

Dynamic Programming for Linear Time Incremental Parsing

Liang Huang

Information Sciences Institute
University of Southern California

Kenji Sagae

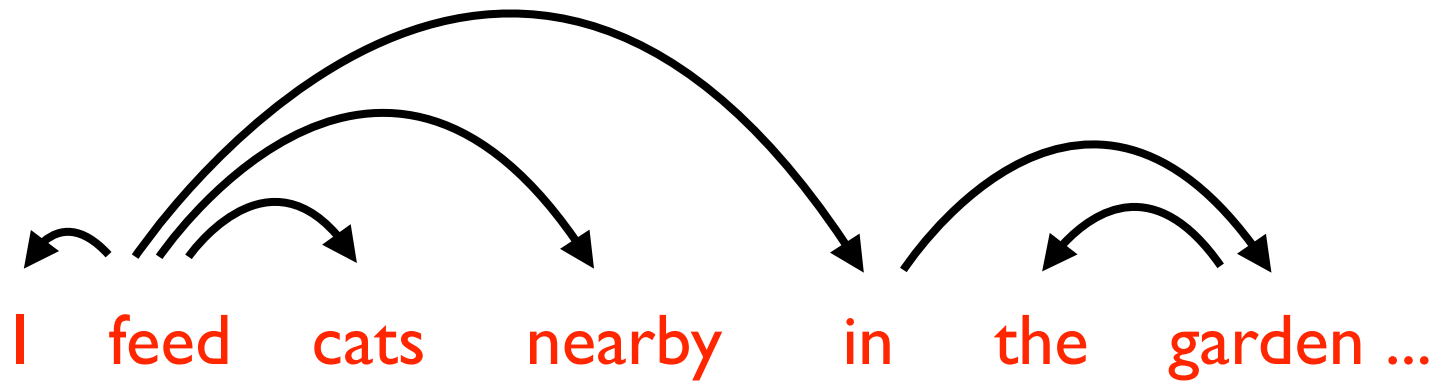
Institute for Creative Technologies
University of Southern California



ACL 2010, Uppsala, Sweden, July 2010 (slightly expanded)

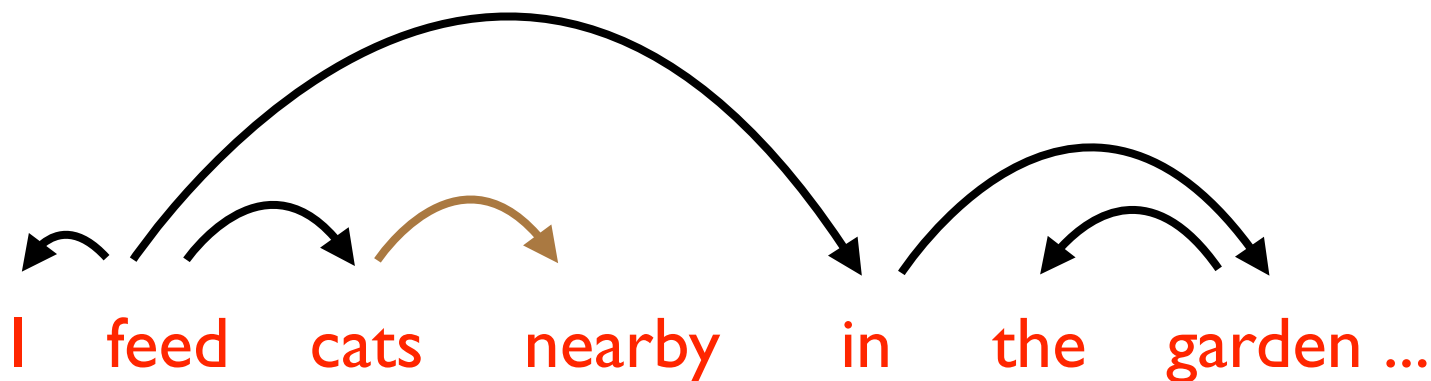


Ambiguities in Parsing



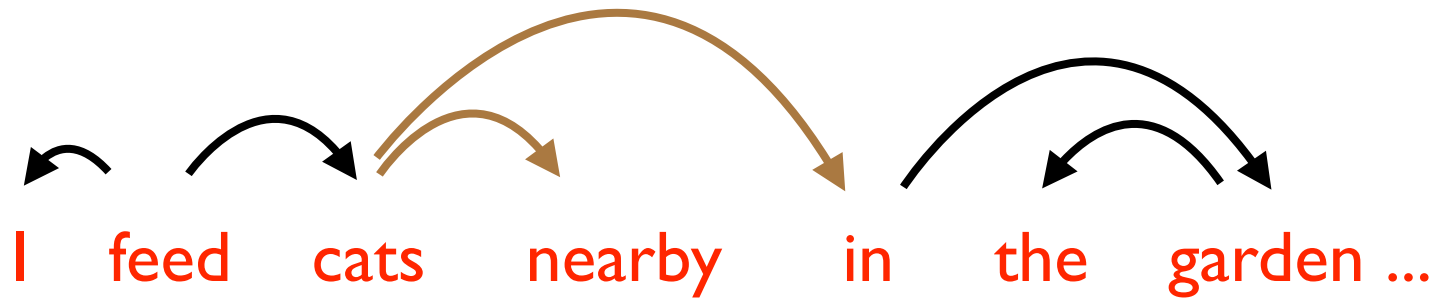
- let's focus on dependency structures for simplicity
- ambiguous attachments of **nearby** and **in**
- ambiguity explodes **exponentially** with sentence length
- must design efficient (polynomial) search algorithm
 - typically using dynamic programming (DP); e.g. CKY

Ambiguities in Parsing



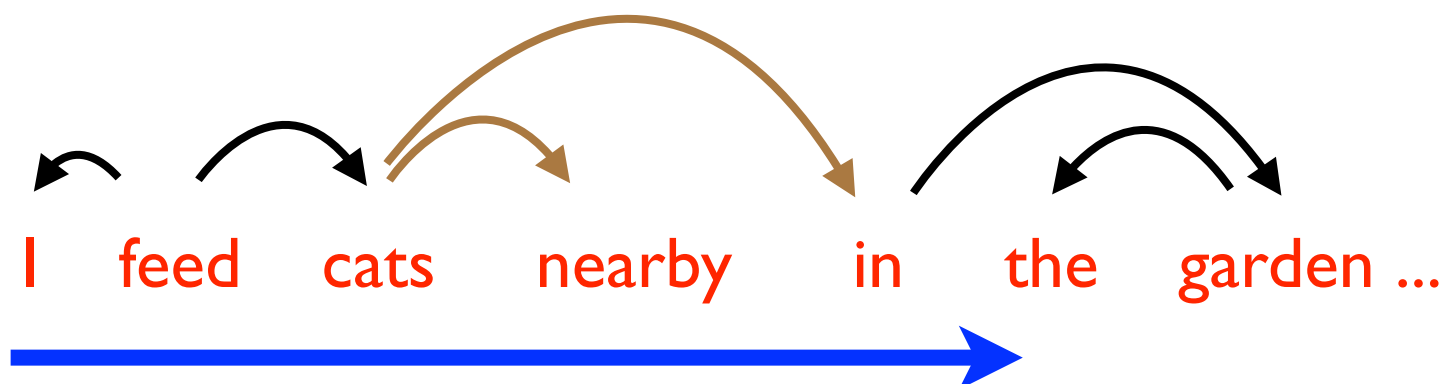
- let's focus on dependency structures for simplicity
- ambiguous attachments of **nearby** and **in**
- ambiguity explodes **exponentially** with sentence length
- must design efficient (polynomial) search algorithm
 - typically using dynamic programming (DP); e.g. CKY

Ambiguities in Parsing



- let's focus on dependency structures for simplicity
- ambiguous attachments of **nearby** and **in**
- ambiguity explodes **exponentially** with sentence length
- must design efficient (polynomial) search algorithm
 - typically using dynamic programming (DP); e.g. CKY

Ambiguities in Parsing



- let's focus on dependency structures for simplicity
- ambiguous attachments of **nearby** and **in**
- ambiguity explodes **exponentially** with sentence length
- must design efficient (polynomial) search algorithm
 - typically using dynamic programming (DP); e.g. CKY

But full DP is too slow...

I feed cats nearby in the garden ...



- full DP (like CKY) is **too slow (cubic-time)**
- while human parsing is **fast & incremental (linear-time)**

But full DP is too slow...

I feed cats nearby in the garden ...



- full DP (like CKY) is **too slow (cubic-time)**
- while human parsing is **fast & incremental (linear-time)**
- how about incremental parsing then?
 - yes, but only with greedy search (accuracy suffers)
 - explores tiny fraction of trees (even w/ beam search)





But full DP is too slow...

I feed cats nearby in the garden ...



