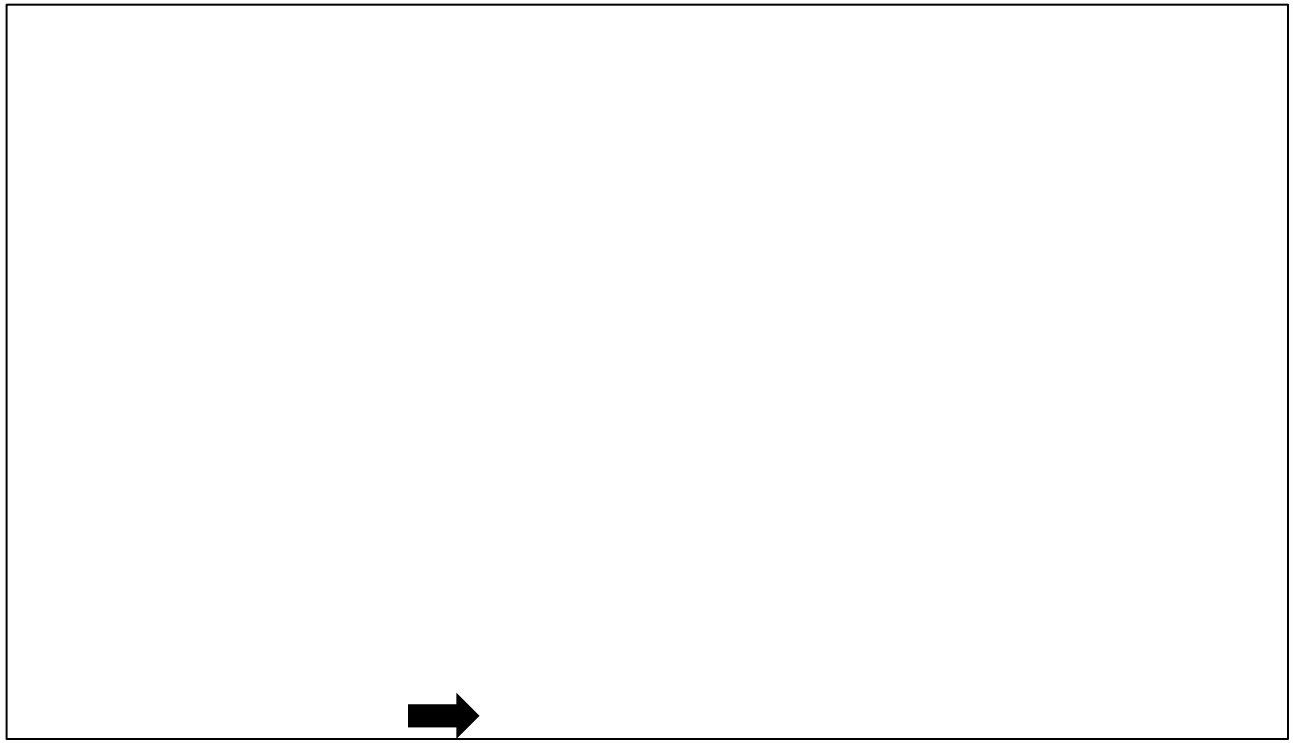
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.



Greedy Parsing with a Stack

Stack:



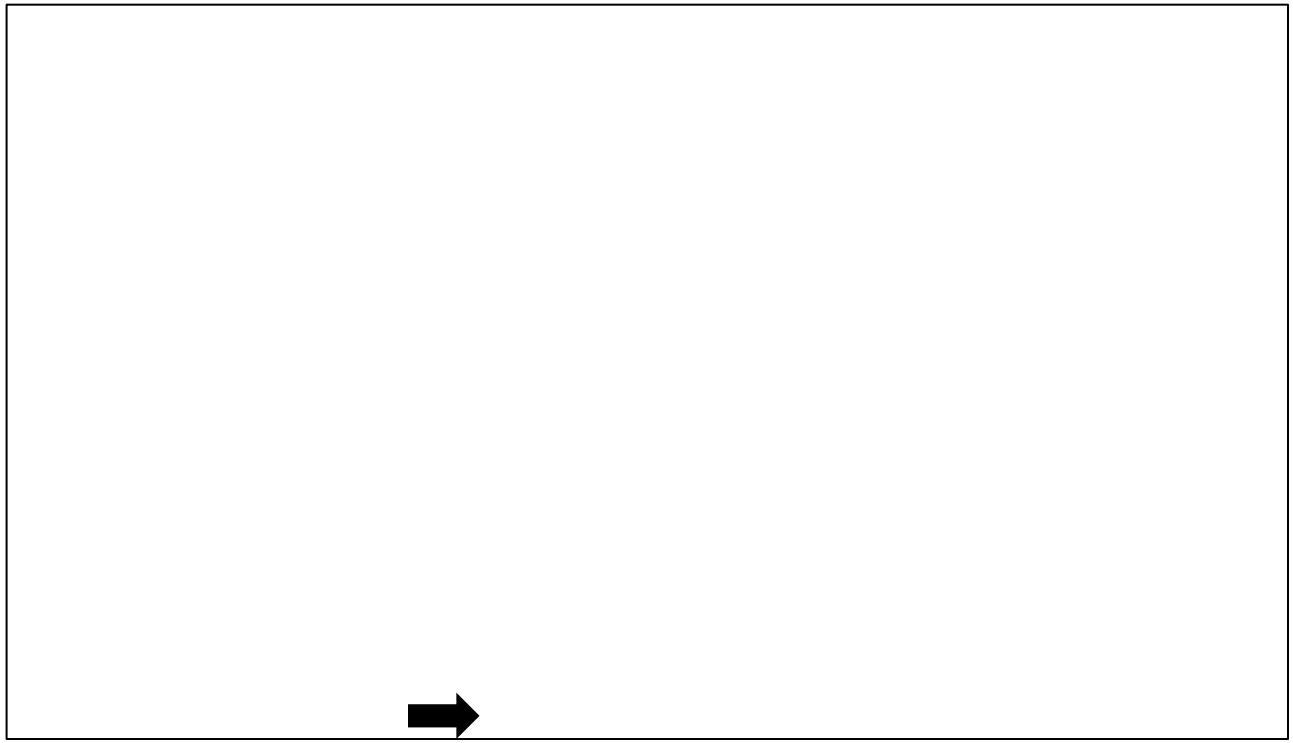
See:
Nivre & Scholz, 2004
Henderson, 2004

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:

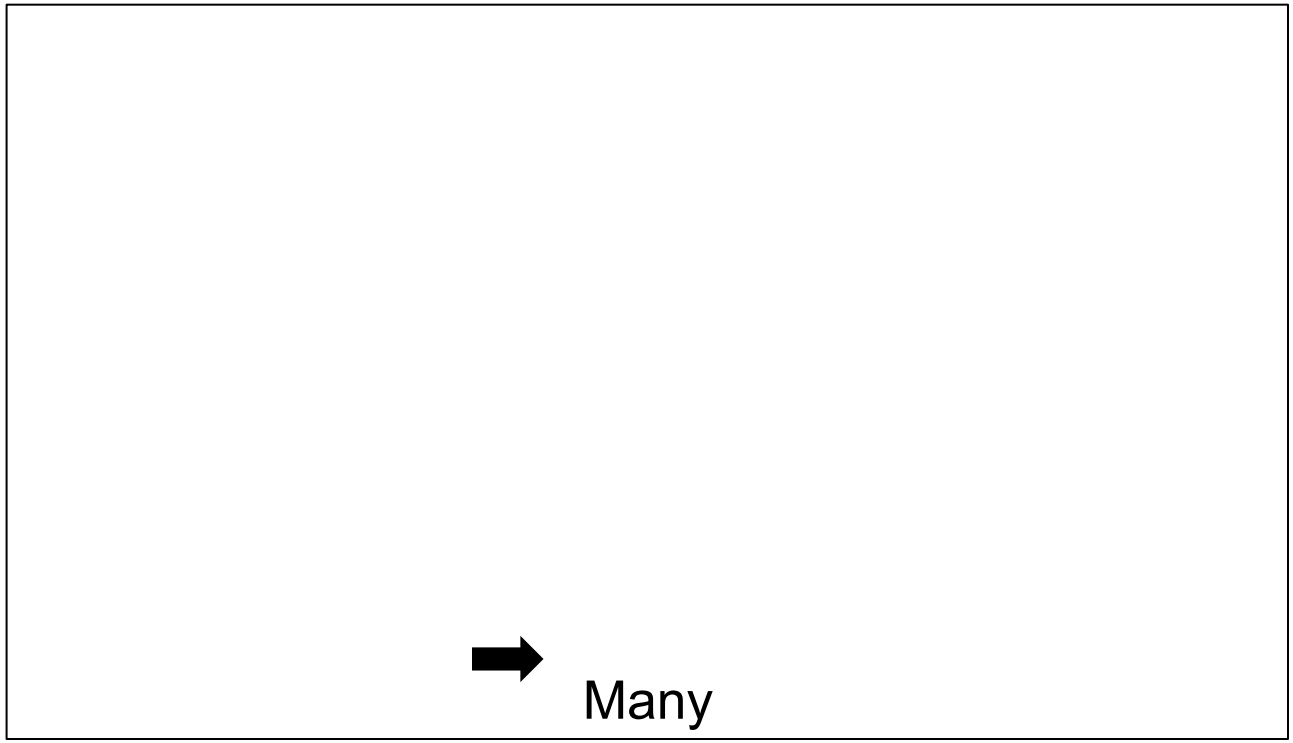


Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



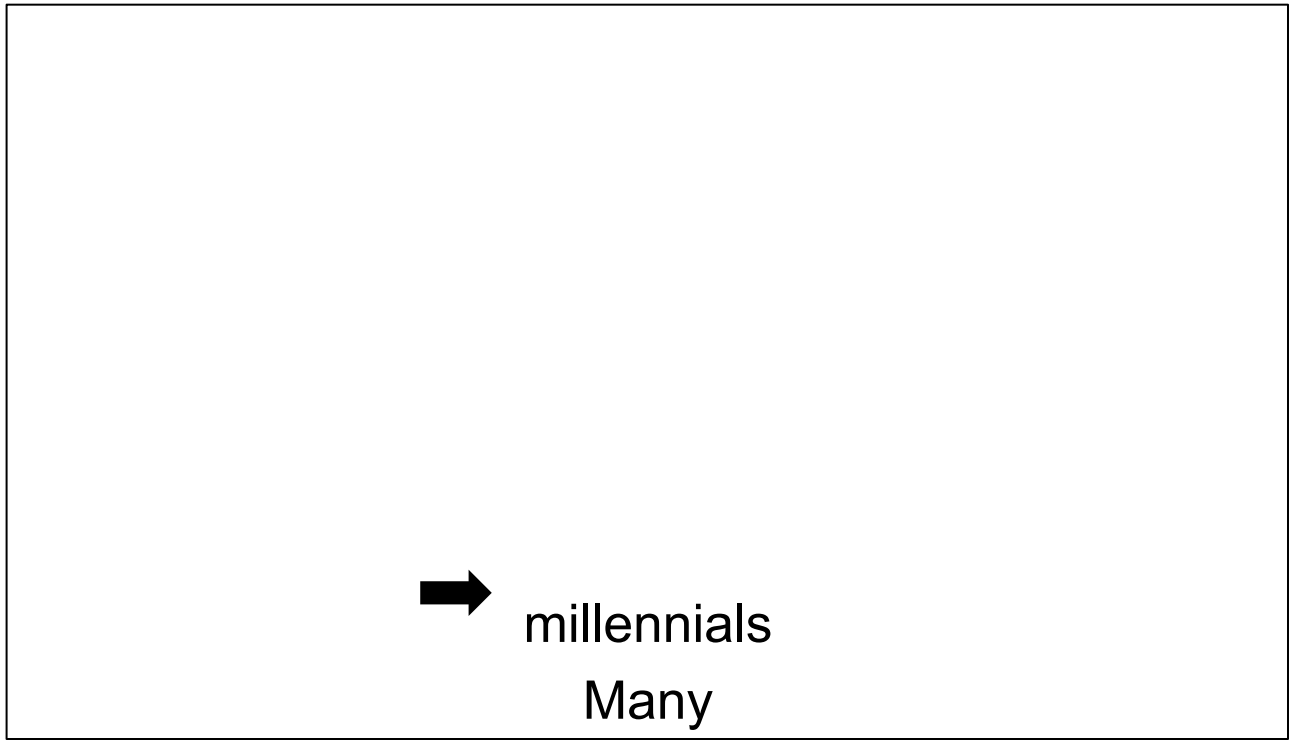
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



