Dependenzgrammatik-Parsing

LMT-/Watson-Parser, MaltParser, Stanford Parser

Kurt Eberle

k.eberle@lingenio.de

28. Februar 2020

Outline

Outline

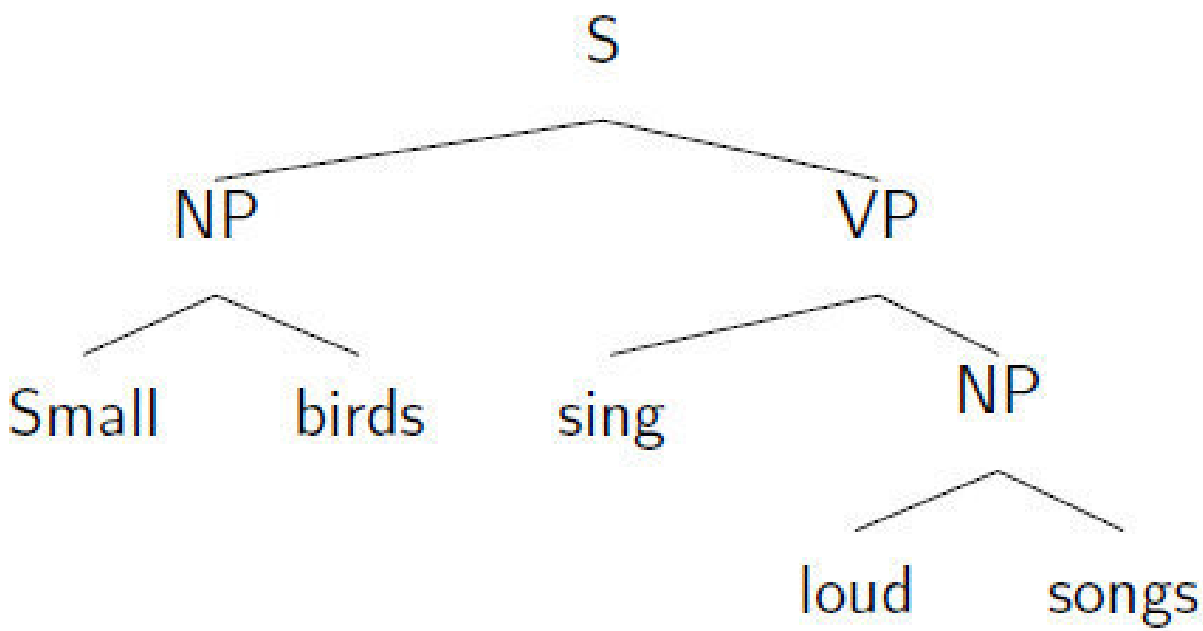
Dependency Grammar

- Dependency Grammar vs Phrase-Structure Grammar
- Features of Dependency Grammar
- Dependency Parsers: Rule-based and statistical
- (IBM's) LMT (WebSphere Translation Server), Watson
- Malt and Stanford

DG and PSG

- *Small birds sing loud songs*
- Constituent structure . . .

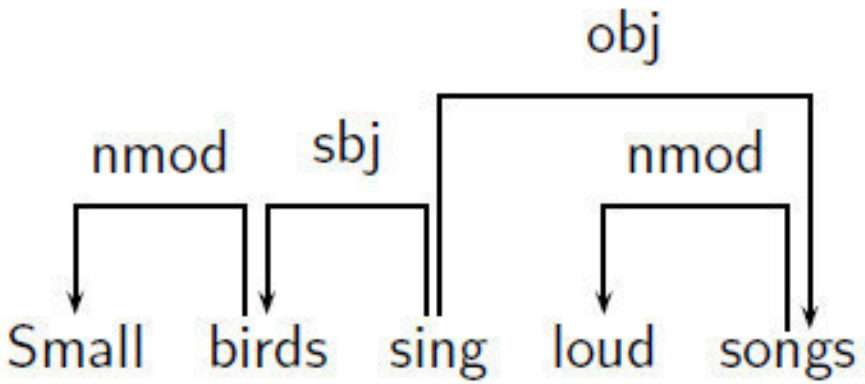
What you might be more used to seeing:

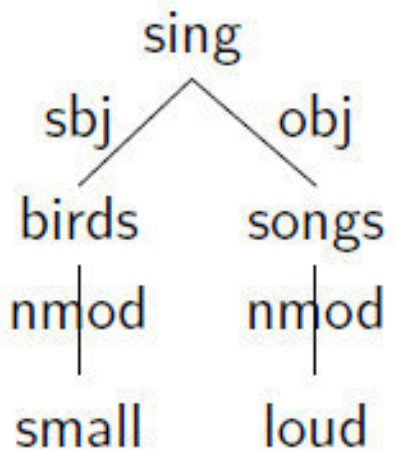


DG and PSG

- *Small birds sing loud songs*
- Dependency Structure . . .

The corresponding dependency tree representations [Hudson(2000)]:





Dependency Syntax

- The basic idea:
  - Syntactic structure consists of lexical items, linked by binary asymmetric relations called dependencies.
- In the (translated) words of Lucien Tesnière [Tesnière(1959)]:
  - The sentence is an *organized whole*, the constituent elements of which are *words*. [1.2] Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives *connections*, the totality of which forms the structure of the sentence. [1.3] The structural connections establish *dependency* relations between the words. Each connection in principle unites a *superior* term and an *inferior* term. [2.1] The superior term receives the name *governor*. The inferior term receives the name *subordinate*. Thus, in the sentence *Alfred parle* [. . .], *parle* is the governor and *Alfred* the subordinate. [2.2]

Constituency vs. Relations

- DG is based on relationships between words, i.e., **dependency relations**
  - A $\rightarrow$ B means *A governs B* or *B depends on A* ...
  - Dependency relations can refer to syntactic properties, semantic properties, or a combination of the two
    - $\rightarrow$ Some variants of DG separate syntactic and semantic relations by representing different layers of dependency structures
  - These relations are generally things like subject, object/complement, (pre-/post-)adjunct, etc.
    - Subject/Agent: *John* fished.
    - Object/Patient: Mary hit *John*.
- PSG is based on groupings (called *phrases* or *constituents*)
  - Grammatical relations are not usually seen as primitives, but as being derived from structure

