

# Dependency Parsing

## lecture 5

Jakub Waszczuk, Kilian Evang

Heinrich Heine Universität

Summer semester 2021

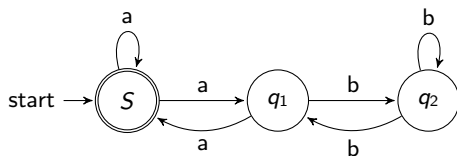
- Transition-based parsing
  - Deterministic, linear time
  - Projective dependencies only

(slides from ESSLII 2007 course by McDonald & Nivre)

- Training a classifier to predict the next transition

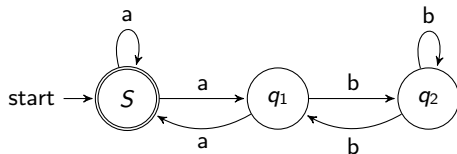
## Transition system

- Simple example—Finite State Automaton:
  - Finite set of atomic states
  - Transitions between states, with input conditions
  - A start state
  - An end state; result: Input is accepted or not

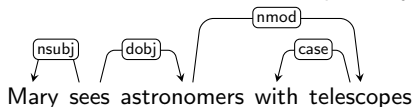


## Transition system

- Simple example—Finite State Automaton:
  - Finite set of atomic states
  - Transitions between states, with input conditions
  - A start state
  - An end state; result: Input is accepted or not



- Dependency parsing:
  - More complex: states have structure (data structures of the parser)
  - Transitions are parser operations
  - Start state: empty data structures, sentence as input
  - Result: transitions define a dependency graph



We will consider a **stack-based** form of transition parsing.

# Stack-Based Transition Systems

- ▶ A **stack-based** configuration for a sentence  $x = w_0, w_1, \dots, w_n$  is a triple  $c = (\sigma, \beta, A)$ , where not all of them!
  1.  $\sigma$  is a stack of tokens  $i \leq m$  (for some  $m \leq n$ ),
  2.  $\beta$  is a buffer of tokens  $j > m$ ,
  3.  $A$  is a set of dependency arcs such that  $G = (\{0, 1, \dots, n\}, A)$  is a dependency graph for  $x$ .
  
- ▶ A **stack-based** transition system is a quadruple  $S = (C, T, c_s, C_t)$ , where
  1.  $C$  is the set of all stack-based configurations, partial
  2.  $c_s(x = w_0, w_1, \dots, w_n) = ([0], [1, \dots, n], \emptyset)$ ,
  3.  $T$  is a set of transitions, each of which is a function  $t : C \rightarrow C$ ,
  4.  $C_t = \{c \in C \mid c = (\sigma, [], A)\}$ .
  
- ▶ Notation: delta | i | j = (delta | i) | j
  - ▶  $\sigma|i$  = stack with top  $i$  ( $|$  left-associative)
  - ▶  $i|\beta$  = buffer with next token  $i$  ( $|$  right-associative)

# Shift-Reduce Dependency Parsing

► Transitions:

► Left-Arc<sub>k</sub>:

$$(\sigma|i,j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$

► Right-Arc<sub>k</sub>:

$$(\sigma|i,j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j, k)\})$$

► Shift:

$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$$

► Preconditions:

► Left-Arc<sub>k</sub>:

$$\neg[i = 0]$$

$$\neg\exists i' \exists k' [(i', i, k') \in A]$$

► Right-Arc<sub>k</sub>:

$$\neg\exists i' \exists k' [(i', j, k') \in A]$$

superfluous

## Example: Shift-Reduce Parsing

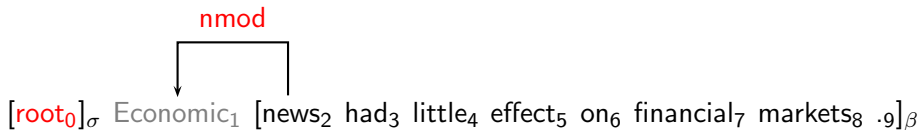
[root<sub>0</sub>]<sub>σ</sub> [Economic<sub>1</sub> news<sub>2</sub> had<sub>3</sub> little<sub>4</sub> effect<sub>5</sub> on<sub>6</sub> financial<sub>7</sub> markets<sub>8</sub> .9]<sub>β</sub>

## Example: Shift-Reduce Parsing

[root<sub>0</sub> Economic<sub>1</sub>]<sub>σ</sub> [news<sub>2</sub> had<sub>3</sub> little<sub>4</sub> effect<sub>5</sub> on<sub>6</sub> financial<sub>7</sub> markets<sub>8</sub> .9]<sub>β</sub>