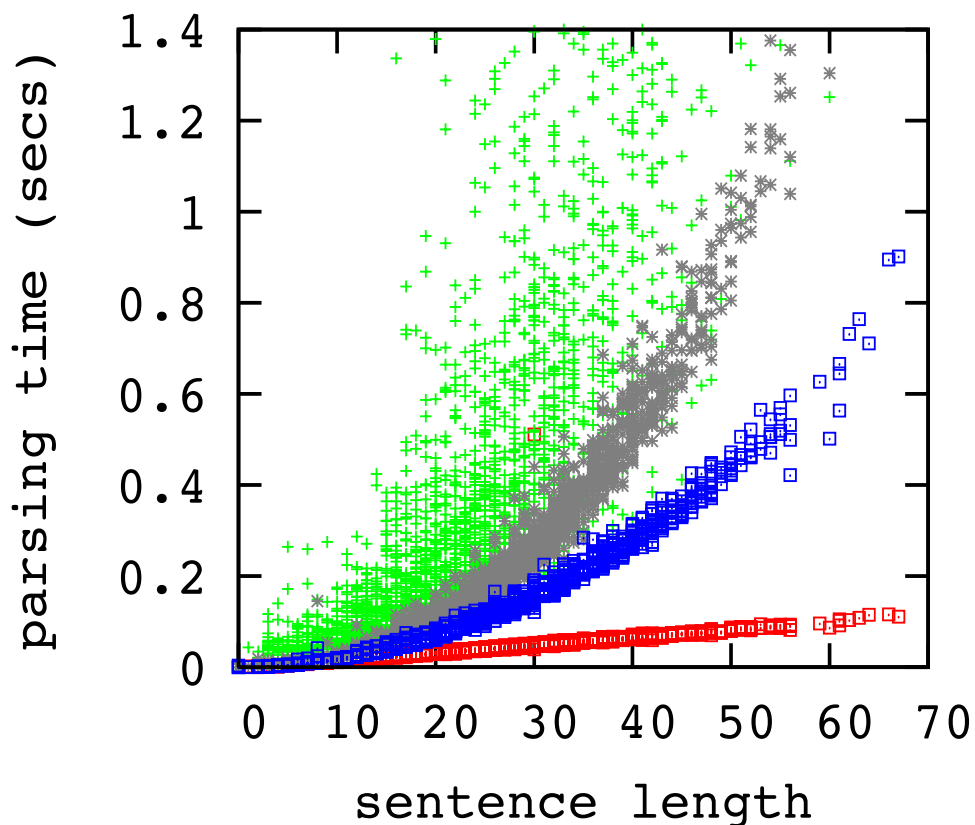
- full DP (like CKY) is **too slow (cubic-time)**
- while human parsing is **fast & incremental (linear-time)**
- how about incremental parsing then?
 - yes, but only with greedy search (accuracy suffers)
 - explores tiny fraction of trees (even w/ beam search)
- can we combine the merits of both approaches?
 - a fast, incremental parser with dynamic programming?
 - explores **exponentially** many trees in **linear-time**?

Linear-Time Incremental DP

greedy search 	incremental parsing (e.g. shift-reduce) (Nivre 04; Collins/Roark 04; ...)	
principled search 	this work: fast shift-reduce parsing with dynamic programming	full DP (e.g. CKY) (Eisner 96; Collins 99; ...)
	fast  (linear-time)	slow  (cubic-time)

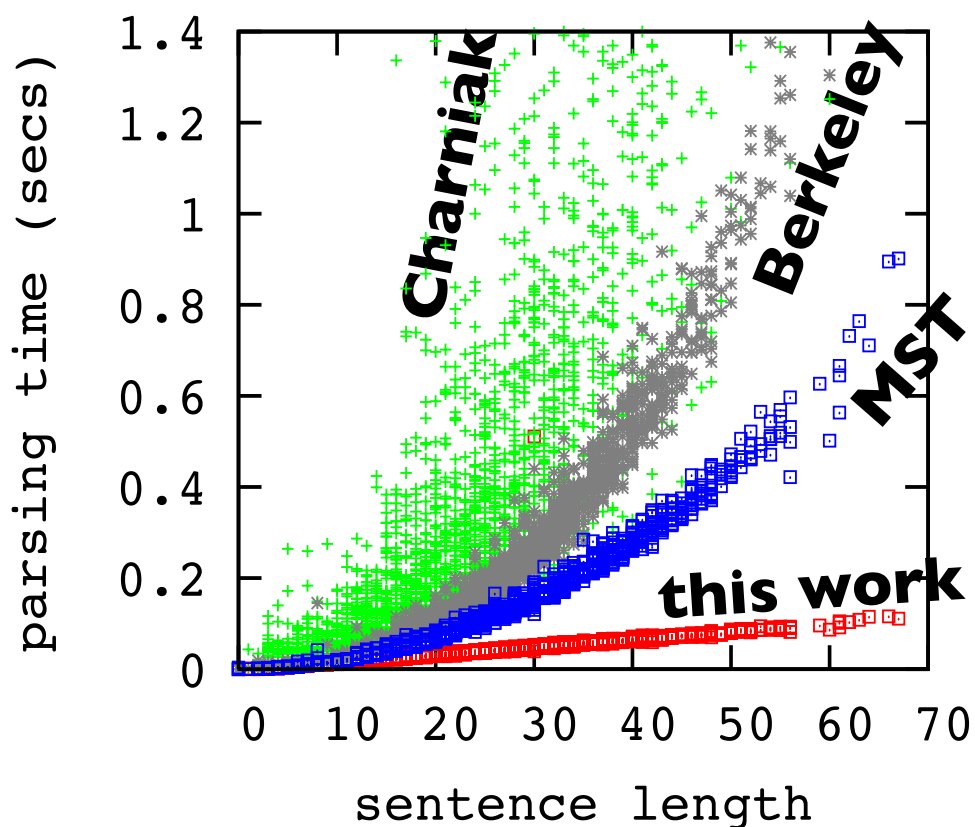
Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores *exponentially* many trees (and outputs forest)



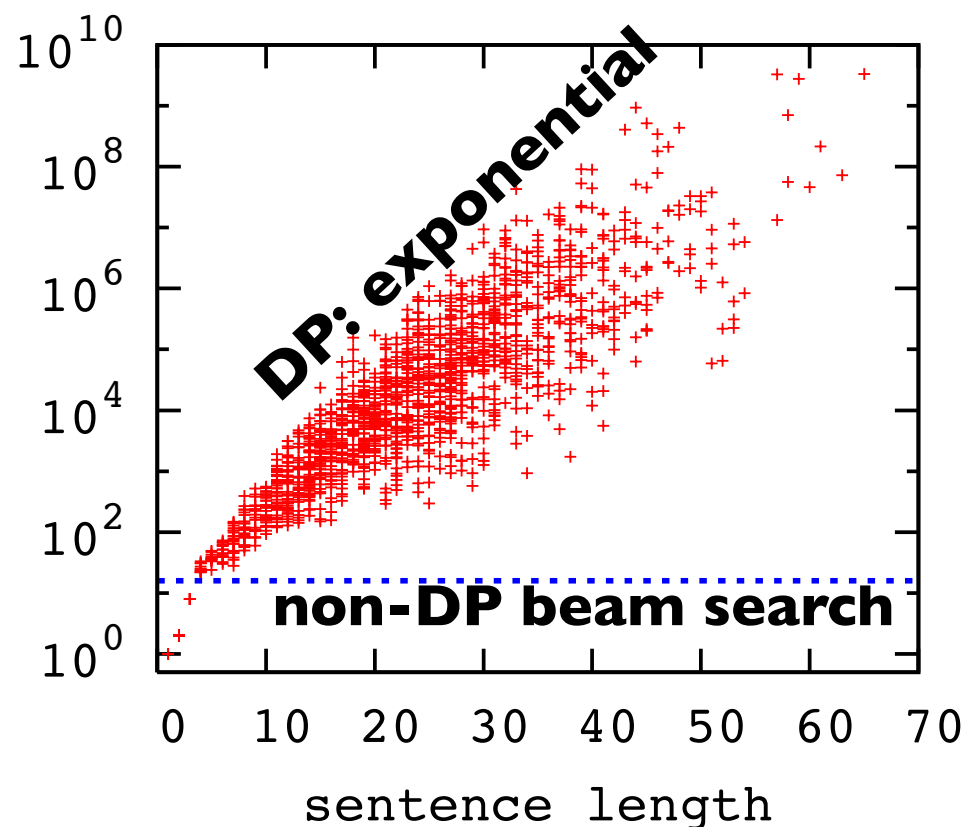
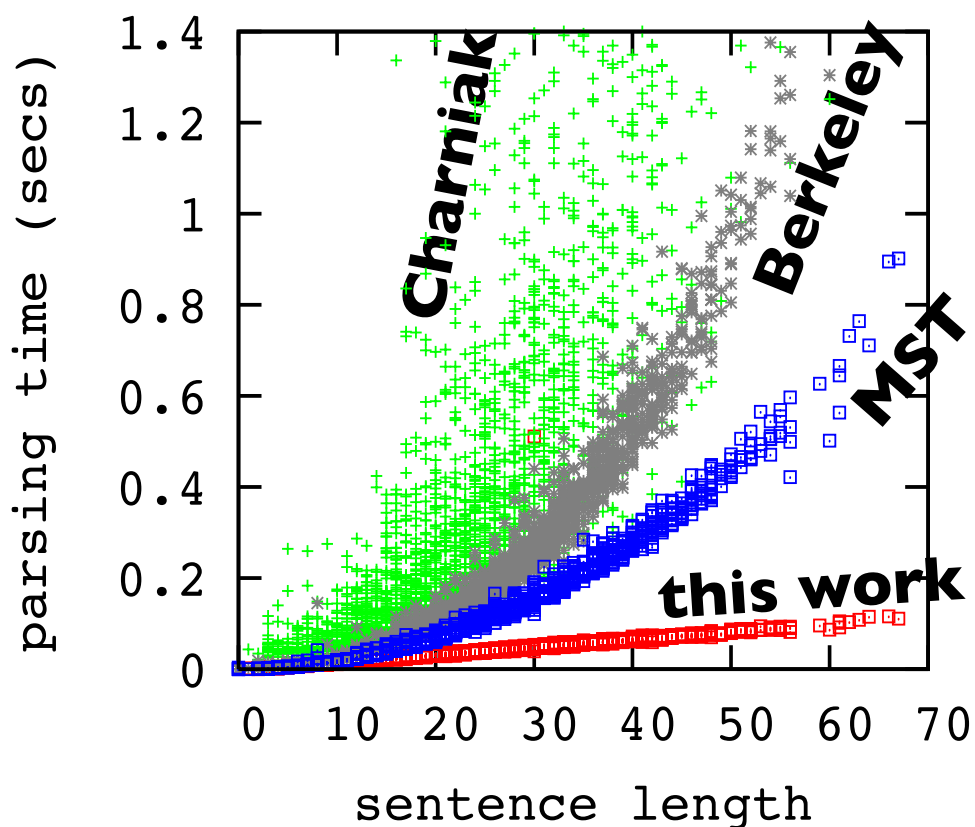
Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores *exponentially* many trees (and outputs forest)



Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores *exponentially* many trees (and outputs forest)



Outline

- Motivation
- Incremental (Shift-Reduce) Parsing
- Dynamic Programming for Incremental Parsing
- Experiments

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-	□	I feed cats ...