Simple relation example

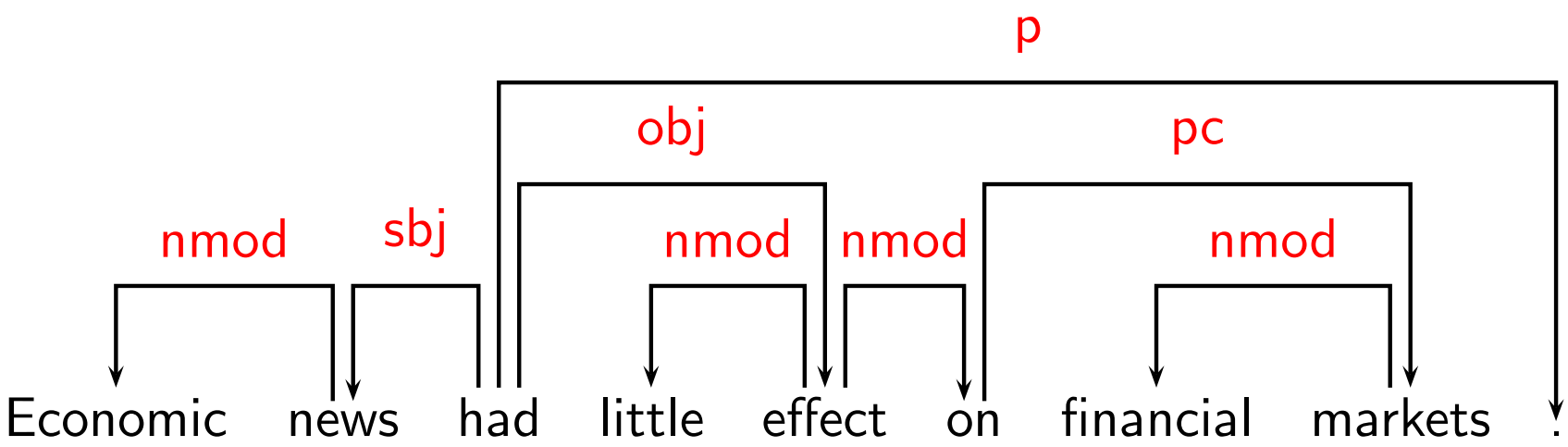
For the sentence *John loves Mary*, we have the relations:

- loves $\rightarrow_{\text{subj}}$ John
- loves $\rightarrow_{\text{obj}}$ Mary

Both *John* and *Mary* depend on *loves*, which makes *loves* the head, or **root**, of the sentence (i.e., there is no word that governs *loves*)

- The structure of a sentence, then, consists of the set of pairwise relations among words.

Dependency Structure

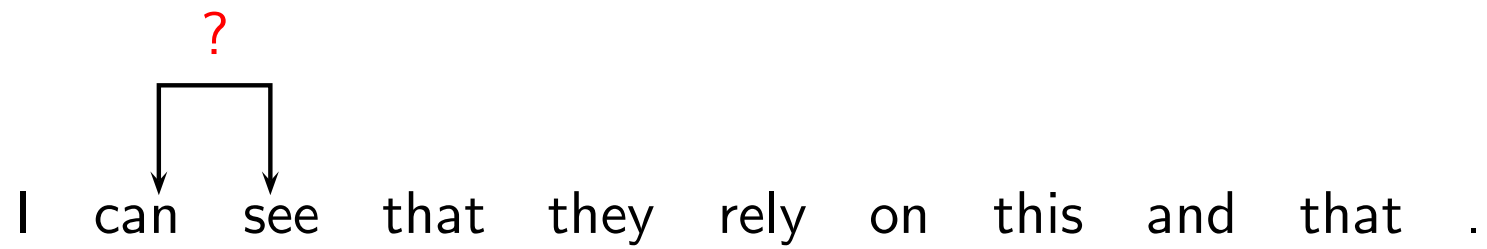


Terminology

Superior	**Inferior**
Head	Dependent
Governor	Modifier
Regent	Subordinate
⋮	⋮

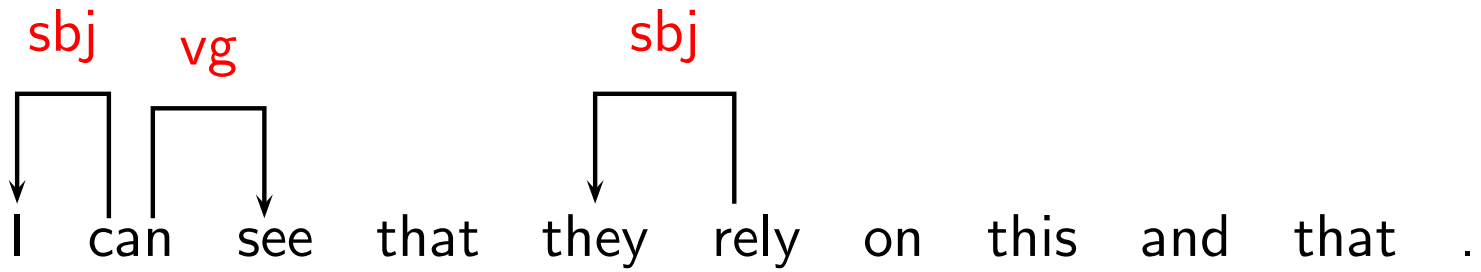
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



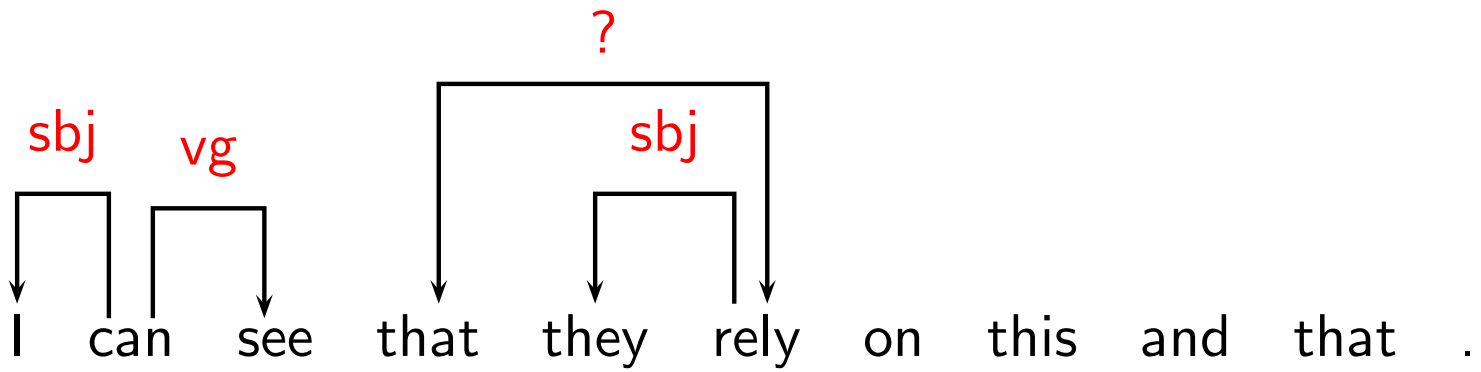
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



