TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel

Spring 2017

Lecture 16:
Structured Prediction in NLP, Syntactic & Semantic Formalisms

- Assignment 3 due tomorrow
- Final project report due Friday, June 9
  - guidelines for final project report have been posted

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\text{classify}(x, \boldsymbol{\theta}) = \underset{y}{\text{argmax}} \; \text{score}(x, y, \boldsymbol{\theta})$$

learning: choose $\boldsymbol{\theta}$

Structured Prediction:

output space is exponentially-sized or unbounded

(we can't just enumerate all possible outputs)

- 2 categories of structured prediction:
  score-based and search-based

Score-Based Structured Prediction

- focus on defining the score function of the structured input/output pair:

$$\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

- cleanly separates score function, inference algorithm, and training loss

Inference in Score-Based SP

- inference algorithms are defined based on decomposition of score into parts

$$\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \sum_{\langle \boldsymbol{x}_r, \boldsymbol{y}_r \rangle \in \text{parts}(\boldsymbol{x}, \boldsymbol{y})} \text{score}(\boldsymbol{x}_r, \boldsymbol{y}_r, \boldsymbol{\theta})$$

- smaller parts = easier to define efficient exact inference algorithms

Loss Functions for Score-Based SP

name	loss	where used
cost ("0-1")	$\text{cost}(\boldsymbol{y}, \text{predict}(\boldsymbol{x}, \boldsymbol{\theta}))$	MERT (Och, 2003)
percep-tron	$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} \text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta})$	structured perceptron (Collins, 2002)
hinge	$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} (\text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta}) + \text{cost}(\boldsymbol{y}, \boldsymbol{y}'))$	structured SVMs (Taskar et al., *inter alia*)
log	$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \log \sum_{\boldsymbol{y}'} \exp \{\text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta})\}$	CRFs (Lafferty et al., 2001)
softmax-margin	$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \log \sum_{\boldsymbol{y}'} \exp \{\text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta}) + \text{cost}(\boldsymbol{y}, \boldsymbol{y}')\}$	Povey et al. (2008), Gimpel & Smith (2010)

Inference Algorithms for Score-Based SP

- dynamic programming
  - exact, but parts must be small for efficiency
- dynamic programming + "cube pruning"
  - permits approximate incorporation of large parts ("non-local features") while still using dynamic program backbone
- integer linear programming

Search-Based Structured Prediction

- focus on the procedure for searching through the structured output space (usually involves simple greedy or beam search)
- design a classifier to score a small number of decisions at each position in the search
  - this classifier can use information about the current state as well as the entire history of the search

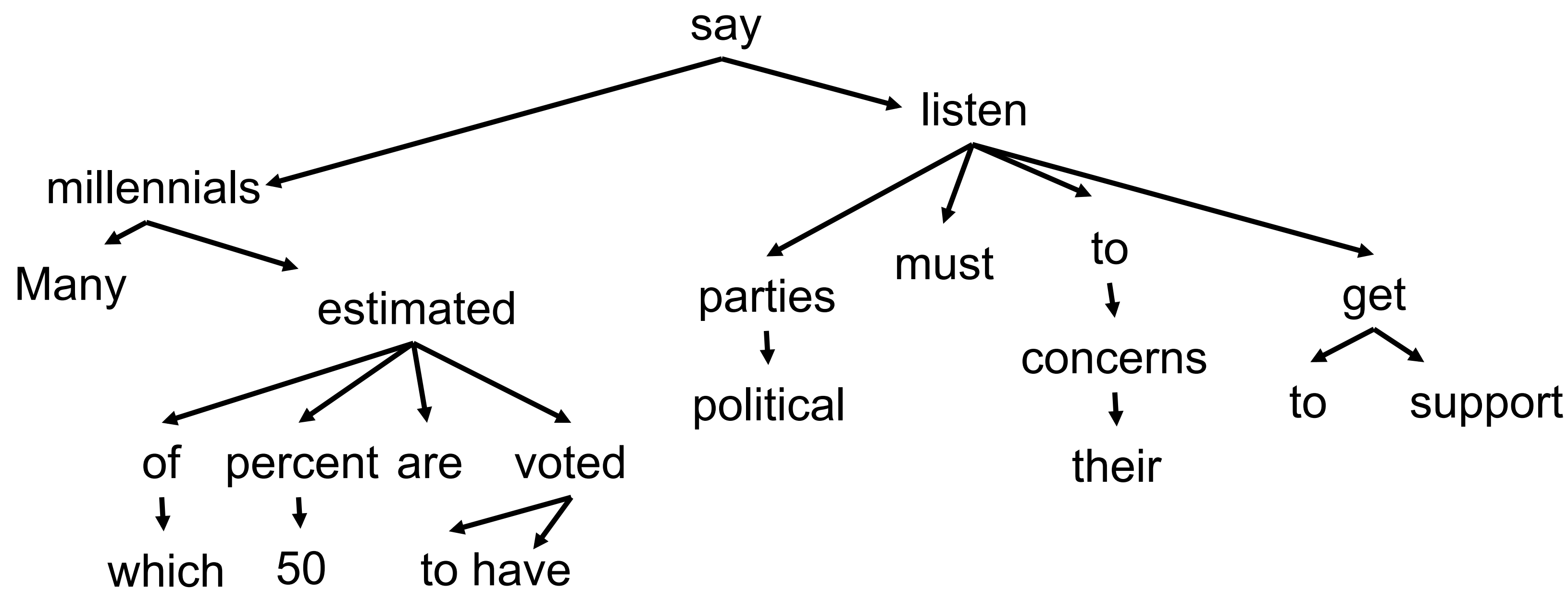
- in dependency parsing, this is called "transition-based parsing" because it consists of greedily, sequentially deciding what parsing decision to make

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!)

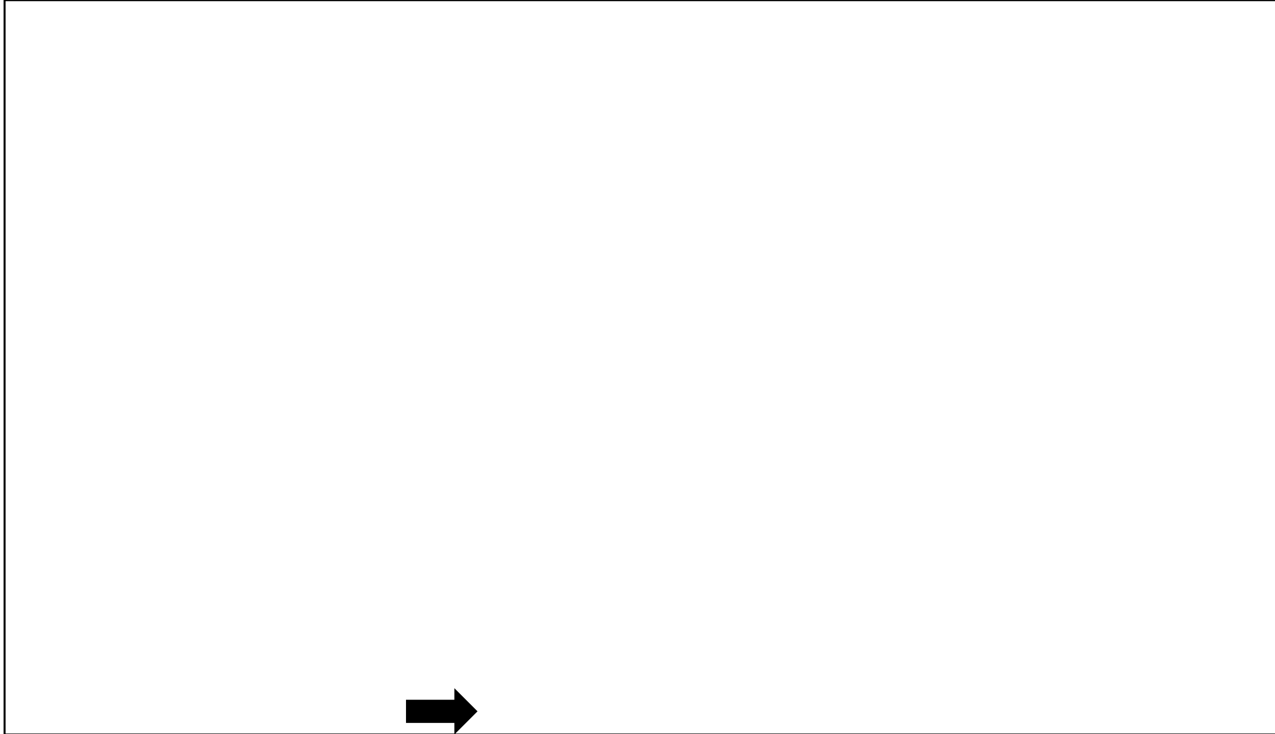
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

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



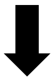
Greedy Parsing with a Stack

Stack:



See:
Nivre & Scholz, 2004
Henderson, 2004

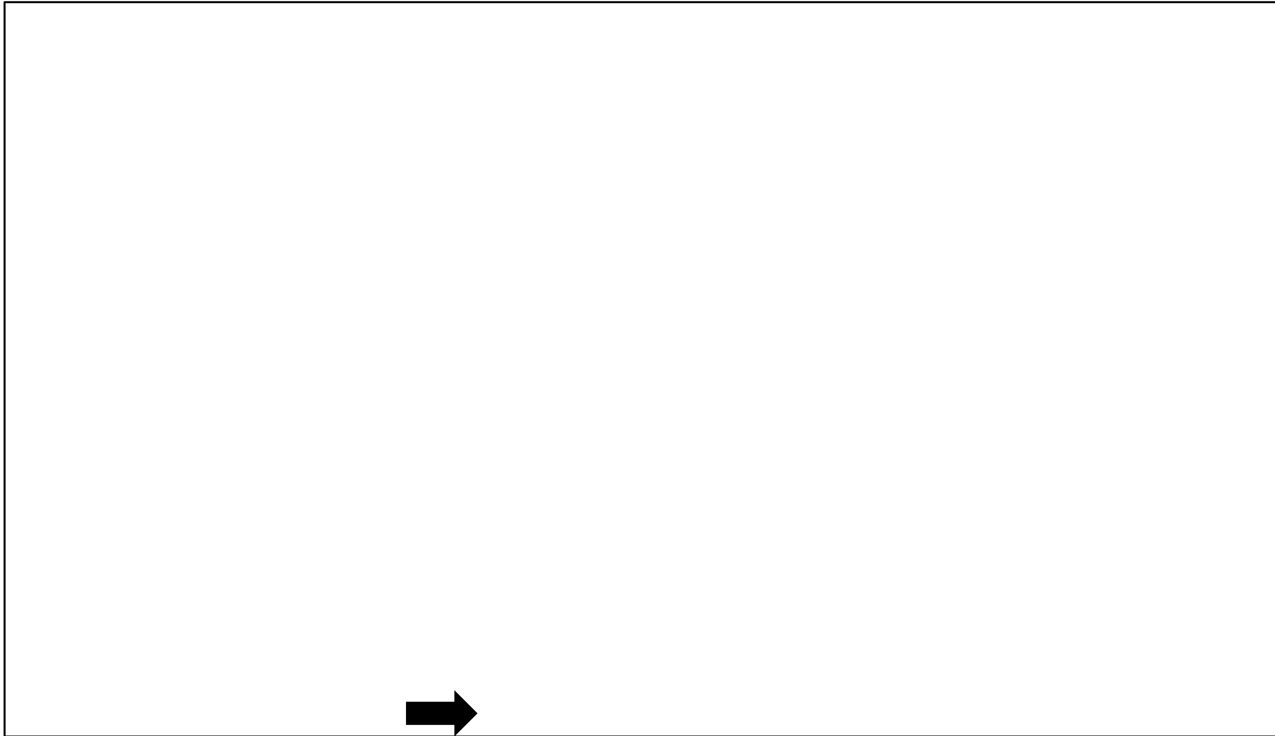
Buffer:



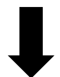
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.