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue	
0	-		<table border="1"><tr><td>I feed cats ...</td></tr></table>	I feed cats ...
I feed cats ...				
1	shift		<table border="1"><tr><td>feed cats nearby ...</td></tr></table>	feed cats nearby ...
feed cats nearby ...				

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...
2	shift	I feed	cats nearby in ...

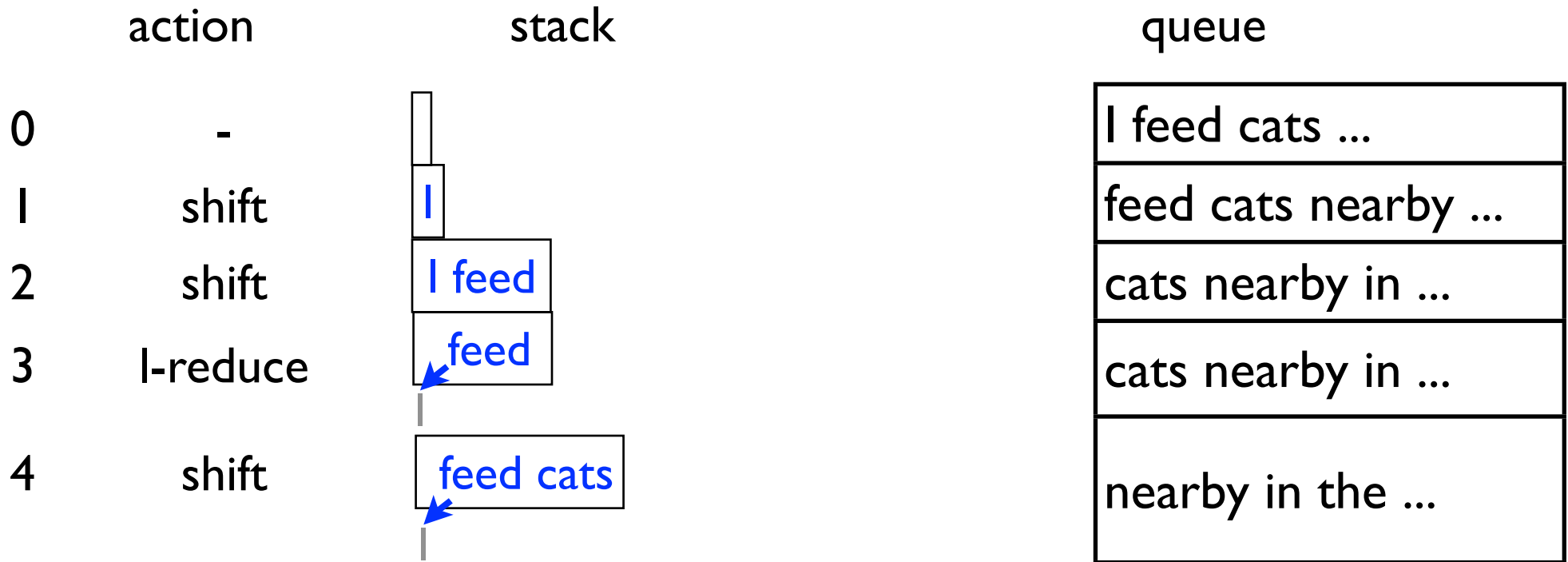
Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift		feed cats nearby ...
2	shift		cats nearby in ...
3	I-reduce		cats nearby in ...

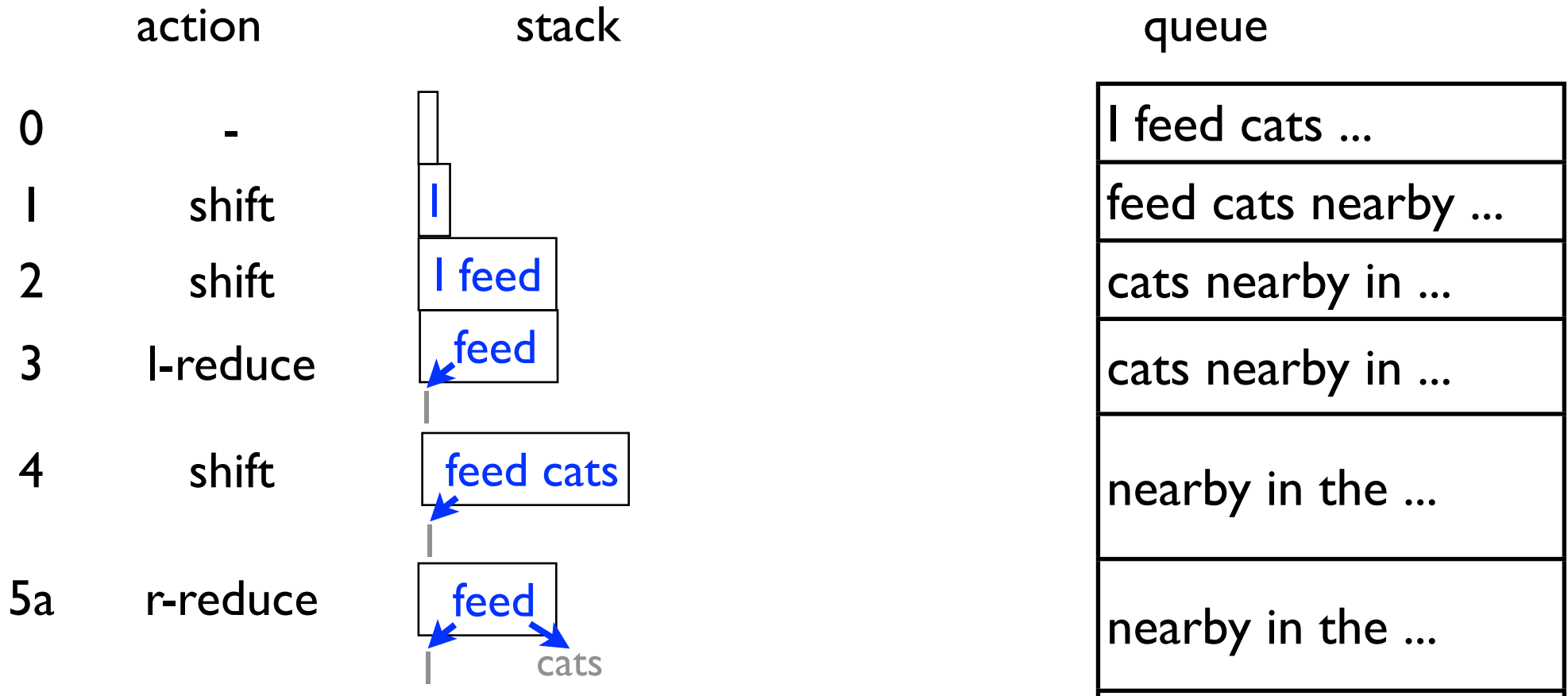
Shift-Reduce Parsing

I feed cats nearby in the garden.



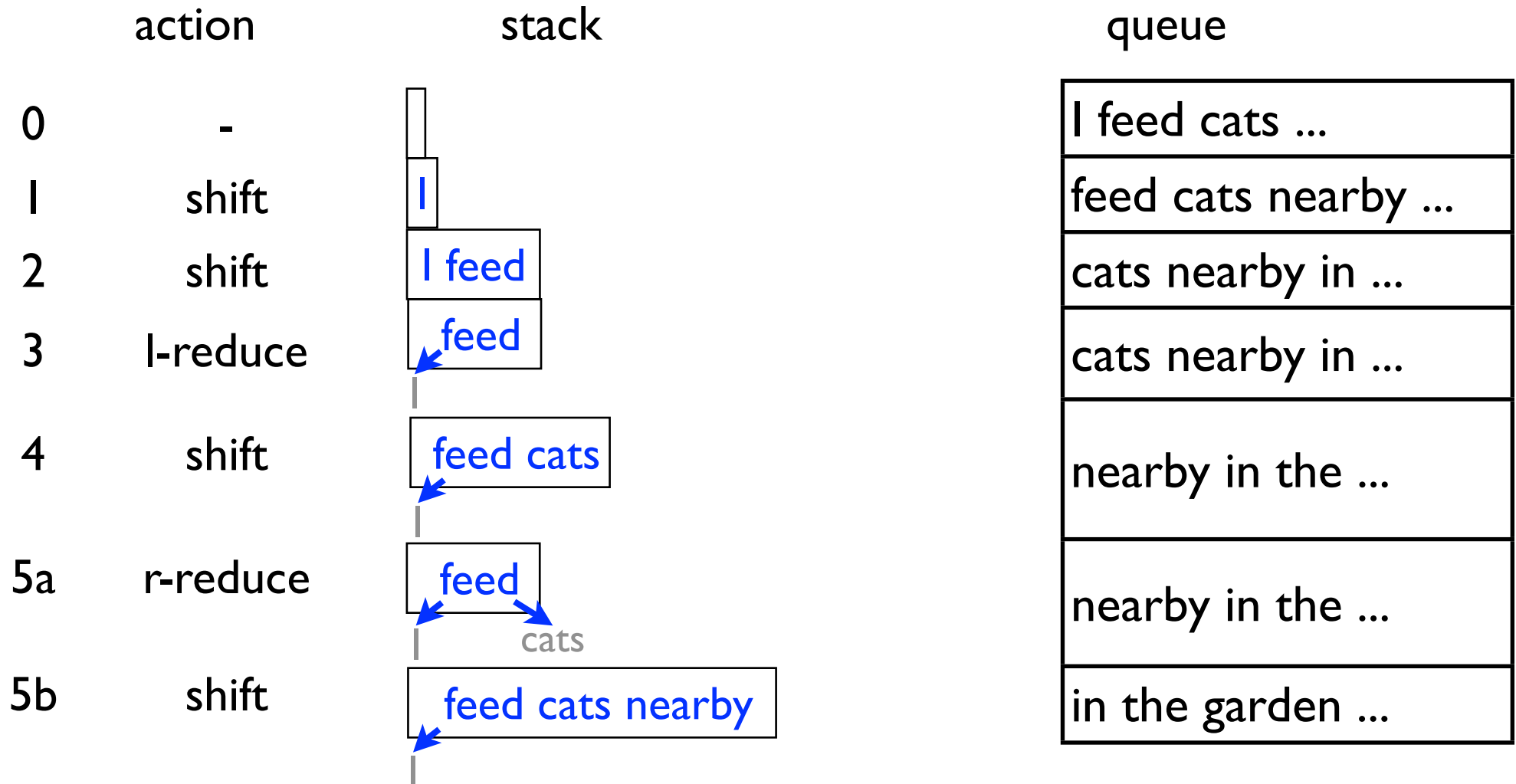
Shift-Reduce Parsing

I feed cats nearby in the garden.