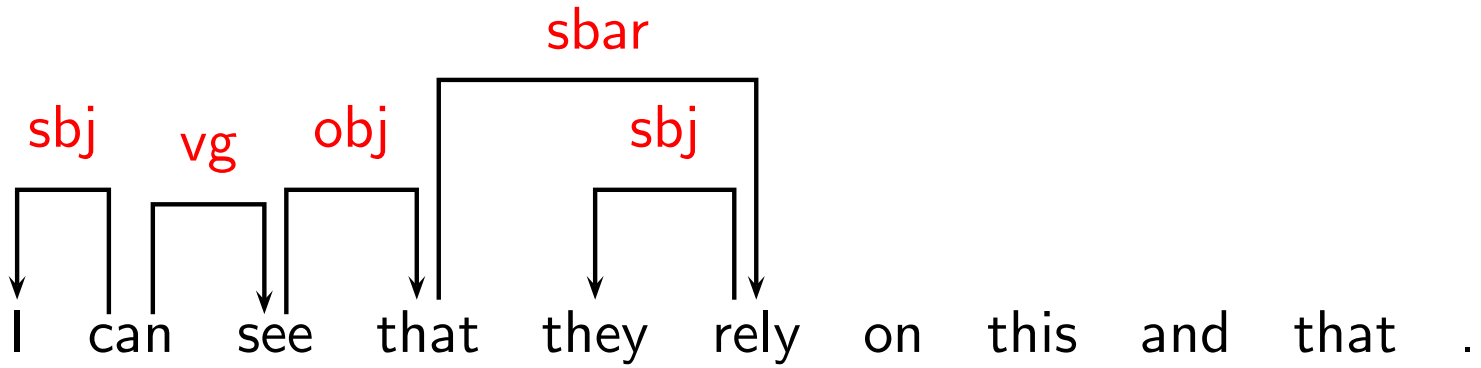
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



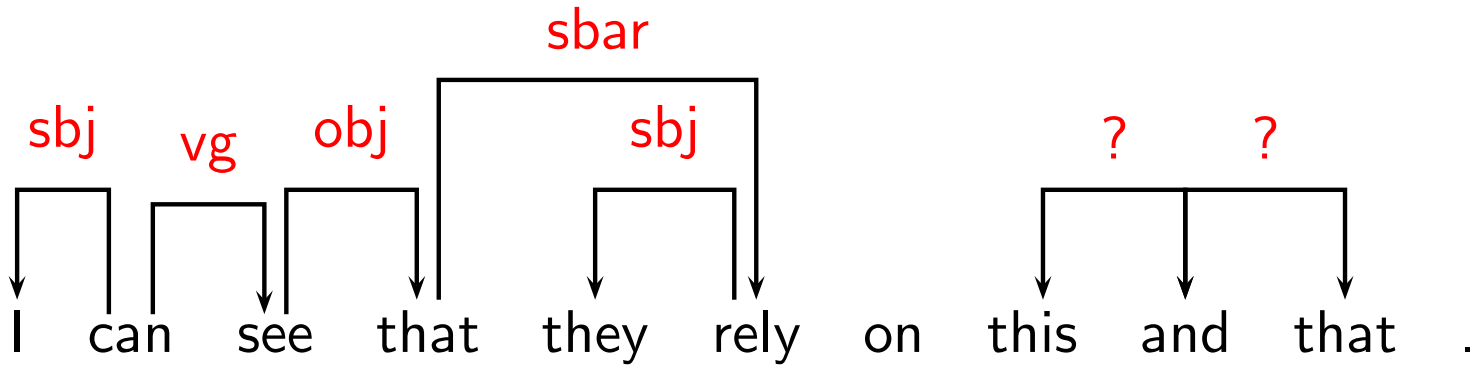
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



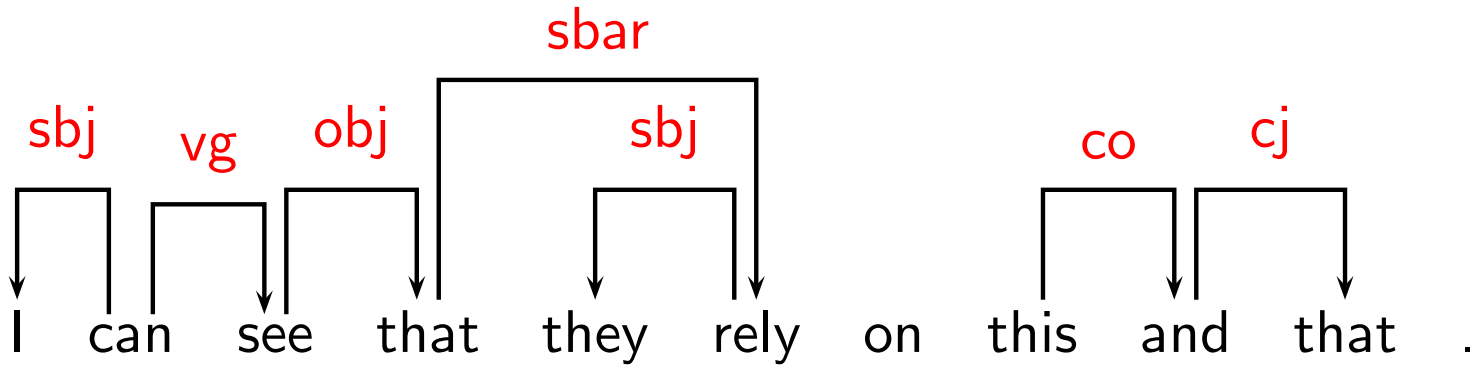
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



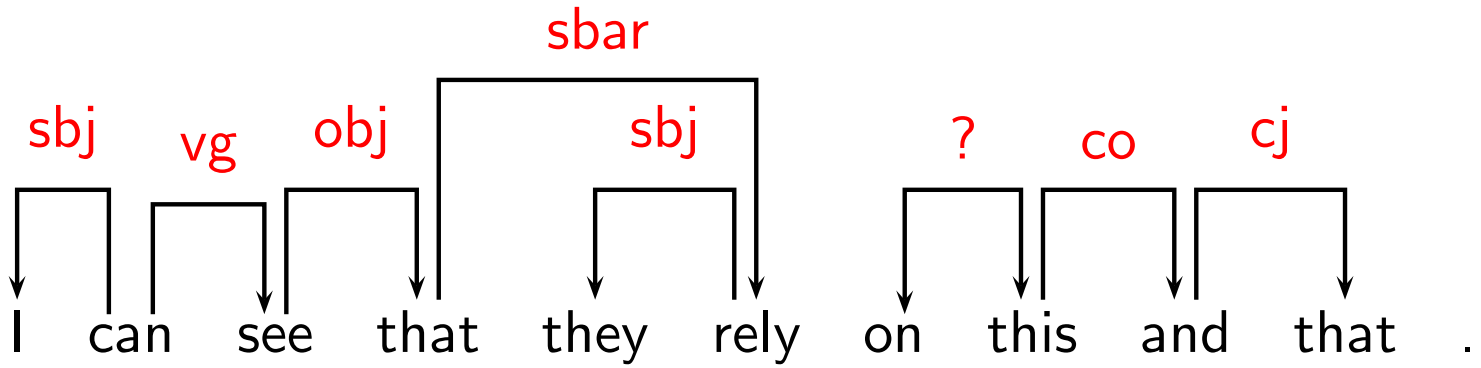
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



