Statistical Parsing

Dependency parsing

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

November 2016

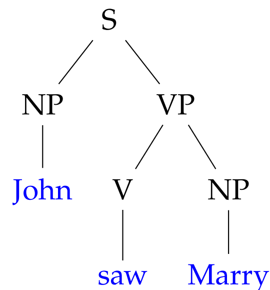
Ingredients of a parser

- A grammar - useful and easy to process representations
- A parsing algorithm - efficient enumeration of possible representations
- A disambiguation method - finding most likely analyses

Context-free parsing: grammars

A phrase structure grammar is a tuple (Σ, N, S, R)

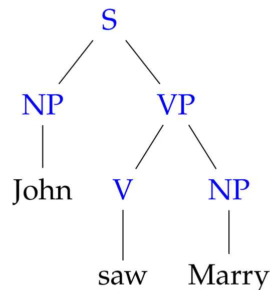
- Σ is a set of terminal symbols



Context-free parsing: grammars

A phrase structure grammar is a tuple (Σ, N, S, R)

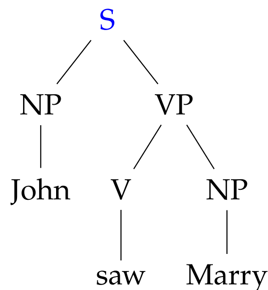
- Σ is a set of terminal symbols
- N is a set of non-terminal symbols



Context-free parsing: grammars

A phrase structure grammar is a tuple (Σ, N, S, R)

- Σ is a set of terminal symbols
- N is a set of non-terminal symbols
- S ∈ N is a distinguished *start* symbol

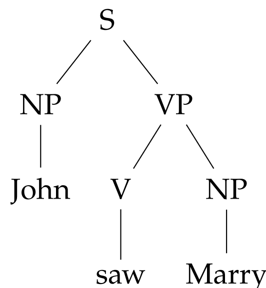


Context-free parsing: grammars

A phrase structure grammar is a tuple (Σ, N, S, R)

- Σ is a set of terminal symbols
- N is a set of non-terminal symbols
- S $\in$ N is a distinguished *start* symbol
- R is a set of rules of the form
  $A \rightarrow \alpha$ for $A \in N$ $\alpha \in \Sigma \cup N$

$$\begin{array}{llll} S & \rightarrow NP\ VP & VP & \rightarrow V\ NP \\ NP & \rightarrow John \mid Marry & V & \rightarrow saw \end{array}$$



Context-free parsing: parsing algorithms

- Top-down parsers start with S, and try to derive the input
- Bottom-up parsers start with the input, and try to reduce it to S
- Naive search (in both directions) has exponential time complexity in the length of the input
- Chart parsing methods (CKY, Earley) do recognition in polynomial time
- Chart parsers also represent ambiguity in a space efficient manner (but recovering all parses can require exponential time complexity)

Context-free parsing: disambiguation

- PCFGs provide a first approximation to finding most likely parse
- But their independence assumptions are too strong:
  - They cannot model structural or lexical preferences/constraints
  - It is also difficult to incorporate arbitrary/global features
- Lexicalized grammars (or parent annotation) may help with the independence assumption
- Discriminative (re-ranking) models can incorporate richer set of (global) features

Short divergence: deterministic parsing

- Unlike natural languages, programming languages are designed not to be ambiguous
- Every programming language sentence (program) has to have a single (semantic) interpretation
- Local ambiguity may happen, but deterministic (without backtracking) parsing is possible with a short lookahead

LR(k) grammars and shift-reduce parsing

- Shift-reduce parsers are bottom-up, table-based, deterministic parsers used in compilers
- For the classes of grammar LR(k) grammars can be parsed by such parsers
  - L means left-to-right
  - R means rightmost derivation
  - k is the number of lookahead symbols needed (typically 1)
- Constructing an LR(k) grammar tables by hand is difficult, often parser-generators (e.g., yacc) are used for converting appropriate CFG grammars written by hand

Shift-reduce parsing

- A shift-reduce parser does a single pass over the input string
- It makes use of a *stack*, the *lookahead* and a *buffer* of unseen tokens
- It deterministically applies two operations:

SHIFT the input symbol from the buffer to the stack

REDUCE if the symbols on top of the stack match the RHS of a rule, pop them and push the LHS

- Accepts the input, if the buffer is empty, and S is on top of the stack

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	**buffer**	**action**
[	2 * 3]	shift
[2	* 3]	reduce

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce
[term	* 3]	shift

Grammar:

exp → exp + term
exp → term
term → term * factor
term → factor
factor → (exp)
factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce
[term	* 3]	shift (?)

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce
[term	* 3]	shift (?)
[term *	3]	shift

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce
[term	* 3]	shift (?)
[term *	3]	shift
[term * 3	]	reduce

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce
[term	* 3]	shift (?)
[term *	3]	shift
[term * 3	]	reduce
[term * factor	]	reduce

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce
[term	* 3]	shift (?)
[term *	3]	shift
[term * 3	]	reduce
[term * factor	]	reduce
[term	]	reduce

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing example

Input: 2 * 3

stack	buffer	action
[	2 * 3]	shift
[2	* 3]	reduce
[factor	* 3]	reduce
[term	* 3]	shift (?)
[term *	3]	shift
[term * 3	]	reduce
[term * factor	]	reduce
[term	]	reduce
[exp	]	accept

Grammar:

exp → exp + term

exp → term

term → term * factor

term → factor

factor → (exp)

factor → [0-9]+

Shift-reduce parsing: summary

- Deterministic parsing is possible for programming languages
- The potential non-determinism (conflicts during shift-reduce parsing) can be avoided
  - by converting the hand-written grammars to LR(k) grammars
  - by heuristics strategies or disambiguation during post-processing

Shift-reduce parsing: summary

- Deterministic parsing is possible for programming languages
- The potential non-determinism (conflicts during shift-reduce parsing) can be avoided
  - by converting the hand-written grammars to LR(k) grammars
  - by heuristics strategies or disambiguation during post-processing

    A well-known ambiguity (just for fun):

```
int t, x;
t = 1;
if (t = 0) x = 0;
else if (t = 1) x = 1;
else x = 2;
```

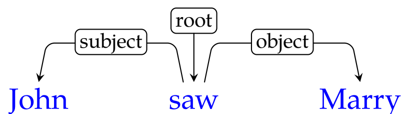
- What is the value of x?
- How to resolve the ambiguity?