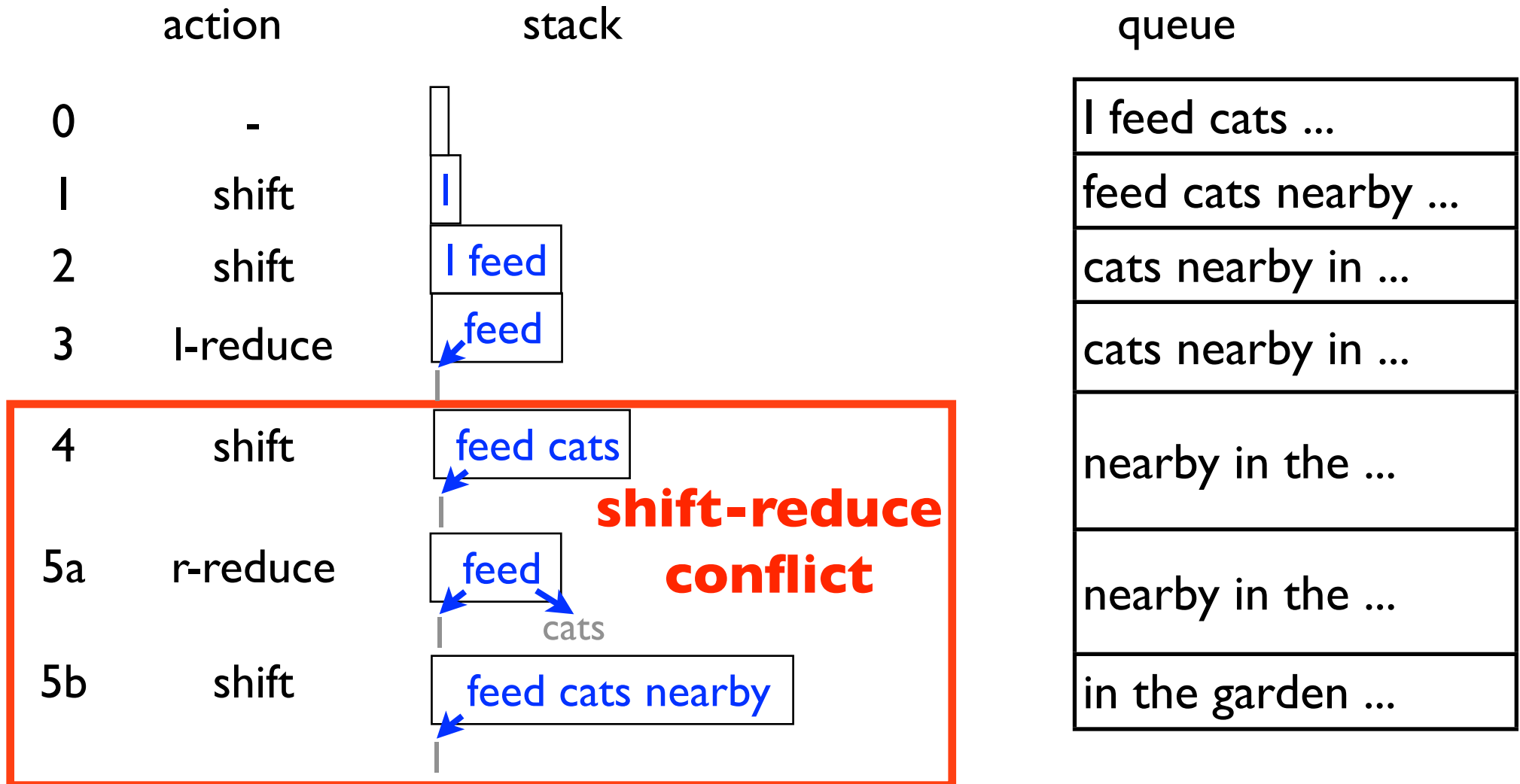
Shift-Reduce Parsing

I feed cats nearby in the garden.

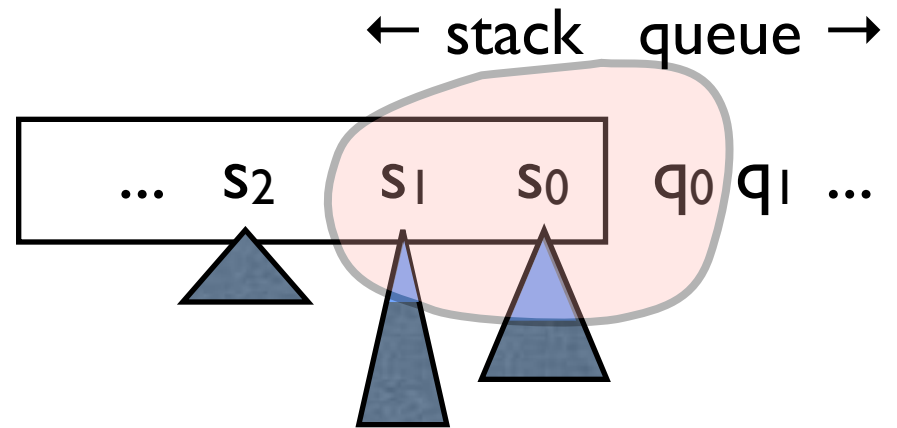
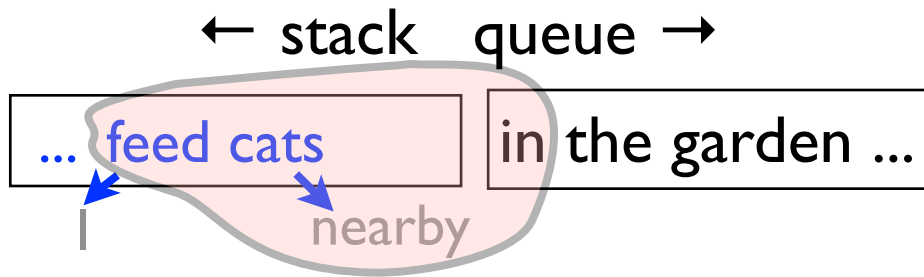


Shift-Reduce Parsing

I feed cats nearby in the garden.



Choosing Parser Actions



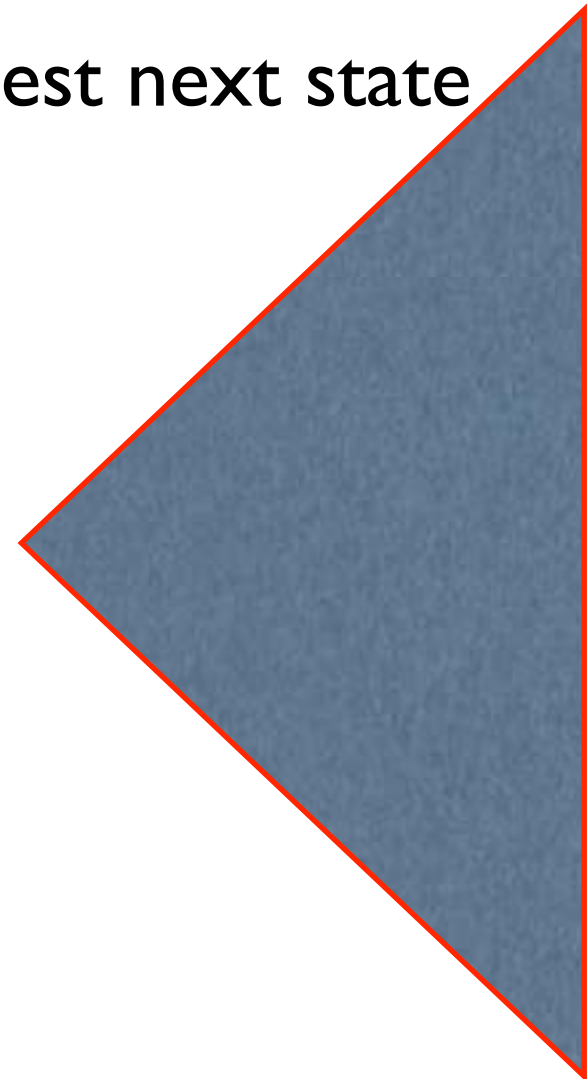
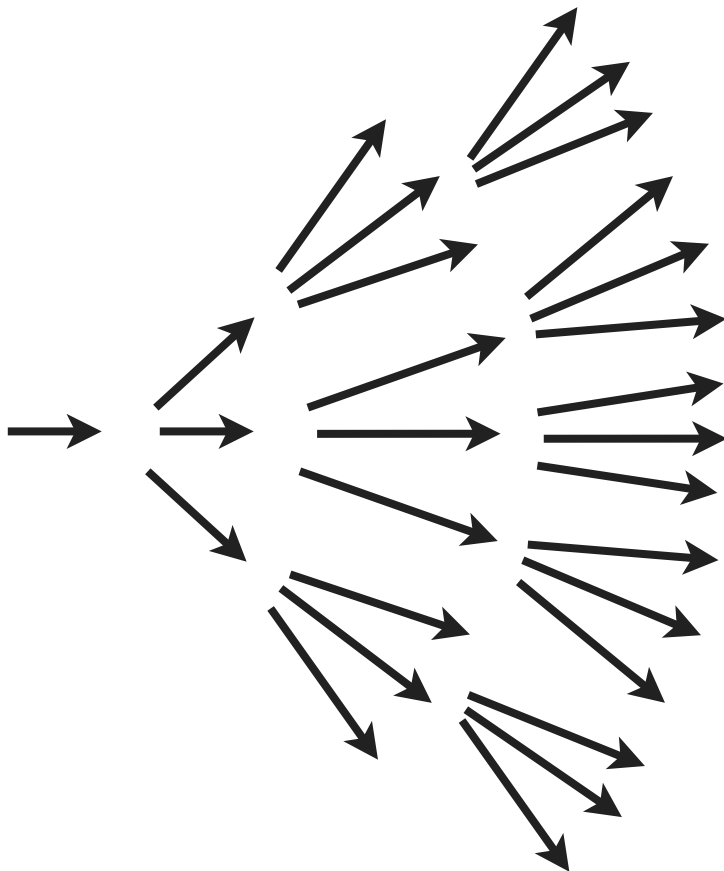
features:

$(s_0.w, s_0.rc, q_0, \dots) = (\text{cats}, \text{nearby}, \text{in}, \dots)$

- score each action using features \mathbf{f} and weights \mathbf{w}
- features are drawn from a *local window*
 - **abstraction** (or **signature**) of a state -- this inspires DP!
- weights trained by structured perceptron (Collins 02)

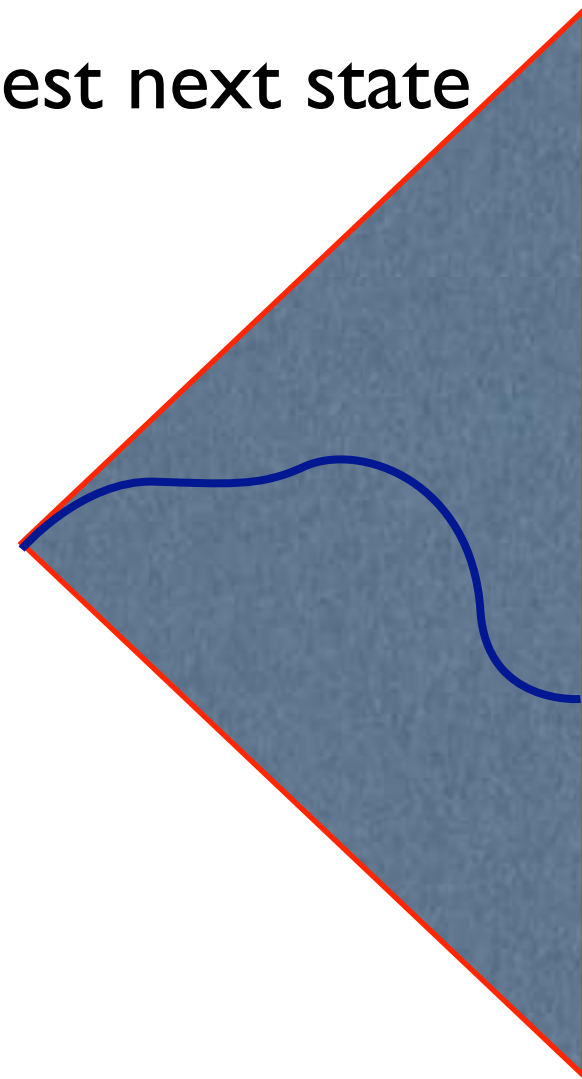
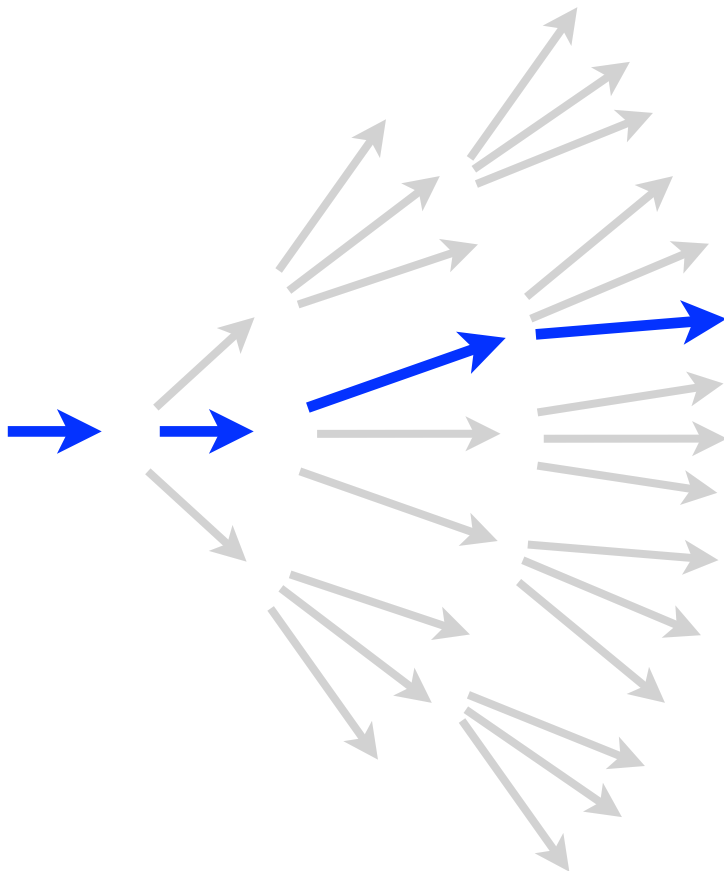
Greedy Search

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
 - search space *should* be exponential
- greedy search: always pick the best next state



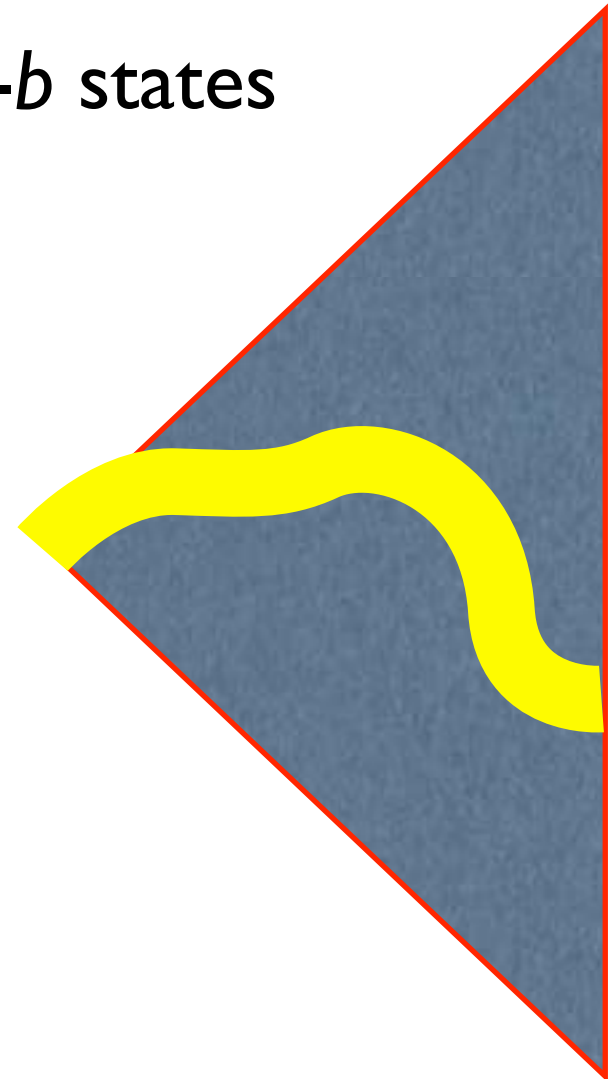
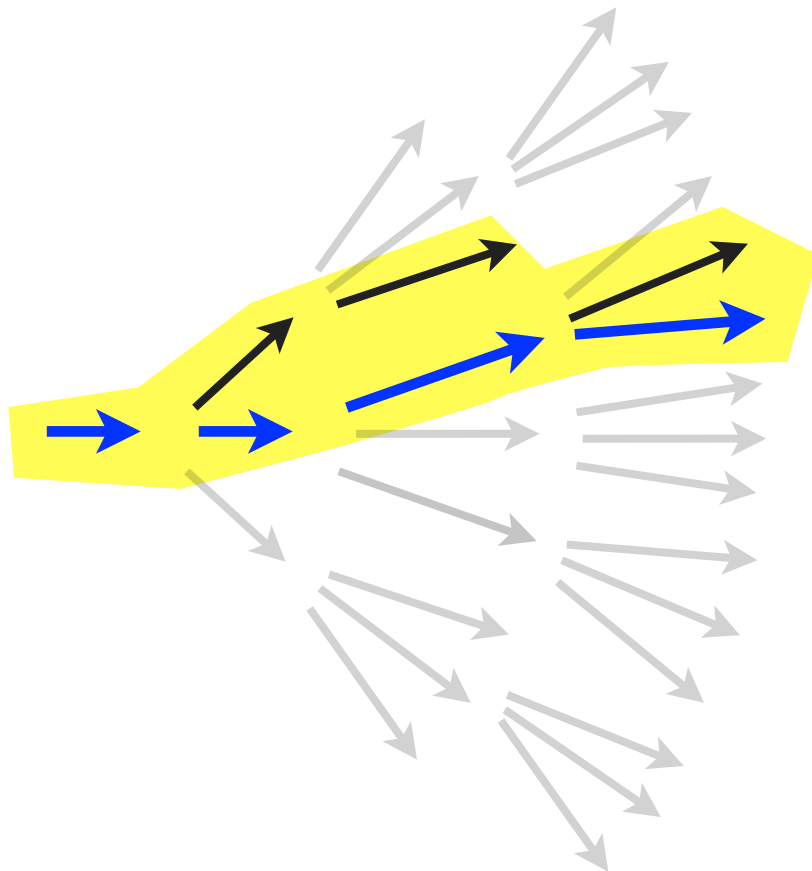
Greedy Search