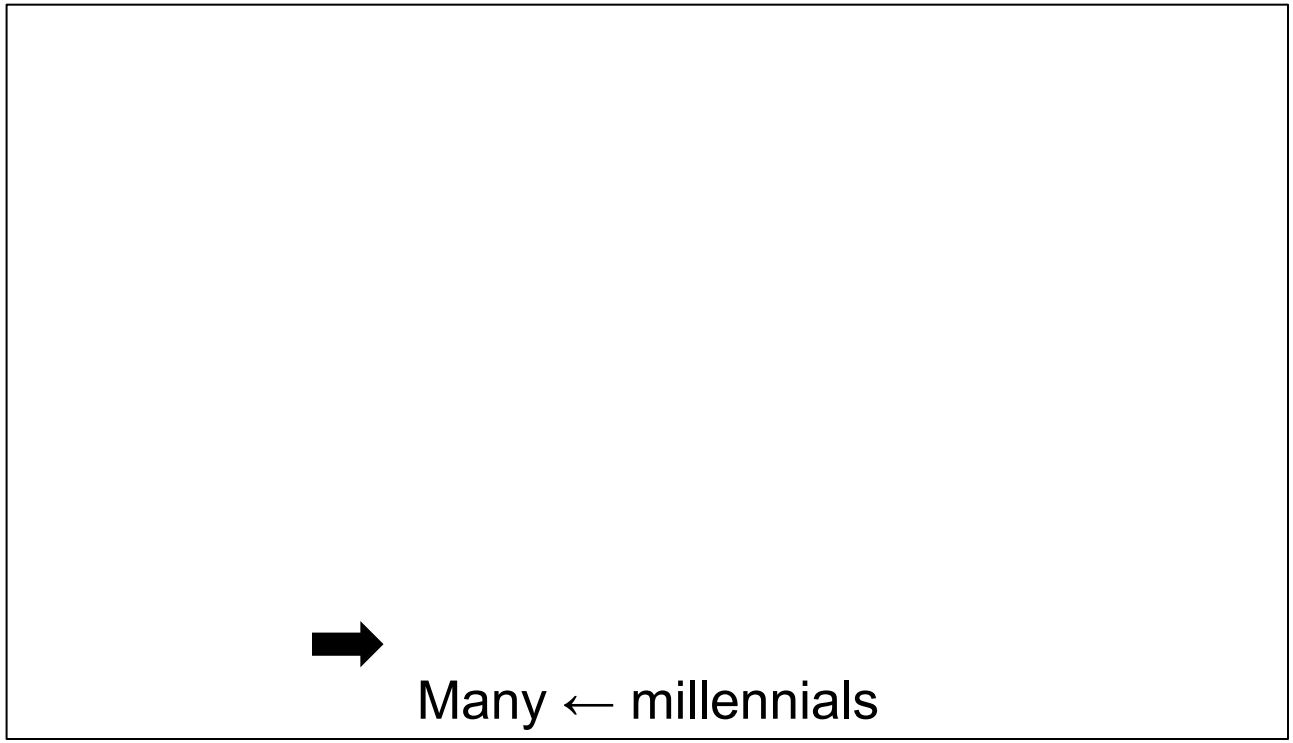
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



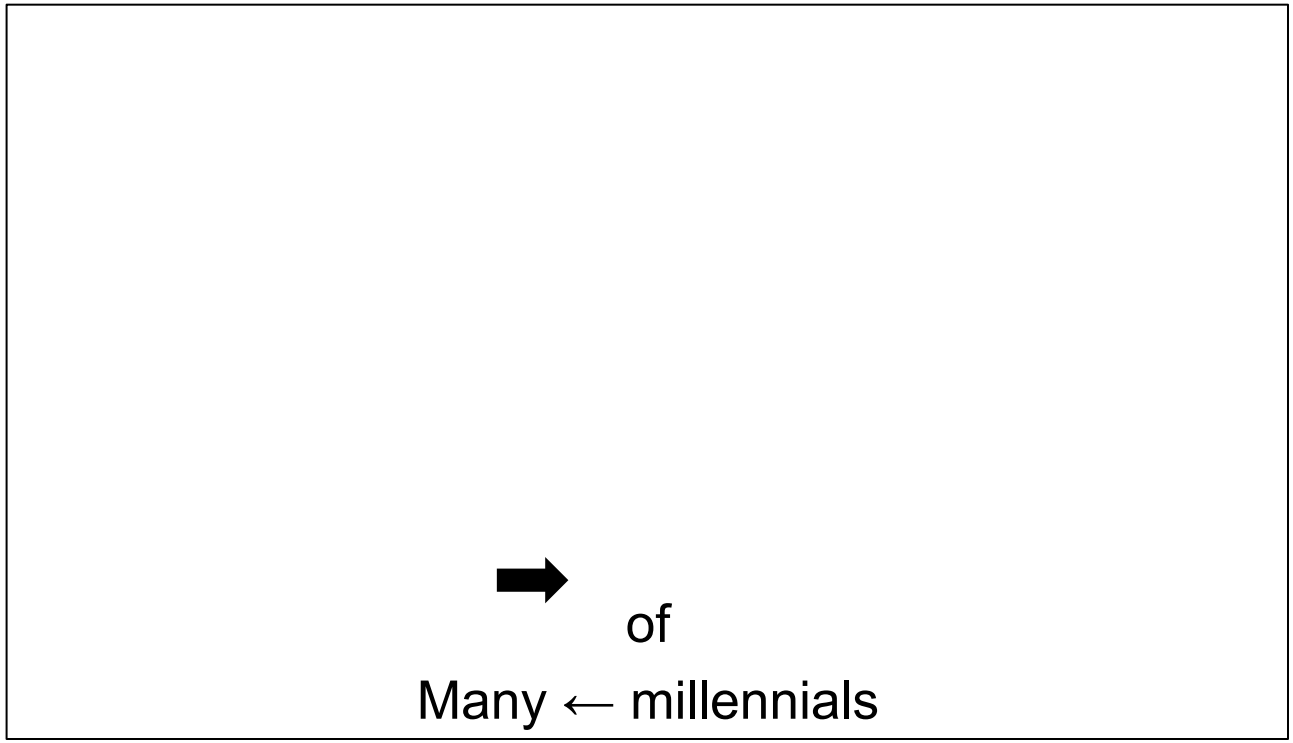
reduce left

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



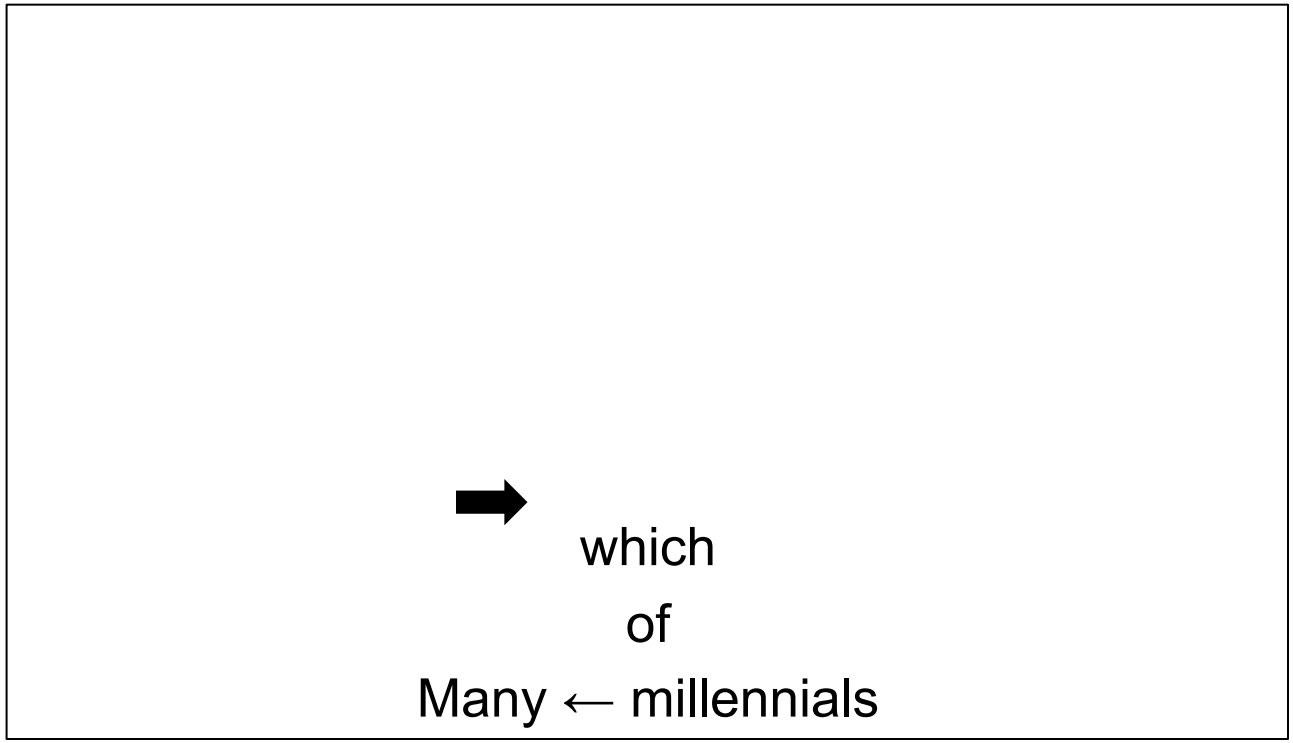
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