Shift-reduce parsing and natural languages

…or why we did went through all these

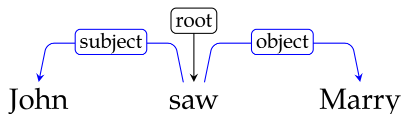
- Natural languages have global ambiguity, standard shift-reduce parsing will not work
- But there are some greedy parsers that follow the same principles (also think about the similarity with Earley parsing)
- Generalized LR (GLR) methods are also suggested for natural language parsing

Dependency grammars



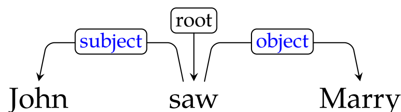
- No constituents, units of syntactic structure are words

Dependency grammars



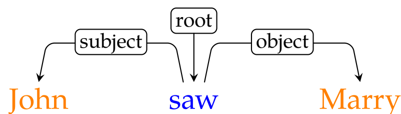
- No constituents, units of syntactic structure are words
- The structure of the sentence is represented by asymmetric binary relations between syntactic units

Dependency grammars



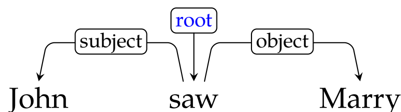
- No constituents, units of syntactic structure are words
- The structure of the sentence is represented by asymmetric binary relations between syntactic units
- The links (relations) have labels (dependency types)

Dependency grammars



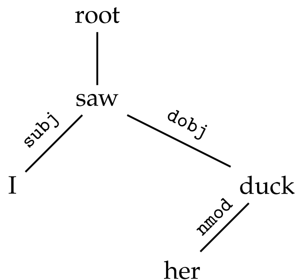
- No constituents, units of syntactic structure are words
- The structure of the sentence is represented by asymmetric binary relations between syntactic units
- The links (relations) have labels (dependency types)
- Each relation defines one of the words as the head and the other as dependent

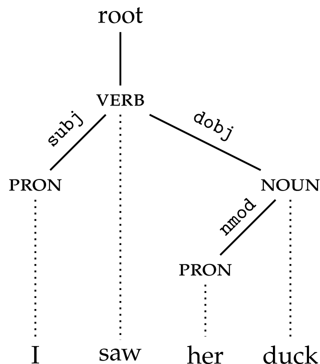
Dependency grammars



- No constituents, units of syntactic structure are words
- The structure of the sentence is represented by asymmetric binary relations between syntactic units
- The links (relations) have labels (dependency types)
- Each relation defines one of the words as the head and the other as dependent
- Often an artificial *root* node is used for computational convenience

Dependency grammars: notational variation





Dependency grammar: definition

A dependency grammar is a tuple (V, A)

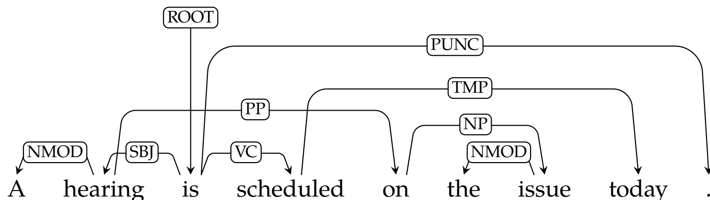
- V is a set of nodes corresponding to the (syntactic) words (we implicitly assume that words have indexes)
- A is a set of arcs of the form (w_i, r, w_j) where
  - $w_i \in V$ is the head
  - r is the type of the relation (arc label)
  - $w_j \in V$ is the dependent

This defines a directed graph.

Dependency grammars: common assumptions

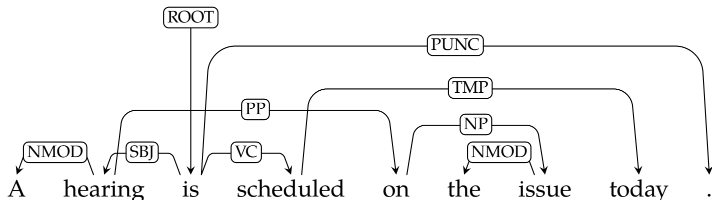
- Every word has a single head
- The dependency graphs are acyclic
- The graph is connected
- With these assumptions, the representation is a tree
- Note that these assumptions are not universal but common for dependency parsing

Dependency grammars: projectivity



- If a dependency graph has no crossing edges, it is said to be *projective*, otherwise *non-projective*
- Non-projectivity stem from long-distance dependencies and free word order
- Projective dependency trees can be represented with context-free grammars
- In general, projective dependencies are parsable more efficiently

Dependency grammars: projectivity



- If a dependency graph has no crossing edges, it is said to be *projective*, otherwise *non-projective*
- Non-projectivity stem from long-distance dependencies and free word order
- Projective dependency trees can be represented with context-free grammars
- In general, projective dependencies are parsable more efficiently

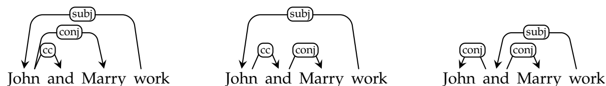
(tree reproduced from McDonald and Satta 2007)