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
 - search space *should* be exponential
- greedy search: always pick the best next state



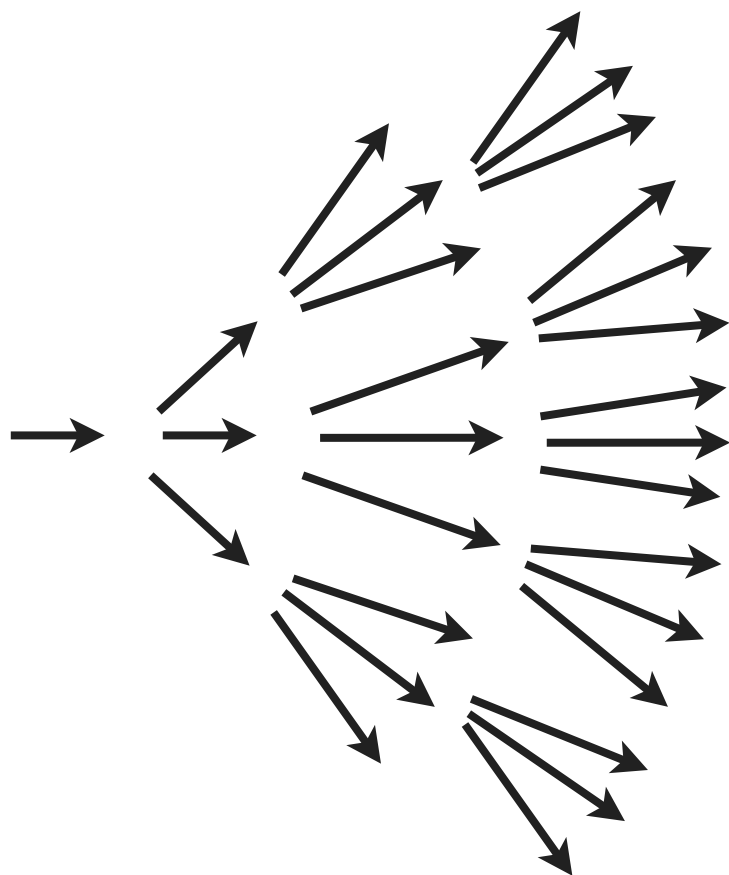
Beam Search

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- search space *should* be exponential
- beam search: always keep top- b states



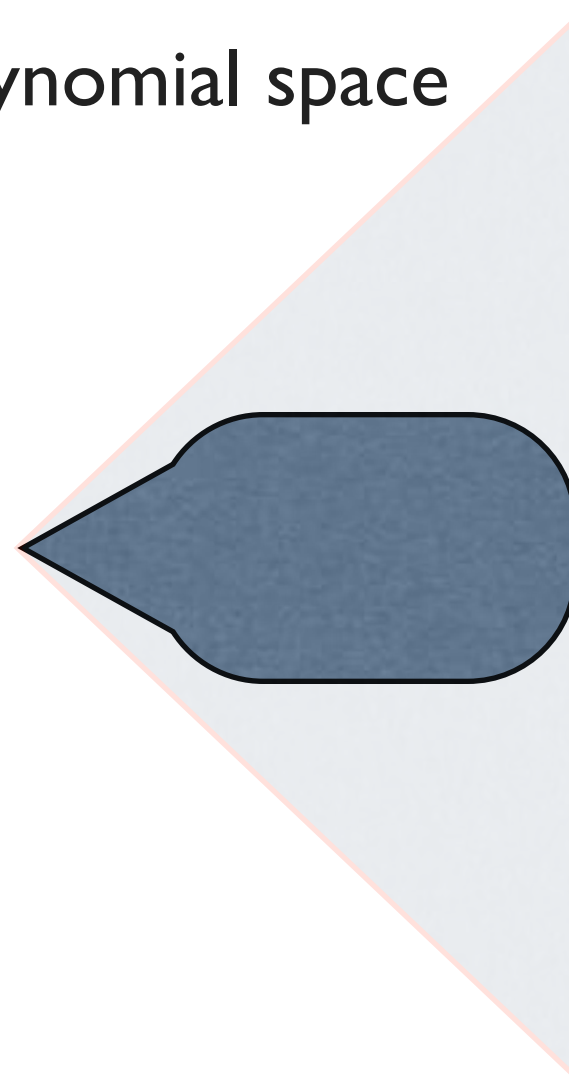
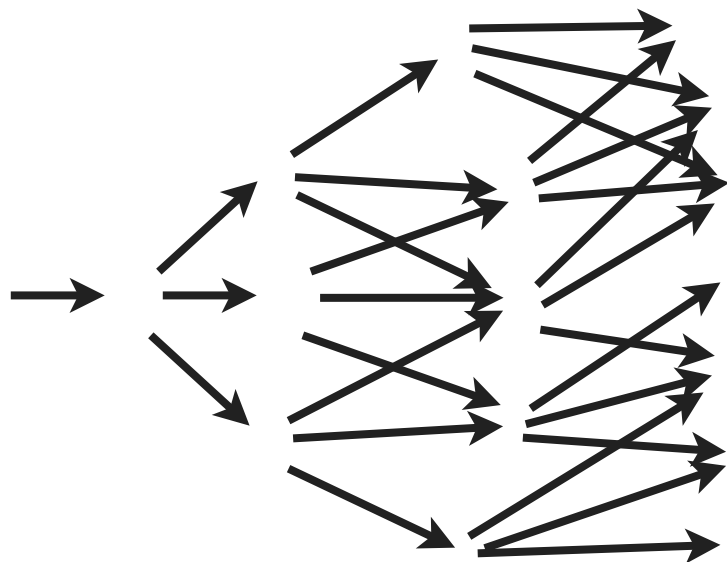
Dynamic Programming

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- key idea of DP: **share** common subproblems
- merge equivalent states \Rightarrow polynomial space



Dynamic Programming

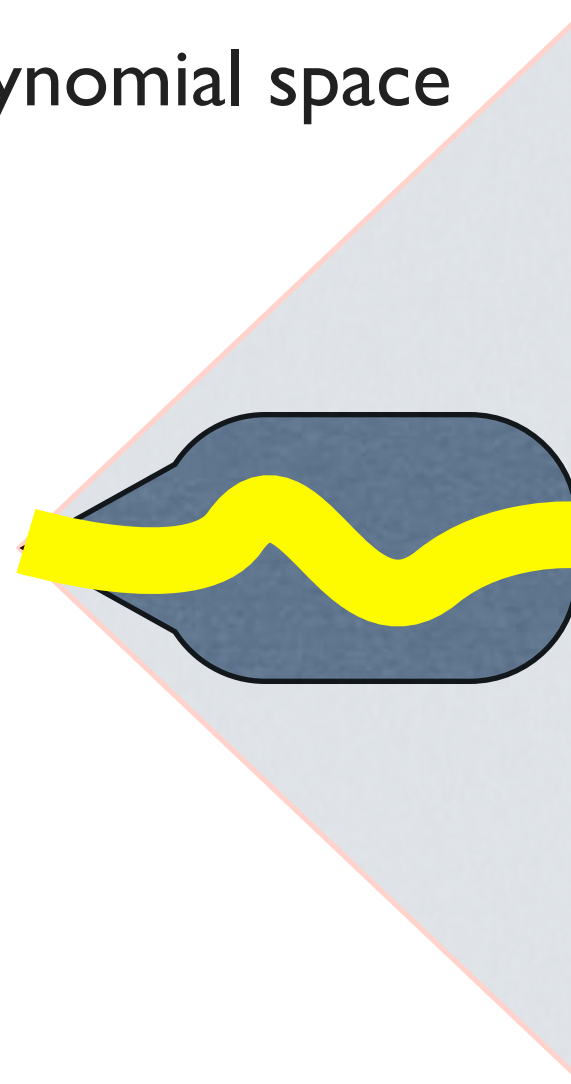
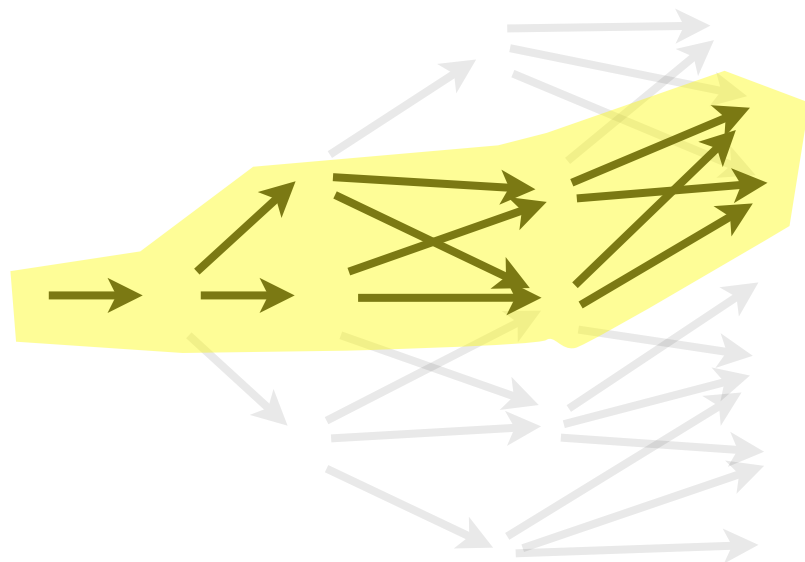
- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- key idea of DP: **share** common subproblems
- merge equivalent states \Rightarrow polynomial space