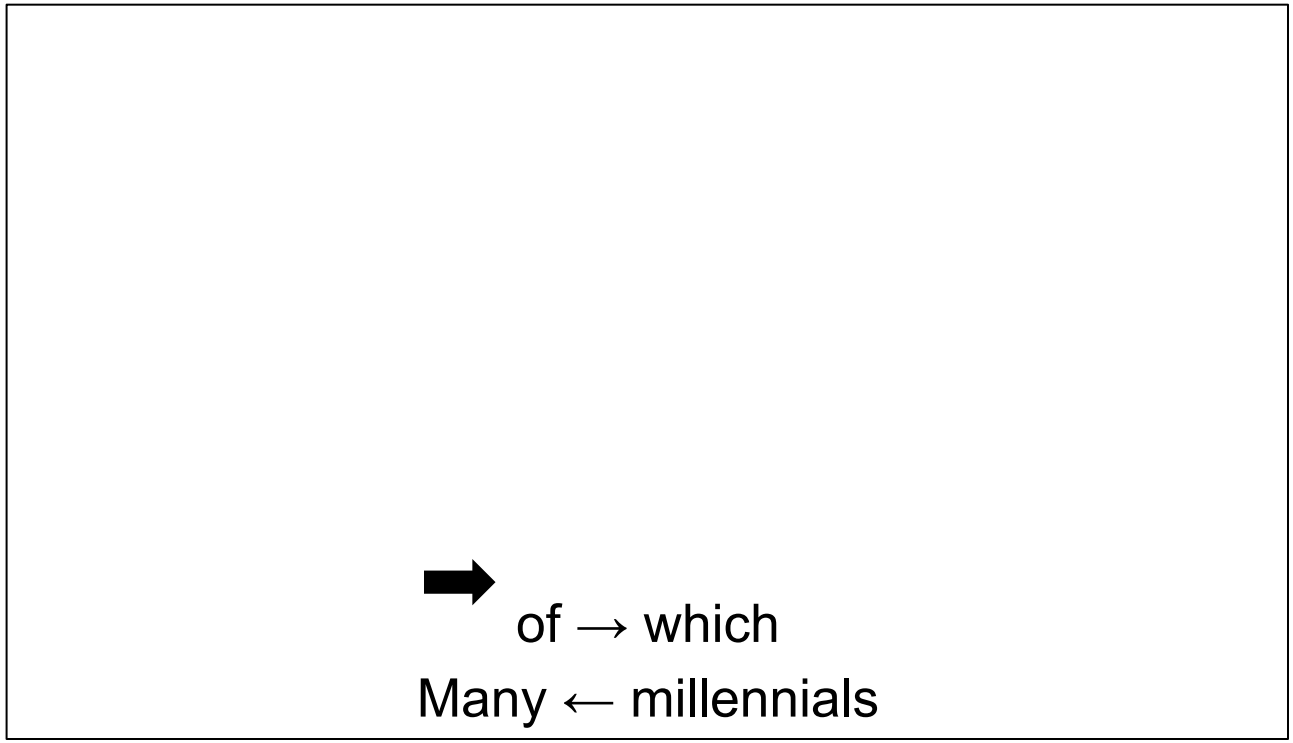
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



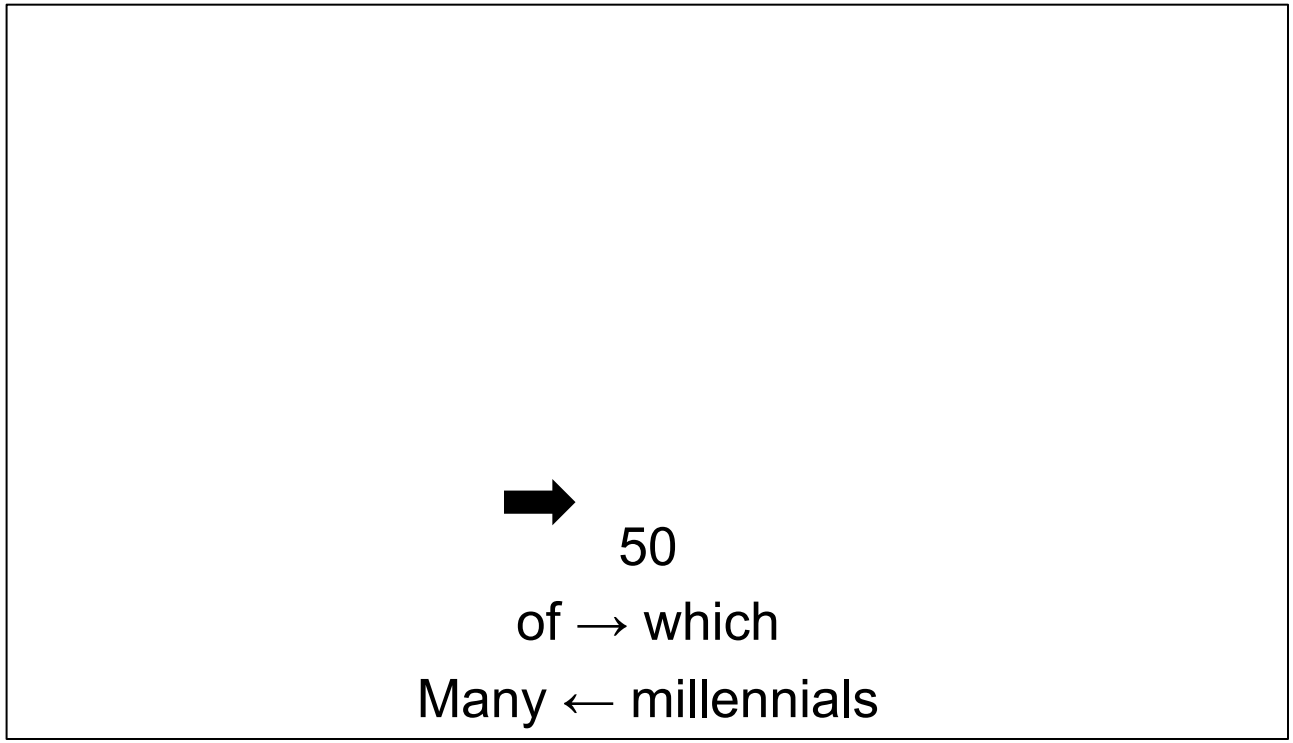
reduce right

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



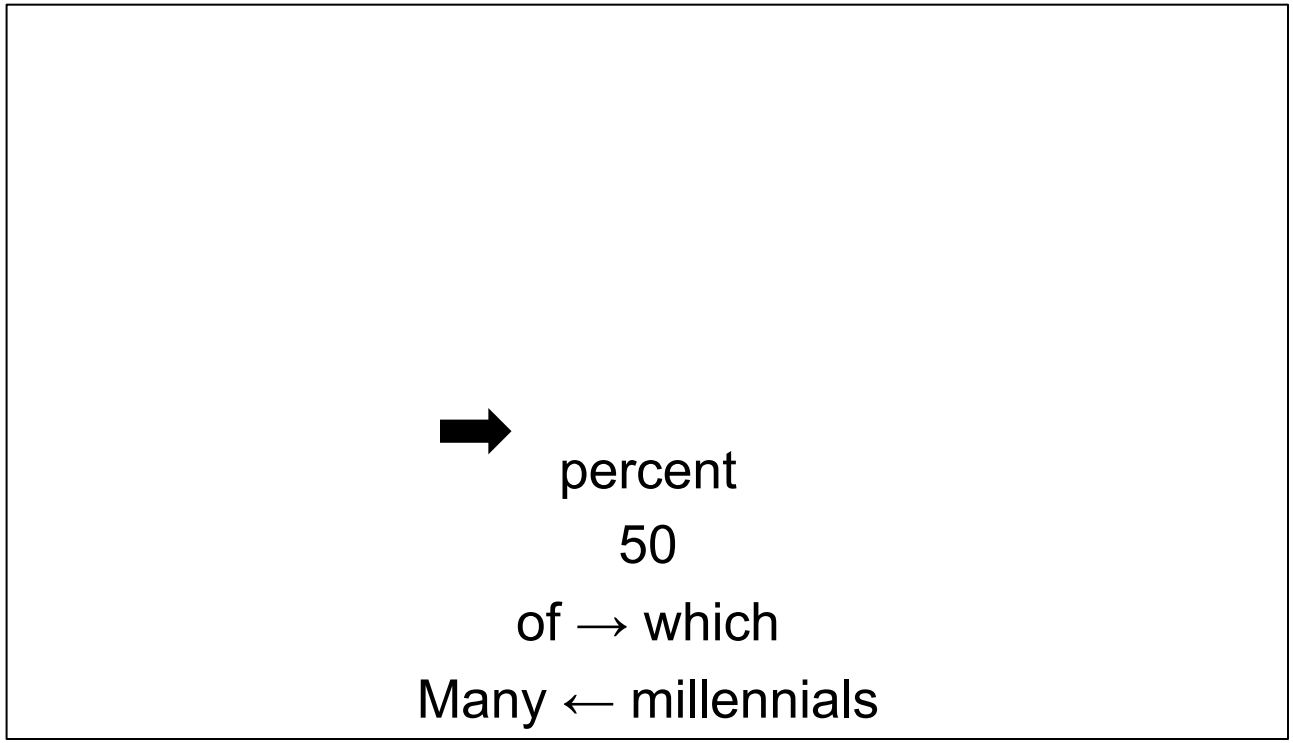
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



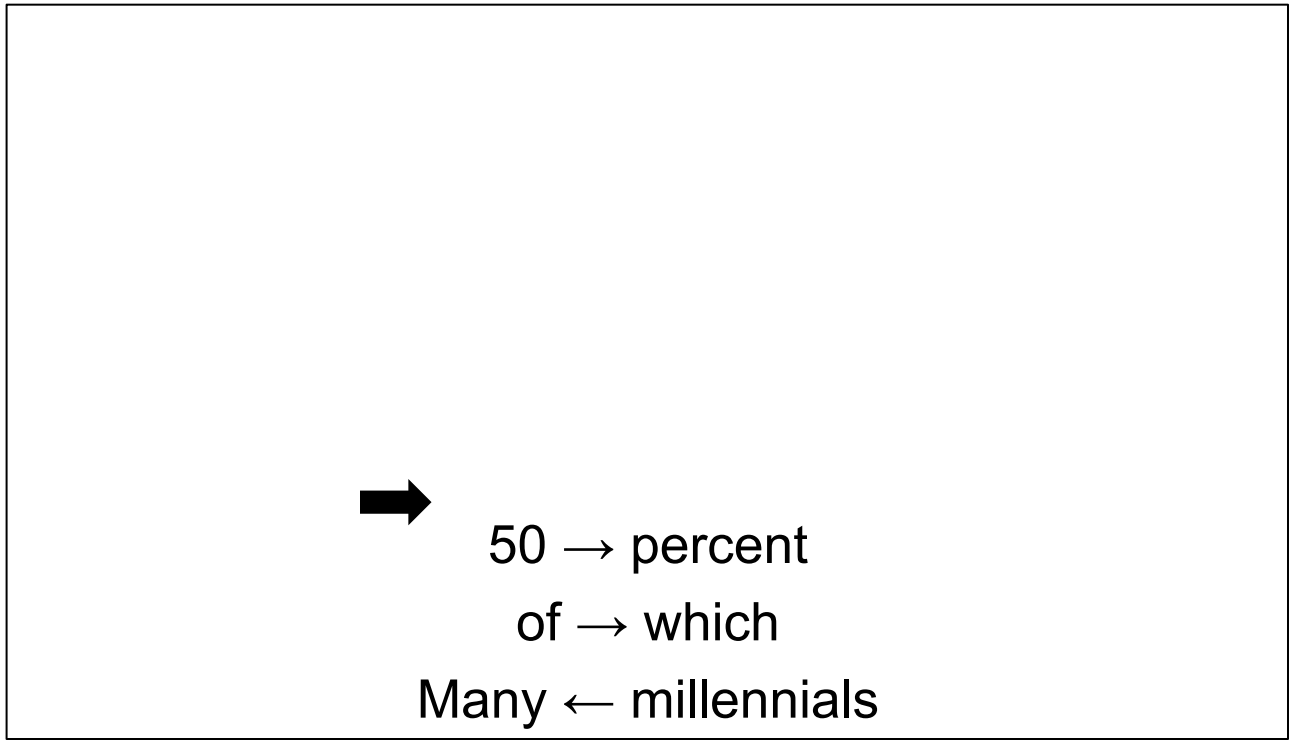
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



