Multiword Expressions in Dependency Parsing

Joakim Nivre

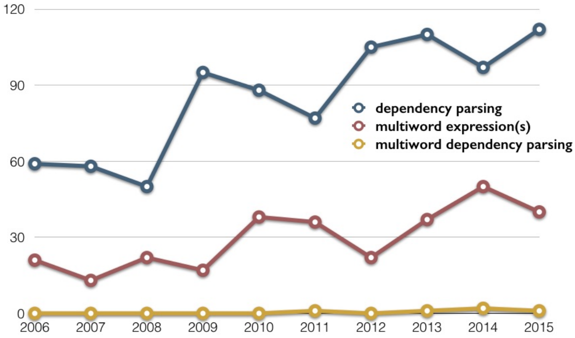
Uppsala University
Department of Linguistics and Philology

Introduction

- Do we really need to talk about MWEs in dependency parsing?
- Research has been booming in both areas over the last decade!

Introduction

- Do we really need to talk about MWEs in dependency parsing?
- Research has been booming in both areas over the last decade!

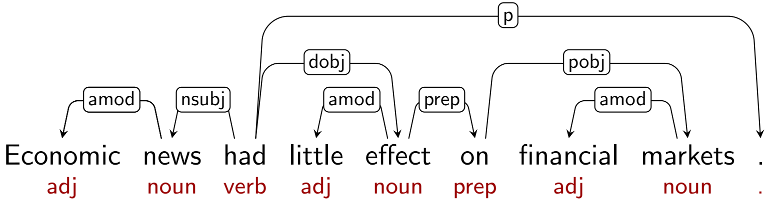


Plan for the Talk

1. Basic concepts of dependency parsing
2. Integrating multiword expressions
   2.1 Linguistic representations
   2.2 Parsing techniques
   2.3 Empirical studies
3. Transition-based joint lexical and syntactic analysis

Dependency Trees

- For a sentence $x = w_1, \ldots, w_n$ and label set L, a dependency tree is a directed tree with labeled arcs over the tokens
- Formally, we model a tree $T = (V, A)$ as follows:
  1. $V = \{1, \ldots, n\}$ is a set of nodes, one for each input token
  2. A is a set of arcs (i, l, j), with $i, j \in V$ and $l \in L$
  3. A is a spanning tree in the graph $G_x = \{(i, l, j) | i, j \in V, l \in L\}$



Dependency Parsing

- The task:
  - Input: sentence $x = w_1, \ldots, w_n$
  - Output: dependency tree $T = (V, A)$ for x
- Graph-based parsing [McDonald et al. 2005]:
  - Learns a (factored) model for scoring spanning trees in G_x
  - Needs efficient spanning tree algorithms for parsing
- Transition-based parsing [Nivre 2003]:
  - Learns a (local) model for scoring parsing actions (transitions)
  - Relies on heuristic search for the optimal sequence of actions

Integrating Multiword Expressions

1. Linguistic representations
   - How do we put MWEs into dependency trees?
2. Parsing techniques
   - How do we process MWEs using dependency parsers?
3. Empirical studies
   - What works and what doesn't?

Linguistic Representations

- How do we represent MWEs in dependency trees?
- Do we need to modify the definition of a dependency tree?
- What about different classes of MWEs?
  - Fixed: by and large, in spite of
  - Semi-fixed: part(s) of speech, kick(s/ed) the bucket
  - Flexible: put off, look for, take a photo
- What about discontiguous MWEs?

The Spanning Tree Assumption

- Basic assumption in (current) dependency parsing:
  - Dependency structure for x is a spanning tree in G_x
  - Every token is a node in the dependency tree (spanning)
  - Every node (except the root) has one incoming arc (tree)
- Possible variations:
  - Give up the tree assumption – allow general graphs
  - Give up the spanning assumption – tokens $\neq$ nodes

Tokens and Nodes

Token	Node	Example
1	1	Ordinary word tokens

Tokens and Nodes

Token	Node	Example
1	1	Ordinary word tokens
1	>1	Clitics, contractions

Tokens and Nodes

Token	Node	Example
1	1	Ordinary word tokens
1	>1	Clitics, contractions
>1	1	Multiword expressions

Tokens and Nodes

Token	Node	Example
1	1	Ordinary word tokens
1	>1	Clitics, contractions
>1	1	Multiword expressions
1	0	Punctuation?