Dependency grammars: some variation

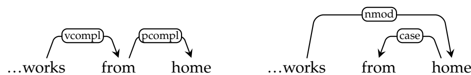
- Choice of dependency types (edge labels) may differ
  - Semantic roles
  - Grammatical/syntactic functions
- The assumption about syntactic units
- Formal properties of dependency structures
  - Projective or non-projective
  - Mono-stratal or multi-stratal

Some tricky constructions

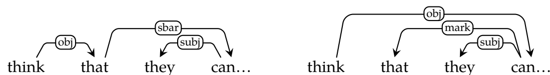
- Coordination



- Prepositional phrases



- Subordinate clauses

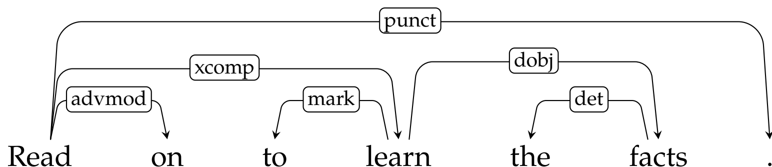


- Auxiliaries vs. main verbs

CONLL-X/U format for dependency annotation

Single-head assumption allows flat representation of dependency trees

```
1    Read   read   VERB  VB   Mood=Imp|VerbForm=Fin 0 root
2    on     on     ADV   RB   _                     1 advmod
3    to     to     PART  TO   _                     4 mark
4    learn  learn  VERB  VB   VerbForm=Inf          1 xcomp
5    the    the    DET   DT   Definite=Def          6 det
6    facts  fact   NOUN  NNS  Number=Plur           4 dobj
7    .      .      PUNCT .    _                     1 punct
```



example from English Universal Dependencies treebank

Dependency parsing

- Dependency parsing has many similarities with context-free parsing (e.g., trees)
- They also have some different properties (e.g., number of edges and depth of trees are limited)
- Dependency parsing can be
  - grammar-driven (hand drafted rules or constraints)
  - data-driven (rules/model is learned from a treebank)
- There are two main approaches:

Graph-based similar to context-free parsing, search for the best tree structure

Transition-based similar to shift-reduce parsing, greedily search for the best transition sequence

Grammar-driven dependency parsing

- Grammar-driven dependency parsers typically based on
  - lexicalized CF parsing
  - constraint satisfaction problem
    - start from fully connected graph, eliminate trees that do not satisfy the constraints
    - exact solution is intractable, often employ heuristics, approximate methods
    - sometime 'soft', or weighted, constraints are used
  - Practical implementations exist
- Our focus will be data-driven methods

Transition based parsing

- Inspired by shift-reduce parsing, single pass over the input
- Use a stack and a buffer of unprocessed words
- Parsing as predicting a sequence of transitions like
  - LEFT-ARC: similar to REDUCE, mark current word the head of the word on top of the stack
  - RIGHT-ARC: similar to REDUCE, mark current word a dependent of the word on top of the stack
  - SHIFT: push the current word to the stack
- Algorithm terminates when all words in the input are processed
- The transitions are not naturally deterministic, best transition is predicted using a machine learning method

(Yamada and Matsumoto 2003; Nivre, Hall, and Nilsson 2004)

A typical transition system

$$(\underbrace{\sigma \mid \overbrace{w_i}^{\text{stack top}}}_{\text{stack}},\ \underbrace{\overbrace{w_j}^{\text{next word}} \mid \beta}_{\text{buffer}},\ \underbrace{A}_{\text{arcs}})$$

LEFT-ARC$_r$: $(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_j|\beta, A \cup \{(w_j, r, w_i)\})$

- pop w_i,
- add arc (w_j, r, w_i) to A (keep w_j in the buffer)

RIGHT-ARC$_r$: $(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_i|\beta, A \cup \{(w_i, r, w_j)\})$

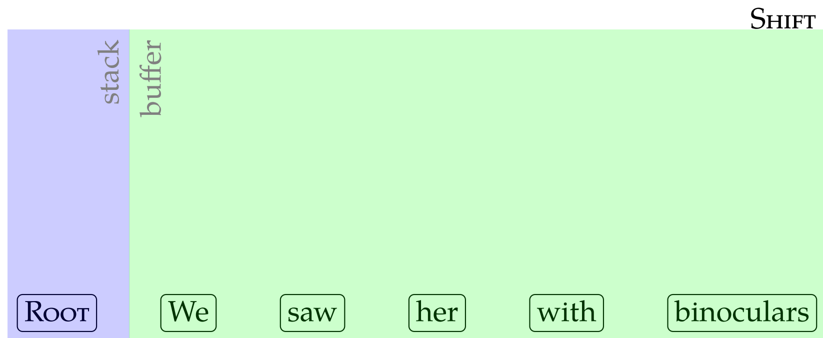
- pop w_i,
- add arc (w_i, r, w_j) to A,
- move w_i to the buffer

SHIFT: $(\sigma, w_j|\beta, A) \Rightarrow (\sigma|w_j, \beta, A)$

- push w_j to the stack
- remove it from the buffer

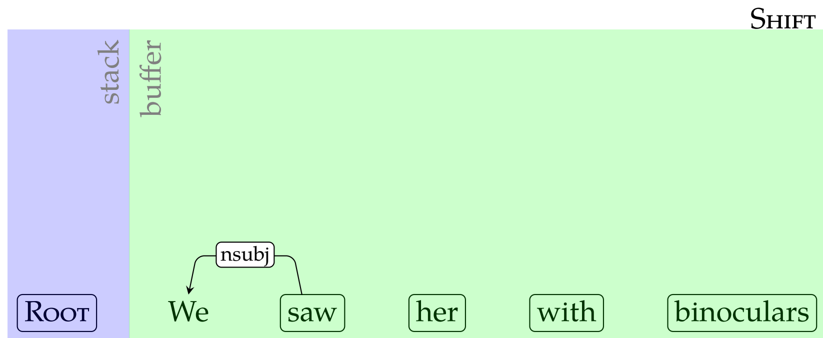
(Kübler, McDonald, and Nivre 2009, p.23)