Shift

# Example: Shift-Reduce Parsing



Left-Arc<sub>*nmod*</sub>

# Example: Shift-Reduce Parsing

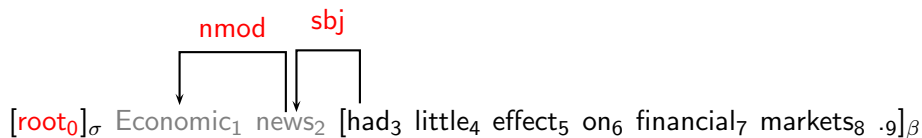
"Economic\_1" is not  
part of the stack

nmod

[root<sub>0</sub> Economic<sub>1</sub> news<sub>2</sub>]<sub>σ</sub> [had<sub>3</sub> little<sub>4</sub> effect<sub>5</sub> on<sub>6</sub> financial<sub>7</sub> markets<sub>8</sub> .9]<sub>β</sub>

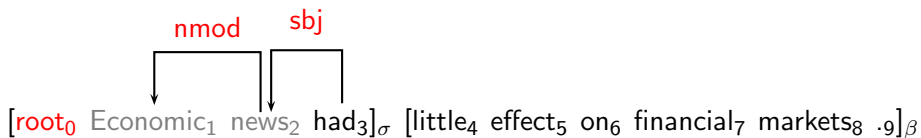
Shift

# Example: Shift-Reduce Parsing



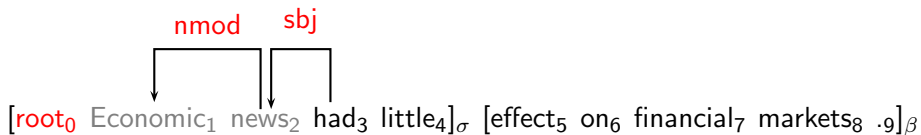
Left-Arc<sub>subj</sub>

# Example: Shift-Reduce Parsing



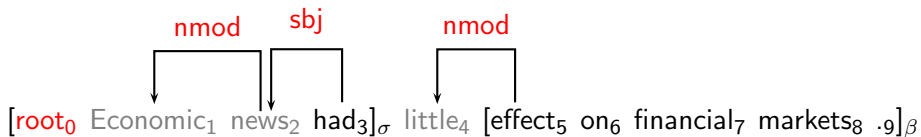
Shift

# Example: Shift-Reduce Parsing



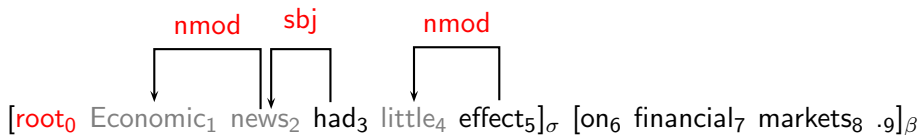
Shift

# Example: Shift-Reduce Parsing



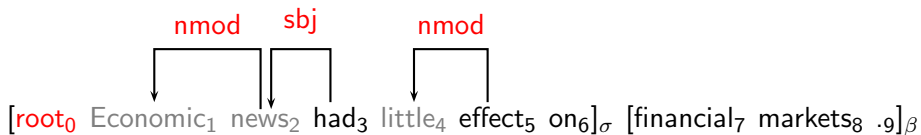
Left-Arc<sub>*nmod*</sub>

# Example: Shift-Reduce Parsing



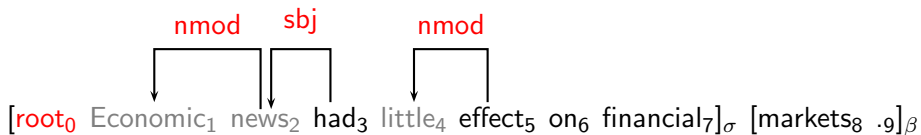
Shift

# Example: Shift-Reduce Parsing



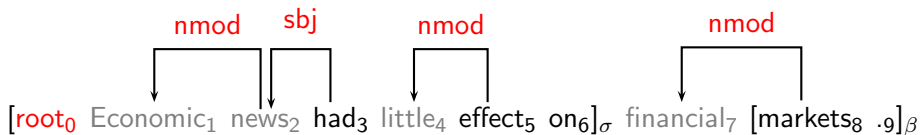
Shift

# Example: Shift-Reduce Parsing



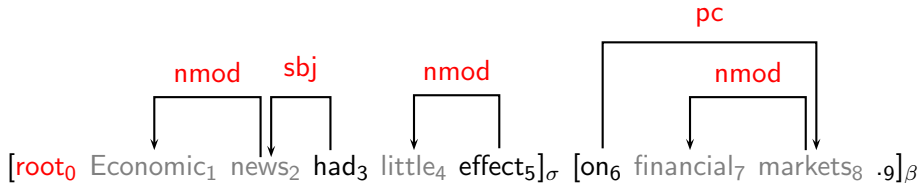
Shift

# Example: Shift-Reduce Parsing



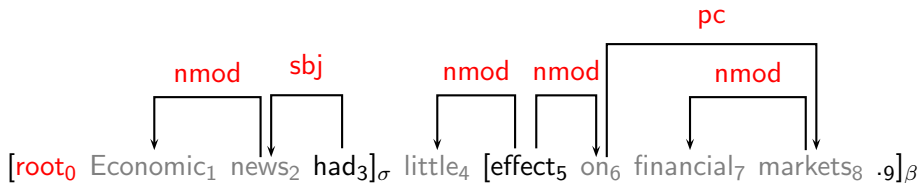
Left-Arc<sub>*nmod*</sub>

# Example: Shift-Reduce Parsing



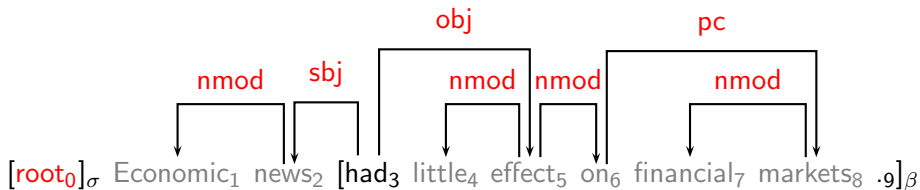
Right-Arc<sub>pc</sub>

# Example: Shift-Reduce Parsing



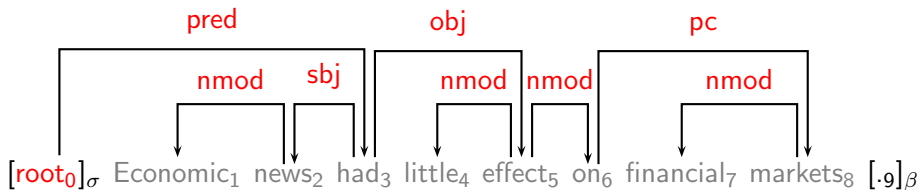
Right-Arc  $nmod$

# Example: Shift-Reduce Parsing



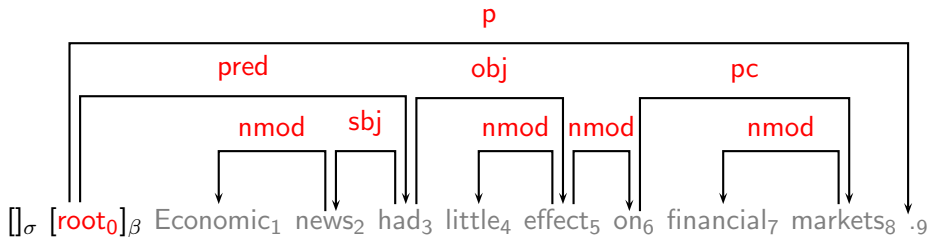
Right-Arc<sub>obj</sub>

# Example: Shift-Reduce Parsing



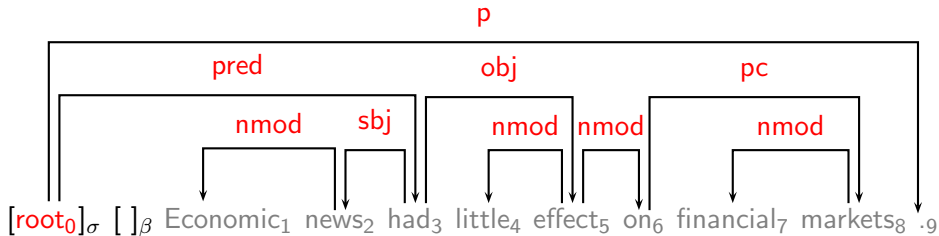
Right-Arc<sub>pred</sub>

# Example: Shift-Reduce Parsing



Right-Arc<sub>p</sub>

# Example: Shift-Reduce Parsing



Shift

# Theoretical Results

- ▶ Complexity:
  - ▶ Deterministic shift-reduce parsing has **time** and **space** complexity  $O(n)$ , where  $n$  is the length of the input sentence.
- ▶ Correctness:
  - ▶ For every transition sequence  $C_{0,m}$ ,  $G_{C_m}$  is a projective dependency forest (**soundness**).
  - ▶ For every projective dependency forest  $G$ , there is a transition sequence  $C_{0,m}$  such that  $G_{C_m} = G$  (**completeness**).
- ▶ Note:
  - ▶ A **dependency forest** is (here) a dependency graph satisfying **Root**, **Single-Head**, and **Acyclicity** (but not **Connectedness**).
  - ▶ A dependency forest  $G = (V, A)$  can be transformed into a **dependency tree** by adding arcs of the form  $(0, i, k)$  (for some  $l_k \in L$ ) for every root  $i \in V$  ( $i \neq 0$ ).

### Definition

A configuration  $c = (\sigma, \beta, A)$

- $\sigma$ : Stack
- $\beta$ : Input buffer
- $A$ : Dependencies already added, of the form: (head, dependent, label)

Parsing algorithm:

- Initial configuration: ( $[\text{ROOT}]$ ,  $[\ ]$ ,  $\emptyset$ )
- Choose a transition, get new configuration
- Repeat, until terminal configuration:  $(\sigma, [\ ], A)$ , s.t.  $A$  determines a dependency graph

Problem for classifier:

- We are at a configuration  $c_i$
- Which transition to  $c_{i+1}$  ?
  - Arc-Left, Arc-Right, Shift?
  - if Arc-Left or Arc-Right, which dependency label?

it is not checked that  $A$  determines a dependency graph

## Probabilistic grammar vs training on parser decisions

In probabilistic grammars (e.g., PCFG):

- complete derivations have a probability
- ... based on scores/probabilities of their productions
- parser can arrive at same (analysis, probability) with different strategies: bottom-up, top-down, left-corner, etc.
- using probabilistic model, can find most likely (sub)tree, n-best,  $\Pr(S)$