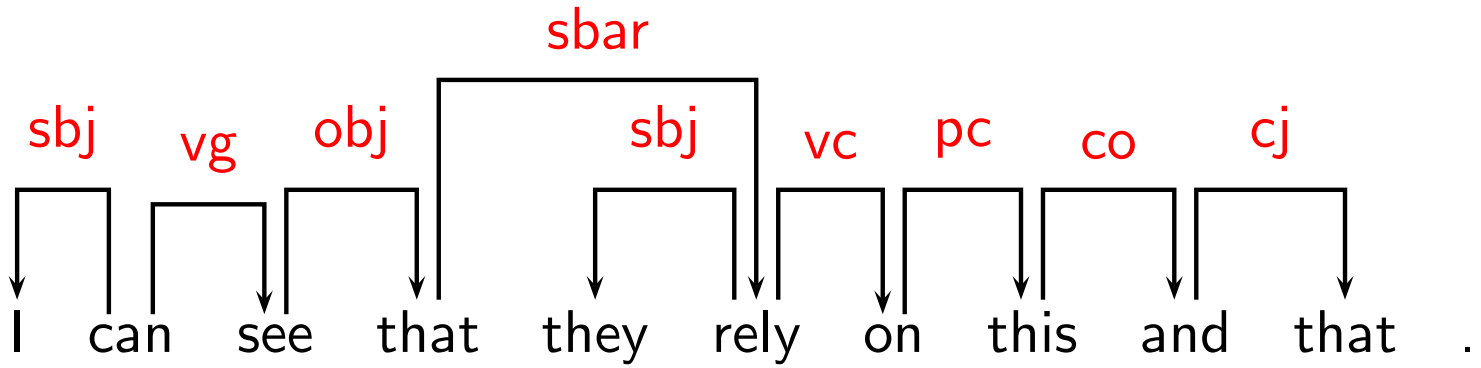
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



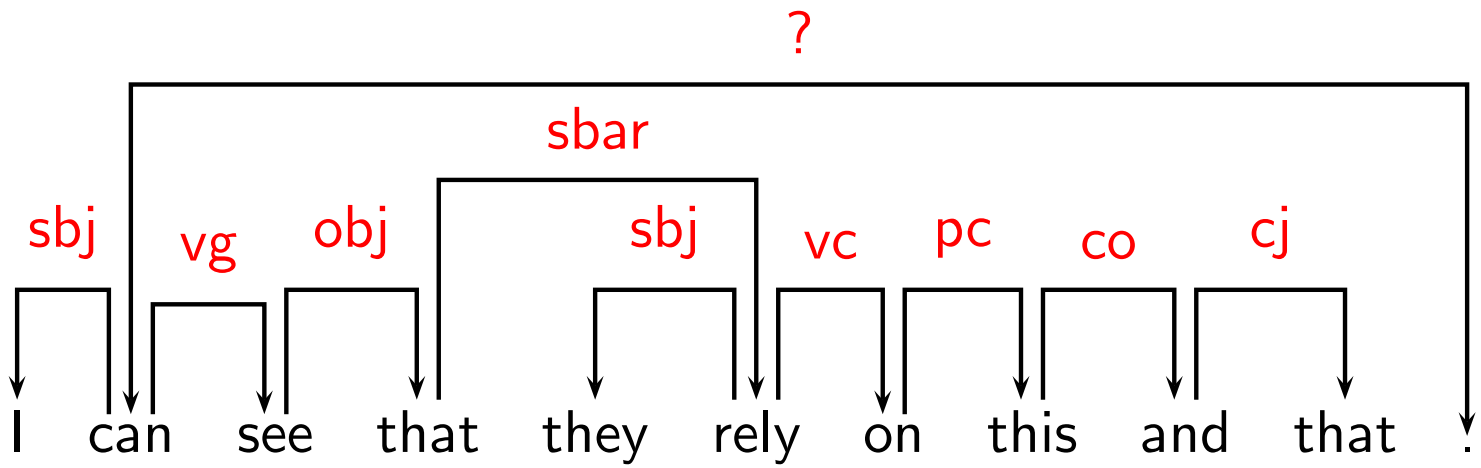
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



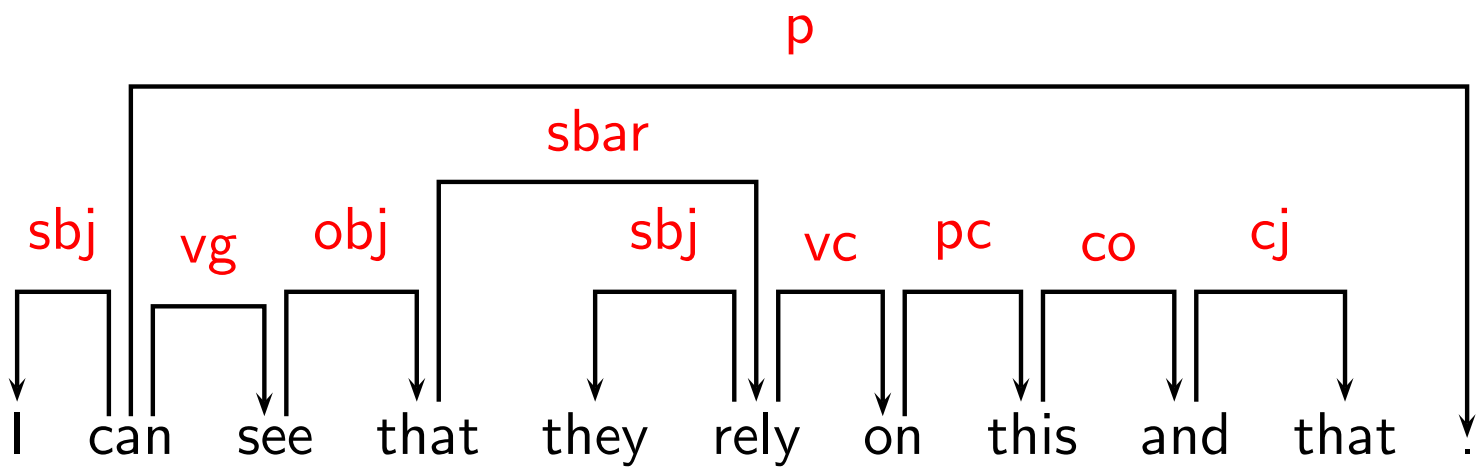
Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



Some Tricky Cases

- Complex verb groups (auxiliary $\leftrightarrow$ main verb)
- Subordinate clauses (complementizer $\leftrightarrow$ verb)
- Coordination (coordinator $\leftrightarrow$ conjuncts)
- Prepositional phrases (preposition $\leftrightarrow$ nominal)
- Punctuation