Transition based parsing: example

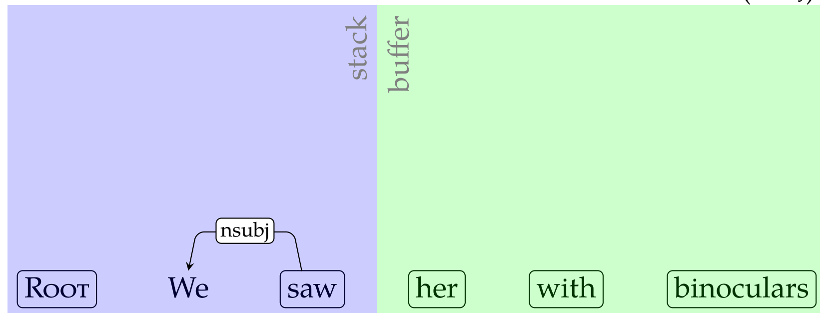


Transition based parsing: example

Transition based parsing: example

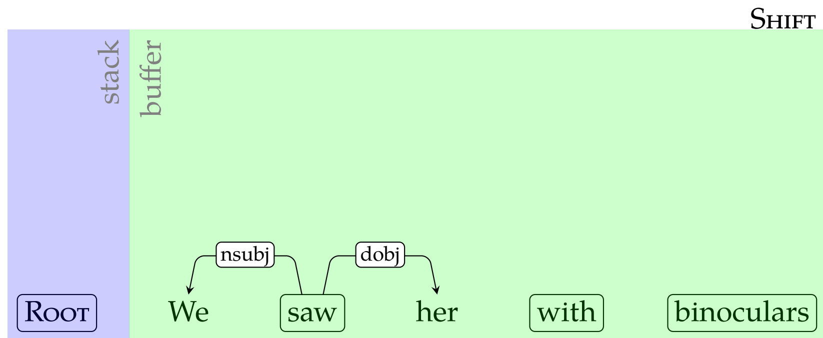


Transition based parsing: example

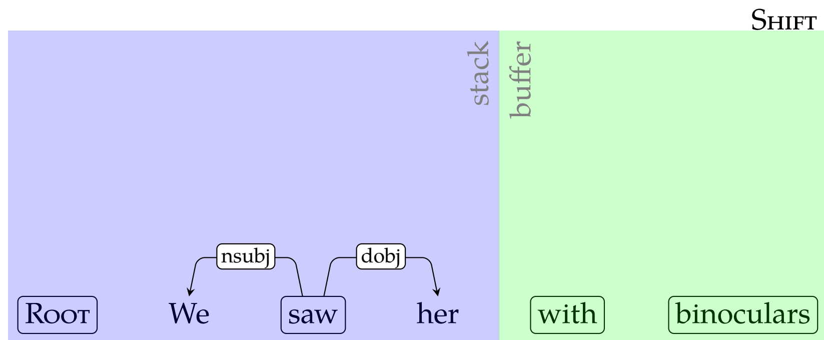


Note: We need SHIFT for NP attachment.