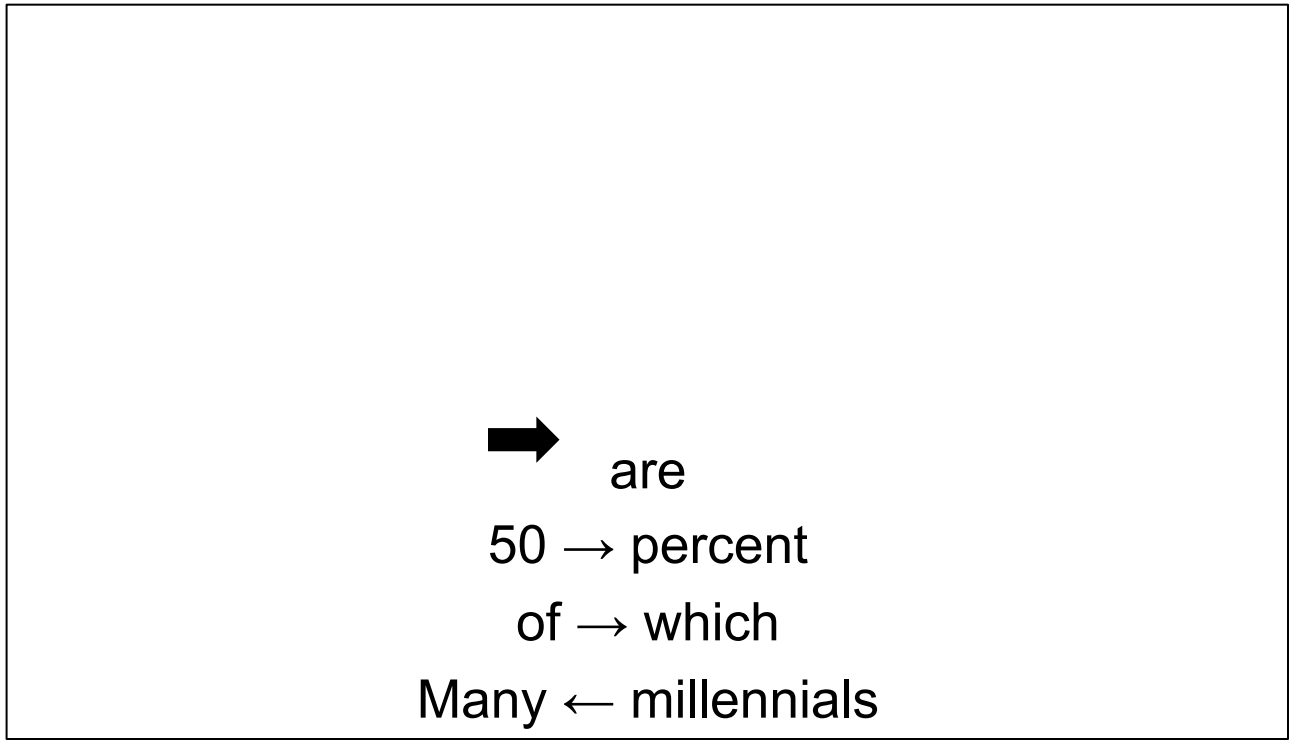
reduce right

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



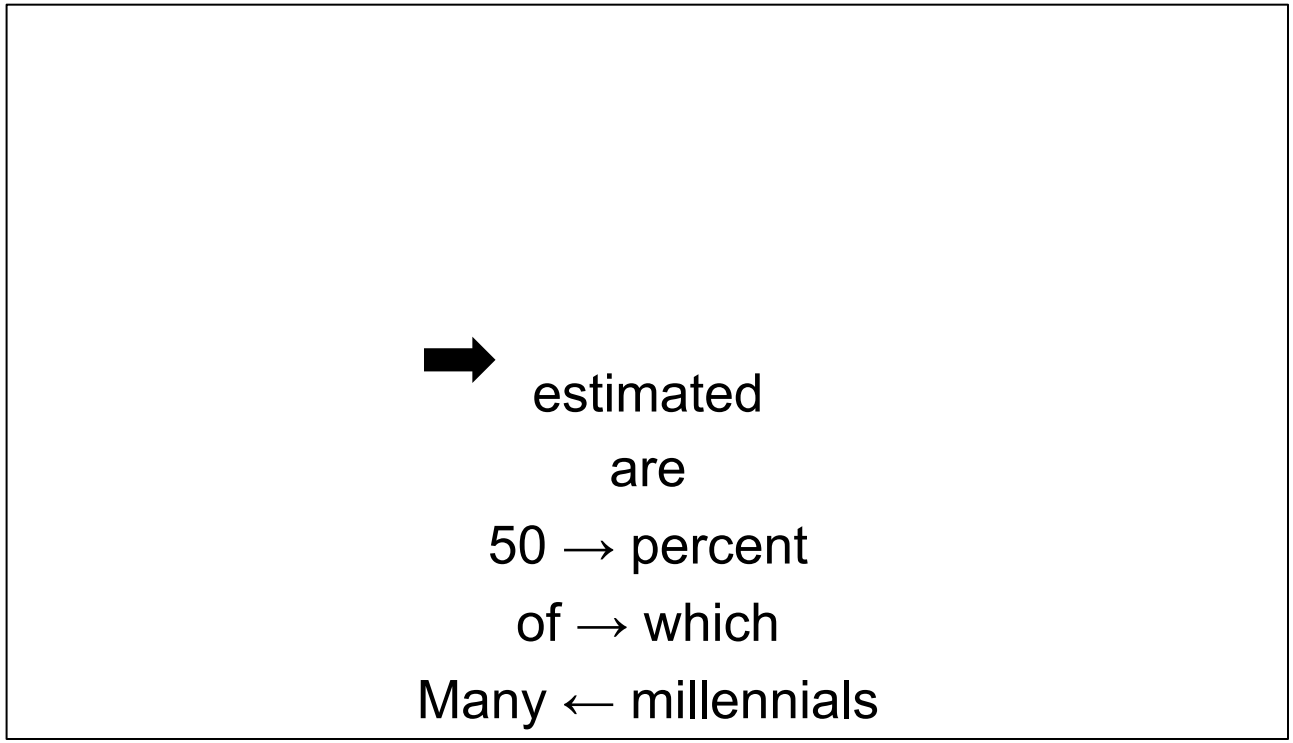
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



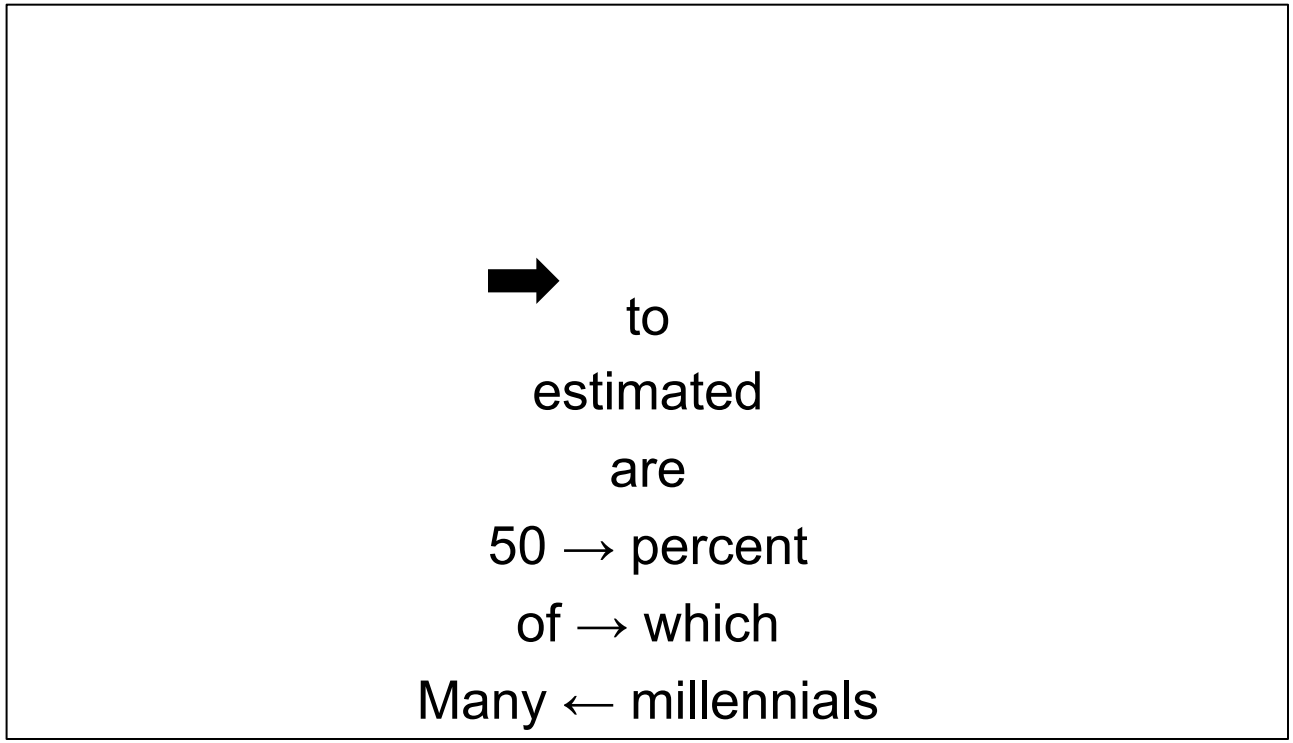
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