In transition-based parsing:

- train on parser **operations**, not its output
- a history-based approach
- decisions can be based on any information the parser has
- only need to know the single most likely action at each point:
  - easier to estimate (operations are considered independently)
  - parsing in linear time

not entirely true these days?

## Training a classifier

- Given dependency trees to train on ...
- Extract a sequence of configurations and transitions for each tree
- For each configuration, extract features according to a template
- Train classifier to predict transition from features

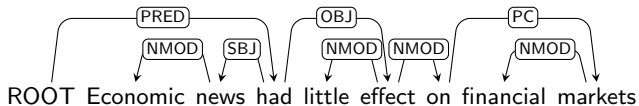
## Extracting transitions from dependency trees

- Oracle: function from configurations to transitions given a dependency graph  $G_d = (V_d, A_d)$  from the training set
- For the arc-standard transition system: top of the buffer

$$o(c = (\sigma, \beta, A)) = \begin{cases} \text{LEFT-ARC}_r & \text{if } (\beta[0], \sigma[0], r) \in A_d \\ \text{RIGHT-ARC}_r & \text{if } (\sigma[0], \beta[0], r) \in A_d \text{ and, for all } w, r', \\ & \text{if } (\beta[0], w, r') \in A_d \text{ then } (\beta[0], w, r') \in A \\ \text{SHIFT}_r & \text{otherwise} \end{cases}$$

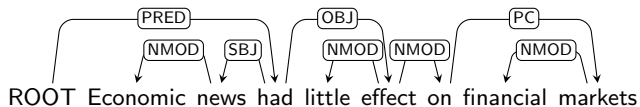
- Now we can construct a sequence of transitions for a sentence  $S_d$  and dependency graph  $G_d = (V_d, A_d)$  in the training set:
  - Start with  $c_0 = ([\text{ROOT}], S_d, \emptyset)$
  - Apply transition from oracle to get next configuration
  - ... until terminal configuration  $(\sigma, [], A_d)$  is reached

## Example: getting configurations from dependency graph



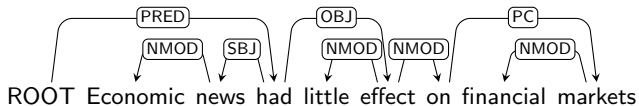
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$