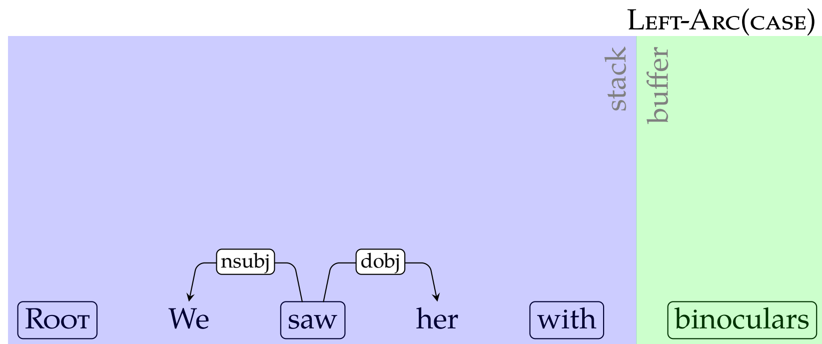
Transition based parsing: example



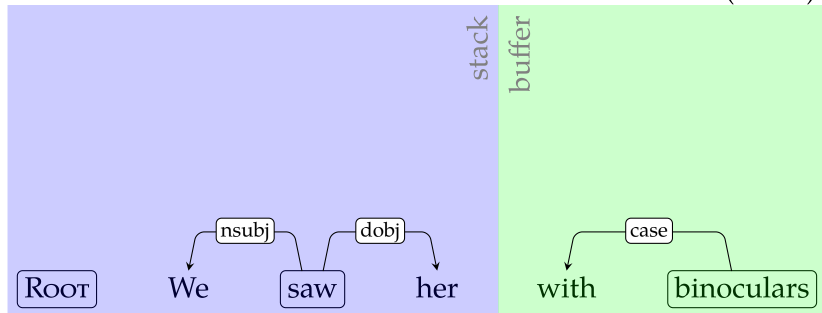
Transition based parsing: example



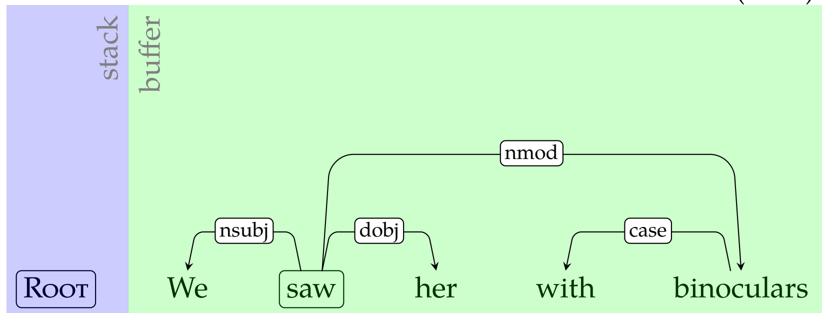
Transition based parsing: example



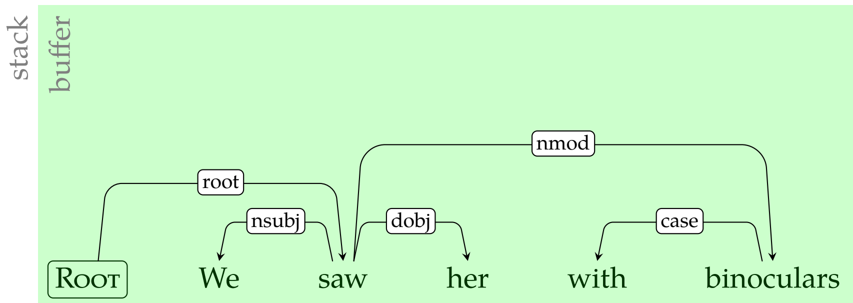
Transition based parsing: example



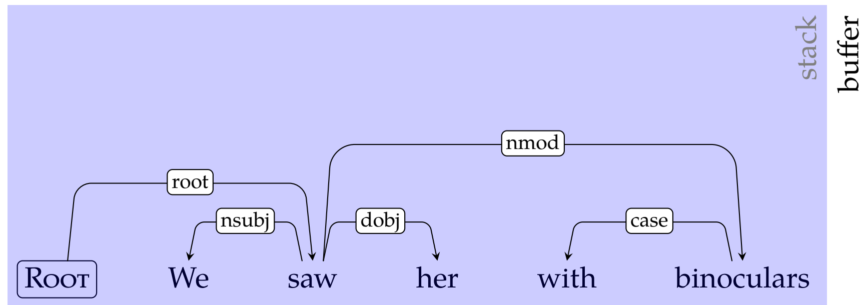
Transition based parsing: example



Transition based parsing: example



Transition based parsing: example



Making transition decisions

- In classical shift-reduce parsing the actions are deterministic
- In transition-based dependency parsing we need to choose among all possible transitions
- The typical method is to train a (discriminative) classifier trained on features extracted from gold-standard *transition sequences*
- Almost any machine learning method method is applicable. Common choices include
  - Memory-based learning
  - Support vector machines
  - (Deep) neural networks

Features for transition-based parsing

- The features come from the parser configuration, for example
  - The word at the top of the stack, (peeking towards the bottom of the stack is also fine)
  - The first/second word on the buffer
  - Right/left dependents of the word on top of the stack/buffer
- For each possible ‘address’, we can make use of features like
  - Word form, lemma, POS tag, morphological features, word embedding
  - Dependency relations – (w_i, r, w_j) triples
- Note that for some ‘address’–‘feature’ combinations and in some configurations the values may be missing

The training data

- The features for transition-based parsing have to be extracted from *parser configurations*
- The data (treebanks) need to be preprocessed for obtaining the training data
- Construct a transition sequence by parsing the sentences, and using treebank annotations (the set A) as an 'oracle'
- Decide for

$$
\begin{array}{rl}
\textsc{Left-Arc}_r & \text{if } (\beta[0], r, \sigma[0]) \in A \\
\textsc{Right-Arc}_r & \text{if } (\sigma[0], r, \beta[0]) \in A \\
 & \text{and all dependents of } \beta[0] \text{ are attached} \\
\textsc{Right-Arc}_r & \text{otherwise}
\end{array}
$$

- There may be multiple sequences that yield to the same dependency tree, the above defines a 'canonical' transition sequence

Alternative transition systems

- A common alternative to the transition system we defined (known as *arc-standard*) is the *arc-eager* transitions system

$$\textsc{Left-Arc}_r\text{:}\ (\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_j|\beta, A \cup \{(w_j, r, w_i)\}) \quad \text{if } (w_k, r', w_i) \notin A$$

$$\textsc{Right-Arc}_r\text{:}\ (\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma|w_i|w_j, \beta, A \cup \{(w_i, r, w_j)\})$$

$$\textsc{Reduce}\text{:}\ (\sigma|w_i, \beta, A) \Rightarrow (\sigma, \beta, A) \quad \text{if } (w_k, r', w_i) \notin A$$

$$\textsc{Shift}\text{:}\ (\sigma, w_j|\beta, A) \Rightarrow (\sigma|w_j, \beta, A)$$

- This system does not have to wait until all dependents of $\beta[0]$ to be attached before a Right-Arc

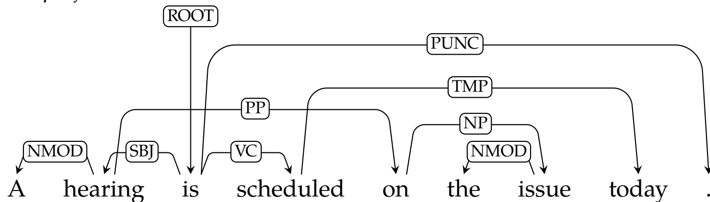
(Kübler, McDonald, and Nivre 2009, p.34)

Non-projective parsing

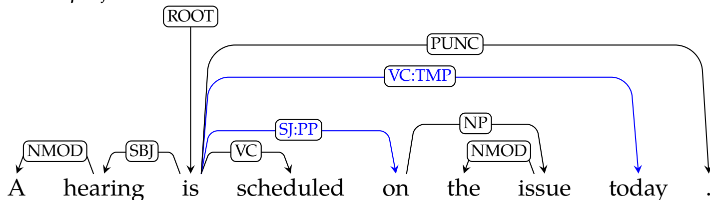
- The transition-based parsing we defined so far works only for projective dependencies
- One way to achieve (limited) non-projective parsing is to add special LEFT-ARC and RIGHT-ARC transitions to/from non-top words from the stack
- Another method is pseudo-projective parsing:
  - preprocessing to 'projectivize' the trees before training
    - The idea is to attach the dependents to a higher level head that preserves projectivity, while marking it on the change on the new dependency
  - postprocessing for restoring the projectivity after parsing
    - Re-introduce projectivity for the marked dependencies