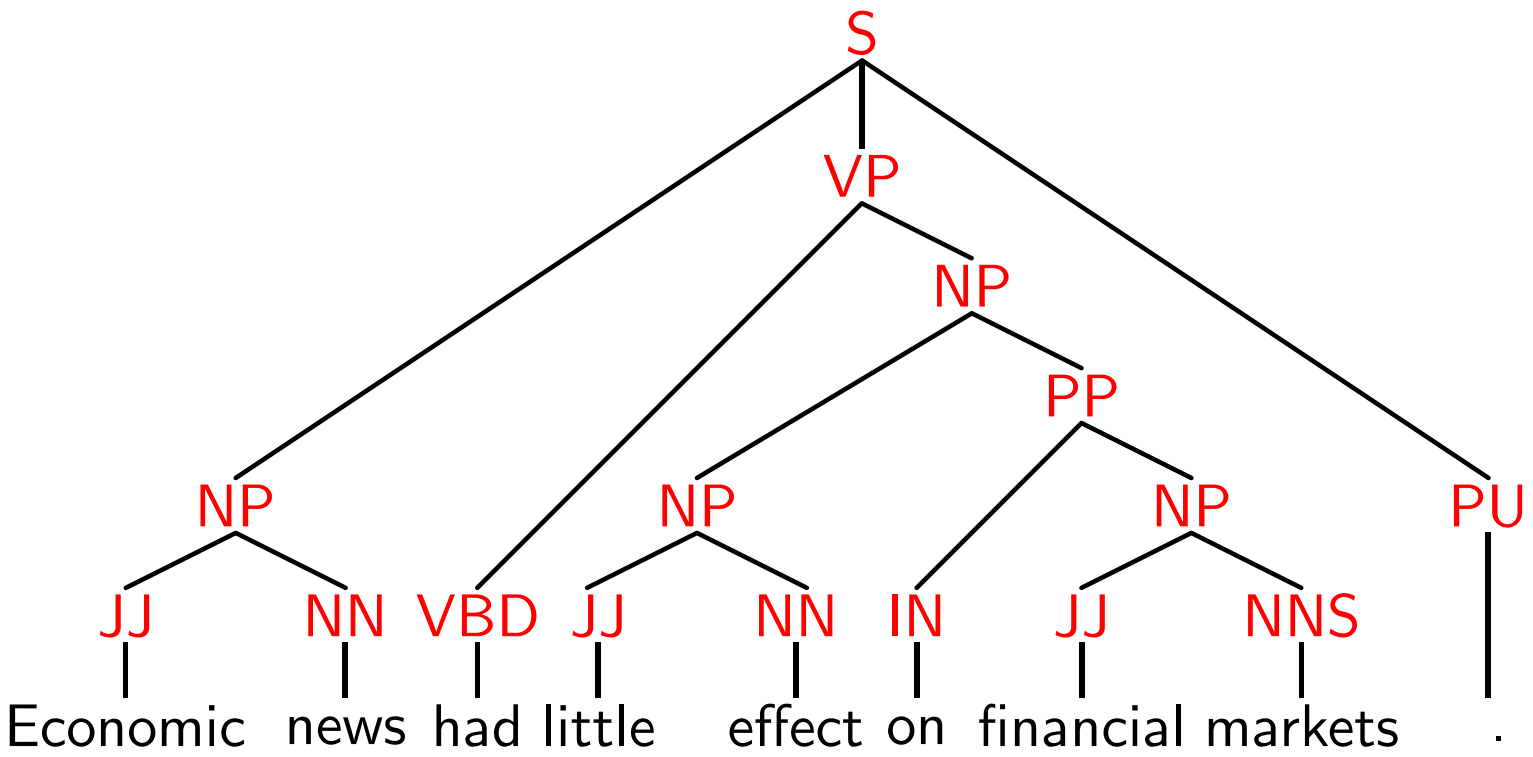
Dependency Grammar

- Not a coherent grammatical framework: wide range of different kinds of DG
  - just as there are wide ranges of "generative syntax"
- Different core ideas than phrase structure grammar
- We will base a lot of our discussion on [Mel'čuk(1988)]

Dependency grammar is important for those interested in CL:

- Increasing interest in dependency-based approaches to syntactic parsing in recent years (e.g., CoNLL-X shared task, 2006)

Phrase Structure



Comparison

- Dependency structures explicitly represent
  - head-dependent relations (directed arcs),
  - functional categories (arc labels),
  - possibly some structural categories (parts-of-speech).
- Phrase structures explicitly represent
  - phrases (nonterminal nodes),
  - structural categories (nonterminal labels),
  - possibly some functional categories (grammatical functions).
- Hybrid representations may combine all elements.

Some Theoretical Frameworks

- Word Grammar (WG) [Hudson(1984), Hudson(1990)]
- Functional Generative Description (FGD)
  [Sgall et al.(1986)Sgall, Hajičová and Panevová]
- Dependency Unification Grammar (DUG)
  [Hellwig(1986), Hellwig(2003)]
- Meaning-Text Theory (MTT) [Mel'čuk(1988)]
- (Weighted) Constraint Dependency Grammar ([W]CDG)
  [Maruyama(1990), Harper and Helzerman(1995),
  Menzel and Schröder(1998), Schröder(2002)]
- Functional Dependency Grammar (FDG)
  [Tapanainen and Järvinen(1997), Järvinen and Tapanainen(1998)]
- Topological/Extensible Dependency Grammar ([T/X]DG)
  [Duchier and Debusmann(2001),
  Debusmann et al.(2004)Debusmann, Duchier and Kruijff]

Dependency Graphs

- A dependency structure can be defined as a directed graph G, consisting of
  - a set V of nodes,
  - a set E of arcs (edges),
  - a linear precedence order $<$ on V
    (not in every theory)
- Labeled graphs:
  - Nodes in V are labeled with word forms (and annotation).
  - Arcs in E are labeled with dependency types.
- Notational conventions ($i, j \in V$):
  - $i \rightarrow j \;\equiv\; (i,j) \in E$
  - $i \rightarrow^* j \;\equiv\; i = j \vee \exists k : i \rightarrow k,\ k \rightarrow^* j$