Tokens and Nodes

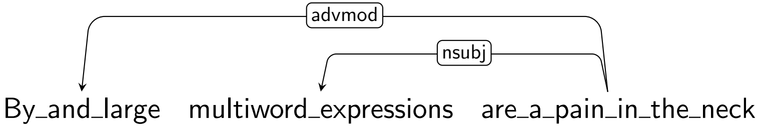
Token	Node	Example
1	1	Ordinary word tokens
1	>1	Clitics, contractions
>1	1	Multiword expressions
1	0	Punctuation?
0	1	Ellipsis?

Tokens and Nodes

Token	Node	Example
1	1	Ordinary word tokens
1	>1	Clitics, contractions
>1	1	Multiword expressions
1	0	Punctuation?
0	1	Ellipsis?

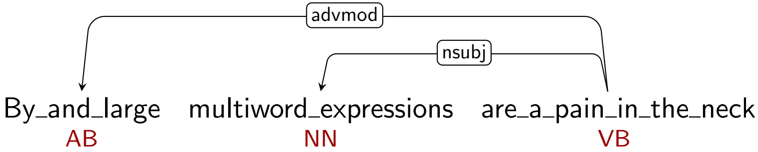
This requires a new type of dependency parser!

MWEs as Special Tokens



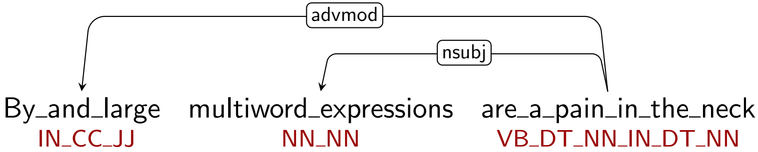
- Simplifies parsing **if** MWEs can be identified prior to parsing
- Works for contiguous MWEs, awkward for flexible MWEs
- Common in treebanks (about half of the CoNLL-X data sets)
- What about part-of-speech tags?

MWEs as Special Tokens



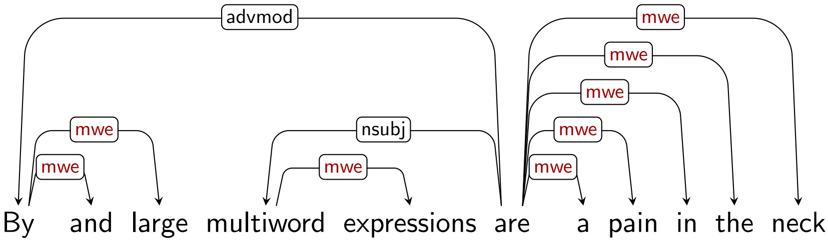
- Simplifies parsing **if** MWEs can be identified prior to parsing
- Works for contiguous MWEs, awkward for flexible MWEs
- Common in treebanks (about half of the CoNLL-X data sets)
- What about part-of-speech tags?

MWEs as Special Tokens



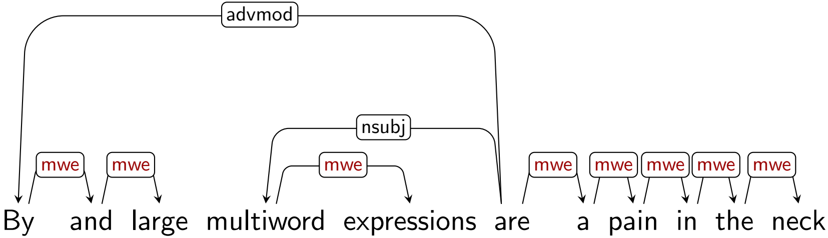
- Simplifies parsing **if** MWEs can be identified prior to parsing
- Works for contiguous MWEs, awkward for flexible MWEs
- Common in treebanks (about half of the CoNLL-X data sets)
- What about part-of-speech tags?