Greedy Parsing with a Stack

Stack:



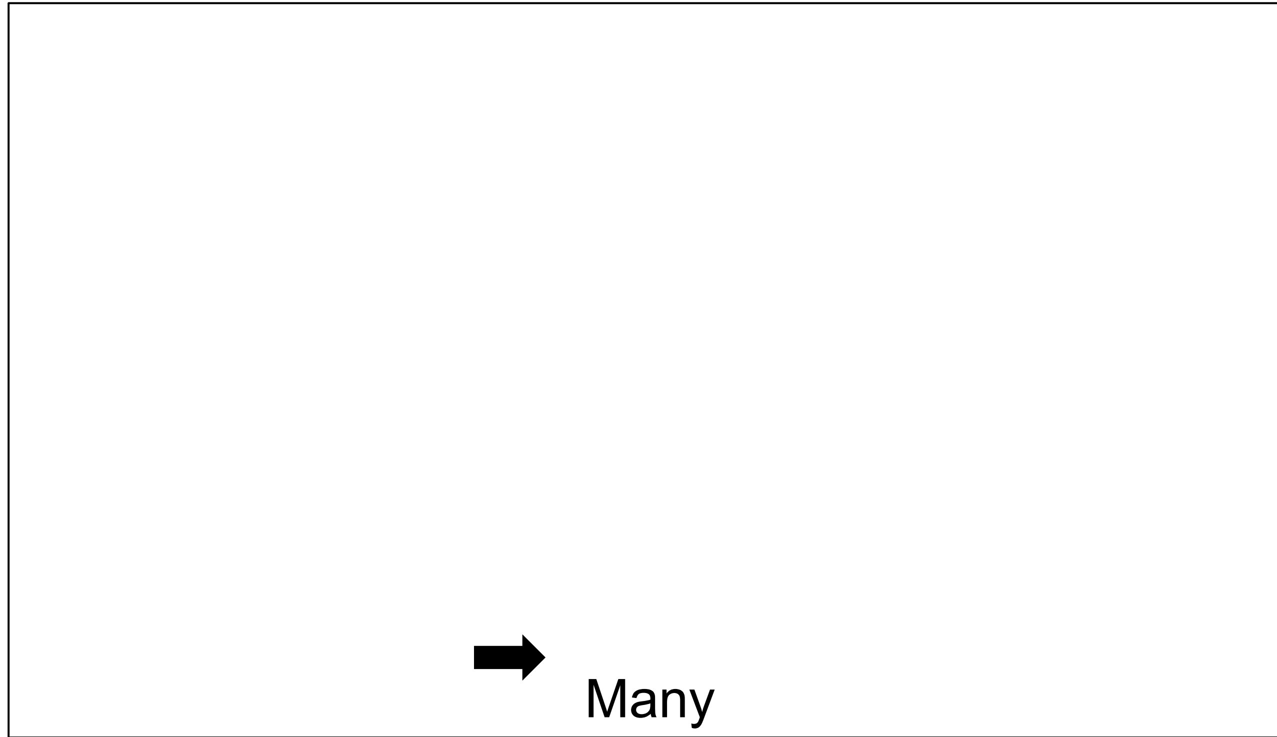
Buffer:



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

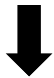
Greedy Parsing with a Stack

Stack:



shift

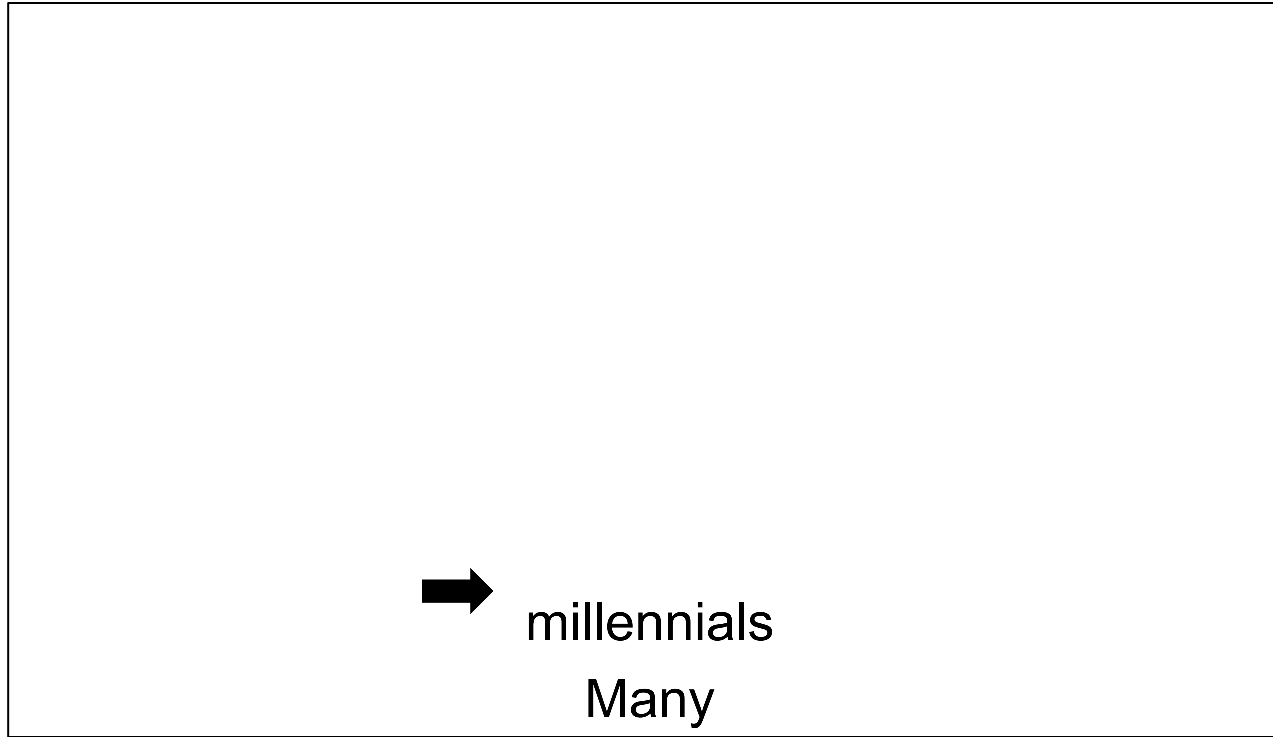
Buffer:



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

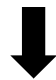
Greedy Parsing with a Stack

Stack:



shift

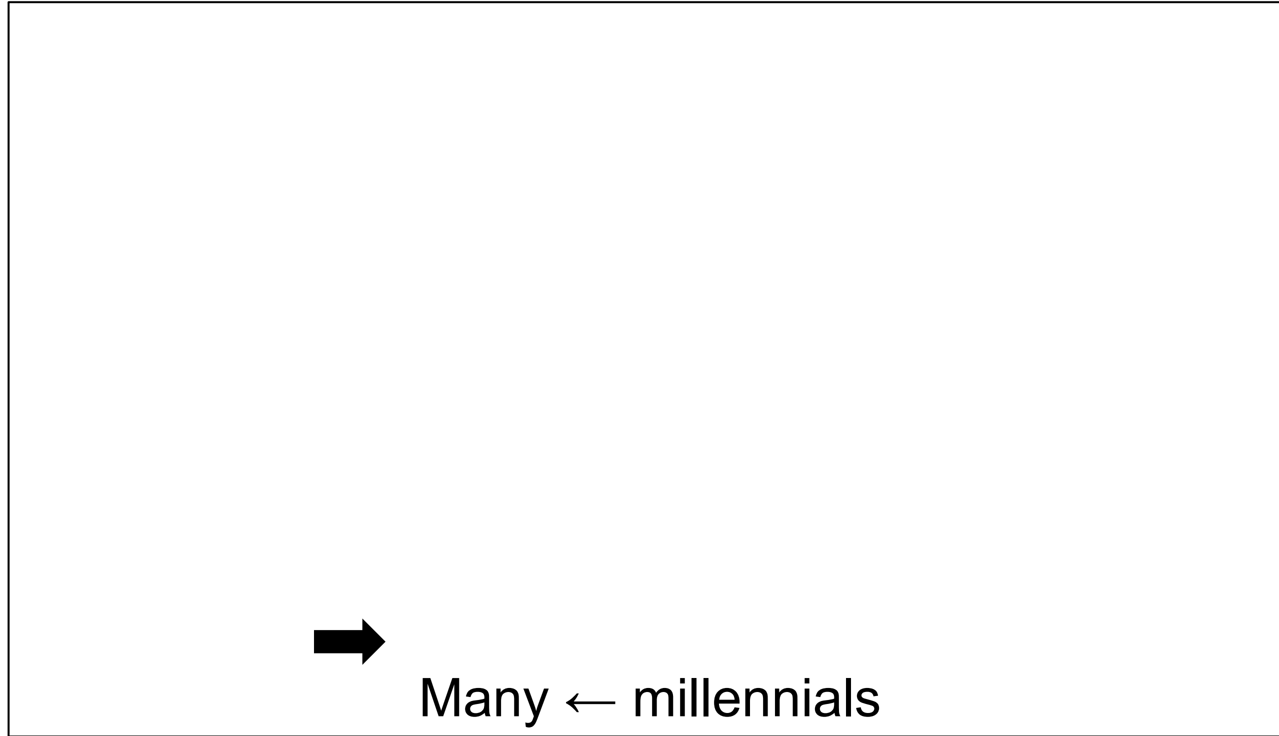
Buffer:



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

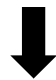
Greedy Parsing with a Stack

Stack:



reduce left

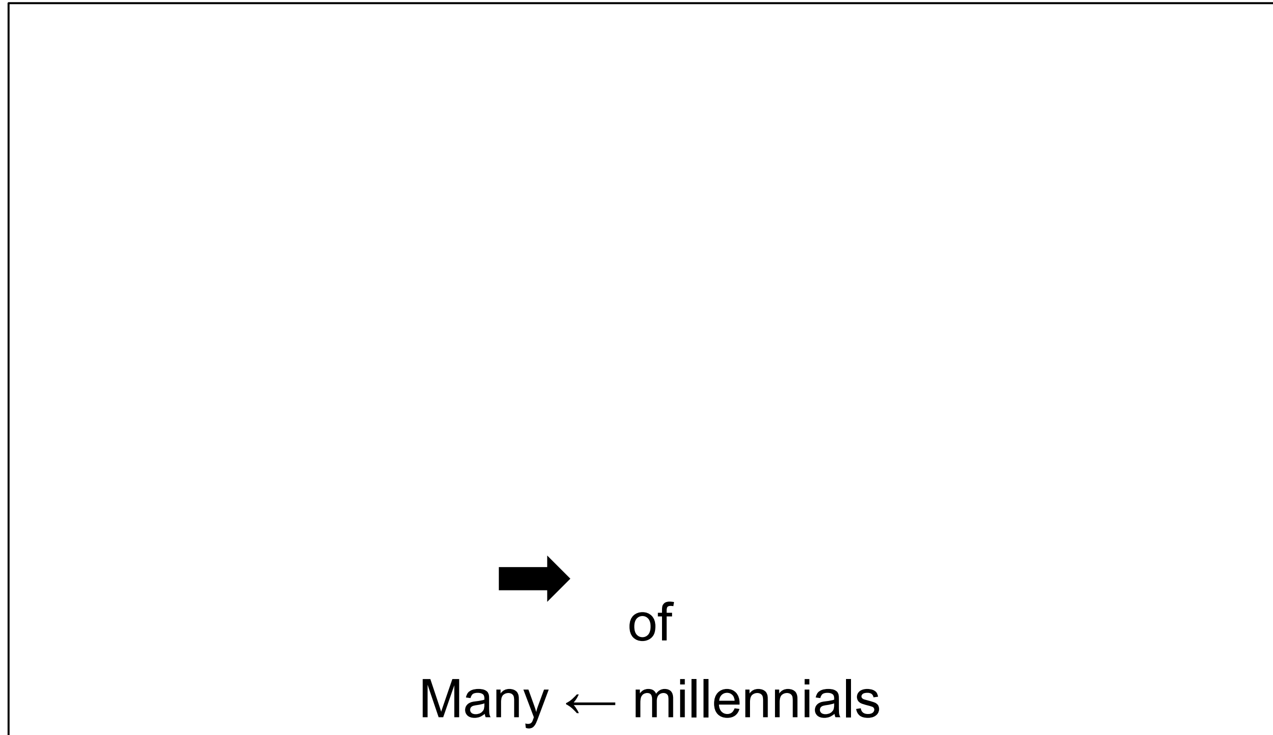
Buffer:



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:



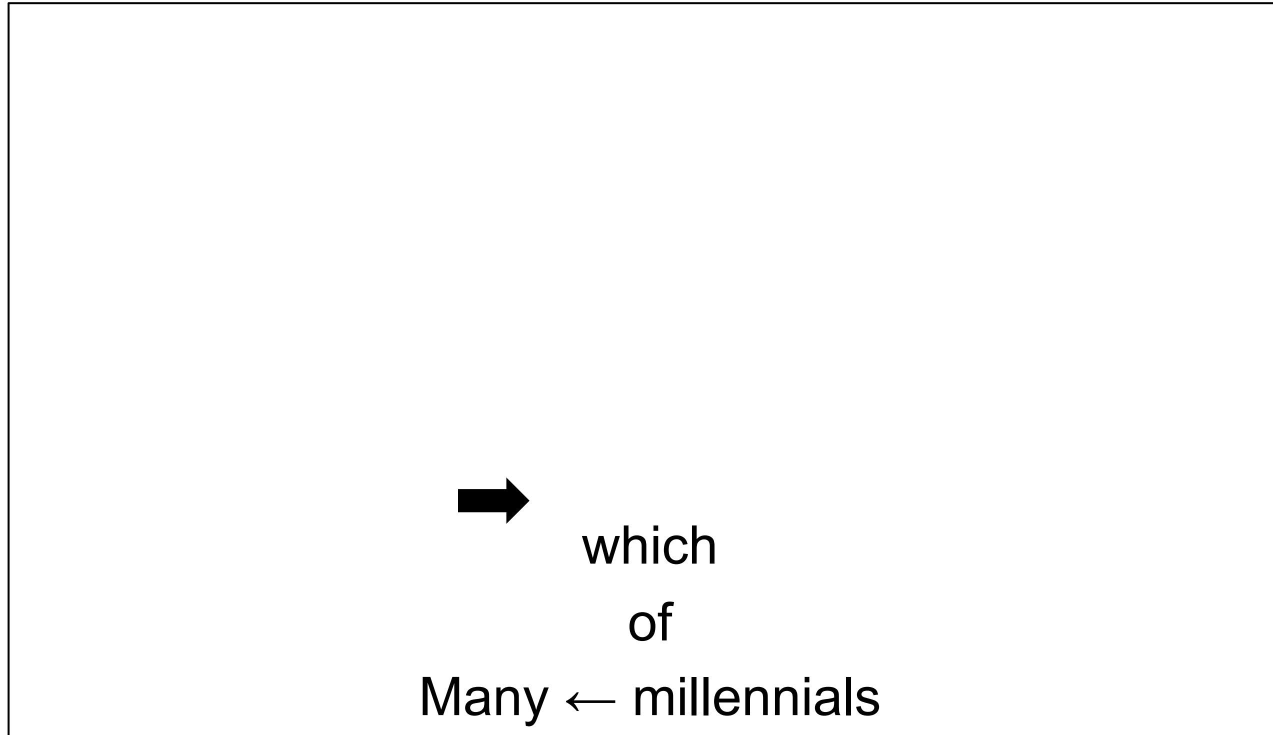
shift

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:



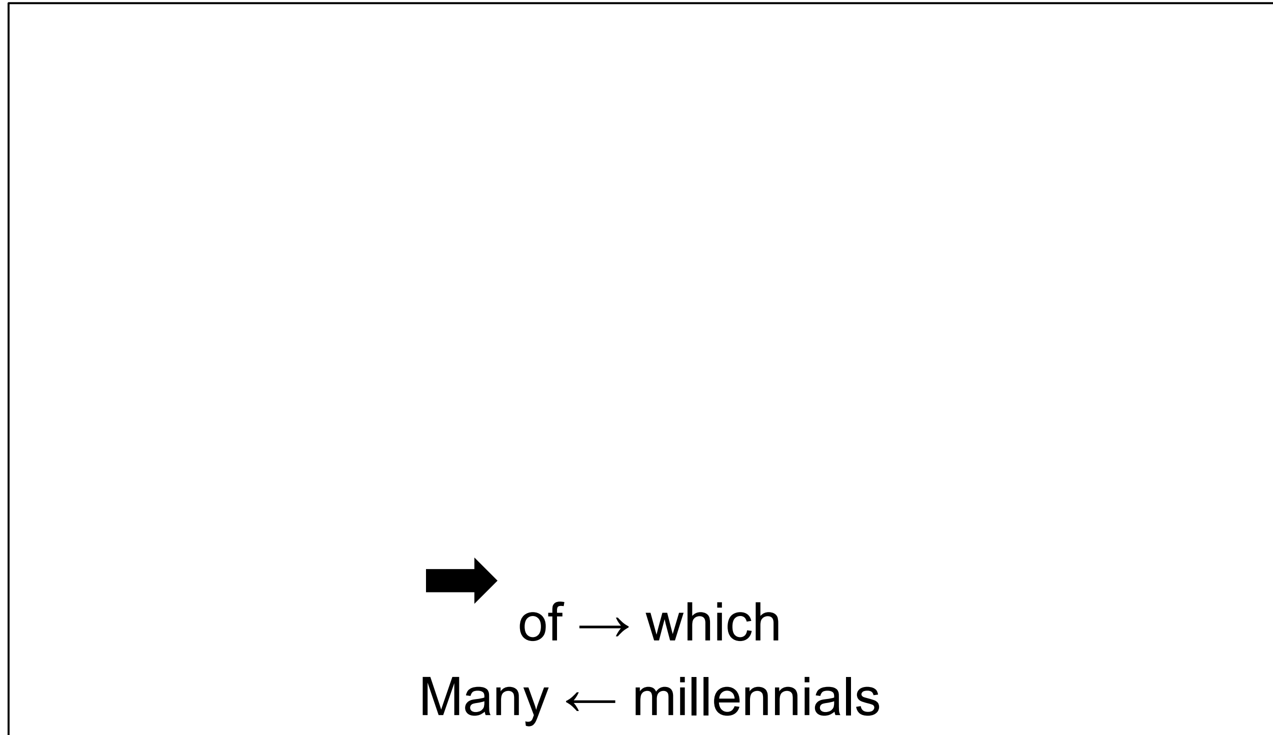
shift

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:



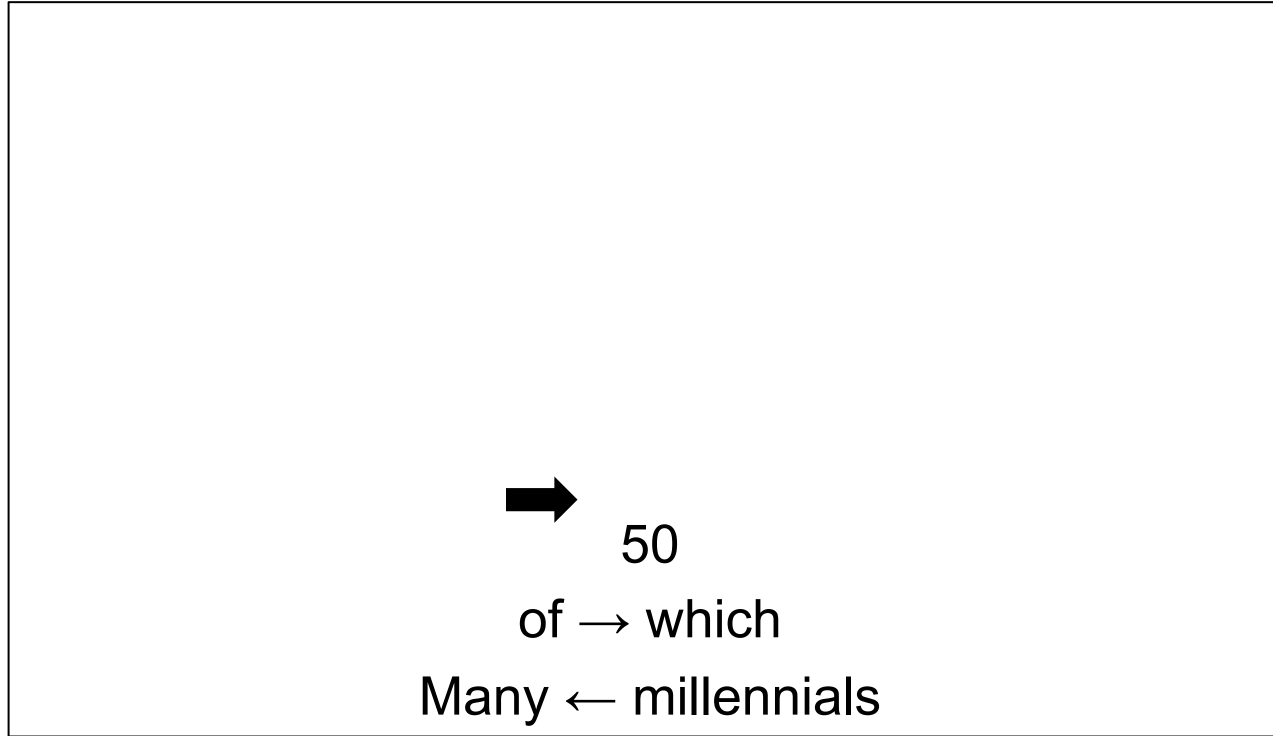
reduce right

Buffer:

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

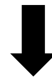
Greedy Parsing with a Stack

Stack:



shift

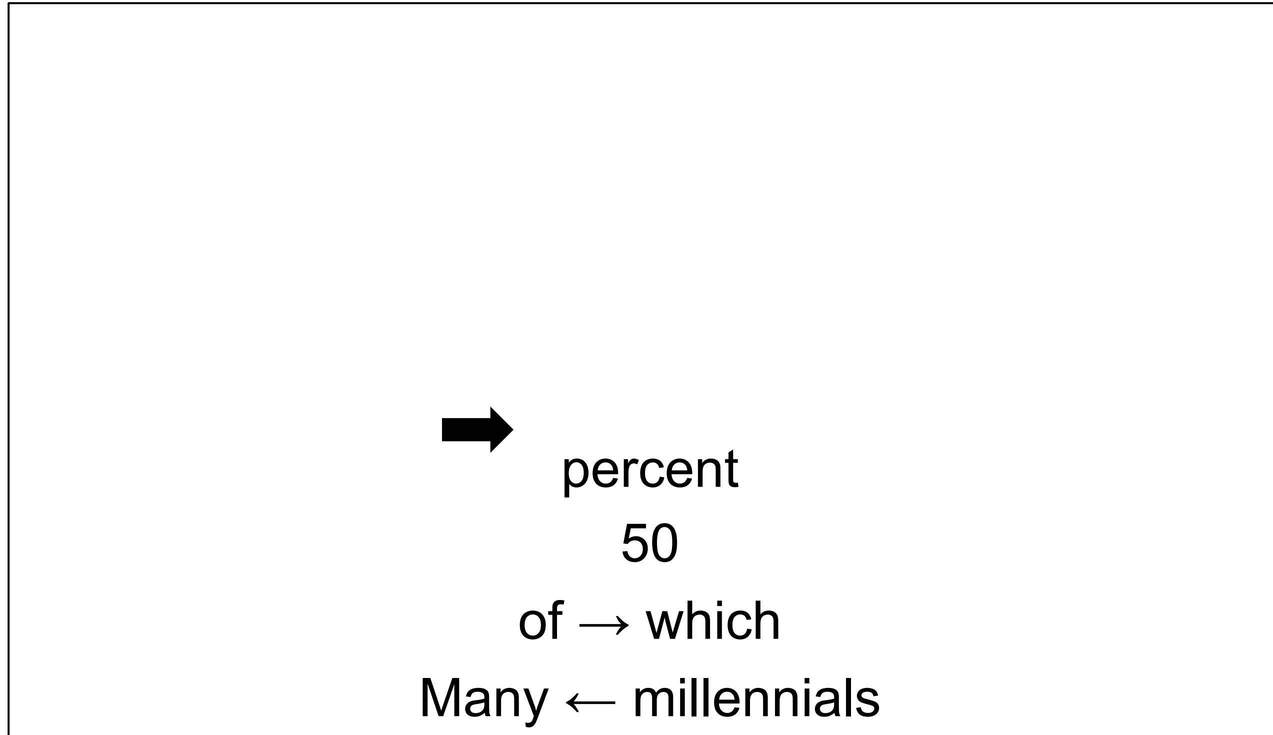
Buffer:



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:



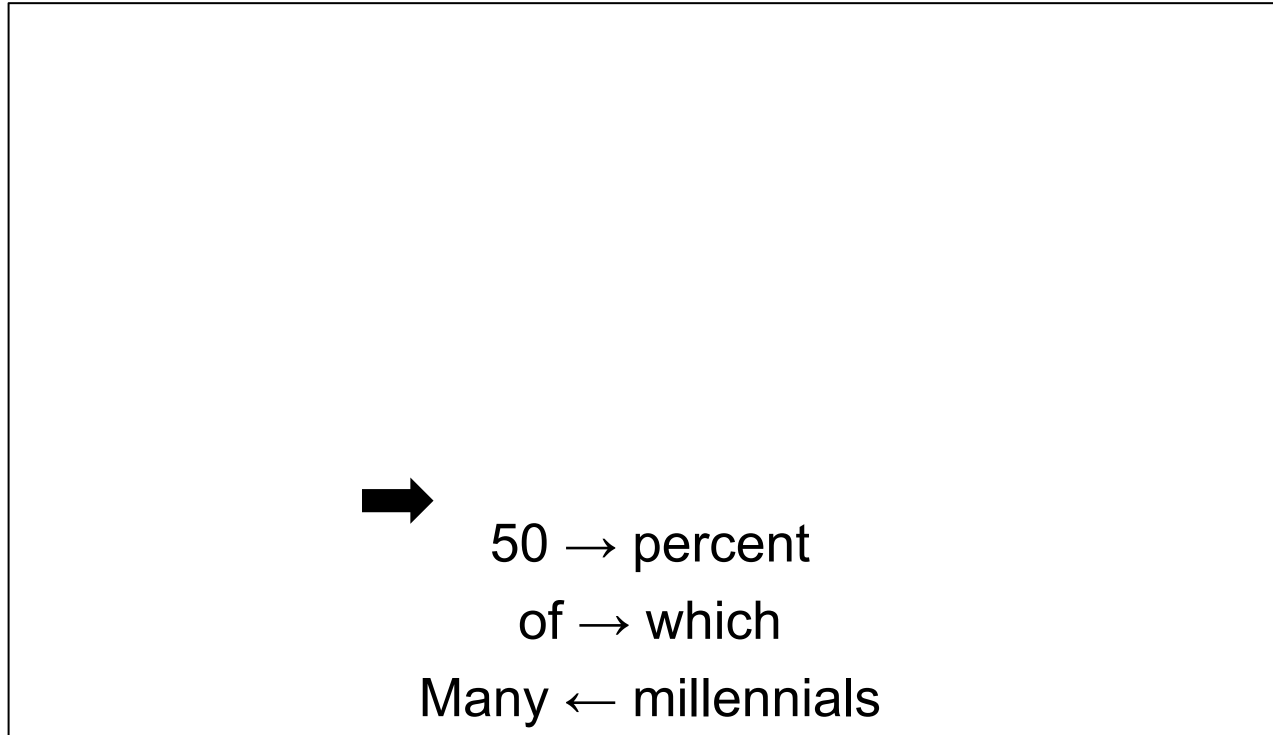
shift

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:



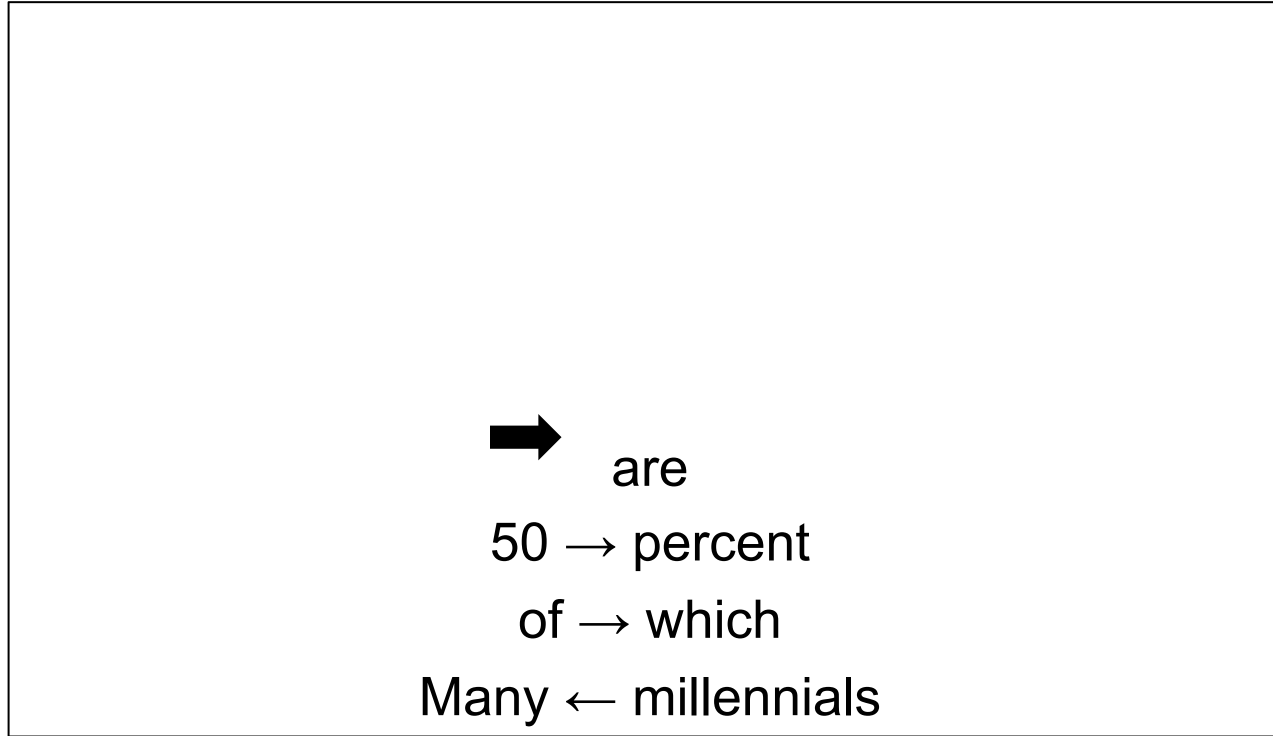
reduce right

Buffer:

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

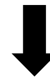
Greedy Parsing with a Stack

Stack:



shift

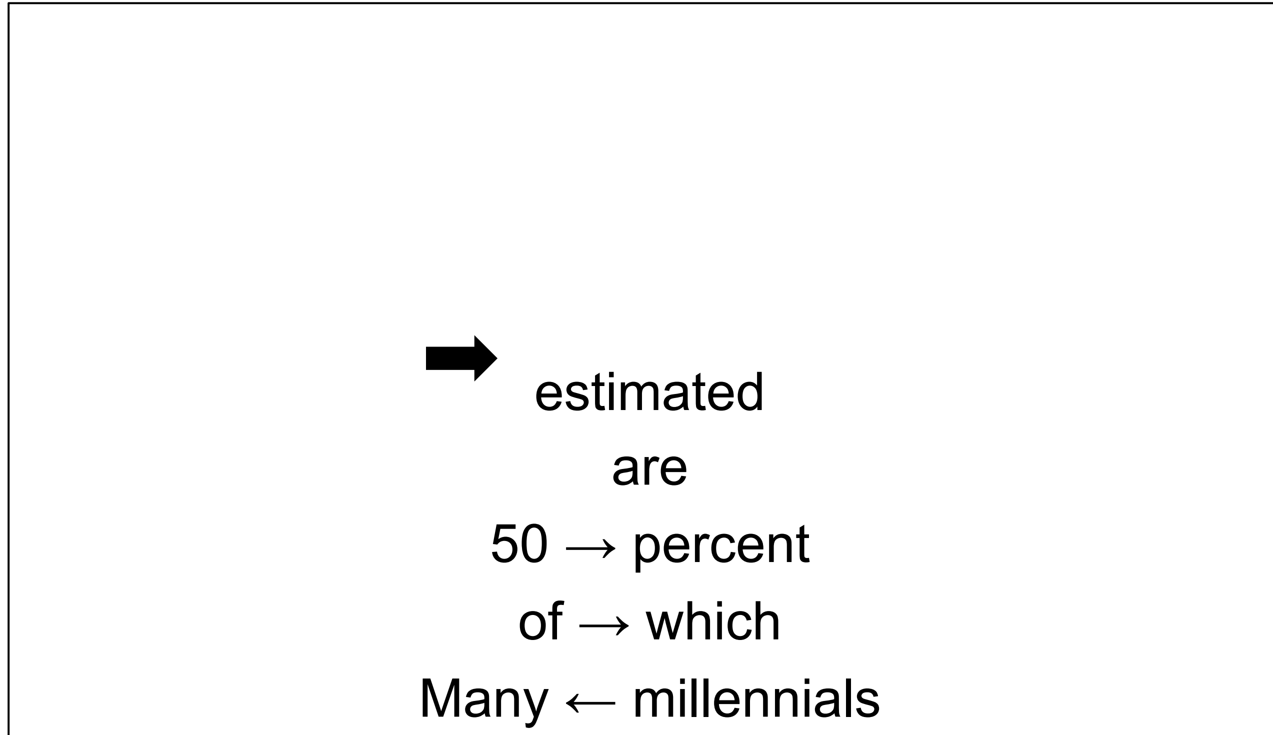
Buffer:



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:



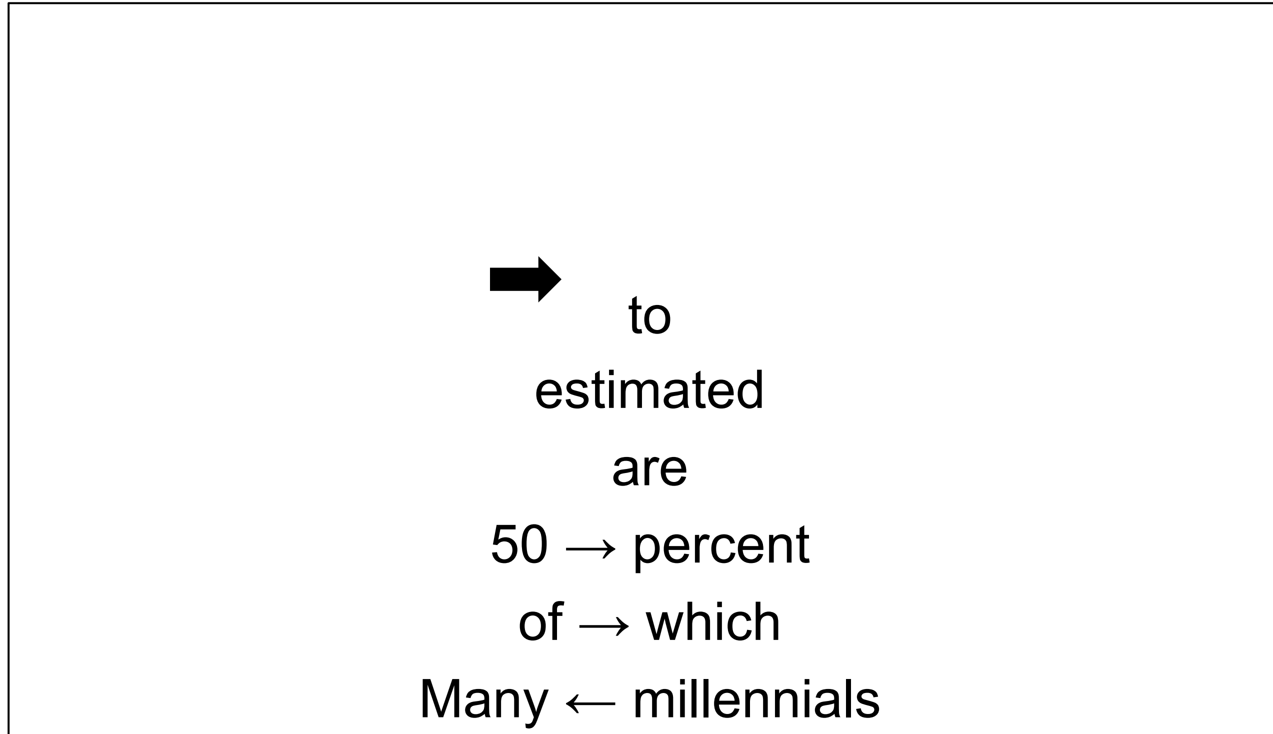
shift

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:



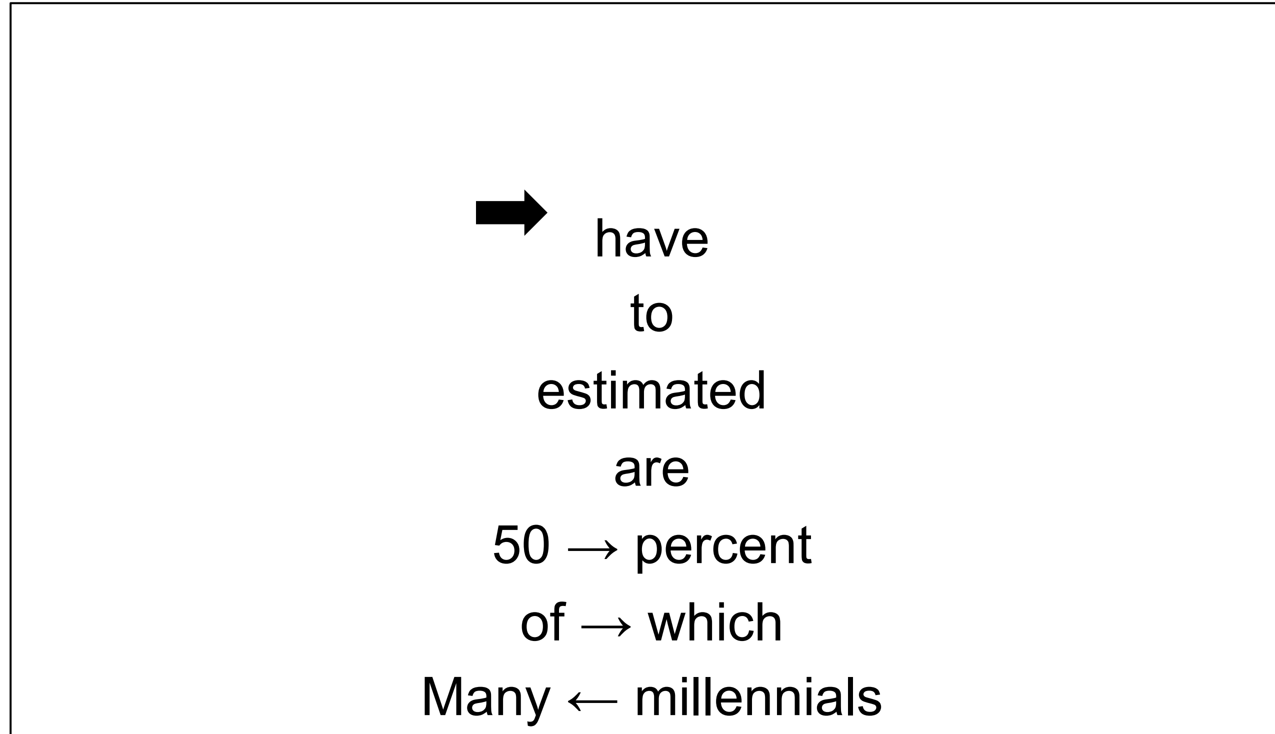
shift

Buffer:

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

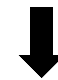
Greedy Parsing with a Stack

Stack:



shift

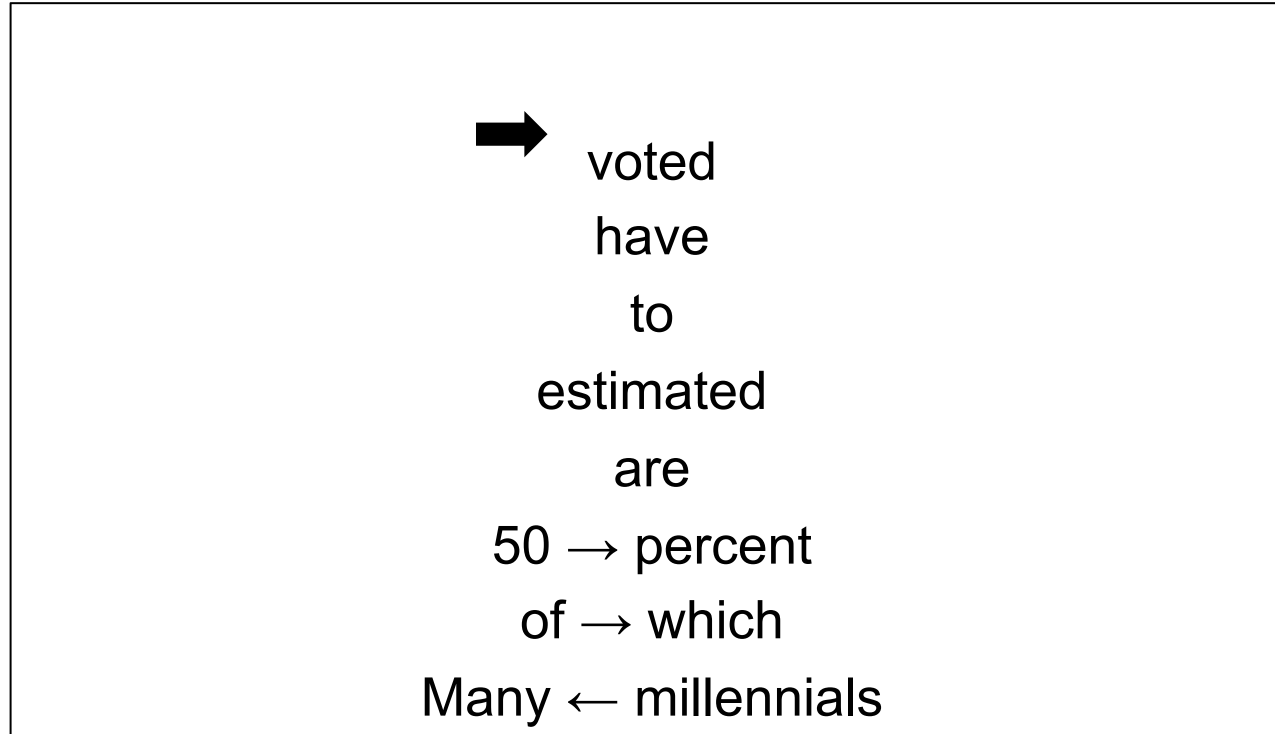
Buffer:



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

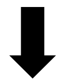
Greedy Parsing with a Stack

Stack:



shift

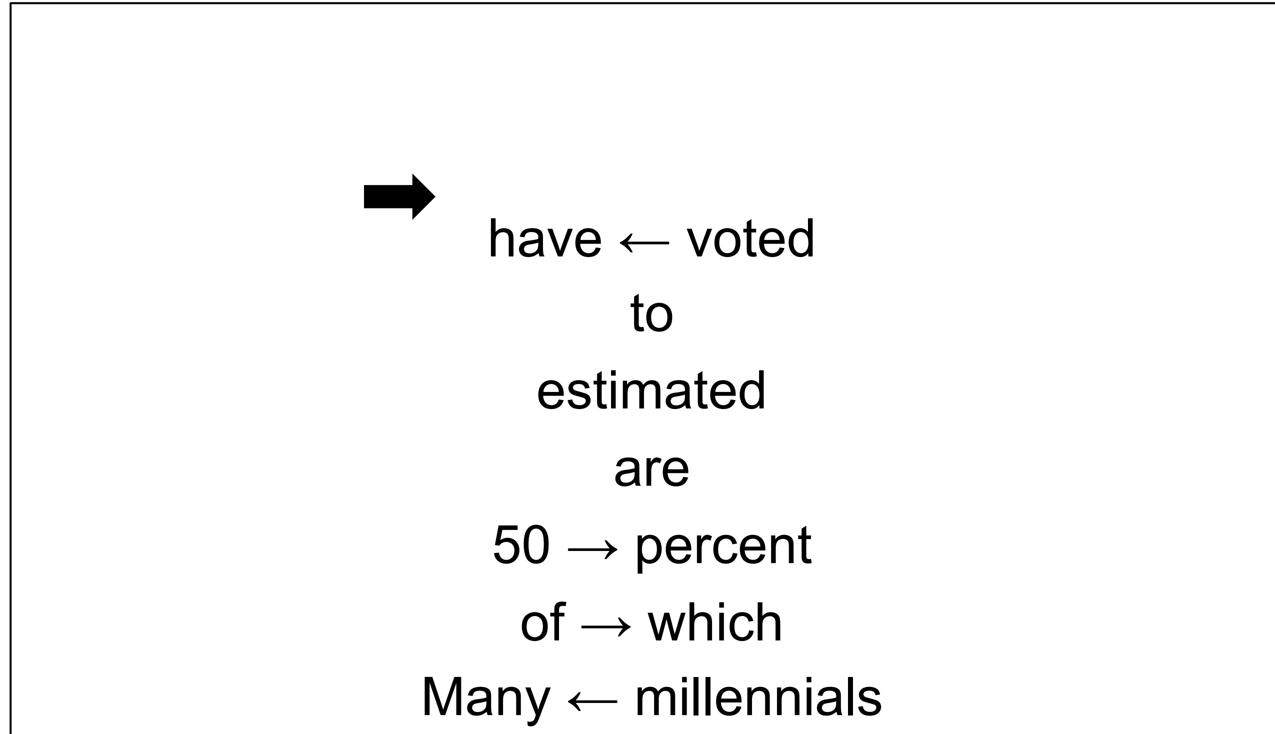
Buffer:



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

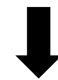
Greedy Parsing with a Stack

Stack:



reduce left

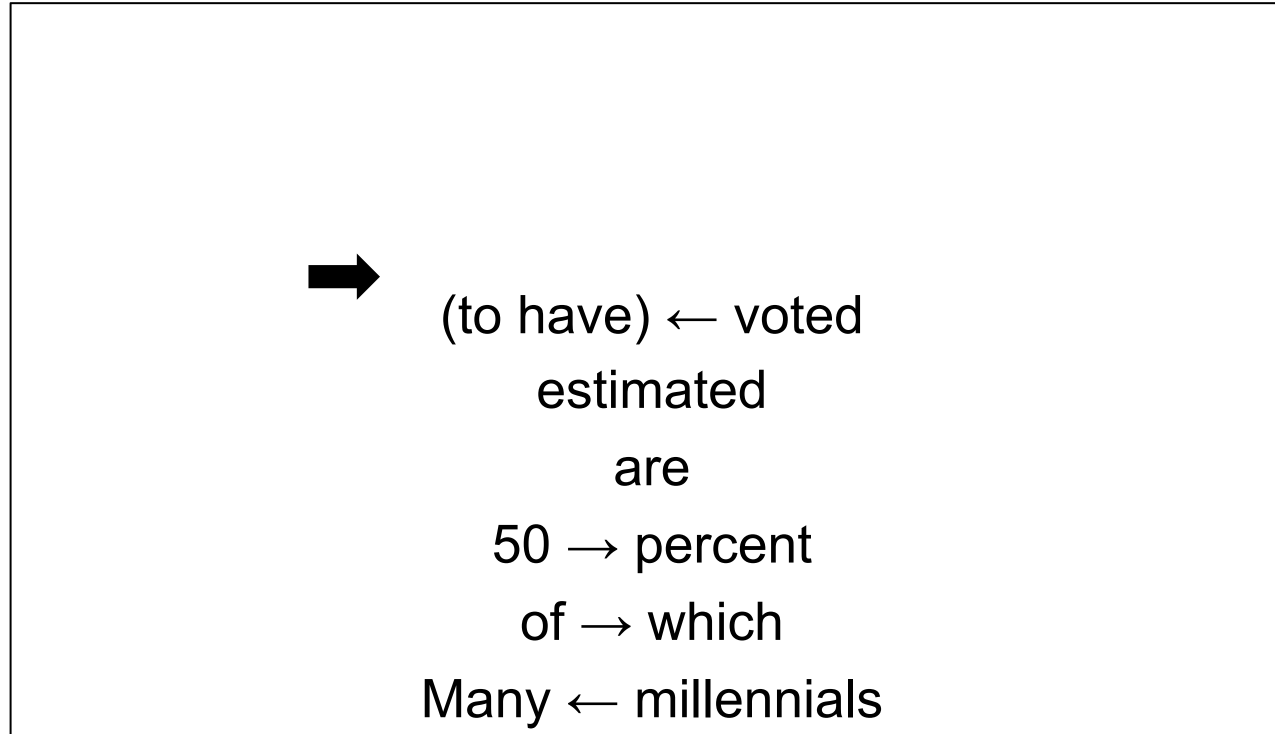
Buffer:



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

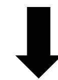
Greedy Parsing with a Stack

Stack:



reduce left

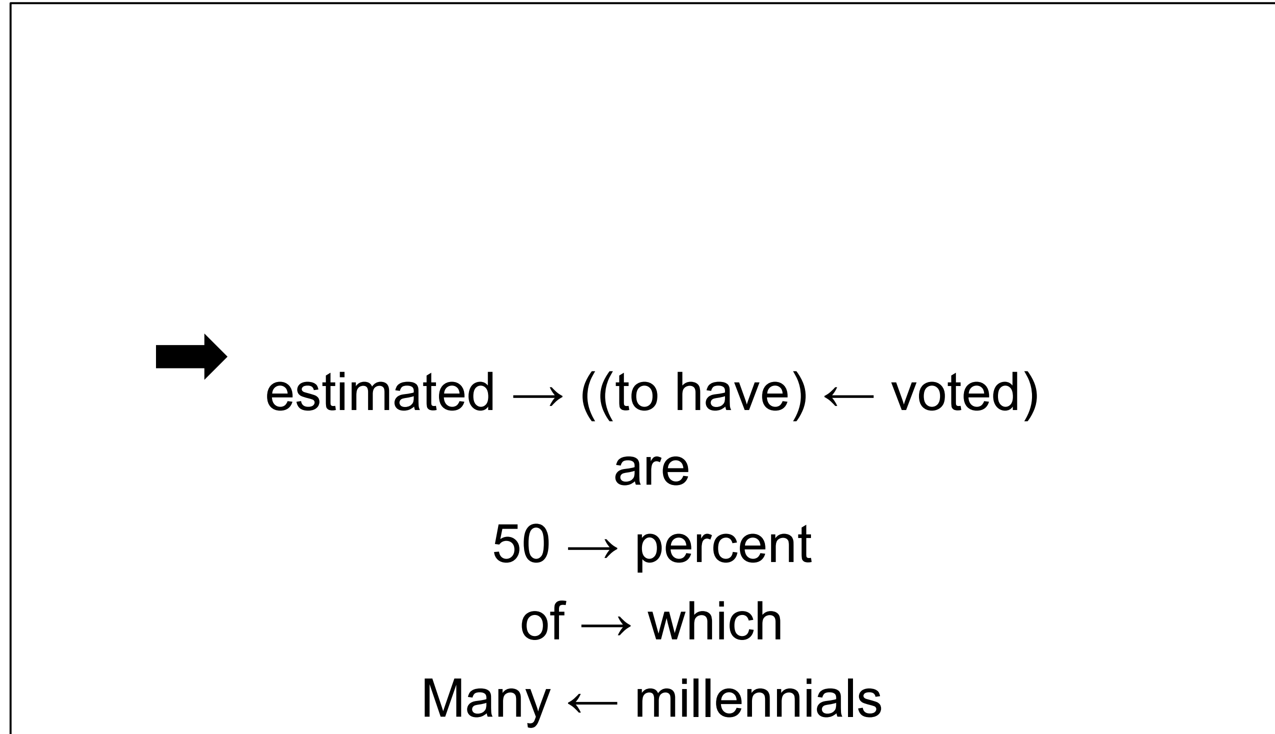
Buffer:



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

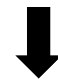
Greedy Parsing with a Stack

Stack:



reduce right

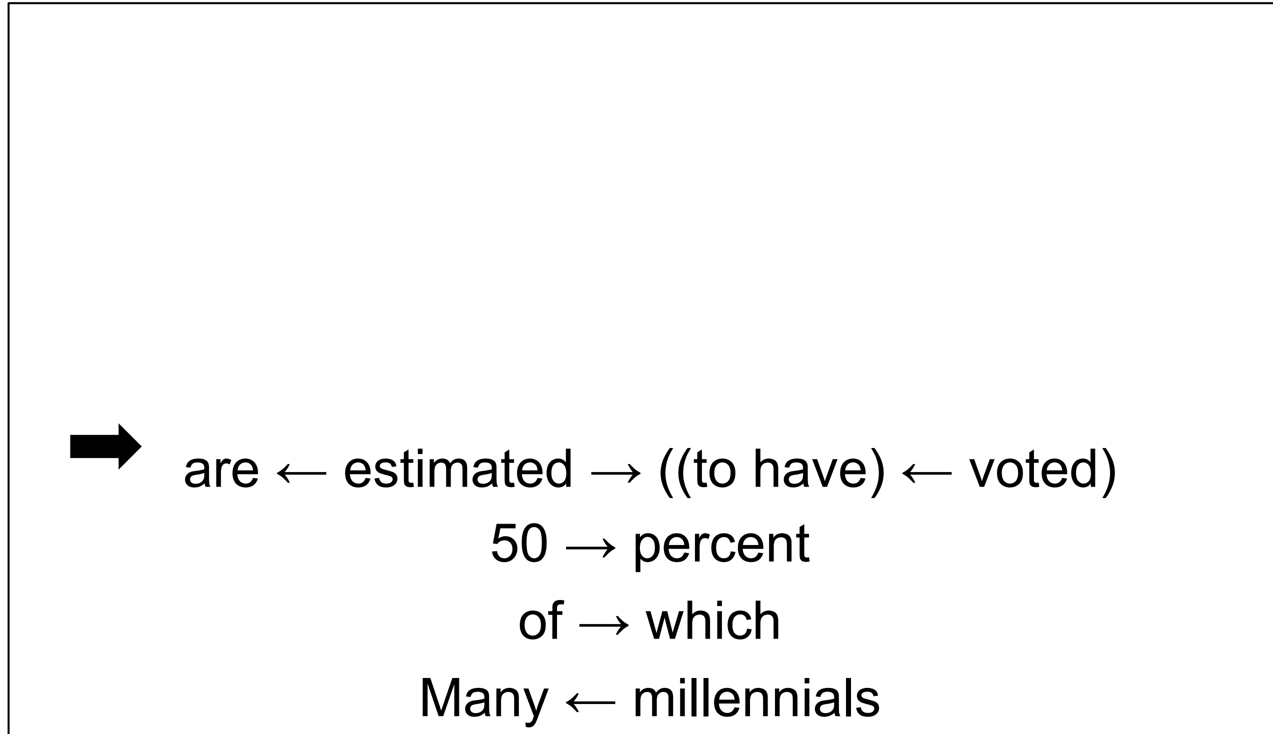
Buffer:



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

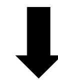
Greedy Parsing with a Stack

Stack:

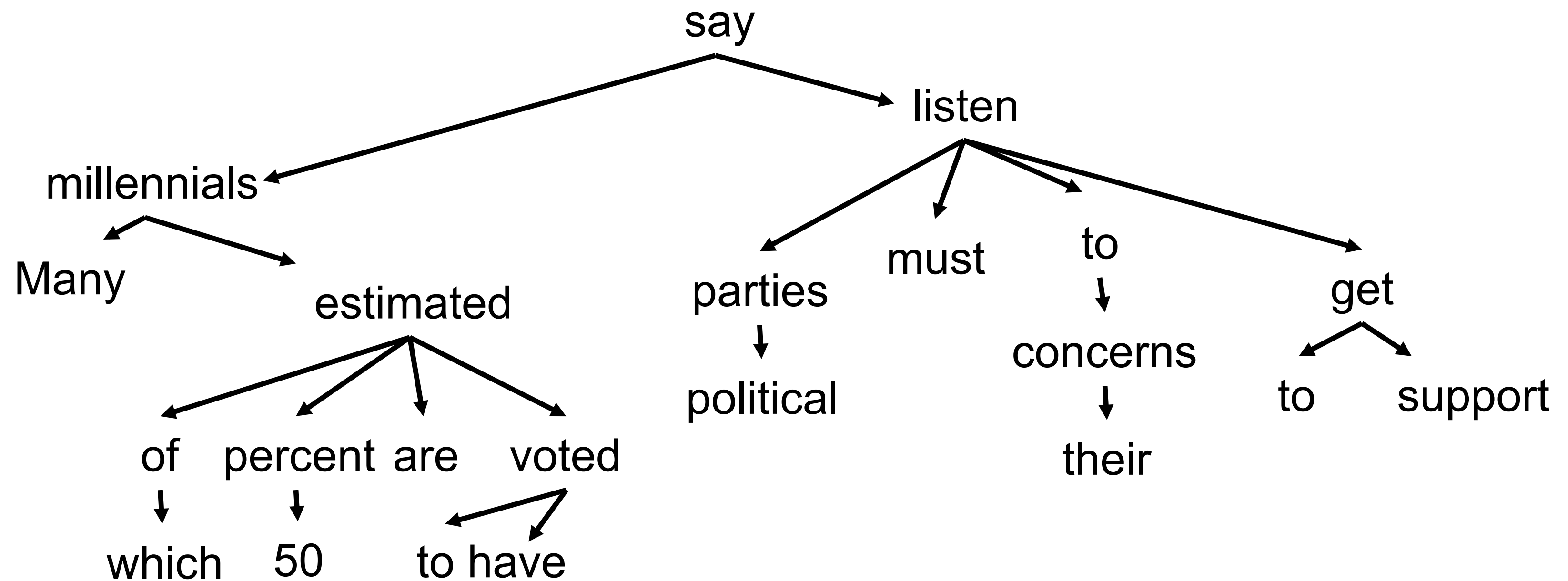


reduce left

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.

$\textbf{Stack}_t$	$\textbf{Buffer}_t$	**Action**	$\textbf{Stack}_{t+1}$	$\textbf{Buffer}_{t+1}$	**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)

- Chen et al. (2014) used a feed-forward network to output a parsing decision (shift, reduce-left, or reduce-right)
- 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

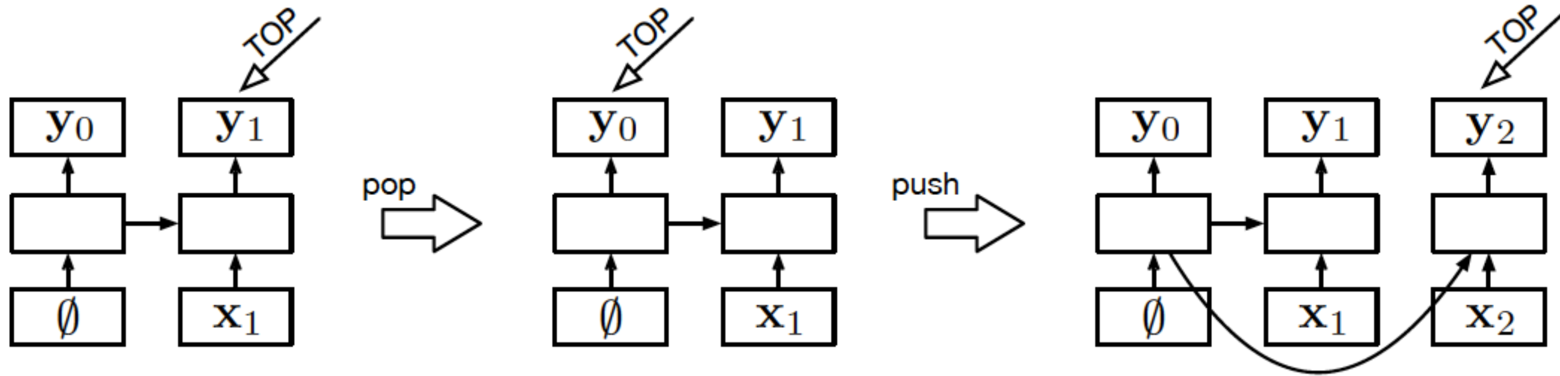


Figure 1: A stack LSTM extends a conventional left-to-right LSTM with the addition of a stack pointer (notated as TOP in the figure). This figure shows three configurations: a stack with a single element (left), the result of a pop operation to this (middle), and then the result of applying a push operation (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)