Pseudo-projective parsing

Non-projective tree:



Pseudo-projective tree:



Transition based parsing: summary/notes

- Linear time, greedy parsing
- Can be extended to non-projective dependencies
- One can use arbitrary features,
- We need some extra work for generating gold-standard transition sequences from treebanks
- Early errors propagate, transition-based parsers make more mistakes on long-distance dependencies
- The greedy algorithm can be extended to beam search for better accuracy (still linear time complexity)

Graph-based parsing: preliminaries

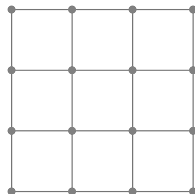
- Enumerate all possible dependency trees
- Pick the best scoring tree
- Features are based on limited parse history (like CFG parsing)
- Two well-known flavors:
  - Maximum (weight) spanning tree (MST)
  - Chart-parsing based methods

J. M. Eisner 1996; McDonald et al. 2005

MST parsing: preliminaries

Spanning tree of a graph

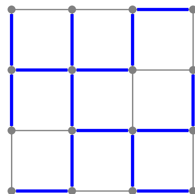
- Spanning tree of a connected graph is a sub-graph which is a tree and traverses all the nodes



MST parsing: preliminaries

Spanning tree of a graph

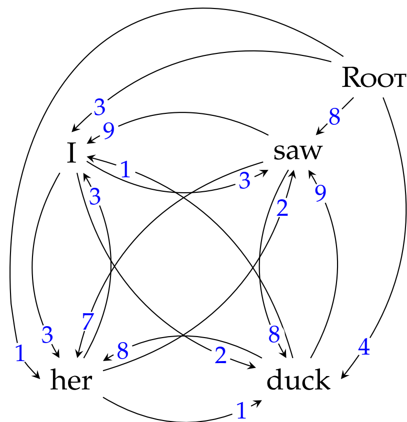
- Spanning tree of a connected graph is a sub-graph which is a tree and traverses all the nodes
- For fully-connected graphs, the number of spanning trees are exponential in the size of the graph
- The problem is well studied
- There are efficient algorithms for enumerating, and finding the optimum spanning tree on weighted graphs



MST algorithm for dependency parsing

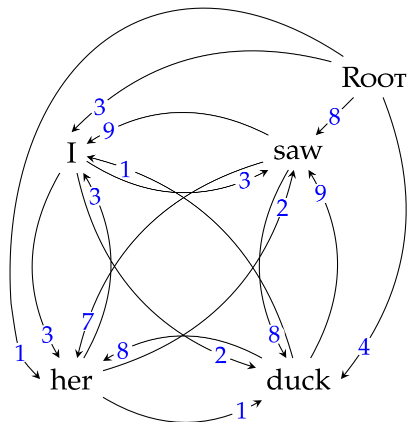
- For directed graphs, there is a polynomial time algorithm that finds the minimum/maximum spanning tree (MST) of a fully connected graph (Chu-Liu-Edmonds algorithm)
- The algorithm starts with a dense/fully connected graph
- Removes edges until the resulting graph is a tree

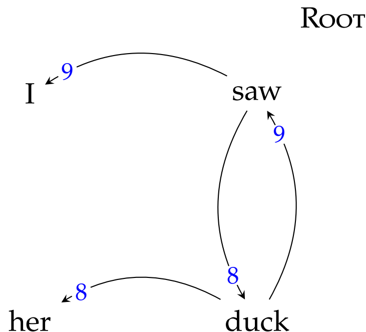
MST example



For each node select the incoming arc with highest weight

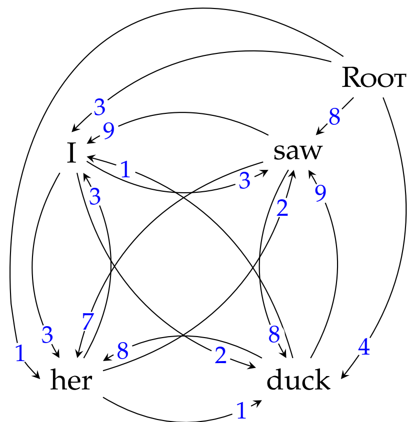
MST example

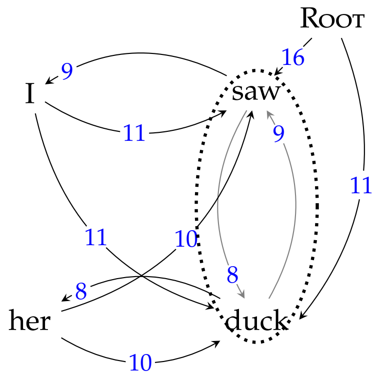




Detect the cycles, contract them to a 'single node'

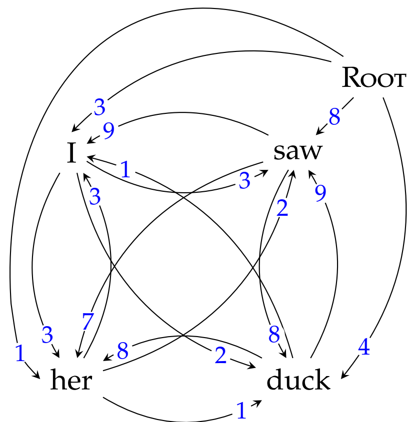
MST example

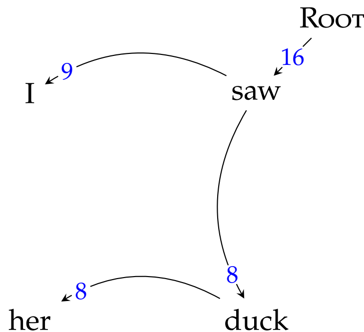




Pick the best arc into the combined node, break the cycle

MST example





Once all cycles are eliminated, the result is the MST

Properties of the MST parser

- The MST parser is non-projective
- There is an alrgorithm with $O(n^2)$ time complexity (Tarjan 1977)
- The time complexity increases with typed dependencies (but still close to quadratic)
- The weights/parameters are associated with edges (often called 'arc-factored')
- We can learn the arc weights directly from a treebank
- However, it is difficult to incorporate non-local features