MWEs as Dummy Dependency Structures



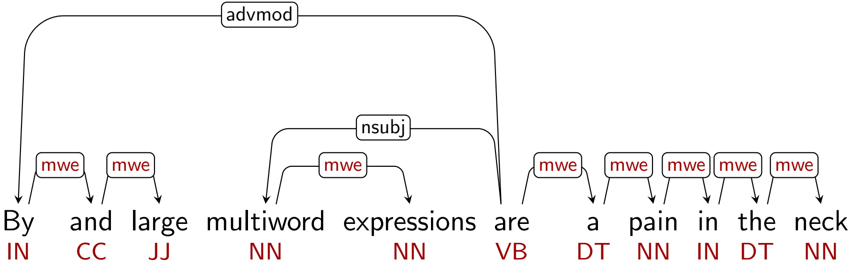
- Canonical structure without syntactic significance
- Special labels distinguish from real dependencies
- The problem with part-of-speech tags remain

MWEs as Dummy Dependency Structures



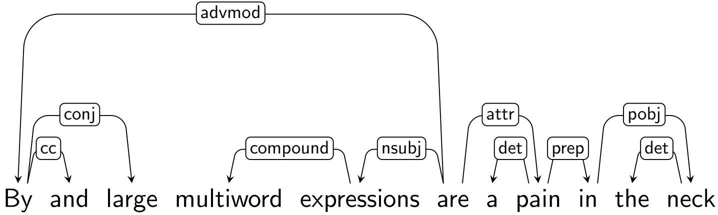
- Canonical structure without syntactic significance
- Special labels distinguish from real dependencies
- The problem with part-of-speech tags remain

MWEs as Dummy Dependency Structures



- Canonical structure without syntactic significance
- Special labels distinguish from real dependencies
- The problem with part-of-speech tags remain

MWEs as Real Dependency Structures



- Dependency structure reflects real internal structure
- Special labels may be used for subtypes (for example, LVCs)
- Part-of-speech tags do not reflect MWE status

So what representations should we use?

- Different types of MWEs require different representations
- At one end of the spectrum: by and large
  - No point in representing internal syntactic structure
  - Equivalent to a single node in dependency structure
  - Special tokens or dummy dependencies?
- At the other end: take a photo
  - Needs internal structure to allow modification and inflection
  - Real dependencies, special labels?
- What about everything in between?

Parsing Techniques

- Three main approaches:
  - Pre-processing – analyze MWEs before parsing
  - Post-processing – analyze MWEs after parsing
  - Joint processing – analyze MWEs during parsing
- Key question:
  - Does MWE identification help parsing or vice versa or both?
  - The answer may be different for different types of MWEs!

Techniques and Representations

	Pre	Joint	Post
Special tokens	yes	no	yes

Techniques and Representations

	Pre	Joint	Post
Special tokens	yes	no	yes
Dummy dependencies	?	yes	?

Techniques and Representations

	Pre	Joint	Post
Special tokens	yes	no	yes
Dummy dependencies	?	yes	?
Real dependencies	no	yes	yes

Techniques and Representations

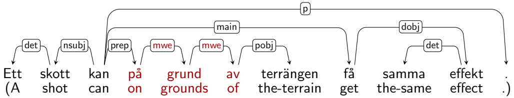
	Pre	Joint	Post
Special tokens	yes	no	yes
Dummy dependencies	?	yes	?
Real dependencies	no	yes	yes

If different types of MWEs require different representations, they may require different processing techniques as well!

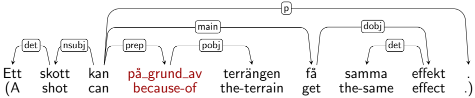
An Early Study [Nivre and Nilsson 2004]

- Swedish treebank with (limited) MWE annotation:
  - Function words like in spite of, at large
  - Names like Carl XVI Gustaf, Swedish Academy
  - Numerical expressions like $2 + 2 = 4$

1. Joint processing with dummy dependencies:



2. Preprocessing with special tokens (gold input):



Results

	MWE	Other
Joint	71.1	80.7
Preprocessing	–	81.6

- Perfect MWE recognition improves parsing accuracy (slightly)
- Typical effects of failing to recognize MWEs:
  - Unusual part-of-speech patterns leads to distorted structure (vad beträffar = as regards)
  - Different attachment preferences for MWEs and compositional phrases (i regel = as a rule)