## Example: getting configurations from dependency graph



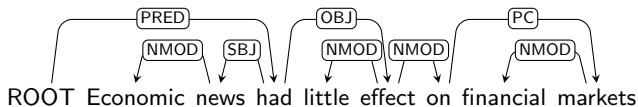
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$

## Example: getting configurations from dependency graph



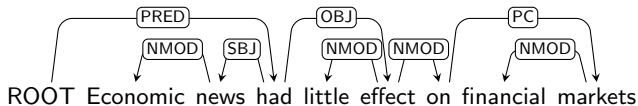
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)

## Example: getting configurations from dependency graph



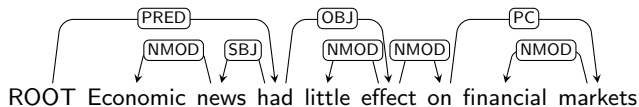
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+ (news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	

## Example: getting configurations from dependency graph



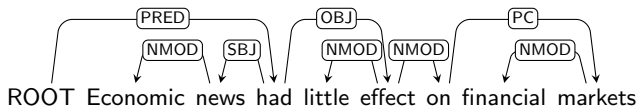
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)

## Example: getting configurations from dependency graph



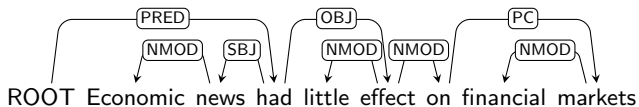
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	

## Example: getting configurations from dependency graph



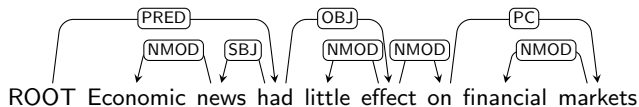
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	

## Example: getting configurations from dependency graph



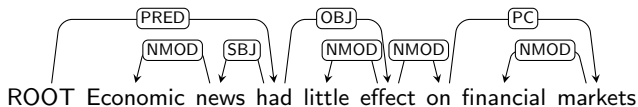
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)

## Example: getting configurations from dependency graph



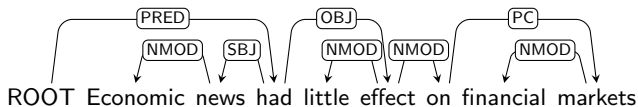
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)
SH	[ROOT, had, effect],	[on, ...],	

## Example: getting configurations from dependency graph



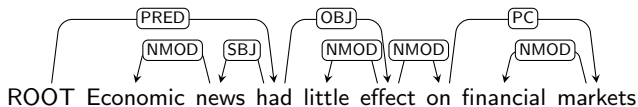
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)
SH	[ROOT, had, effect],	[on, ...],	
SH	[..., on],	[financial, ...],	

## Example: getting configurations from dependency graph



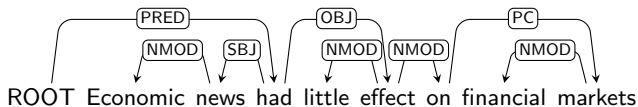
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)
SH	[ROOT, had, effect],	[on, ...],	
SH	[..., on],	[financial, ...],	
SH	[..., on, financial],	[markets, ...],	

## Example: getting configurations from dependency graph



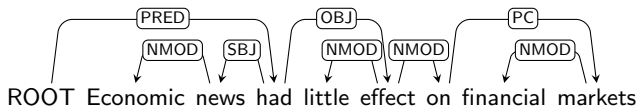
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)
SH	[ROOT, had, effect],	[on, ...],	
SH	[..., on],	[financial, ...],	
SH	[..., on, financial],	[markets, ...],	
LA <sub>NMOD</sub>	[..., on],	[markets, ...]	+(markets, financial, NMOD)

## Example: getting configurations from dependency graph



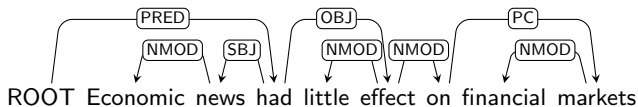
Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)
SH	[ROOT, had, effect],	[on, ...],	
SH	[..., on],	[financial, ...],	
SH	[..., on, financial],	[markets, ...],	
LA <sub>NMOD</sub>	[..., on],	[markets, ...]	+(markets, financial, NMOD)
RA <sub>PC</sub>	[ROOT, had, effect]	[on, ...]	+(on, markets, PC)