“graph-structured stack” (Tomita, 1988)

Dynamic Programming

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- key idea of DP: **share** common subproblems
- merge equivalent states \Rightarrow polynomial space

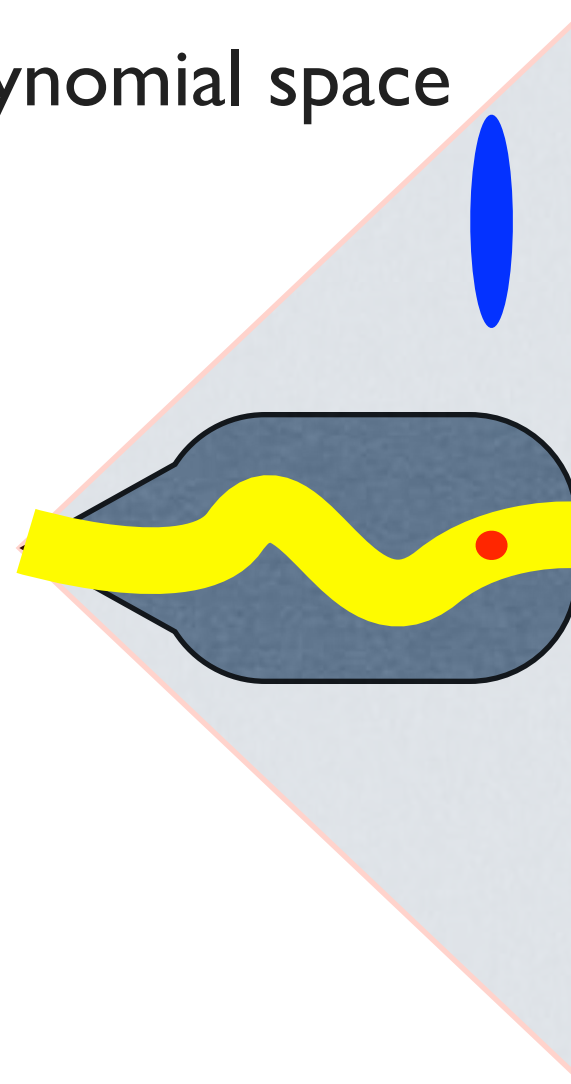
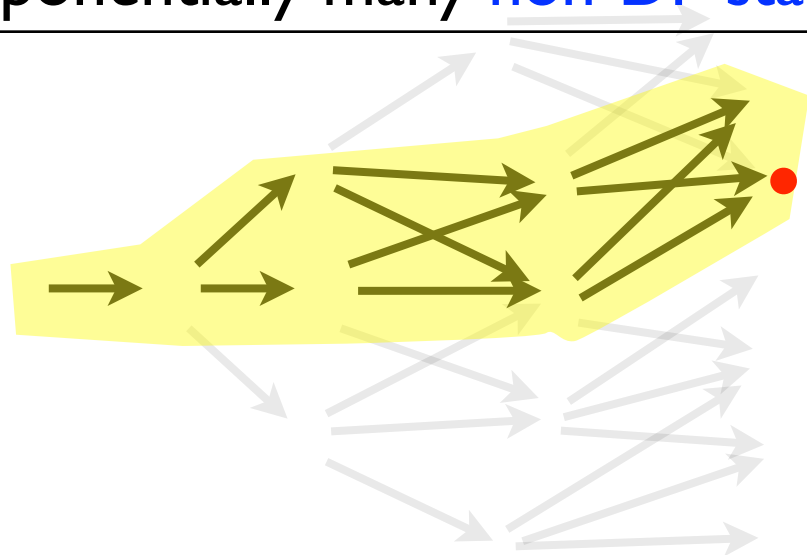


“graph-structured stack” (Tomita, 1988)

Dynamic Programming

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- key idea of DP: **share** common subproblems
 - merge equivalent states \Rightarrow polynomial space

each **DP state** corresponds to exponentially many **non-DP states**



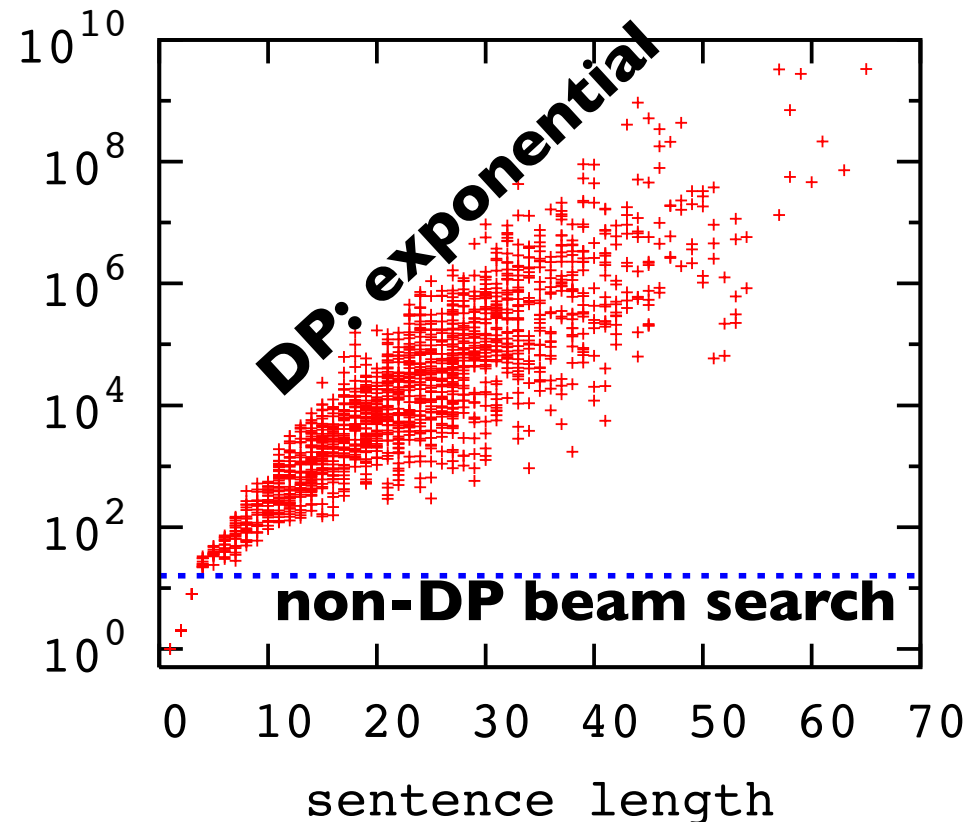
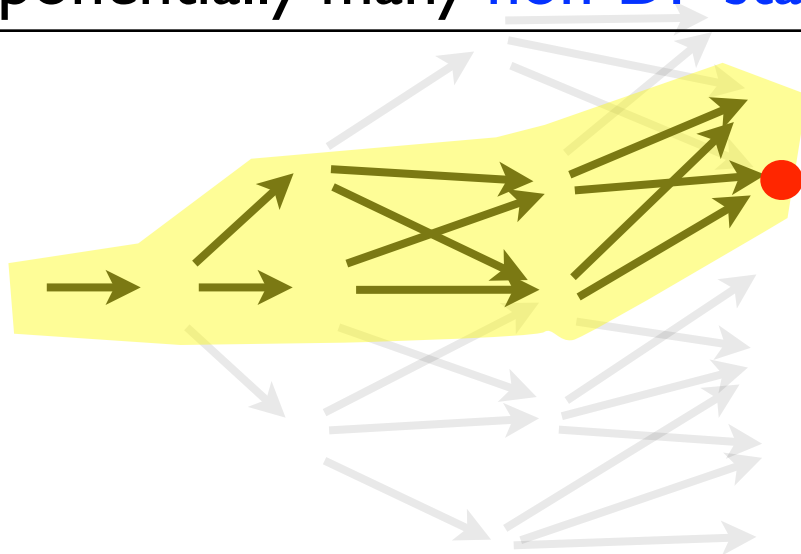
“graph-structured stack” (Tomita, 1988)

DP for Incremental Parsing (Huang and Sagae)

Dynamic Programming

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- key idea of DP: **share** common subproblems
- merge equivalent states \Rightarrow polynomial space

each **DP state** corresponds to exponentially many **non-DP states**



“graph-structured stack” (Tomita, 1988)

Merging Equivalent States

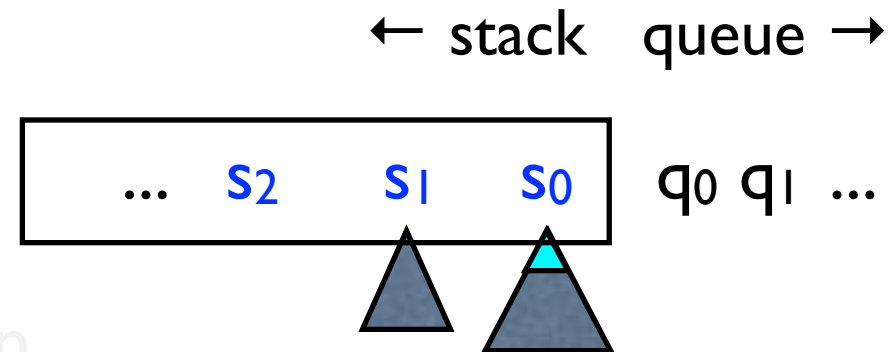
- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