Results

	MWE	Other
Joint	71.1	80.7
Preprocessing	–	81.6

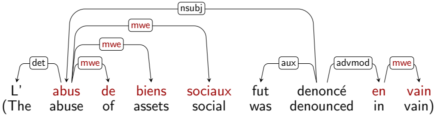
- Perfect MWE recognition improves parsing accuracy (slightly)
- Typical effects of failing to recognize MWEs:
  - Unusual part-of-speech patterns leads to distorted structure (vad beträffar = as regards)
  - Different attachment preferences for MWEs and compositional phrases (i regel = as a rule)

Similar results observed later for Turkish [Eryiğit et al. 2011]

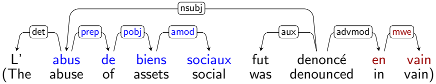
Regular and Irregular MWEs

[Candito and Constant 2014]

- French dependency treebank with dummy MWE dependencies:



- Alternative representations for regular MWEs:



- PoS patterns used to distinguish regular and irregular MWEs

Processing Models

	Irregular	Regular
Joint	Parser	Parser
Joint-Reg	Pre	Parser
Joint-Irreg	Parser	Post
Pipeline	Pre	Post

- Pre = MWEs pre-recognized and merged to single tokens
- Post = MWEs recognized after parsing

Results

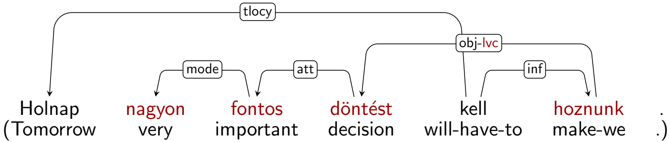
	Dummy		Real	
	MWE[1]	Overall	MWE[2]	Overall
Joint	73.5	84.5	81.4	86.9
Joint-Reg	73.3	84.2	80.4	86.6
Joint-Irreg	75.4	84.4	82.1	87.0
Pipeline	74.4	83.9	80.4	86.5

[1]) All MWEs. [2]) Irregular MWEs only.

- Syntactic accuracy higher with real dependencies
- Irregular MWEs benefit from joint processing
- Regular MWEs better identified after parsing?

Light Verb Constructions [Vincze et al. 2013]

- Hungarian dependency treebank with LVC annotation:



- Can a parser learn to identify light verb constructions?
- How is overall parsing accuracy affected?

Results

	LVC	Overall
Parser plain	–	90.6
Parser LVC	75.6	90.4
Post dictionary	21.3	–
Post C4.5	74.5	–

- Parser improves LVC identification with a marginal drop in overall labeled attachment score
- Parser significantly better than post-classifier on discontiguous LVCs ($64.0 > 60.0$)

Interim Conclusion

- We have only scratched the surface ...
- Complex interaction between several factors:
  - MWE types
  - Linguistic representations
  - Processing techniques
- Tentative conclusions:
  - MWE identification can benefit from syntactic context
  - Regular MWEs should be assigned regular syntactic structure

Joint Lexical and Syntactic Analysis

- Joint work with **Mathieu Constant** [Constant and Nivre 2016]
- Factored representation for lexical and syntactic analysis
  - Lexical trees represent lexical structure (including MWEs)
  - Dependency trees represent syntactic structure
  - Two orthogonal dimensions synchronized at the token level
- Transition-based system that processes both dimensions jointly

Lexical and Syntactic Structure

- Sentence = sequence of tokens $w_1, \ldots, w_n$
- Lexical tree = tree over tokens (possibly discontiguous)

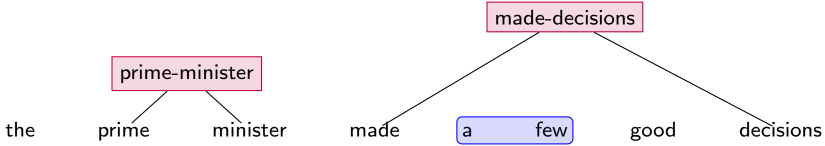
Lexical unit	Tokens	Syntactic unit
Word	1	yes
Fixed MWE	>1	yes
Non-fixed MWE	>1	no

- Syntax tree = tree over syntactic units (words, F-MWEs)