Formal Properties of Dependency Graphs

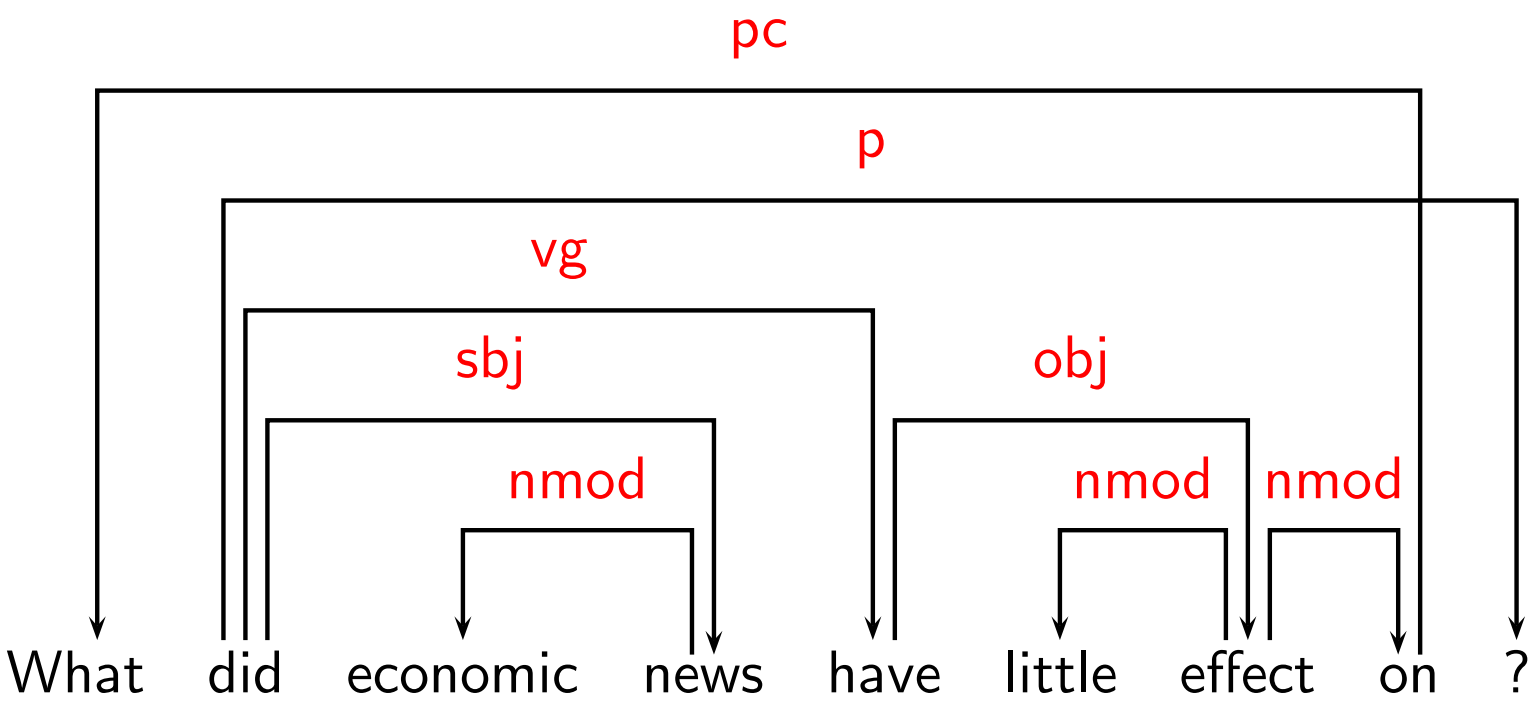
- **antisymmetric**: if A $\rightarrow$ B, then B $\nrightarrow$ A
  - cf. *box lunch* (lunch $\rightarrow$ box) vs. *lunch box* (box $\rightarrow$ lunch)
- **antireflexive**: if A $\rightarrow$ B, then B $\neq$ A
- **antitransitive**: if A $\rightarrow$ B and B $\rightarrow$ C, then A $\nrightarrow$ C
  - These are *direct* dependency relations
  - cf. *a usually reliable source*: source $\rightarrow$ reliable & reliable $\rightarrow$ usually, but source $\nrightarrow$ usually
- **labeled**: $\forall \rightarrow$, $\rightarrow$ has a label (r)

Formal Conditions on Dependency Graphs

- G is (weakly) connected:
  - For every node i there is a node j such that $i \rightarrow j$ or $j \rightarrow i$.
- G is acyclic:
  - If $i \rightarrow j$ then not $j \rightarrow^* i$.
- G obeys the single-head constraint:
  - If $i \rightarrow j$, then not $k \rightarrow j$, for any $k \neq i$.
- G is projective:
  - If $i \rightarrow j$ then $i \rightarrow^* k$, for any k such that $i < k < j$ or $j < k < i$.

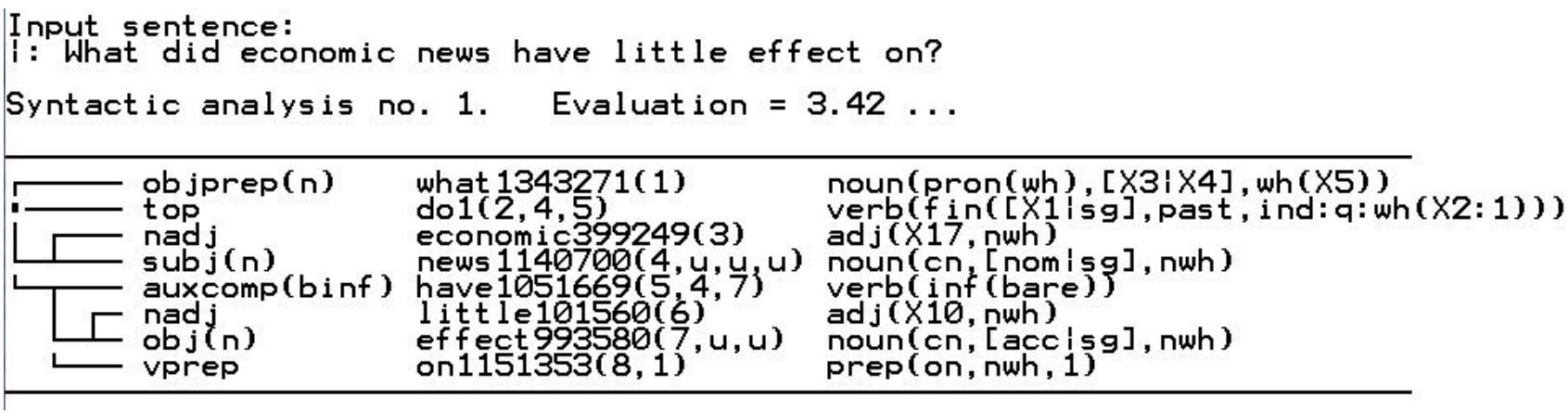
Projectivity

- Most theoretical frameworks do not assume projectivity.
- Non-projective structures are needed to account for
  - long-distance dependencies,
  - free word order.



LMT/Watson-Type Representation

- Slot Grammar representation
- tree rotated by 90 degrees
- mirrored at middle axis ...
- uses second level representation to express semantic links



Slot Grammar Rules

- Construction
  - adjunct declaration (what are the admitted adjuncts?)
  - obligation declaration (control phenomena, raising)
  - slot filler rules
  - slot ordering rules
  - extraposition rules
  - coordination
  - punctuation
- Evaluation

Basic representation unit

Phrase

phrase(Span,Sense,Feas,Frame,Ext,Mods)

[LB,H|RB] **mods(LM,RM)**

LM = [slot:phrase(...),slot:phrase(...)]

(inverse order:)

RM = [slot:phrase(...),slot:phrase(...)]