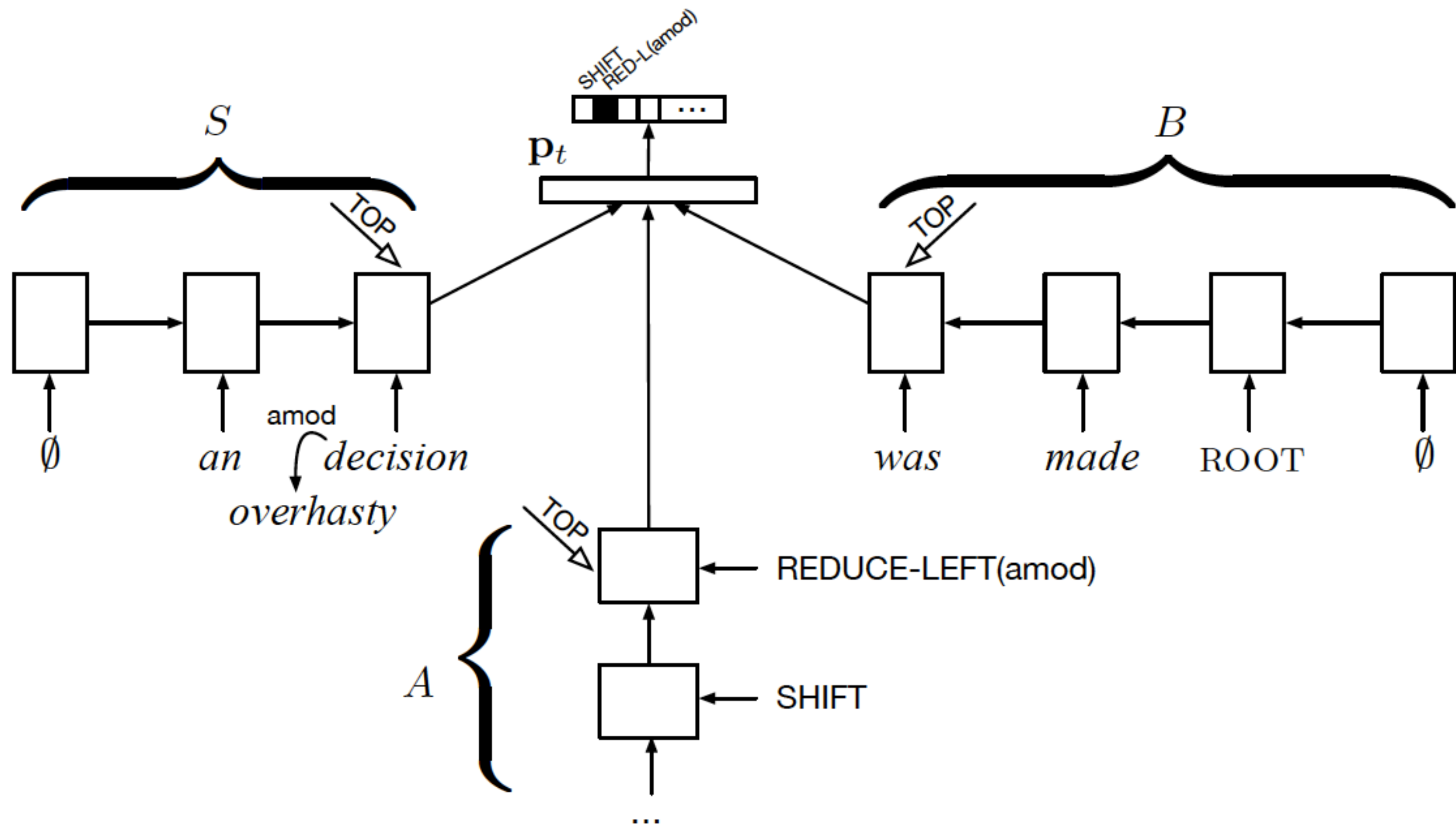
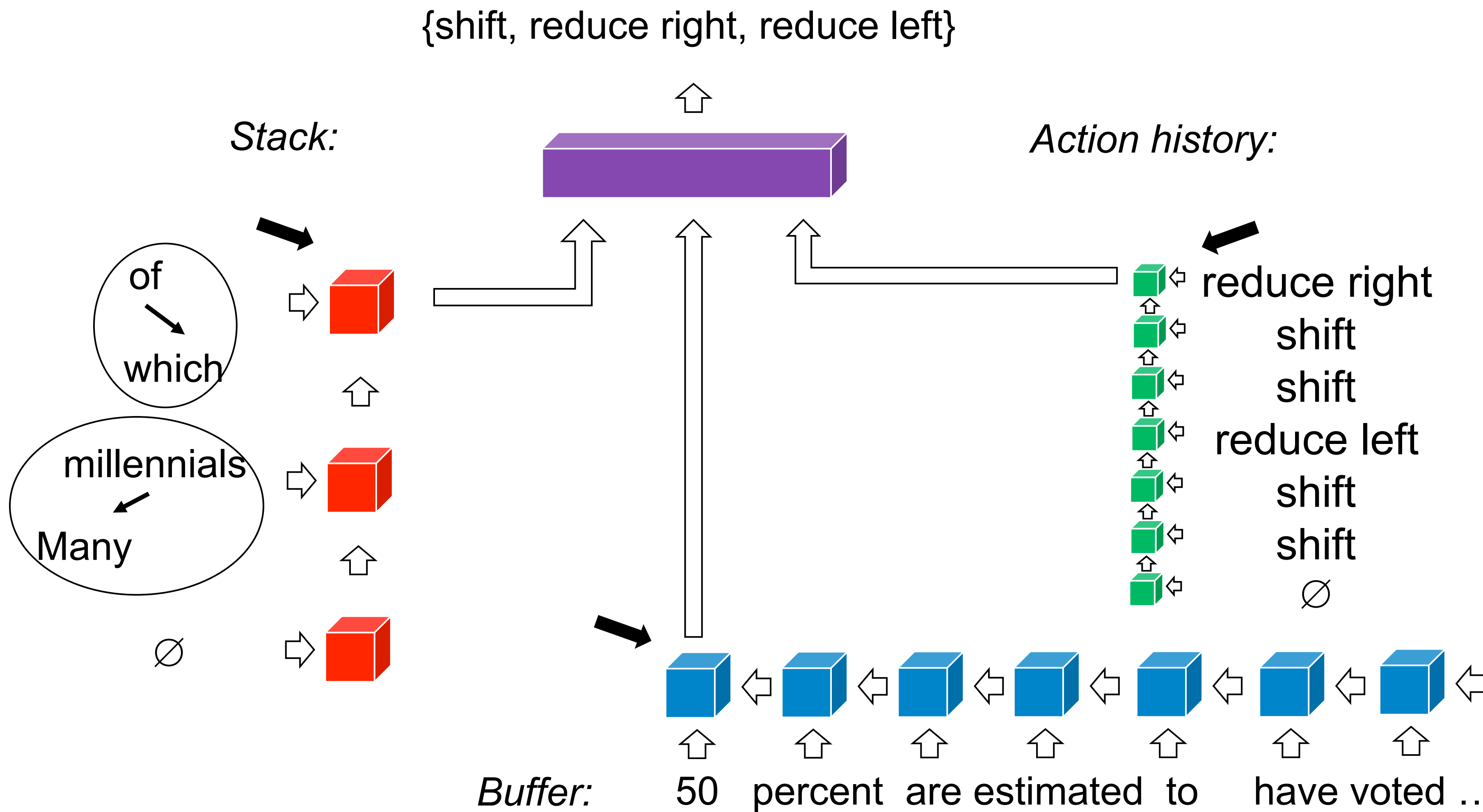


Figure 2: Parser state computation encountered while parsing the sentence "*an overhasty decision was made*." Here S designates the stack of partially constructed dependency subtrees and its LSTM encoding; B is the buffer of words remaining to be processed and its LSTM encoding; and A is the stack representing 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



- we’ve talked about constituency and dependency parsing in this course and in 31190
- what about other syntactic & semantic formalisms?
- today we’ll cover 2 you should know about:
  - AMR
  - CCG

http://tiny.cc/amrtutorial

The Logic of AMR

Practical, Unified, Graph-Based Sentence Semantics for NLP

Nathan Schneider University of Edinburgh
Jeff Flanigan CMU
Tim O'Gorman CU-Boulder

Note: slides from this section have been removed due to large size. Please see the original tutorial slides by Schneider/Flanigan/O'Gorman

Combinatory Categorial Grammar
(Steedman, 1987)

- family of grammars that focus on **function application**
- CCGs are useful for semantic parsing and parsing to logical forms
- in one simple CCG instantiation, there are only 2 atomic types: nouns (N) and sentences (S)

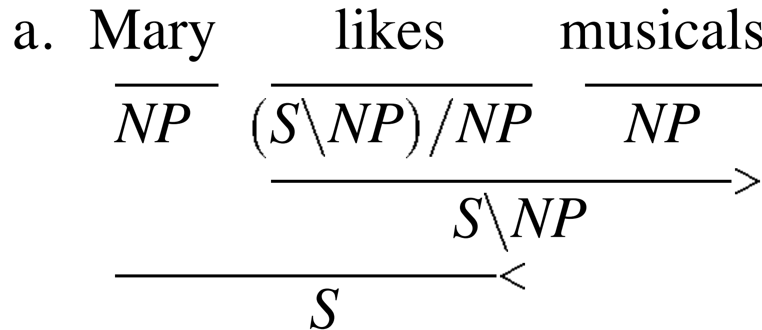
CCG

- 2 atomic types: nouns (N) and sentences (S)
- complex types created by using “slash” rules; think of these as “functions”:
  - X/Y = “something that combines with a Y **to its right** to form an X”
  - X\Y = “something that combines with a Y **to its left** to form an X”
- Consider the type S\N:
  - what are some examples of words that would have this type?
  - that is, what are some words that, when preceded by a noun, form a sentence?
  - verbs like sleeps, ate, walked

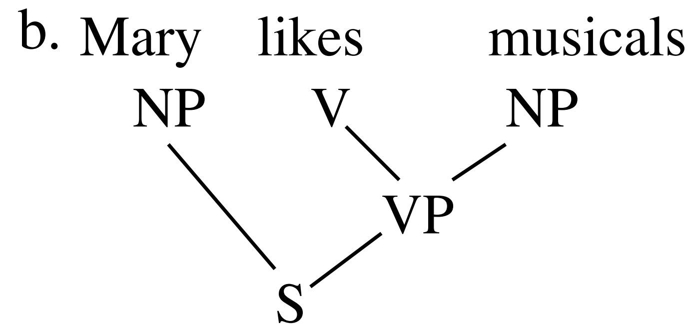
Other CCG Types

- How about (S\N)/N?
  - transitive verbs: likes, sees, ate, etc

CCG



PCFG



Forward Application: (>)

$X/Y \quad Y \quad \Rightarrow \quad X$

Backward Application: (<)

$Y \quad X\backslash Y \quad \Rightarrow \quad X$

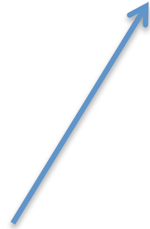
Steedman (1996)

Other CCG Types

- How about N/N?
  - determiners, adjectives, nouns

Function Application as an Isomorphic Hierarchical Procedure:

likes := $(S\backslash NP_{3s})/NP : like'$



the part after the colon (:) is the "semantic" component

Function Application as an Isomorphic Hierarchical Procedure:

We must also expand the rules of functional application in the same way:

(6) *Forward Application:* ($>$)

$$X/Y : f \quad Y : a \quad \Rightarrow \quad X : fa$$

(7) *Backward Application:* ($<$)

$$Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : fa$$

Function Application as an Isomorphic Hierarchical Procedure:

(5) likes := $(S\backslash NP_{3s})/NP : like'$

We must also expand the rules of functional application in the same way:

(6) *Forward Application:* (>)

$$X/Y : f \quad Y : a \quad \Rightarrow \quad X : fa$$

(7) *Backward Application:* (<)

$$Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : fa$$

They yield derivations like the following:

(8)

Mary	likes	musicals
$NP_{3sm} : mary'$	$(S\backslash NP_{3s})/NP : like'$	$NP : musicals'$
	$S\backslash NP_{3s} : like' musicals'$ (>)	
$S : like' musicals' mary$ (<)		

Conclusions

- we've focused on core techniques in this course

- hope is that you can now understand 90% of ACL papers published in recent years

- we've glossed over many details of particular NLP problems and linguistic theories
  - some of that was covered in TTIC 31190: NLP