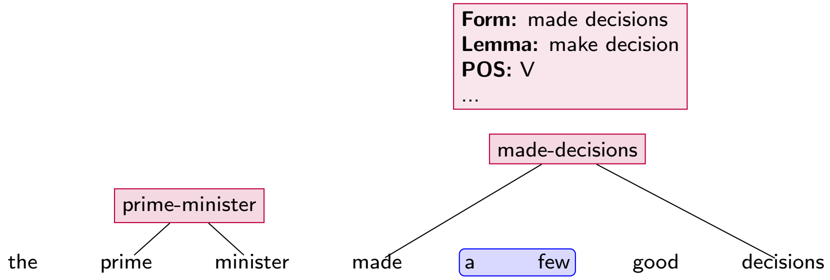
Lexical and Syntactic Structure

the prime minister made a few good decisions

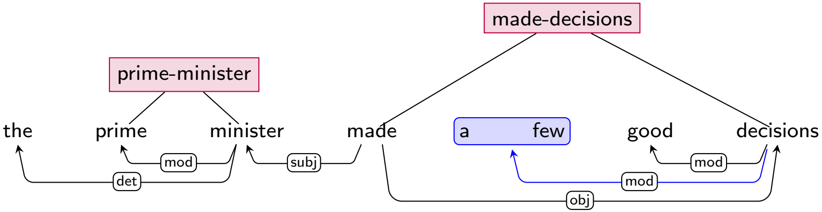
Lexical and Syntactic Structure



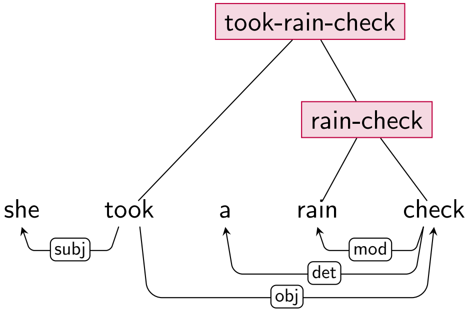
Lexical and Syntactic Structure



Lexical and Syntactic Structure



MWE Embedding



A Transition-Based System

- Handling two linguistic dimensions:
  - Two stacks: one syntactic stack and one lexical stack
  - One buffer to synchronize the two dimensions
  - Output: one set of dependency arcs and one set of lexical trees
- Handling MWEs:
  - Mild extension of arc-standard transition system
  - Specific transitions to deal with MWE identification

Transition system

- Configuration:

  (Buffer, SynStack, SynArcs, LexStack, LexTrees)

- Initialization:

  $([w_1, \ldots, w_n], [\,], \{\,\}, [\,], \{\,\})$

  Input: $w_1, \ldots, w_n$

- Termination:

  $([\,], [x]$, SynArcs, $[\,]$, LexTrees)

  Output: SynArcs, LexTrees

Transition system

- **Shift**
  - Moves next token from buffer to both stacks
- **Right-Arc(l), Left-Arc(l)**
  - Adds syntactic arc between top items on syntactic stack
- **Merge$_F$(t)**
  - Creates lexical tree from top items on both stacks – F-MWE
- **Merge$_N$(t)**
  - Creates lexical tree from top items on lexical stack – NF-MWE
- **Complete**
  - Adds lexical tree from lexical stack

Example Parse

Transition

–

Buffer

[he made a few decisions]

SynStack

[ ]

SynArcs

–

LexStack

[ ]

LexTrees

–

Example Parse

Transition
Shift

Buffer
[made a few decisions]

SynStack
[he]

SynArcs
–

LexStack
[he]

LexTrees
–

Example Parse

Transition
Complete

Buffer
[made a few decisions]

SynStack
[he]

SynArcs
–

LexStack
[ ]

LexTrees
he

Example Parse

Transition
Shift

Buffer
[a few decisions]

SynStack
[he made]

SynArcs
–

LexStack
[made]

LexTrees
he

Example Parse

Transition
Left-Arc(subj)

Buffer
[a few decisions]

SynStack
[made]

SynArcs
subj(made, he)

LexStack
[made]

LexTrees
he

Example Parse

Transition
Shift

Buffer
[few decisions]

SynStack
[made a]

SynArcs
subj(made, he)

LexStack
[made a]