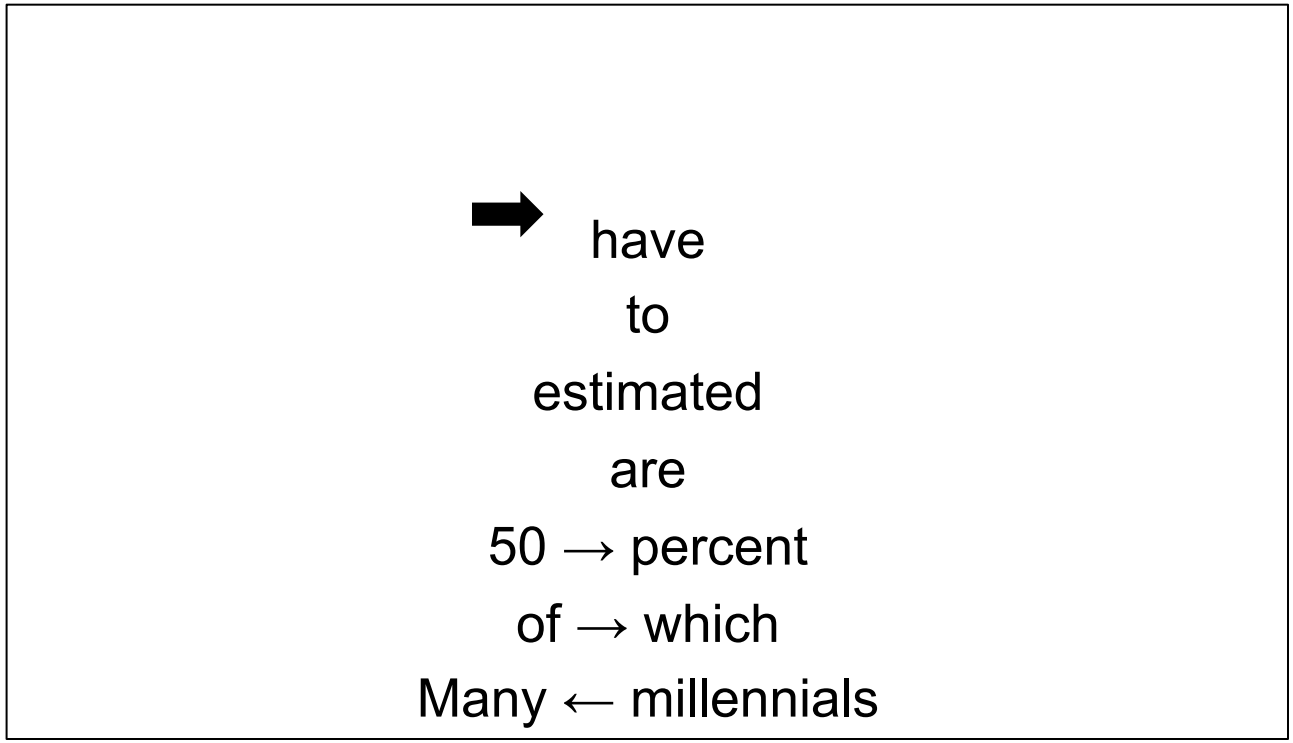
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



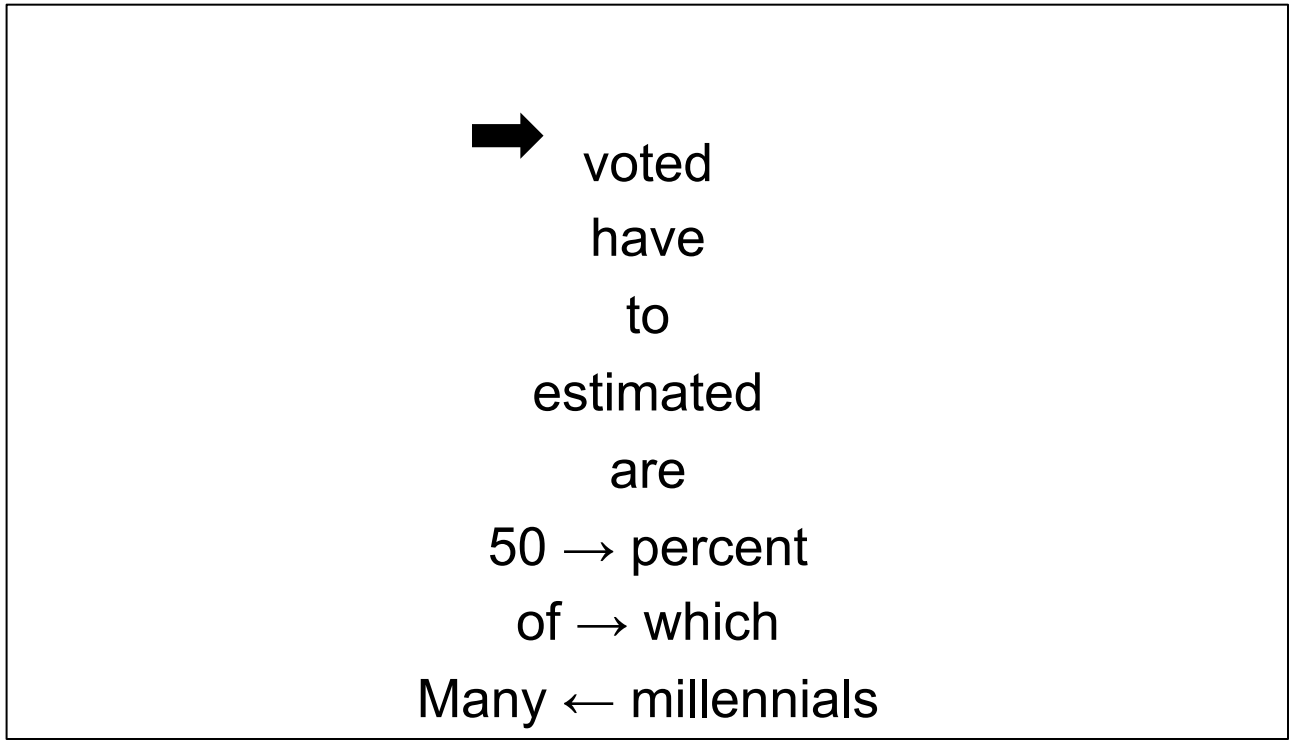
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



