



# Dependency Parsing

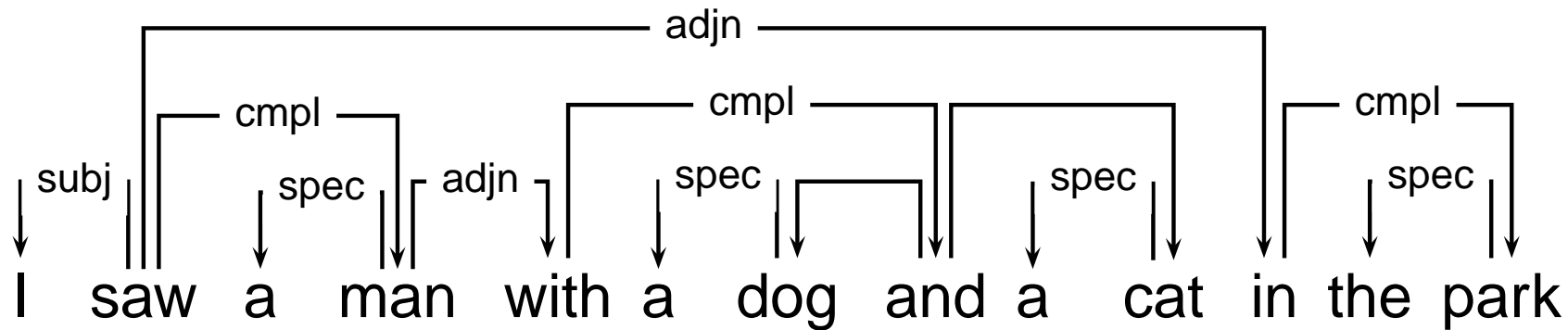
Erhard W. Hinrichs and Sandra Kübler

SfS-CL

Eberhard-Karls-Universität Tübingen



- dependency annotation:



- important: relations between word pairs, no recursive structure
- parsing can be formulated as classification task: decide whether two words share a dependency relation
- possible problem: based on words  $\Rightarrow$  sparse data



- uses Support Vector Machines
- parsing in several rounds: deterministic bottom-up
- looks at pairs of words
- 3 actions: shift, left, right



- uses Support Vector Machines
- parsing in several rounds: deterministic bottom-up
- looks at pairs of words
- 3 actions: shift, left, right
- shift: shifts focus to next word pair

I    

<b>saw</b>	<b>a</b>
<b>VBD</b>	<b>DT</b>

    girl    with     $\Rightarrow$     I    saw    

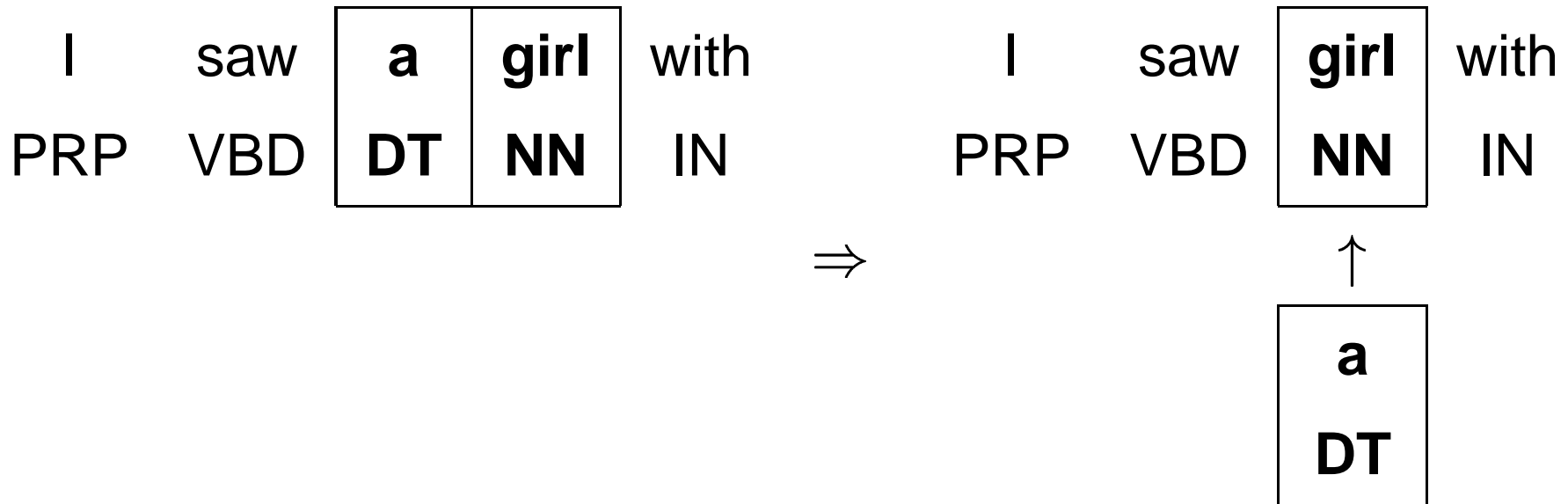
<b>a</b>	<b>girl</b>
<b>DT</b>	<b>NN</b>

