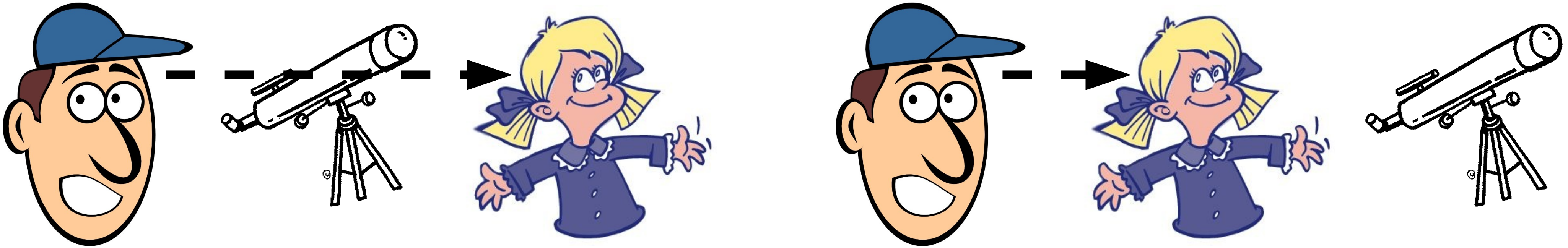
NLP Programming Tutorial 12 - Dependency Parsing

Graham Neubig
Nara Institute of Science and Technology (NAIST)

Interpreting Language is Hard!

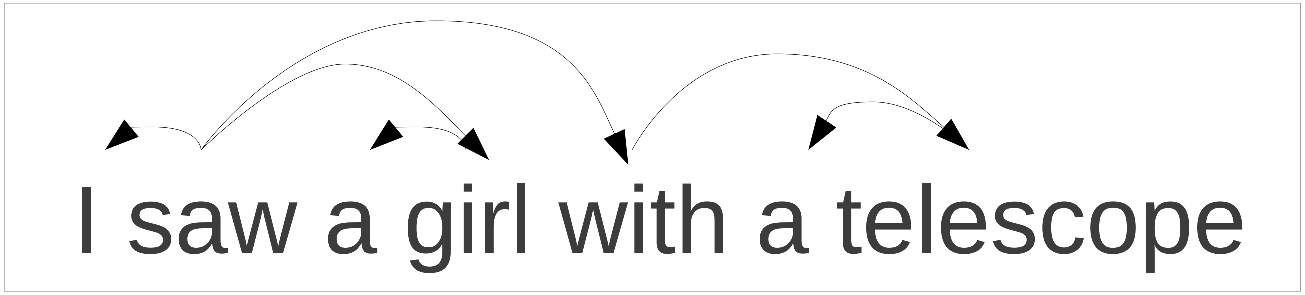
I saw a girl with a telescope



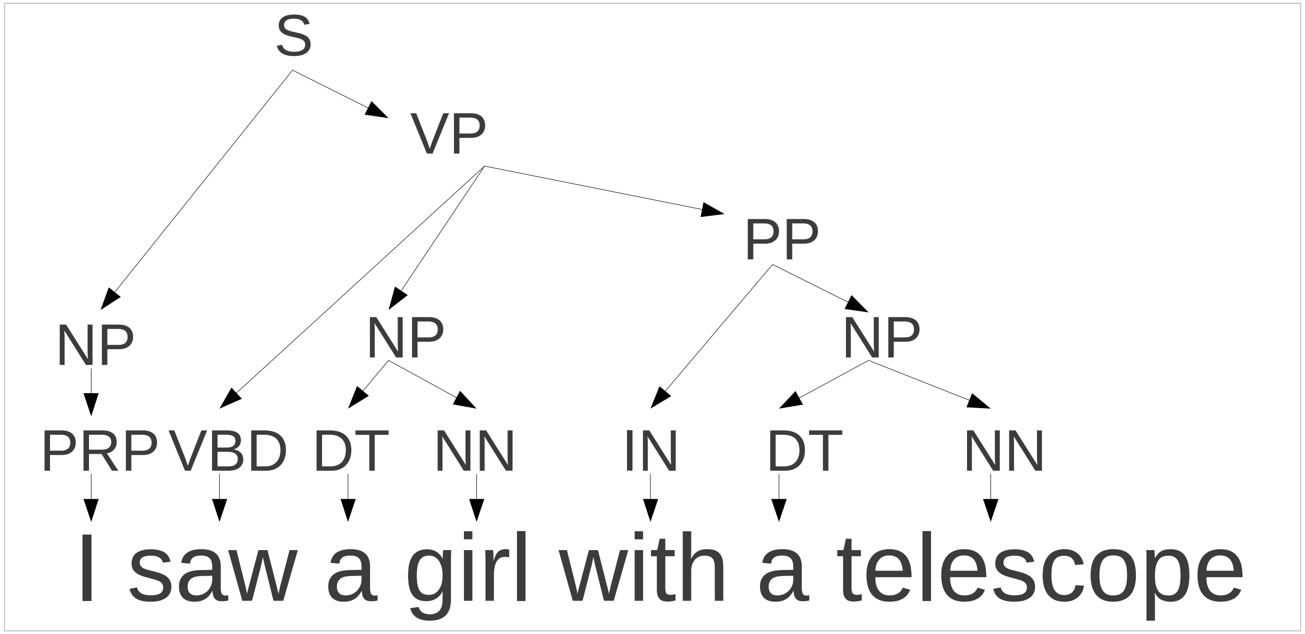
- "Parsing" resolves structural ambiguity in a formal way