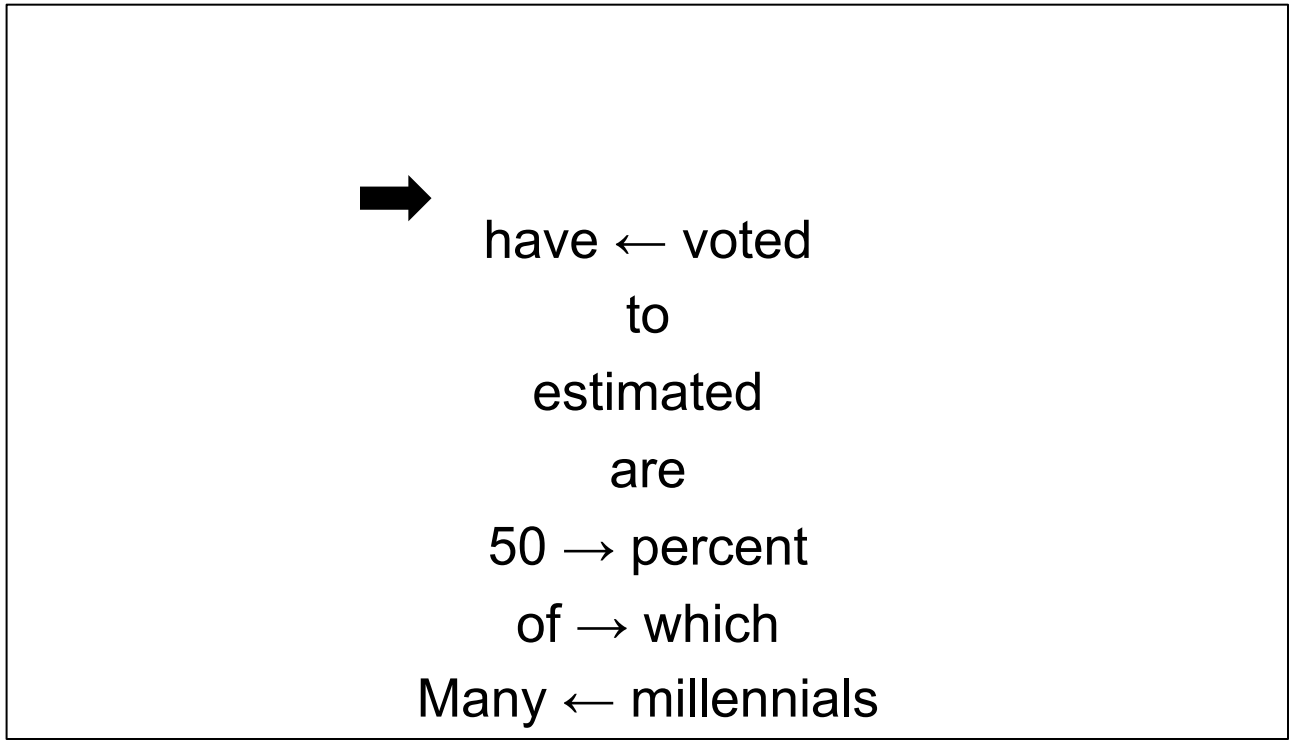
shift

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



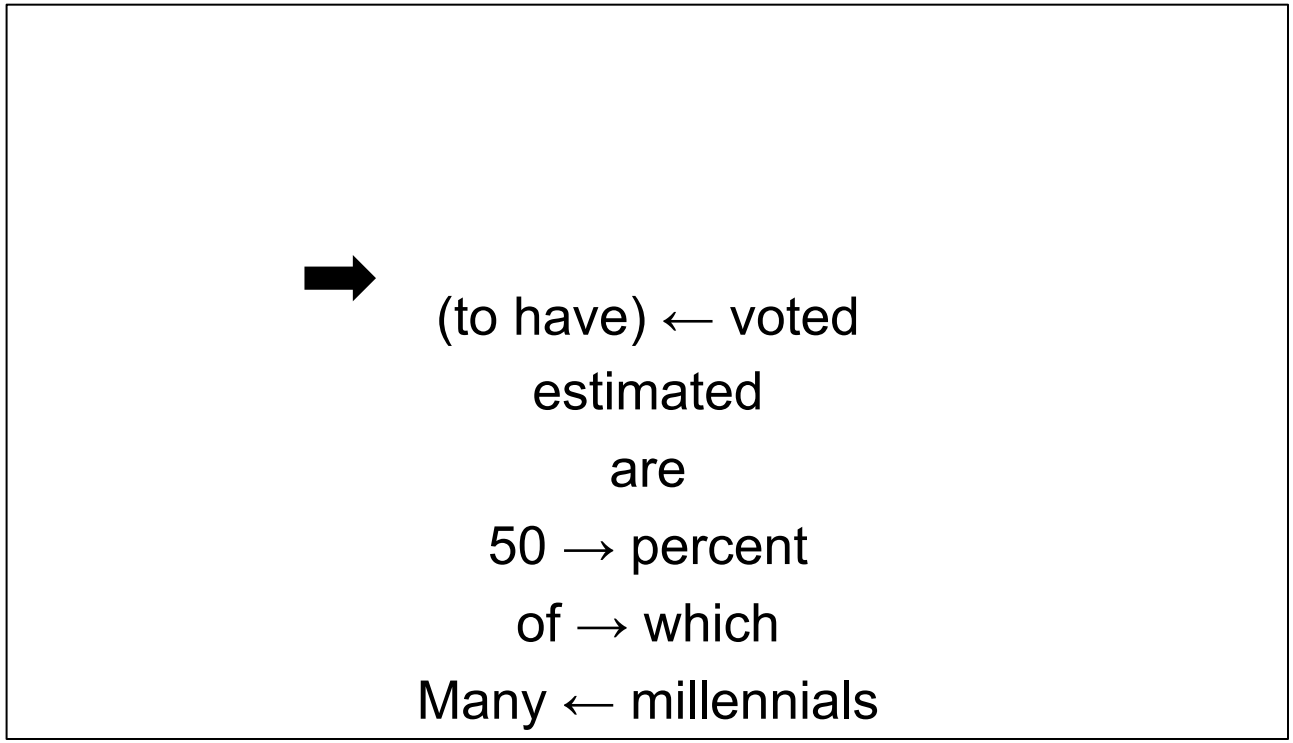
reduce left

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



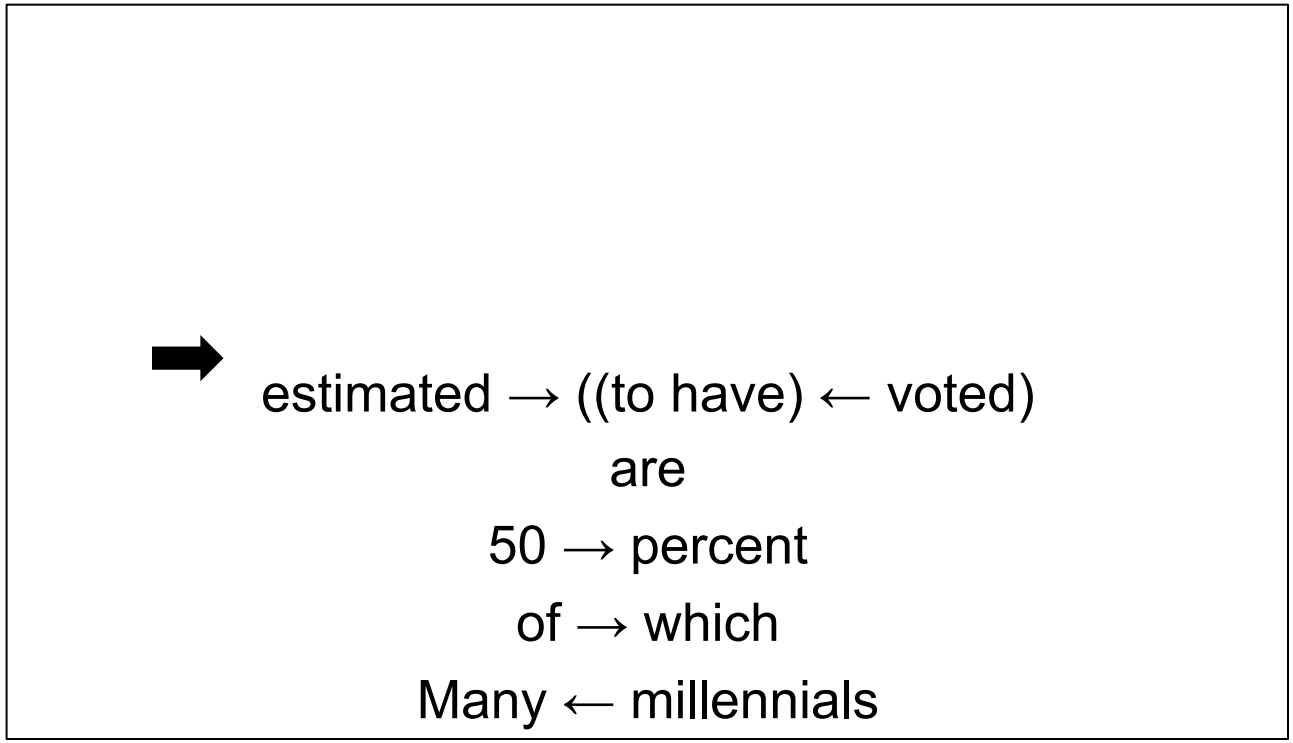
reduce left

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



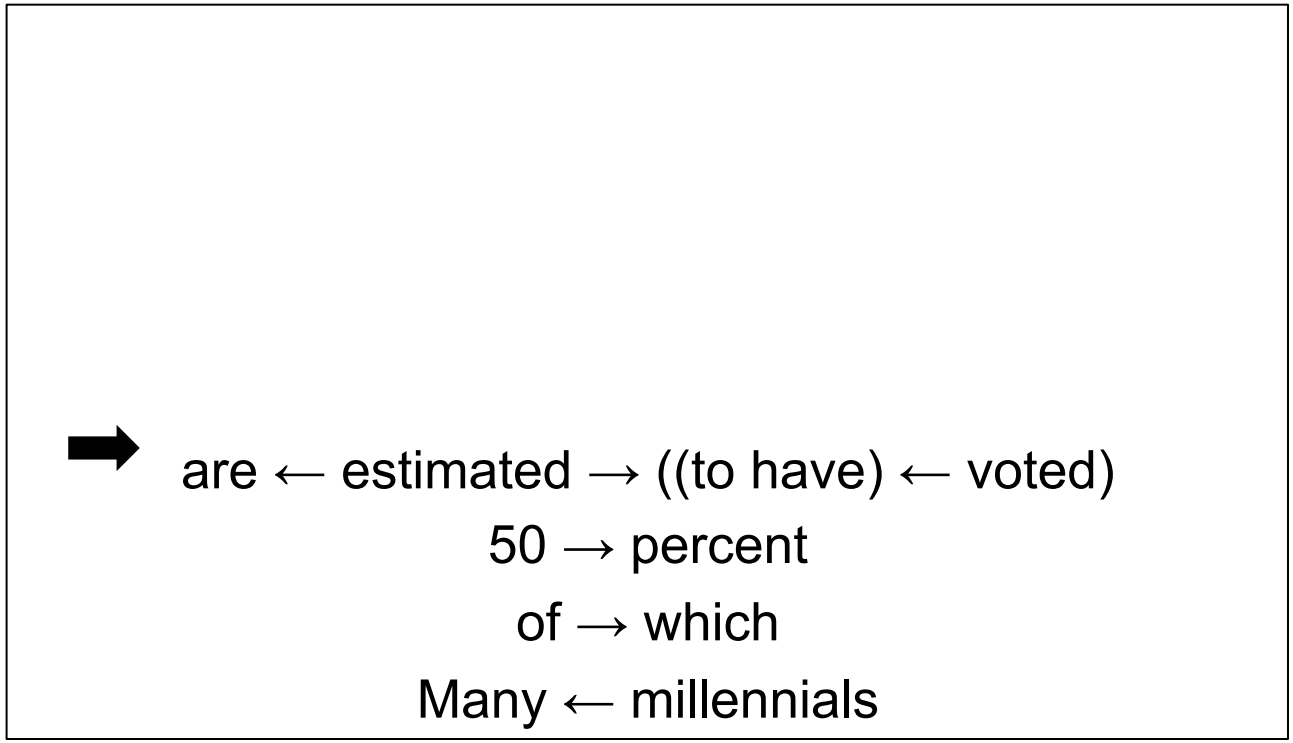
reduce right

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

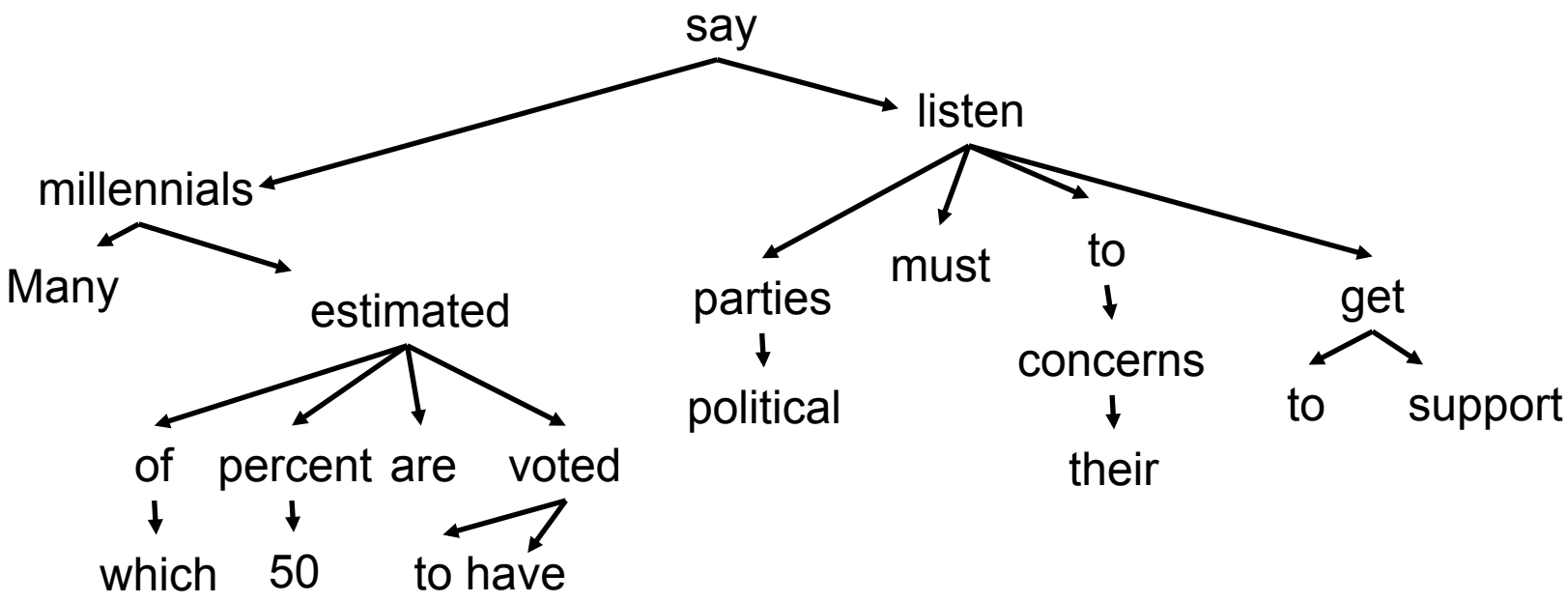
Stack:



reduce left

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.



Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Stack$_t$	**Buffer**$_t$	**Action**	**Stack**$_{t+1}$	**Buffer**$_{t+1}$	**Dependency**
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-RIGHT(r)	$(g_r(\mathbf{u}, \mathbf{v}), u), S$	B	$u \xrightarrow{r} v$
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-LEFT(r)	$(g_r(\mathbf{v}, \mathbf{u}), v), S$	B	$u \xleftarrow{r} v$
S	$(\mathbf{u}, u), B$	SHIFT	$(\mathbf{u}, u), S$	B	—

Figure 3: Parser transitions indicating the action applied to the stack and buffer and the resulting stack and buffer states. Bold symbols indicate (learned) embeddings of words and relations, script symbols indicate the corresponding words and relations.