CKY reminder

```
function CKY(words, grammar)
    for j ← 1 to LENGTH(words) do
        table[j − 1, j] ← {A|A → words[j] ∈ grammar}
        for i ← j − 1 downto 0 do
            for k ← i + 1 to j − 1 do
                table[i, j] ←  table[i, j] ∪
                              {A|A → BC ∈ grammar and
                                 B ∈ table[i, k] and
                                 C ∈ table[k, j]}
    return table
```

CKY for dependency parsing

- The CKY algorithm can be adopted to projective dependency parsing
- For a naive implementation the complexity increases drastically $O(n^6)$
  - Any of the words within the span can be the head
  - Inner loop has to consider all possible splits
- For projective parsing, the observation that the left and right dependents of a head are independently generated reduces the comlexity to $O(n^3)$

(J. Eisner 1997)

Non-local features

- The graph-based dependency parsers use edge-based features
- This limits the use of more global features
- Some extensions for using 'more' global features are possible
- This often leads non-projective parsing to become intractable

External features

- For both type of parsers, one can obtain features that are based on unsupervised methods such as
  - clustering
  - dense vector representations
  - alignment/transfer from bilingual corpora/treebanks

(Koo, Carreras, and Collins 2008)

Errors from different parsers

- Different parsers make different errors
  - Transition based parser do well on local arcs, worse on long-distance arcs
  - Graph based parser tend to do better on long-disntance dependencies
- Parser combination is a good way to comibine the powers of different models. Two common methods
  - Mojority voting: train parsers separately, use the weighted combination of their results
  - Stacking: use the output of a parser as features for another

(McDonald and Satta 2007; Sagae and Lavie 2006; Nivre and McDonald 2008)

Dependency parsing: summary

- Two general methods:

  transition based: greedy search, non-local features, fast, less accurate

  graph based: exact search, local features, slower, accurate (within model limitations)
- Combination of different methods often result in better performance
- Non-projective parsing is more difficult
- Most of the recent parsing research has focused on better machine learning methods (mainly using neural networks)

Evaluation metrics for dependency parsers

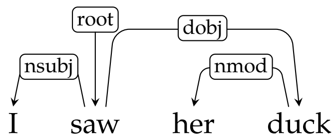
- Like CF parsing, exact match is often too strict
- *Attachment score* is the ratio of words whose heads are identified correctly.
  - *Labeled attachment score* (LAS) requires the dependency type to match
  - *Unlabeled attachment score* (UAS) disregards the dependency type
- *Precision/recall/F-measure* often used for quantifying success on identifying a particular dependency type

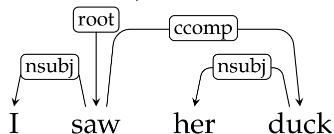
precision
: is the ratio of correctly identified dependencies (of a certain type)

recall
: is the ratio of dependencies in the gold standard that parser predicted correctly

f-measure
: is the harmonic mean of precision and recall $\left(\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}\right)$

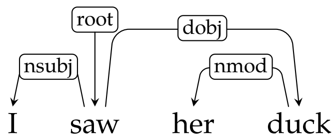
Evaluation example

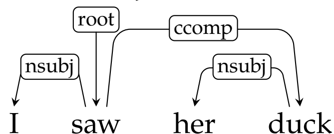




UAS
LAS
Precision_{nsubj}
Recall_{nsubj}
Precision_{dobj}
Recall_{dobj}

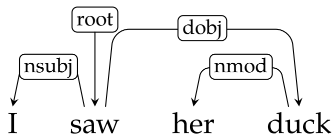
Evaluation example

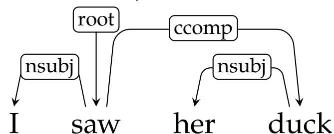




UAS	100%
LAS	
$\text{Precision}_{\text{nsubj}}$	
$\text{Recall}_{\text{nsubj}}$	
$\text{Precision}_{\text{dobj}}$	
$\text{Recall}_{\text{dobj}}$	

Evaluation example

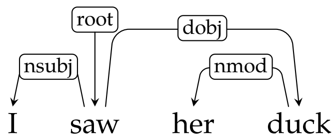




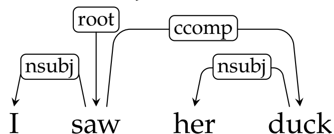
UAS	100%
LAS	50%
Precision_{nsubj}	
Recall_{nsubj}	
Precision_{dobj}	
Recall_{dobj}	

Evaluation example

Gold standard

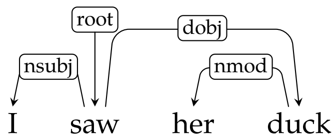


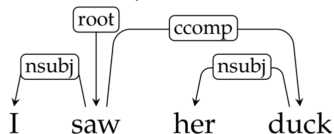
Parser output



UAS	100%
LAS	50%
$\text{Precision}_{\text{nsubj}}$	50%
$\text{Recall}_{\text{nsubj}}$	
$\text{Precision}_{\text{dobj}}$	
$\text{Recall}_{\text{dobj}}$	