Two Types of Parsing

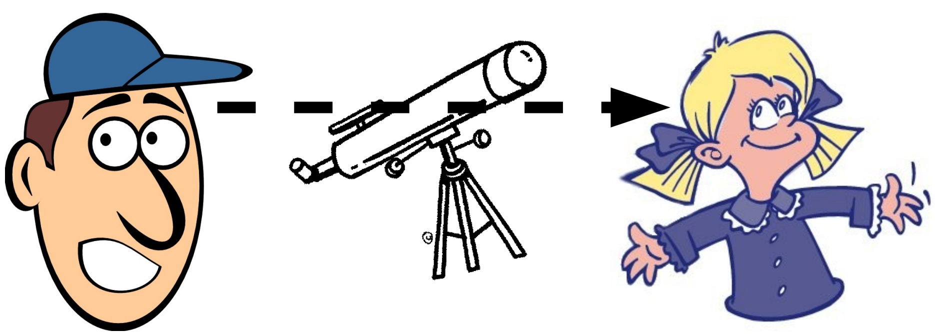
- Dependency: focuses on relations between words

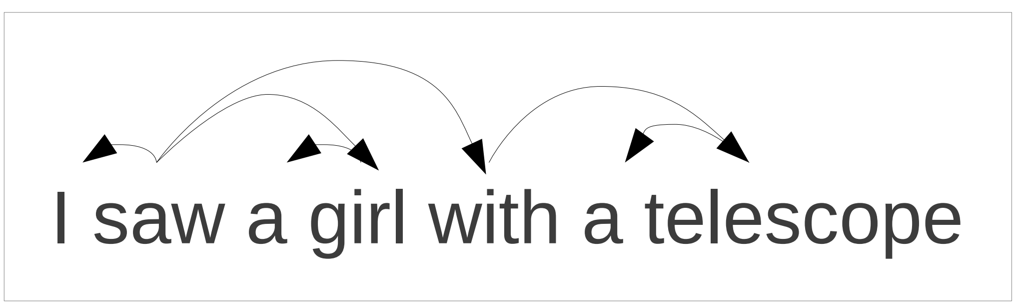


- Phrase structure: focuses on identifying phrases and their recursive structure



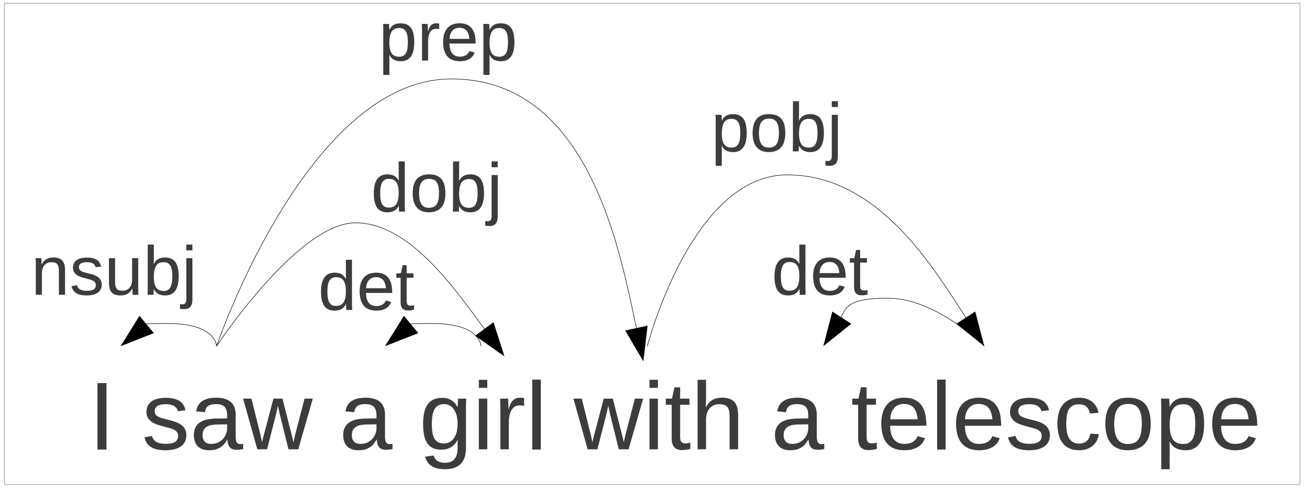
Dependencies Also Resolve Ambiguity



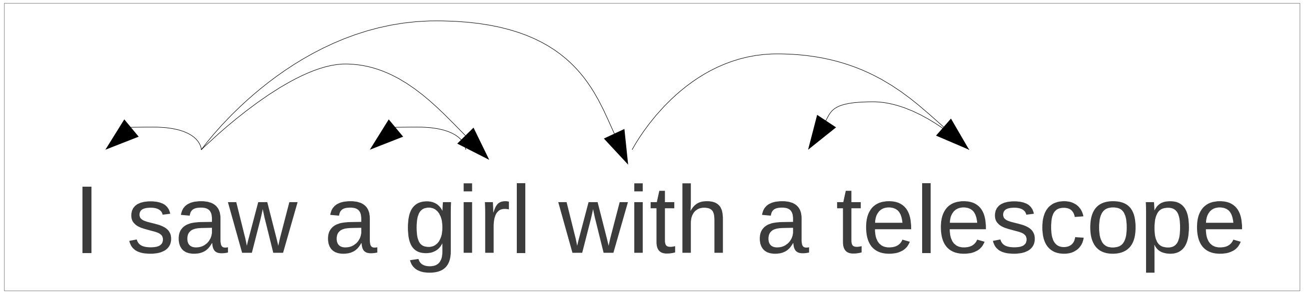