LexTrees
he

Example Parse

Transition
Shift

Buffer
[decisions]

SynStack
[made a few]

SynArcs
subj(made, he)

LexStack
[made a few]

LexTrees
he

Example Parse

Transition

Merge_F(A)

Buffer

[decisions]

SynStack

[made A(a, few)]

SynArcs

subj(made, he)

LexStack

[made A(a, few)]

LexTrees

he

Example Parse

Transition
Complete

Buffer
[decisions]

SynStack
[made A(a, few)]

SynArcs
subj(made, he)

LexStack
[made]

LexTrees
he, A(a, few)

Example Parse

Transition
Shift

Buffer
[ ]

SynStack
[made A(a, few) decisions]

SynArcs
subj(made, he)

LexStack
[made decisions]

LexTrees
he, A(a, few)

Example Parse

Transition
Left-Arc(mod)

Buffer
[ ]

SynStack
[made decisions]

SynArcs
subj(made, he)
mod(decisions, A(a, few))

LexStack
[made decisions]

LexTrees
he, A(a, few)

Example Parse

Transition
$\text{Merge}_N(\text{V})$

Buffer
[ ]

SynStack
[made decisions]

SynArcs
subj(made, he)
mod(decisions, A(a, few))

LexStack
[V(made, decisions)]

LexTrees
he, A(a, few)

Example Parse

Transition
Complete

Buffer
[ ]

SynStack
[made decisions]

SynArcs
subj(made, he)
mod(decisions, A(a, few))

LexStack
[ ]

LexTrees
he, A(a, few), V(made, decisions)

Example Parse

Transition
Right-Arc(obj)

Buffer
[ ]

SynStack
[made]

SynArcs
subj(made, he)
mod(decisions, A(a, few))
obj(made, decisions)

LexStack
[ ]

LexTrees
he, A(a, few), V(made, decisions)

Implementation and Evaluation

- Implementation:
  - Greedy parser trained with averaged perceptron
  - Variation: Implicit Completes, triggered by arc transitions
- Evaluation:
  - English Web Treebank and French Treebank
  - Comparisons with
    1. standard parser with special MWE labels
    2. partial systems: only lexical or only syntactic
    3. pipeline systems: fixed MWE identification + parsing

Main Findings

- Compared to standard parser with extended labels:
  - Joint system significantly better for MWE analysis (+2–3%)
  - Implicit Completes help syntactic analysis (+0.1–1%)
- Compared to partial systems:
  - Lexical structure helps syntactic parsing (+0.5–1%)
  - Syntactic structure does not really help lexical analysis
- Comparison with pipeline system
  - Preprocessing F-MWEs improves MWE analysis (+2–3%)
  - But it leads to a slight drop in syntactic analysis (−0.5–1%)

Conclusion

- What we have learned so far (I think):
  - Using dummy trees or special labels to squeeze MWEs into parsing is suboptimal for both lexical and syntactic analysis
  - Non-fixed MWEs should be parsed as syntactically regular – lexical analysis can be done jointly or in post-processing
  - Fixed MWEs should be parsed as atomic syntactic units – lexical analysis can be done jointly or in pre-processing
- Where do we go from here?
  - Explore novel (multi-dimensional) representations
  - Annotate treebanks using these representations
  - Develop more flexible parsing systems

- Marie Candito and Matthieu Constant. 2014.
  Strategies for contiguous multiword expression analysis and dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 743–753.
- Mathieu Constant and Joakim Nivre. 2016.
  A transition-based system for joint lexical and syntactic analysis. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.
- Gülşen Eryiğit, Tugay Ilbay, and Ozan Arkan Can. 2011.
  Multiword expressions in statistical dependency parsing. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 45–55.
- Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005.
  Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.
- Joakim Nivre and Jens Nilsson. 2004.
  Multiword units in syntactic parsing. In *Proceedings of the Workshop on Methodologies and Evaluation of Multiword Units in Real-World Applications (LREC)*, pages 39–46.
- Joakim Nivre. 2003.
  An efficient algorithm for projective dependency parsing. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.
- Veronika Vincze, János Zsibrita, and István Nagy T. 2013.
  Dependency parsing for identifying hungarian light verb constructions. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 207–215.