Evaluation example

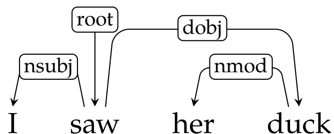




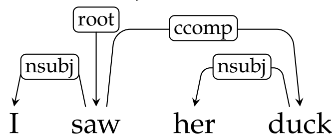
UAS	100%
LAS	50%
$\text{Precision}_{\text{nsubj}}$	50%
$\text{Recall}_{\text{nsubj}}$	100%
$\text{Precision}_{\text{dobj}}$	
$\text{Recall}_{\text{dobj}}$	

Evaluation example

Gold standard



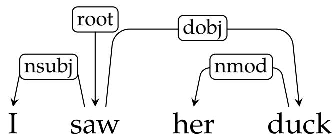
Parser output



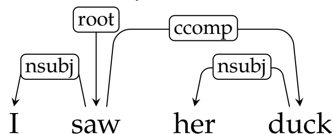
UAS	100%	
LAS	50%	
$\text{Precision}_{\text{nsubj}}$	50%	
$\text{Recall}_{\text{nsubj}}$	100%	
$\text{Precision}_{\text{dobj}}$	0%	(assumed)
$\text{Recall}_{\text{dobj}}$		

Evaluation example

Gold standard



Parser output



UAS	100%	
LAS	50%	
$\text{Precision}_{\text{nsubj}}$	50%	
$\text{Recall}_{\text{nsubj}}$	100%	
$\text{Precision}_{\text{dobj}}$	0%	(assumed)
$\text{Recall}_{\text{dobj}}$	0%	

Averaging evaluation scores

- As in context-free parsing, average scores can be
  macro-average or sentence-based
  micro-average or word-based
- Consider a two-sentence test set with

	words	correct
sentence 1	30	10
sentence 2	10	10

  - word-based average attachment score:
  - sentence-based average attachment score:

Averaging evaluation scores

- As in context-free parsing, average scores can be
  macro-average or sentence-based
  micro-average or word-based
- Consider a two-sentence test set with

	words	correct
sentence 1	30	10
sentence 2	10	10

  - word-based average attachment score: 50% (20/40)
  - sentence-based average attachment score: 66% ((1 + 1/3)/2)

Summary

- Dependency relations are often semantically easier to interpret
- It is also claimed that dependency parsers are more suitable for parsing free-word-order langauges
- Dependency relations are between words, no phrases or other abstract nodes are postulated
- This often leads to more efficient parsing
- We reviewed two major classes of parsers:
  - Transition based
  - Graph based

Summary

- Dependency relations are often semantically easier to interpret
- It is also claimed that dependency parsers are more suitable for parsing free-word-order langauges
- Dependency relations are between words, no phrases or other abstract nodes are postulated
- This often leads to more efficient parsing
- We reviewed two major classes of parsers:
  - Transition based
  - Graph based

Next:

Thursday More work practical work on of-the-shelf dependency parsers

Next Tue Michael Collins (2003). "Head-driven statistical models for natural language parsing". In: *Computational linguistics* 29.4, pp. 589–637. DOI: 10.1162/089120103322753356

Bibliography

Collins, Michael (2003). "Head-driven statistical models for natural language parsing". In: *Computational linguistics* 29.4, pp. 589–637. DOI: 10.1162/089120103322753356.

Eisner, Jason (1997). "Bilexical grammars and a cubic-time probabilistic parser". In: *Proceedings of the Fifth International Conference on Parsing Technologies (IWPT)*.

Eisner, Jason M. (1996). "Three New Probabilistic Models for Dependency Parsing: An Exploration". In: *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*. COLING '96. Copenhagen, Denmark: Association for Computational Linguistics, pp. 340–345. DOI: 10.3115/992628.992688. URL: http://dx.doi.org/10.3115/992628.992688.

Koo, Terry, Xavier Carreras, and Michael Collins (2008). "Simple Semi-supervised Dependency Parsing". In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 595–603. URL: http://www.aclweb.org/anthology/P/P08/P08-1068.

Kübler, Sandra, Ryan McDonald, and Joakim Nivre (2009). *Dependency Parsing*. Synthesis lectures on human language technologies. Morgan & Claypool. ISBN: 9781598295962.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič (2005). "Non-projective Dependency Parsing Using Spanning Tree Algorithms". In: *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. HLT '05. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 523–530. DOI: 10.3115/1220575.1220641. URL: http://dx.doi.org/10.3115/1220575.1220641.

McDonald, Ryan and Giorgio Satta (2007). "On the complexity of non-projective data-driven dependency parsing". In: *Proceedings of the 10th International Conference on Parsing Technologies*. Association for Computational Linguistics, pp. 121–132.

Bibliography (cont.)

Nivre, Joakim, Johan Hall, and Jens Nilsson (2004). "Memory-based dependency parsing". In: *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*. Ed. by Hwee Tou Ng and Ellen Riloff, pp. 49–56.

Nivre, Joakim and Ryan McDonald (2008). "Integrating Graph-Based and Transition-Based Dependency Parsers". In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 950–958. URL: `http://www.aclweb.org/anthology/P/P08/P08-1108`.

Sagae, Kenji and Alon Lavie (2006). "Parser Combination by Reparsing". In: *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*. New York City, USA: Association for Computational Linguistics, pp. 129–132. URL: `http://www.aclweb.org/anthology/N/N06/N06-2033`.

Tarjan, R. E. (1977). "Finding optimum branchings". In: *Networks* 7.1, pp. 25–35. ISSN: 1097-0037. DOI: `10.1002/net.3230070103`.

Yamada, Hiroyasu and Yuji Matsumoto (2003). "Statistical dependency analysis with support vector machines". In: *Proceedings of 8th international workshop on parsing technologies (IWPT)*. Ed. by Gertjan Van Noord, pp. 195–206.

A small assignment

Find the ratio of the non-projective trees and dependencies in all Universal Dependencies treebanks (version 1.4).

- Information about the treebanks:
  `http://universaldependencies.org/`
- Can be downloaded from:
  `http://hdl.handle.net/11234/1-1827`

Please send your results via email before next Thursday (December 1st).