Dependencies

- **Typed:** Label indicating relationship between words



- **Untyped:** Only which words depend

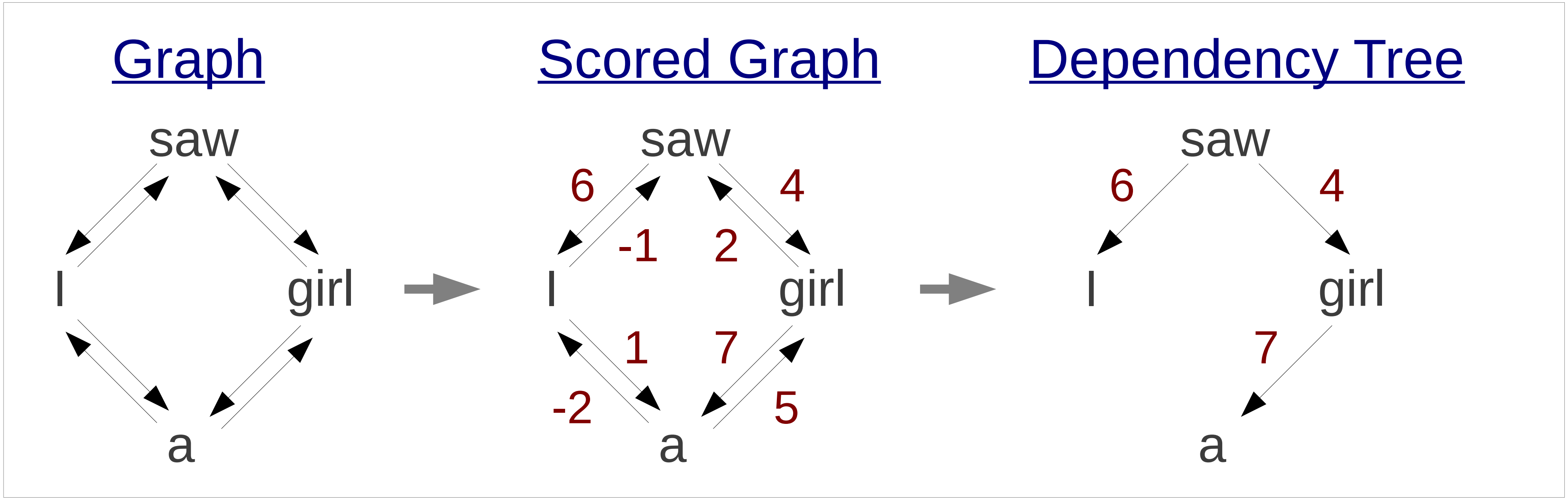


Dependency Parsing Methods

- Shift-reduce
  - Predict from left-to-right
  - Fast (linear), but slightly less accurate?
  - MaltParser
- Spanning tree
  - Calculate full tree at once
  - Slightly more accurate, slower