## Example: getting configurations from dependency graph



Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)
SH	[ROOT, had, effect],	[on, ...],	
SH	[..., on],	[financial, ...],	
SH	[..., on, financial],	[markets, ...],	
LA <sub>NMOD</sub>	[..., on],	[markets, ...]	+(markets, financial, NMOD)
RA <sub>PC</sub>	[ROOT, had, effect]	[on, ...]	+(on, markets, PC)
RA <sub>NMOD</sub>	[ROOT, had]	[effect, ...]	+(effect, on, NMOD)

## Example: getting configurations from dependency graph



Transition	Stack	Input	Dependency
	[ROOT],	[Economic, ...],	$\emptyset$
SH	[ROOT, Economic],	[news, ...],	$\emptyset$
LA <sub>NMOD</sub>	[ROOT],	[news, ...],	+(news, Economic, NMOD)
SH	[ROOT, news],	[had, ...],	
LA <sub>SBJ</sub>	[ROOT],	[had, ...],	+(had, news, SBJ)
SH	[ROOT, had],	[little, ...]	
SH	[ROOT, had, little],	[effect, ...]	
LA <sub>NMOD</sub>	[ROOT, had],	[effect, ...],	+(effect, little, NMOD)
SH	[ROOT, had, effect],	[on, ...],	
SH	[..., on],	[financial, ...],	
SH	[..., on, financial],	[markets, ...],	
LA <sub>NMOD</sub>	[..., on],	[markets, ...]	+(markets, financial, NMOD)
RA <sub>PC</sub>	[ROOT, had, effect]	[on, ...]	+(on, markets, PC)
RA <sub>NMOD</sub>	[ROOT, had]	[effect, ...]	+(effect, on, NMOD)
...			

Feature template:

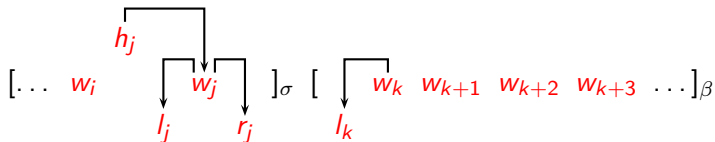
- The top word on the stack
- The first word in the input
- Linear context: the neighboring words in the sentence
- Structural context: parent or dependents of a word

Instead of words, features can also be:

- lemmas
- part-of-speech tags
- morphological features
- dependency labels

# A Typical Model [Nivre et al. 2006]

These days feature templates tend to be more minimalistic



FORM		+	+		+	+		
LEMMA			+		+			
CPOS			+		+			
POS	+		+		+	+	+	+
FEATS			+		+			
DEPREL			+	+	+		+	

## Extracting features from a configuration

Transition	Stack	Input
SH	[ROOT, had, effect]	[on, financial, markets]

Feature template:

- the word on top of the stack
- the POS tags of the two words on top of the stack
- the first word in the input buffer
- the POS tags of the first two words in the input buffer

Features: (effect, VERB, NOUN, on, PREP, ADJ)

Data point for classifier: (effect, VERB, NOUN, on, PREP, ADJ)  $\Rightarrow$  SH

## Summary:

- Shift-reduce transition-based parsing for projective dependencies
- Training a classifier to predict transitions

## Next:

- Arc-eager parsing
- Non-projective parsing
- Dynamic oracles

T H E

E N D

## References and Further Reading

- ▶ Giuseppe Attardi. 2006.  
Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.
- ▶ Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2004.  
Deterministic dependency structure analyzer for Chinese. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP)*, pages 500–508.
- ▶ Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2005.  
Machine learning-based dependency analyzer for Chinese. In *Proceedings of International Conference on Chinese Computing (ICCC)*, pages 66–73.
- ▶ Hideki Isozaki, Hideto Kazawa, and Tsutomu Hirao. 2004.  
A deterministic word dependency analyzer enhanced with preference learning. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 275–281.
- ▶ Taku Kudo and Yuji Matsumoto. 2002.  
Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pages 63–69.
- ▶ Joakim Nivre and Jens Nilsson. 2005.

Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

- ▶ Joakim Nivre and Mario Scholz. 2004.  
Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 64–70.
- ▶ Joakim Nivre, Johan Hall, and Jens Nilsson. 2004.  
Memory-based dependency parsing. In Hwee Tou Ng and Ellen Riloff, editors, *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56.
- ▶ Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiugit, and Svetoslav Marinov. 2006.  
Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.
- ▶ 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.
- ▶ Hiroyasu Yamada and Yuji Matsumoto. 2003.

Statistical dependency analysis with support vector machines. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.