Adjunct declaration

Feas adjuncts {.....}

noun(Ntype,NFeas,Wh) adjuncts

(ndet.ndetgen.ngen.nadj.nnoun.napos.
naposo.nprep.nrel.ncompar.n|selbst.pnspec.
ncompound.npbrk.ncorrel.nadv.nper).

Slot filler rule

Slot ==> Conds

obj(n) ==>
f(noun(NType,[Cases,Gen,StA|_],Wh)),
cf(noun(NType,[acc,Num,Gen,STA|_],Wh)).

Slot filler rule

Slot ==> Conds

```
ndet ==>
        f(det(Feas,Dtype)) ,
        hf(noun(NType,[Cases,Gen,StA|ST],Wh)) ,
        detagree(Feas,Dtype,Cases,Cases1),
        hcf(noun(NType,[Cases1,Gen,StA|det],Wh)).
```

Relation to phrase structure

After all this discussion, what is the relation between DG and PSG?

- If a PS tree has heads marked, then you can derive the dependencies
- Likewise, a DG tree can be converted into a PS tree by grouping a word with its dependents
  - But what the constituents are is still open (binary-branching, flat)
  - And phrases are not categorized

Outline

MaltParser

- Dependency Parser
- Data-Driven Parser-Generation
- Deterministic, probability-based left-to-right Parser
- Pre-trained models for English, French, Swedish, Spanish

Some Papers

- Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. (IWPT03)
- Nivre, J., J. Hall and J. Nilsson (2006) MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. (LREC 06)
- Nivre et al. (2007) MaltParser: A language-independent system for data-driven dependency parsing. Natural Language Engineering, 13(2)
- Hall, J. and J. Nivre (2008) A Dependency-Driven Parser for German Dependency and Constituency Representations (ACL, PaGe 08)

Some Basic Features

MaltParser System

- generates dependency parsers from tree banks
- parsers have a dependency accuracy of 80 - 90 %
- on the basis of modest data resources (on the order of 100k tokens or less).
- freely available for research and educational purposes
- `http://www.maltparser.org`

Dependency Structures

- lexical nodes linked by binary relations called *dependencies*
- Dependency graph

A dependency graph $D = (N_W, A)$ is well-formed iff the following conditions are satisfied:

Single head	$(\forall n\, n'\, n'')\,(n \rightarrow n' \wedge n'' \rightarrow n') \Rightarrow n = n''$
Acyclic	$(\forall n\, n')\,\neg(n \rightarrow n' \wedge n' \rightarrow^* n)$
Connected	$(\forall n\, n')\; n \leftrightarrow^* n'$
Projective	$(\forall n\, n'\, n'')\,(n \leftrightarrow n' \wedge n < n'' < n') \Rightarrow (n \rightarrow^* n'' \vee n' \rightarrow^* n'')$

- Grammar: consists of D-rules

$$w \leftarrow w'$$
$$w \rightarrow w'$$

Given a string of words with nodes n and n' such that $\text{LEX}(n) = w$, $\text{LEX}(n') = w'$ and $n < n'$, the first rule says that n' can be the head of n, while the second rule says that n can be the head of n'.

Grammar transformation

A general dependency grammar can be transformed into D-rule format:

given a grammar in the formalism of Hays [19], we can construct a set of directed D-rules in the following way:

- For every rule $X(X_{-m} \cdots X_{-1} * X_1 \cdots X_n)$ in the original grammar, introduce rules $w_l \leftarrow w$ and $w \rightarrow w_r$ for every $w \in X, w_l \in X_{-m} \cup \cdots \cup X_{-1}, w_r \in X_1 \cup \cdots \cup X_n$.

MaltParser Parsing

- similar to general *shift/reduce* algorithm
- General data structure:
  $\langle S, I, A \rangle$ where
  - S . . . stack of (active) nodes (= tokens = word+position)
  - I . . . list of (remaining) input tokens
  - A . . . dependency relation recognized so far
- Start configuration:
  $\langle nil, W, \emptyset \rangle$
- End configuration:
  $\langle S, nil, A \rangle$
- *accept* if $\langle N_W, A \rangle$ is well-formed

MaltParser example

$W =$ | på (in) | 60-talet (the-60's) | målade (painted) | han (he) | tavlor (pictures)

$R \supseteq \{$ på → 60-talet, på ← målade, målade → han, målade → tavlor $\}$

$\langle \textbf{nil}, \text{på 60-talet målade han tavlor}, \emptyset \rangle$

$\xrightarrow{\textbf{S}}$ $\langle \text{på}, \text{60-talet målade han tavlor}, \emptyset \rangle$

$\xrightarrow{\textbf{RA}}$ $\langle \text{60-talet på}, \text{målade han tavlor}, \{(\text{på}, \text{60-talet})\} \rangle$

$\xrightarrow{\textbf{R}}$ $\langle \text{på}, \text{målade han tavlor}, \{(\text{på}, \text{60-talet})\} \rangle$

$\xrightarrow{\textbf{LA}}$ $\langle \textbf{nil}, \text{målade han tavlor}, \{(\text{på}, \text{60-talet}), (\text{målade}, \text{på})\} \rangle$

$\xrightarrow{\textbf{S}}$ $\langle \text{målade}, \text{han tavlor}, \{(\text{på}, \text{60-talet}), (\text{målade}, \text{på})\} \rangle$

$\xrightarrow{\textbf{RA}}$ $\langle \text{han målade}, \text{tavlor}, \{(\text{på}, \text{60-talet}), (\text{målade}, \text{på}), (\text{målade}, \text{han})\} \rangle$

$\xrightarrow{\textbf{R}}$ $\langle \text{målade}, \text{tavlor}, \{(\text{på}, \text{60-talet}), (\text{målade}, \text{på}), (\text{målade}, \text{han})\} \rangle$

$\xrightarrow{\textbf{RA}}$ $\langle \text{tavlor målade}, \textbf{nil}, \{(\text{på}, \text{60-talet}), (\text{målade}, \text{på}), (\text{målade}, \text{han}), (\text{målade}, \text{tavlor})\} \rangle$

Figure 3: Parse of Swedish sentence

Parsing Actions

Initialization	$\langle \mathbf{nil}, W, \emptyset \rangle$	
Termination	$\langle S, \mathbf{nil}, A \rangle$	
Left-Arc	$\langle n\|S, n'\ I, A \rangle \rightarrow \langle S, n'\ I, A \cup \{(n', n)\} \rangle$	$\textsc{lex}(n) \leftarrow \textsc{lex}(n') \in R$ $\neg\exists n''(n'', n) \in A$
Right-Arc	$\langle n\|S, n'\ I, A \rangle \rightarrow \langle n'\|n\ S, I, A \cup \{(n, n')\} \rangle$	$\textsc{lex}(n) \rightarrow \textsc{lex}(n') \in R$ $\neg\exists n''(n'', n') \in A$
Reduce	$\langle n\|S, I, A \rangle \rightarrow \langle S, I, A \rangle$	$\exists n'(n', n) \in A$
Shift	$\langle S, n\ I, A \rangle \rightarrow \langle n\|S, I, A \rangle$	

Choice of Actions

- per se non-deterministic
- in practice deterministic selection (complexity O(n))

a)	baseline parser	$LA > RA > R > S$
b)	S/R parser	as (a) + $S > R$ if S(0) can be a transitive head of I(0)
c)	S/RA parser	as (b) + lookahead: $S > RA$ if I(0) can be a pre-modifier of I(1)/I(2),...