    with

PRP    VBD    NN    IN    PRP    VBD    DT    NN    IN

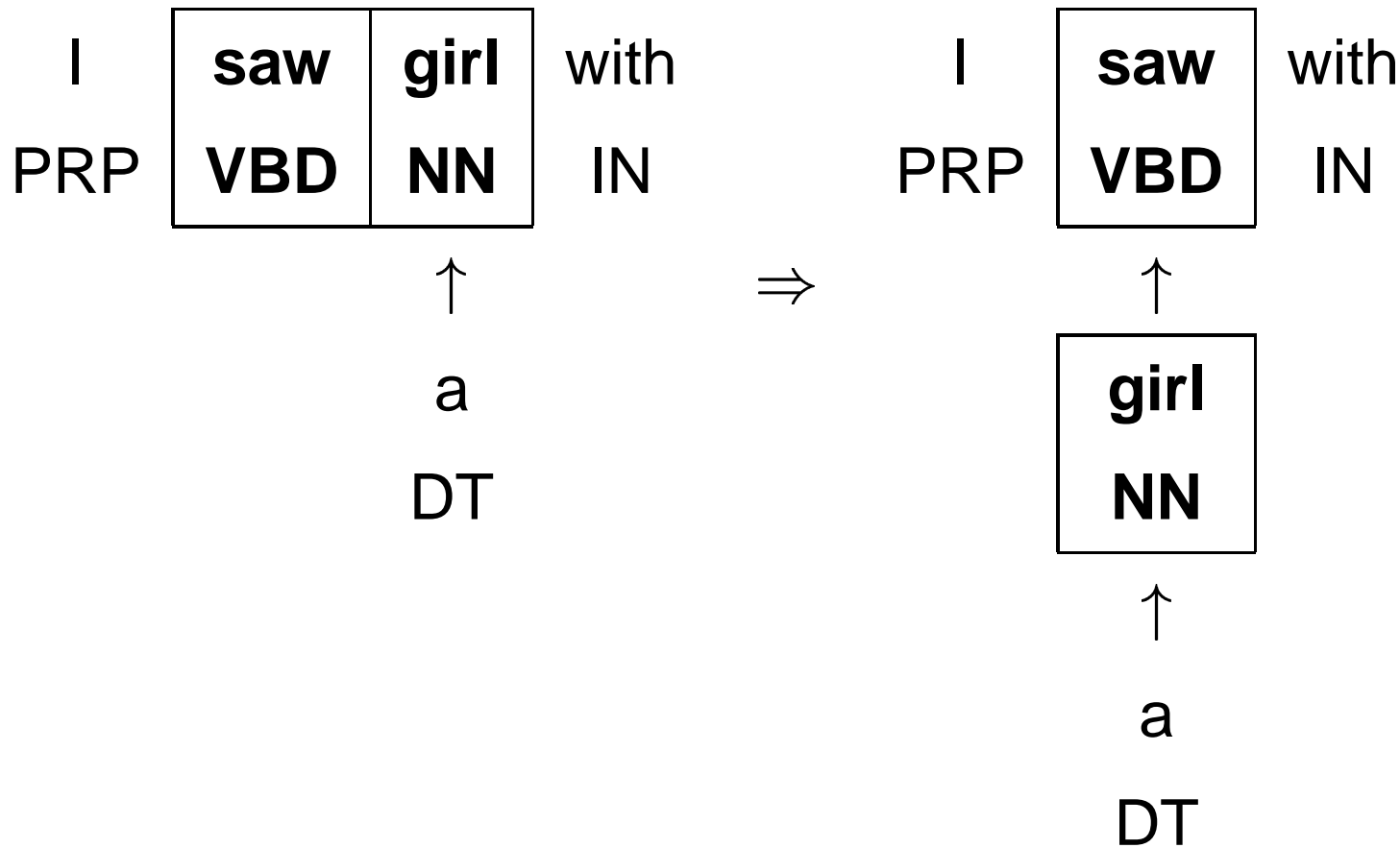


- left: decides that the left word depends on the right one





- right: decides that the right word depends on the left one





- go through each pair of words, decide which action to take
- if a relation was detected in a pass, do another pass
- e.g. the little girl  
first pass: relation between little and girl  
second pass relation between the and girl
- decision on action depends on word pair and context
- SVMs decide which action based on features
- SVMs: binary classifiers  $\Rightarrow$  3 classifiers: shift vs. left, shift vs. right, left vs. right; majority vote



- $n$  features =  $n$ -dimensional space
- linearly separable problem
- slack variable
- support vectors
- not linearly separable: transpose to higher dimension
- kernel



# Context Features

- POS tag (of context word)
- word string
- POS tag of left child node
- word of left child node
- POS tag of right child node
- word of right child node
- what about pair itself???



- POS tag (of context word)
- word string
- POS tag of left child node
- word of left child node
- POS tag of right child node
- word of right child node
- what about pair itself???
- for each POS tag of left focus word, separate classifier pairs



- measures: dependency accuracy, root accuracy, sentence rate (percentage of correctly parsed sentences)
- dependency accuracy: 90.3%, root accuracy: 91.6%, sentence rate: 37.9%
- all context features help
- best context size (wrt. dep./root accuracy): 2 left, 4 right
- best context size (wrt. sentence rate): 3 left, 3 right
- good results with small training set



- use memory-based learning algorithm
- first approach to parse labeled dependencies
- parse Swedish treebank
- deterministic parsing algorithm: similar to shift-reduce parsing
- actions: shift, reduce, left-arc, right-arc
- similar to Yamada and Matsumoto, but needs only one pass through sentence



## NP detection



## NP detection

classification task: decide for each word: NP or not  
instance: word + context



## NP detection

classification task: decide for each word: NP or not  