  - MSTParser, Eda (Japanese)
- Cascaded chunking
  - Chunk words into phrases, find heads, delete non-heads, repeat
  - CaboCha (Japanese)

Maximum Spanning Tree

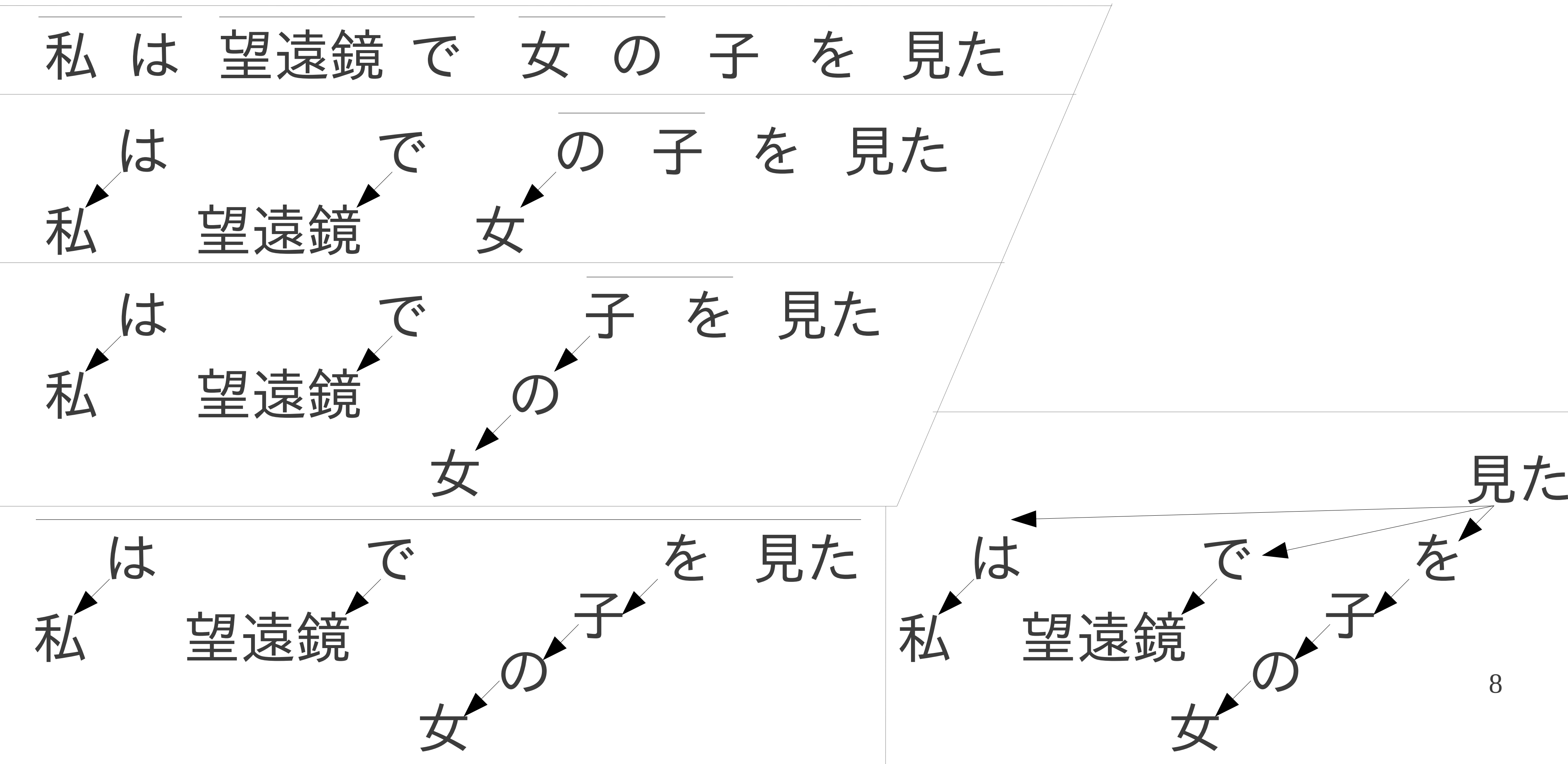
- Each dependency is an edge in a directed graph
- Assign each edge a score (with machine learning)
- Keep the tree with the highest score