Choice of Actions

S/RA ambiguity - example

PN	**VB**	**AB**	**JJ**	NN
han	**målar**	**extremt**	**djärva**	tavlor
(he)	**(paints)**	**(extremely)**	**(bold)**	(pictures)

Training and Results (2003)

- Stockholm-Unmeå Corpus (mixed)
- 4000 tokens
- vocabulary: word-PoS-tag pairs
- 257 sentences with manually annotated dependency graphs
- hand-crafted grammar with 126 rules (90 left-headed, 36 right-headed)

Training and Results (2003)

Attachment Score: percentage of words in the sentence with correct head

Parser	Mean	Std
Baseline	80.0	13.2
S/R	87.8	11.0
S/RA	89.0	10.6

Table 1: Attachment scores (mean and standard deviation)

Feature-based Approach (2006)

Data Structure

- Stack of tokens . . . Stack(0), Stack(1), . . .
- Input list of (remaining) input tokens . . . Input(0), Input(1), . . .
- Unattached tokens between Stack(0) and Input(0) . . . Context(0), Context(1), . . .
- Partial function Head where Head(i)=syntactic head of i
- Function Dep with Dep(i) giving the label of the relation to Head(i)
- Function LC with LC(i) = leftmost child of i
- Function RC with RC(i) = rightmost child of i
- Function LS with LS(i) = next left sibling of i
- Function RS with RS(i) = next right sibling of i

Feature-based Approach (2006)

Feature model

```
<fspec>   ::= <feat>+
<feat>    ::= <lfeat>|<nlfeat>
<lfeat>   ::= LEX\t<dstruc>\t<off>\t<suff>\n
<nlfeat>  ::= (POS|DEP)\t<dstruc>\t<off>\n
<dstruc>  ::= (STACK|INPUT|CONTEXT)
<off>     ::= <nnint>\t<int>\t<nnint>
              \t<int>\t<int>
<suff>    ::= <nnint>
<int>     ::= (...|-2|-1|0|1|2|...)
<nnint>   ::= (0|1|2|...)
```

Offsets

- 3. column: stack/input/context element
- 4. column: negative/positive offset to (3.)
- 5. number of applications of head function
- 6. number of applications of LC/RC
- 7. number of applications of LS/RS

Examples

POS	STACK	0	0	0	0
POS	INPUT	1	0	0	0
POS	INPUT	0	-1	0	0
DEP	STACK	0	0	1	0
DEP	STACK	0	0	0	-1

Abbreviation:

POS	STACK				
POS	INPUT	1			
POS	INPUT	0	-1		
DEP	STACK	0	0	1	
DEP	STACK	0	0	0	-1

The Standard Model

```
POS    STACK
POS    INPUT
POS    INPUT 1
POS    INPUT 2
POS    INPUT 3
POS    STACK 1
DEP    STACK
DEP    STACK 0      0      0      -1
DEP    STACK 0      0      0      1
DEP    INPUT 0      0      0      -1
LEX    STACK
LEX    INPUT
LEX    INPUT 1
LEX    STACK 0      0      1
```

Learning and Parsing

MaltParser 3.0:

- provides 2 learning algorithms
  - Memory-based learning and classification (Daelemans and Van der Bosch 2000)
  - Application of support vector machines
- can be run in 2 modes
  - learning mode
  - parsing mode

Malt-TAB Format

```
This    DT    2    SBJ
is      VBZ   0    ROOT
an      DT    4    DET
old     JJ    4    NMOD
story   NN    2    PRD
.       .     2    P

So      RB    2    PRD
is      VBZ   0    ROOT
this    DT    2    SBJ
.       .     2    P
```

Outline

Stanford Parsers

Different versions

- (lexicalized) PCFG (*probabilistic context-free grammar*) parsers
- (lexicalized)dependency grammar parsers
- (English, Chinese, German, Arabic, . . .)
- `http://nlp.stanford.edu/software/lex-parser.shtml`

Some Papers

- PCFG parser:
  Klein, D., Manning, Ch. (2003) Accurate Unlexicalized Parsing.(ACL 03)
- (English) Stanford Dependencies representation:
  de Marneffe, M.C., MacCartney, B., Manning, Ch. (2006) Generating Typed Dependency Parses from Phrase Structure Parses. (LREC 2006).
- Neural-network dependency parser:
  Chen, D. Manning, Ch. (2014) A Fast and Accurate Dependency Parser using Neural Networks (EMNLP 2014)
- Compositional Vector Grammar parser:
  Socher, R., Bauer, J., Manning, Ch., Ng, A. (2013) Parsing With Compositional Vector Grammars. (ACL 2013)

Example

Stanford Parser

Please enter a sentence to be parsed:

Which chair does Mary believe John said he was sitting in?

Language: English Sample Sentence Parse

Your query

Which chair does Mary believe John said he was sitting in?

Tagging

Which/WDT chair/NN does/VBZ Mary/NNP believe/VB John/NNP said/VBD he/PRP was/VBD sitting/VBG in/RP ?/.

Example

Parse

```
(ROOT
  (SBARQ
    (WHNP (WDT Which) (NN chair))
    (SQ (VBZ does)
      (NP (NNP Mary))
      (VP (VB believe)
        (SBAR
          (S
            (NP (NNP John))
            (VP (VBD said)
              (SBAR
                (S
                  (NP (PRP he))
                  (VP (VBD was)
                    (VP (VBG sitting)
                      (PRT (RP in)))))))))))
    (. ?)))
```

Universal dependencies

```
det(chair-2, Which-1)
dobj(believe-5, chair-2)
aux(believe-5, does-3)
nsubj(believe-5, Mary-4)
root(ROOT-0, believe-5)
nsubj(said-7, John-6)
ccomp(believe-5, said-7)
nsubj(sitting-10, he-8)
aux(sitting-10, was-9)
ccomp(said-7, sitting-10)
compound:prt(sitting-10, in-11)
```

Universal dependencies, enhanced

```
det(chair-2, Which-1)
dobj(believe-5, chair-2)
aux(believe-5, does-3)
nsubj(believe-5, Mary-4)
root(ROOT-0, believe-5)
nsubj(said-7, John-6)
ccomp(believe-5, said-7)
nsubj(sitting-10, he-8)
aux(sitting-10, was-9)
ccomp(said-7, sitting-10)
compound:prt(sitting-10, in-11)
```