instance: word + context

instance base (= training data):

prev. w.	word	next w.	prev. POS	POS	next POS	class
will	book	two	md	vb	cd	no-NP
book	two	flights	vb	cd	nns	NP
two	flights	for	cd	nns	in	NP



## NP detection

classification task: decide for each word: NP or not  
instance: word + context

instance base (= training data):

prev. w.	word	next w.	prev. POS	POS	next POS	class
will	book	two	md	vb	cd	no-NP
book	two	flights	vb	cd	nns	NP
two	flights	for	cd	nns	in	NP

test instance:

prev. w.	word	next w.	prev. POS	POS	next POS	class
two	cars	for	cd	nns	in	???



- data structures: input + stack



- data structures: input + stack
- shift: move next word onto stack



- data structures: input + stack
- shift: move next word onto stack
- reduce: pop uppermost word from stack



- data structures: input + stack
- shift: move next word onto stack
- reduce: pop uppermost word from stack
- left-arc: word  $x$  on stack, word  $y$  next word in input  
add new dependency from word  $y$  to word  $x$ ,  
pop  $x$  from stack



- data structures: input + stack
- shift: move next word onto stack
- reduce: pop uppermost word from stack
- left-arc: word  $x$  on stack, word  $y$  next word in input  
add new dependency from word  $y$  to word  $x$ ,  
pop  $x$  from stack
- right-arc: word  $x$  on stack, word  $y$  next word in input  
add new dependency from word  $x$  to word  $y$ ,  
shift  $y$  onto stack



- actions are subject to certain conditions: unique label, single head, reduce can only pop words that have a head
- next action decided by memory-based learning
- decision based on features concerning the word on top of the stack, the next input word, and the following input word
- word on stack: word form, POS tag, label of its dependency to head, dependency to leftmost dependent, dependency to rightmost dependent
- next input word: word form, POS tag, dependency to leftmost dependent
- following input word: POS tag



	labeled acc.	unlabeled acc.
non-lexicalized	76.5 %	82.9 %
lexicalized	81.7 %	85.7 %

- lexical information important
- most difficult category: adverbial modifiers
- approach can also cover crossing (non-projective) dependencies (Nivre and Nilsson 2005)