(Chu-Liu-Edmonds Algorithm)

Cascaded Chunking

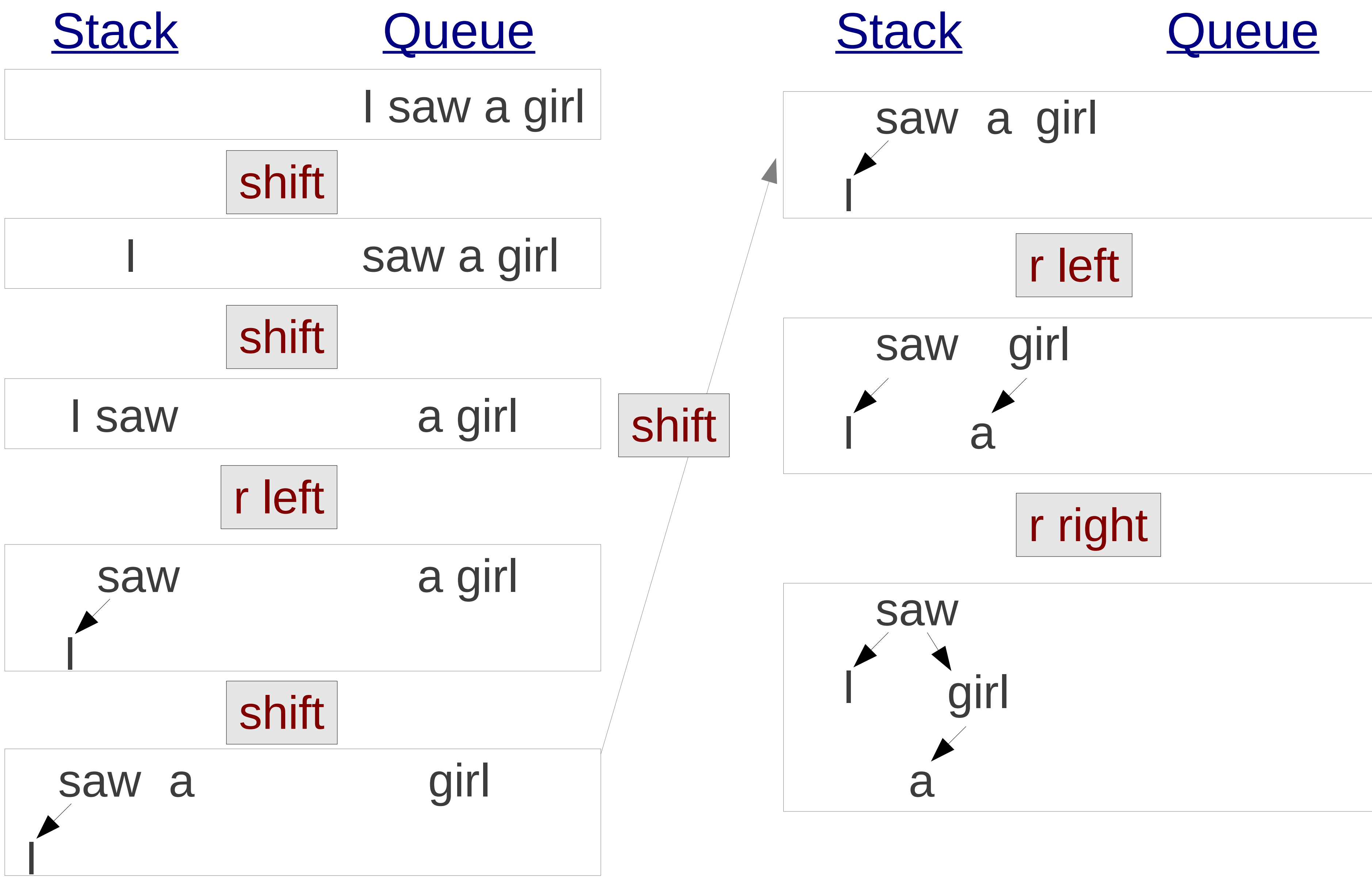
- Works for Japanese, which is strictly head-final
- Divide sentence into chunks, head is rightmost word