Dyer et al. (ACL 2015)

- Early work used multi-class linear classifiers to output a parsing decision (shift, reduce-left, or reduce-right)
- Chen et al. (2014) used a feed-forward network for this
- Dyer et al. (2015) used RNNs to model the history of parsing decisions, the partial parses so far (the “stack”), and the sentence

Stack RNNs

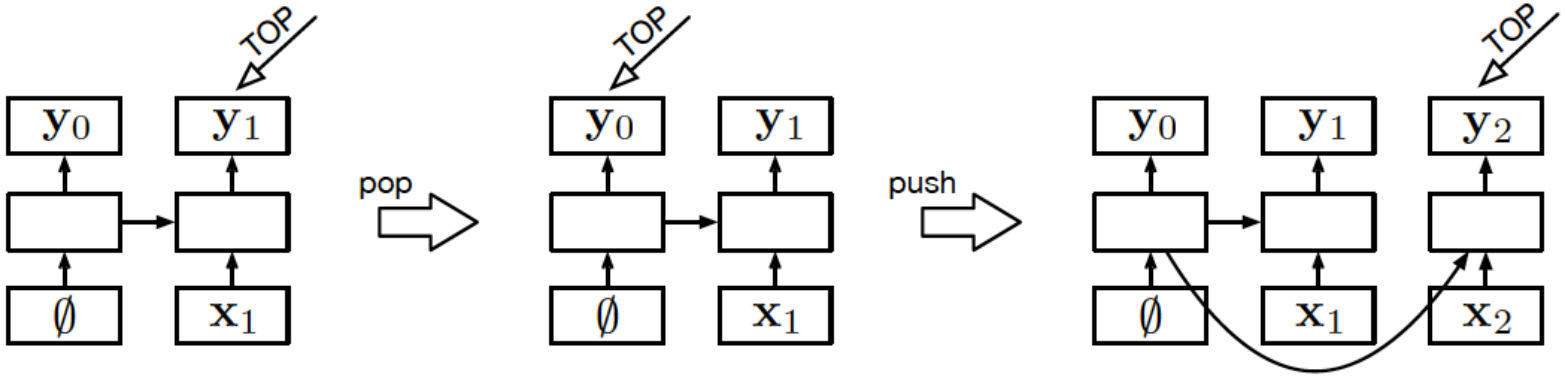


Figure 1: A stack LSTM extends a conventional left-to-right LSTM with the addition of a stack pointer (notated as TOP in the figure). This figure shows three configurations: a stack with a single element (left), the result of a pop operation to this (middle), and then the result of applying a push operation (right). The boxes in the lowest rows represent stack contents, which are the inputs to the LSTM, the upper rows are the outputs of the LSTM (in this paper, only the output pointed to by TOP is ever accessed), and the middle rows are the memory cells (the $\mathbf{c}_t$'s and $\mathbf{h}_t$'s) and gates. Arrows represent function applications (usually affine transformations followed by a nonlinearity), refer to §2.1 for specifics.

Dyer et al. (ACL 2015)

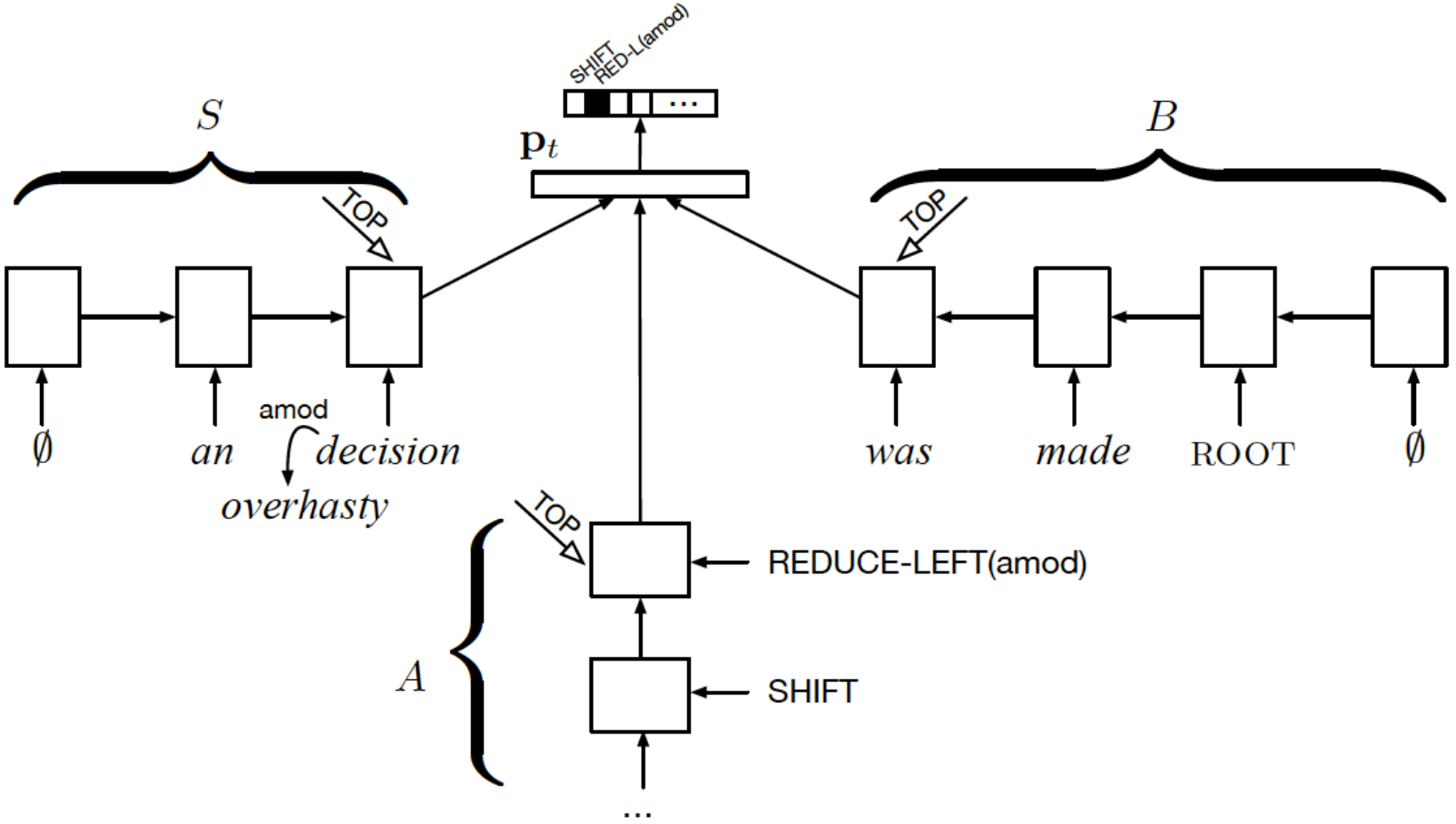
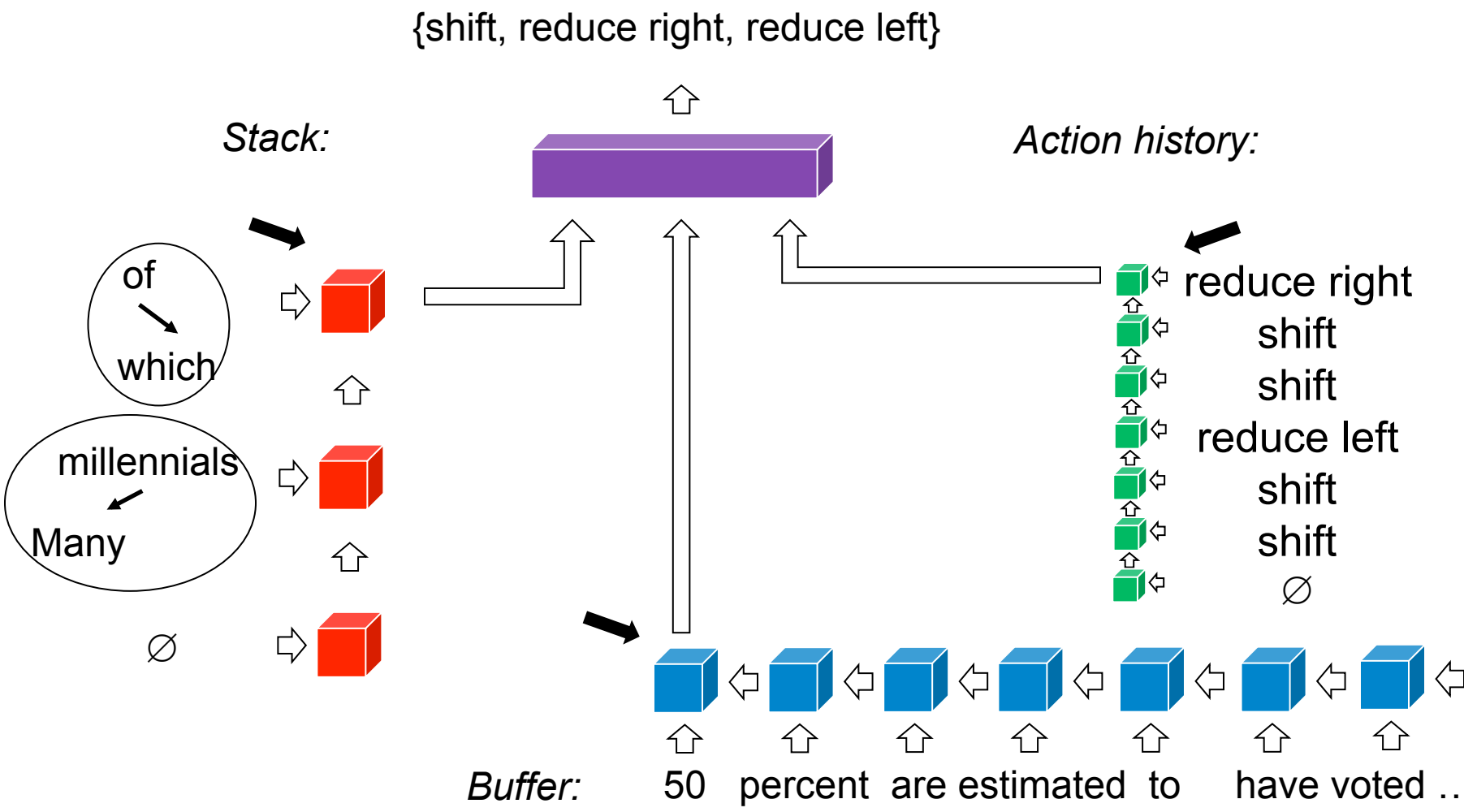


Figure 2: Parser state computation encountered while parsing the sentence "*an overhasty decision was made*." Here S designates the stack of partially constructed dependency subtrees and its LSTM encoding; B is the buffer of words remaining to be processed and its LSTM encoding; and A is the stack representing the history of actions taken by the parser. These are linearly transformed, passed through a ReLU nonlinearity to produce the parser state embedding $\mathbf{p}_t$. An affine transformation of this embedding is passed to a softmax layer to give a distribution over parsing decisions that can be taken.