Shift-Reduce

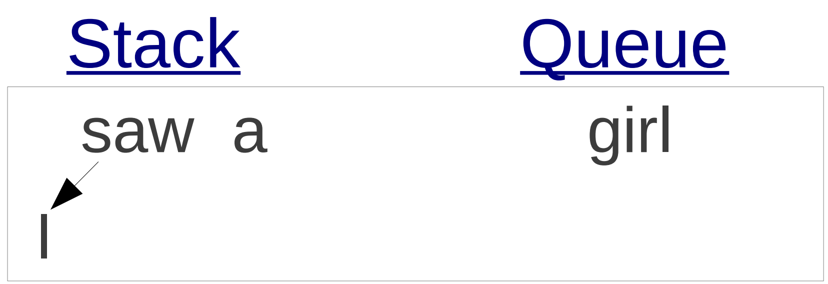
- Process words one-by-one left-to-right
- Two data structures
  - Queue: of unprocessed words
  - Stack: of partially processed words
- At each point choose
  - shift: move one word from queue to stack
  - reduce left: top word on stack is head of second word
  - reduce right: second word on stack is head of top word
- Learn how to choose each action with a classifier

Shift Reduce Example



Classification for Shift-Reduce

- Given a state:



- Which action do we choose?

shift ? r left ? r right ?

- Correct actions → correct tree

Classification for Shift-Reduce

- We have a weight vector for “shift” “reduce left” “reduce right”

$$w_s \quad w_l \quad w_r$$

- Calculate feature functions from the queue and stack

$$\varphi(queue, stack)$$

- Multiply the feature functions to get scores

$$s_s = w_s * \varphi(queue, stack)$$

- Take the highest score

$$s_s > s_l \ \&\& \ s_s > s_r \quad \rightarrow \quad \text{do shift}$$

Features for Shift Reduce

- Features should generally cover at least the last stack entries and first queue entry

	stack[-2]	*stack*[-1]	*queue*[0]
Word:	saw	a	girl
POS:	VBD	DET	NN

(-2 → second-to-last)
(-1 → last)
(0 → first)

$\varphi_{W\text{-}2saw,W\text{-}1a} = 1$ $\varphi_{W\text{-}1a,W0girl} = 1$

$\varphi_{W\text{-}2saw,P\text{-}1DET} = 1$ $\varphi_{W\text{-}1a,P0NN} = 1$

$\varphi_{P\text{-}2VBD,W\text{-}1a} = 1$ $\varphi_{P\text{-}1DET,W0girl} = 1$

$\varphi_{P\text{-}2VBD,P\text{-}1DET} = 1$ $\varphi_{P\text{-}1DET,P0NN} = 1$

Algorithm Definition

- The algorithm **SHIFTREDUCE** takes as input:
  - Weights w_s w_l w_r
  - A *queue*=[(1, $word_1$, POS_1), (2, $word_2$, POS_2), …]
- starts with a stack holding the special ROOT symbol:
  - *stack* = [(0, “ROOT”, “ROOT”)]
- processes and returns:
  - *heads* = [-1, $head_1$, $head_2$, …]

Shift Reduce Algorithm

```
SHIFTREDUCE(queue)
    make list heads
    stack = [ (0, “ROOT”, “ROOT”) ]
    while |queue| > 0 or |stack| > 1:
        feats = MAKEFEATS(stack, queue)
        s_s = w_s * feats                # Score for “shift”
        s_l = w_l * feats                # Score for “reduce left”
        s_r = w_r * feats                # Score for “reduce right”
        if s_s >= s_l and s_s >= s_r and |queue| > 0:
            stack.push( queue.popleft() )     # Do the shift
        elif s_l >= s_r:                      # Do the reduce left
            heads[ stack[-2].id ] = stack[-1].id
            stack.remove(-2)
        else:                                 # Do the reduce right
            heads[ stack[-1].id ] = stack[-2].id
            stack.remove(-1)
```

Training Shift-Reduce

- Can be trained using perceptron algorithm
- Do parsing, if correct answer *corr* different from classifier answer *ans*, update weights
- e.g. if *ans* = SHIFT and *corr* = LEFT

$$w_s \mathrel{-}= \varphi(queue, stack)$$

$$w_l \mathrel{+}= \varphi(queue, stack)$$

Keeping Track of the Correct Answer (Initial Attempt)

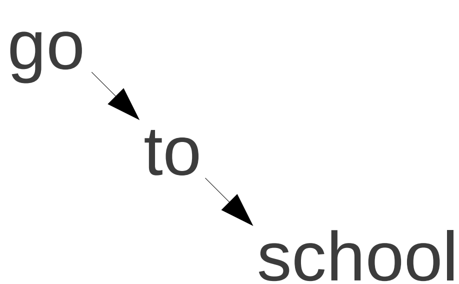
- Assume we know correct head of each stack entry:

stack[-1].*head* == *stack*[-2].*id* (left is head of right)
→ *corr* = RIGHT

stack[-2].*head* == *stack*[-1].*id* (right is head of left)
→ corr = LEFT

else
→ corr = SHIFT

- **Problem:** too greedy for right-branching dependencies



	stack[-2]	*stack*[-1]	*queue*[0]
	go	to	school
id:	1	2	3
head:	0	1	2

→ RIGHT

Keeping Track of the Correct Answer (Revised)

- Count the number of unprocessed children

- *stack*[-1].*head* == *stack*[-2].*id* (right is head of left)
  stack[-1].*unproc* == 0 (left no unprocessed children)
  → *corr* = RIGHT
- *stack*[-2].*head* == *stack*[-1].*id* (left is head of right)
  stack[-2].*unproc* == 0 (right no unprocessed children)
  → *corr* = LEFT
- else
  → *corr* = SHIFT

- Increase *unproc* when reading in the tree
  When we reduce a head, decrement *unproc*
  corr == RIGHT → *stack*[-1].*unproc* -= 1

Shift Reduce Training Algorithm

```
SHIFTREDUCETRAIN(queue)
    make list heads
    stack = [ (0, "ROOT", "ROOT") ]
    while |queue| > 0 or |stack| > 1:
        feats = MAKEFEATS(stack, queue)
        calculate ans          # Same as SHIFTREDUCE
        calculate corr         # Previous slides
        if ans != corr:
            w_ans -= feats
            w_corr += feats
        perform action according to corr
```

CoNLL File Format:

- Standard format for dependencies
- Tab-separated columns, sentences separated by space

ID	Word	Base	POS	POS2	?	Head	Type
1	ms.	ms.	NNP	NNP	_	2	DEP
2	haag	haag	NNP	NNP	_	3	NP-SBJ
3	plays	plays	VBZ	VBZ	_	0	ROOT
4	elianti	elianti	NNP	NNP	_	3	NP-OBJ
5	.	.	.	.	_	3	DEP

Exercise

Exercise

- Write train-sr.py test-sr.py
- Train the program
  - Input: `data/mstparser-en-train.dep`
- Run the program on actual data:
  - `data/mstparser-en-test.dep`
- Measure: accuracy with `script/grade-dep.py`
- Challenge:
  - think of better features to use
  - use a better classification algorithm than perceptron
  - analyze the common mistakes

Thank You!