Dyer et al. (ACL 2015)

Stack LSTM Parser



Dependency Parsers

- Stanford parser
- TurboParser
- Joakim Nivre's MALT parser
- Ryan McDonald's MST parser
- and many others for many non-English languages

Projective vs. Nonprojective Dependency Parsing

- nonprojective parsing can be formulated as a minimum spanning tree problem

- projective parsing cannot, but dynamic programming algorithms can be used (variations of CKY) as well as transition-based parsers

Complexity Comparison

- constituent parsing: **$O(Gn^3)$**
  - parsing complexity depends on grammar structure ("grammar constant" G)
  - since it has lots of nonterminal-only rules at the top of the tree, there are many rule probabilities to estimate
- dependency parsing: **$O(n^3)$**
  - operates directly on words, so parsing complexity has no grammar constant
  - features designed on possible dependencies (pairs of words) and larger structures
  - transition-based parsing algorithms are O(n), though not optimal; also, non-projective parsing is faster

Applications of Dependency Parsing

- widely used for NLP tasks because:
  - faster than constituent parsing
  - captures more semantic information
- text classification (features on dependencies)
- syntax-based machine translation
- relation extraction
  - e.g., extract relation between Sam Smith and AITech:

  Sam Smith was named new CEO of AITech.
  - use dependency path between *Sam Smith* and *AITech:*
    - Smith → named, named ← CEO, CEO ← of, of ← AITech

Summary: two types of grammars

- phrase structure / constituent grammars
  - inspired mostly by Chomsky and others
  - only appropriate for certain languages (e.g., English)
- dependency grammars
  - closer to a semantic representation; some have made this more explicit
  - problematic for certain syntactic structures (e.g., conjunctions, nesting of noun phrases, etc.)
- both are widely used in NLP
- you can find constituent parsers and dependency parsers for several languages online

Recursive Neural Networks for NLP

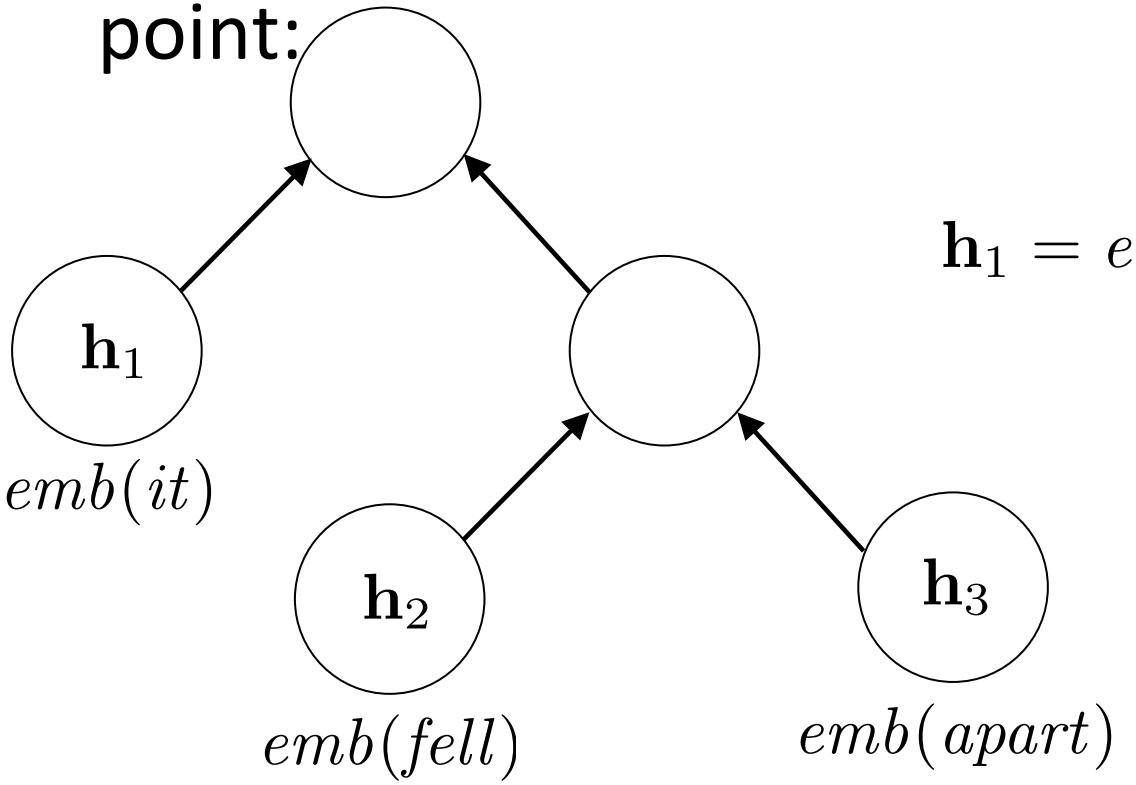
$x =$ *it fell apart*

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

Recursive Neural Networks for NLP

$x =$ *it fell apart*

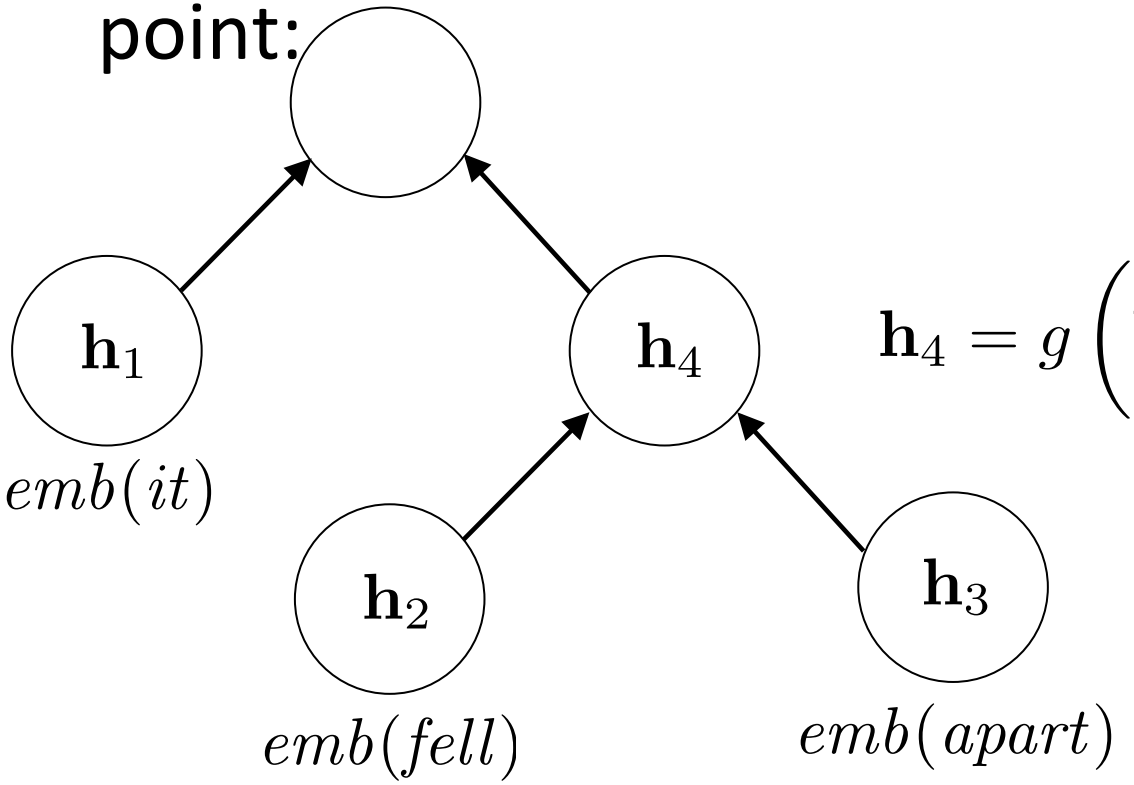
- run a syntactic parser on the sentence
- construct vector recursively at each split point:



Recursive Neural Networks for NLP

$\boldsymbol{x} =$ *it fell apart*

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

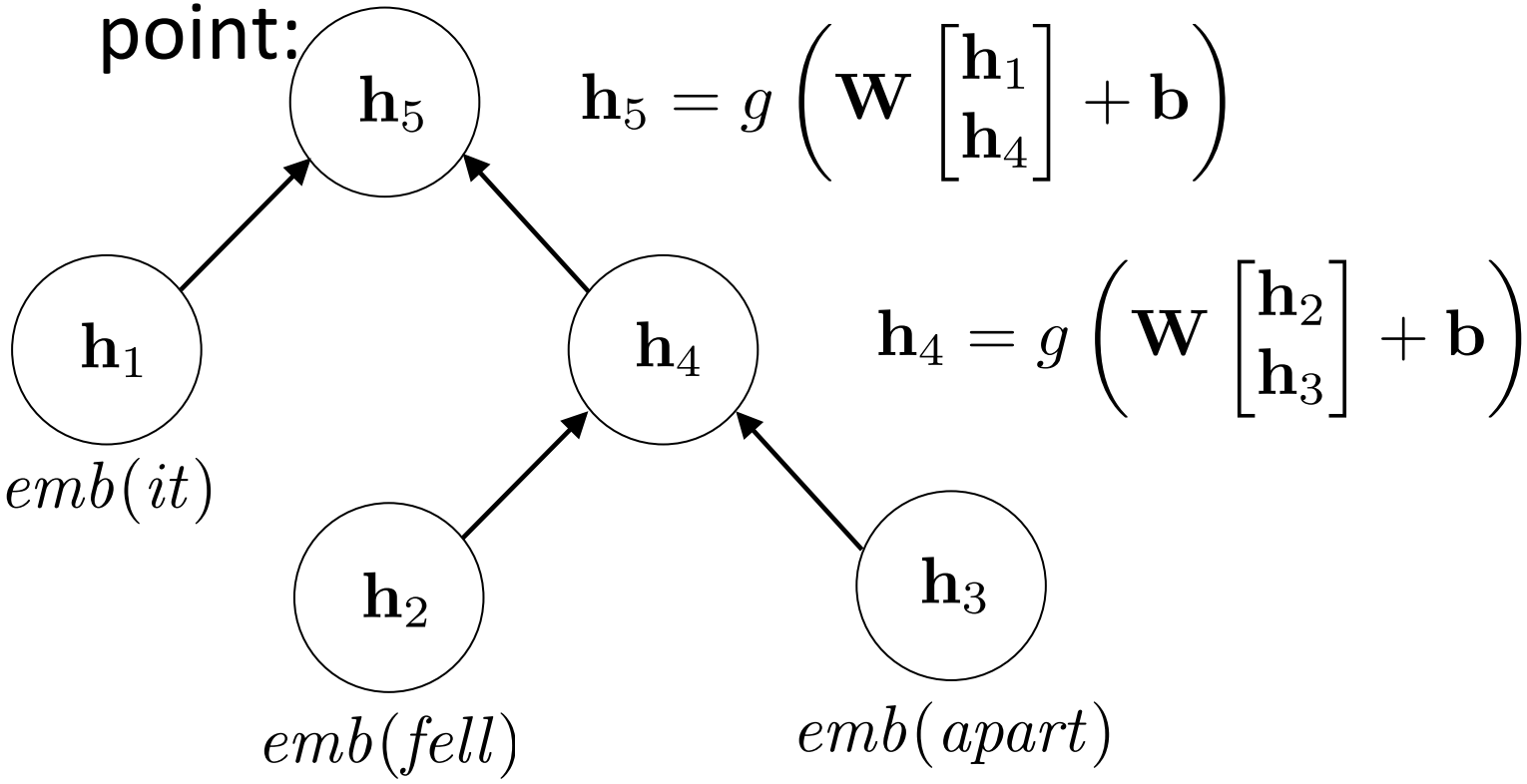


$$\mathbf{h}_4 = g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_2\\ \mathbf{h}_3\end{bmatrix} + \mathbf{b}\right)$$

Recursive Neural Networks for NLP

$x =$ *it fell apart*

- run a syntactic parser on the sentence
- construct vector recursively at each split point:



Recursive Neural Networks for NLP

- same parameters used at every split point
- order of children matters (different weights used for left and right child)