- feed cats nearby in the garden



- feed cats nearby in the garden



Merging Equivalent States

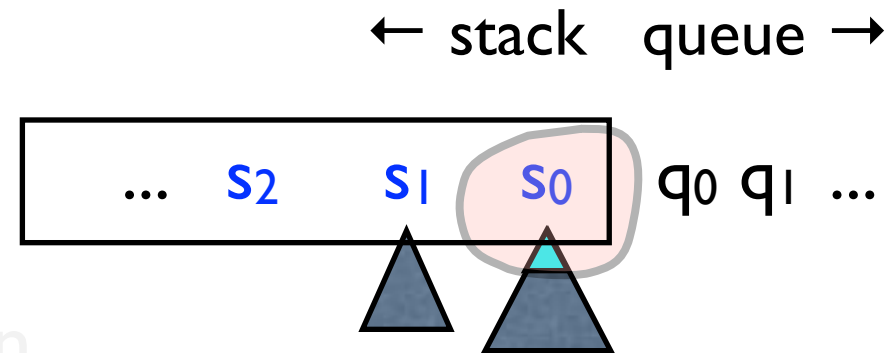
- two states are equivalent if they agree on features
- because same features guarantee same cost

• shift-reduce conflict:

• `feed cats` nearby in the garden



• `feed cats` nearby in the garden



assume features only
look at root of `s0`

Merging Equivalent States

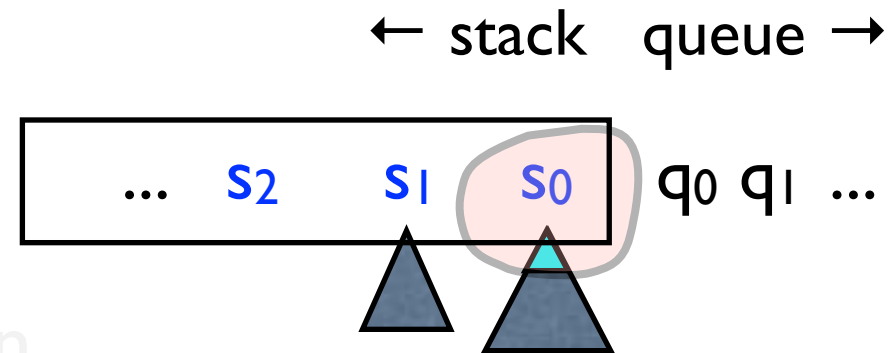
- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

- `feed cats` nearby in the garden



- `feed cats` nearby in the garden



assume features only
look at root of s_0

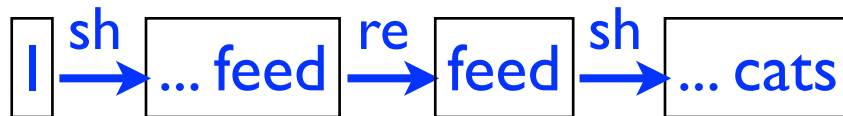
two states are equivalent
if they agree on root of s_0

Merging Equivalent States

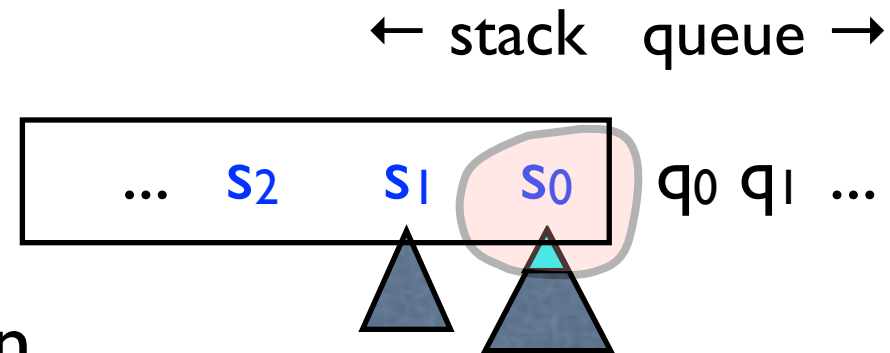
- two states are equivalent if they agree on features
 - because same features guarantee same cost

- shift-reduce conflict:

- **feed cats** nearby in the garden



- **feed cats** nearby in the garden



assume features only
look at root of s_0

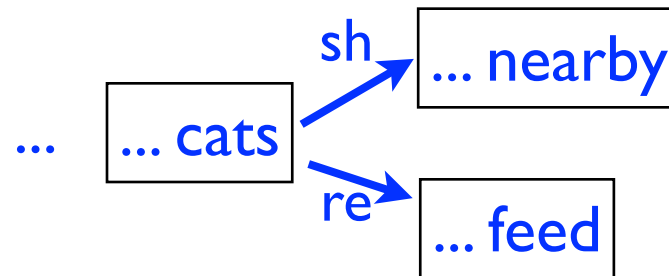
two states are equivalent
if they agree on root of s_0

Merging Equivalent States

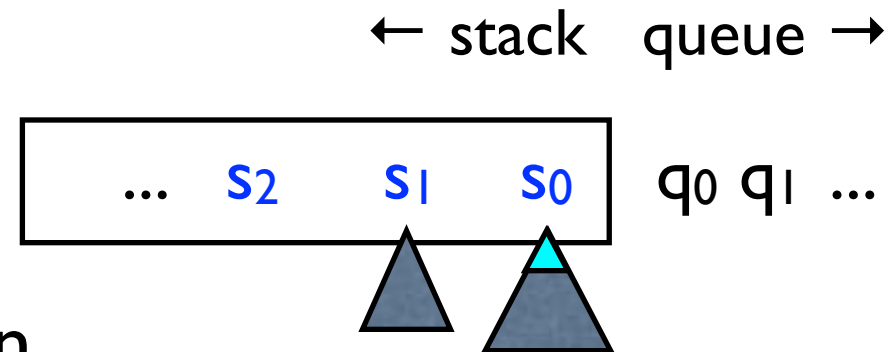
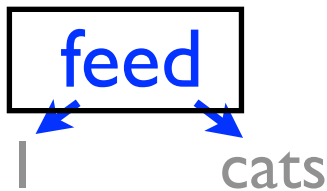
- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

- **feed cats nearby** in the garden



- **feed** nearby in the garden

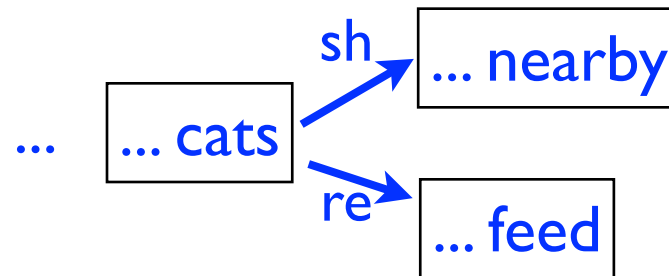


Merging Equivalent States

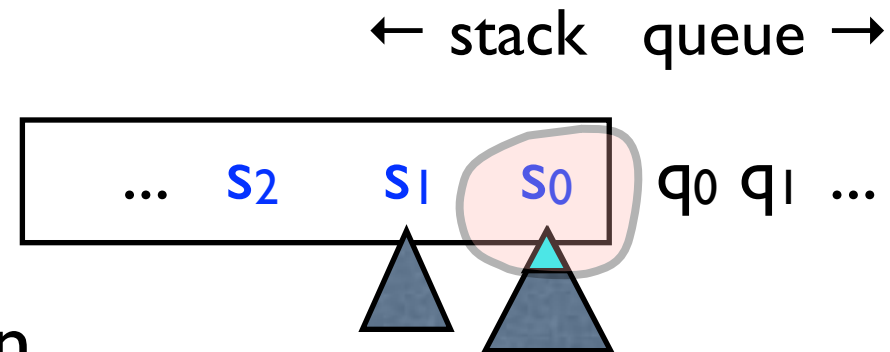
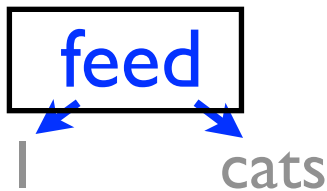
- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

- **feed cats nearby** in the garden



- **feed** nearby in the garden

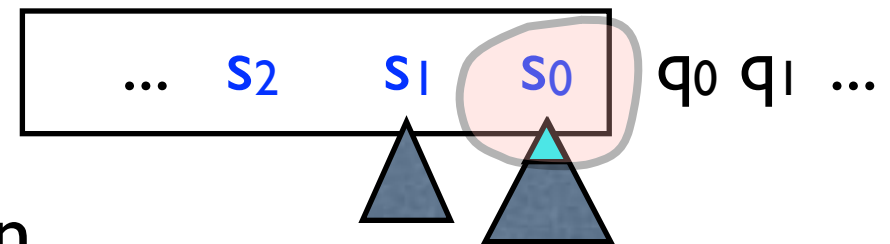


Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

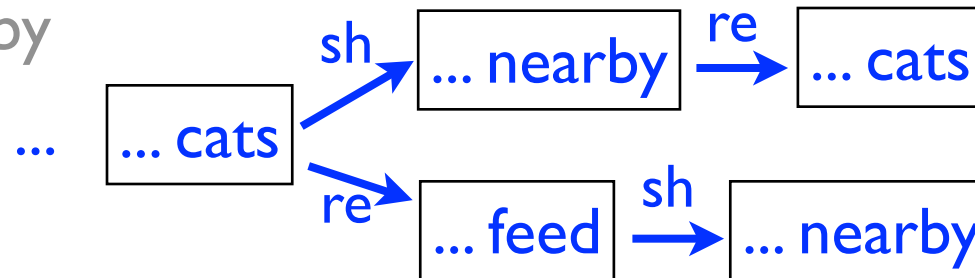
← stack queue →



- **feed cats**



in the garden



- **feed nearby** in the garden

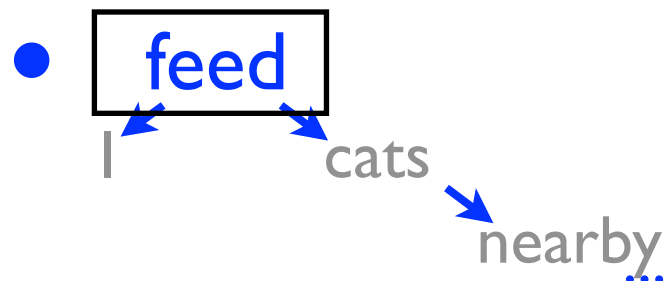
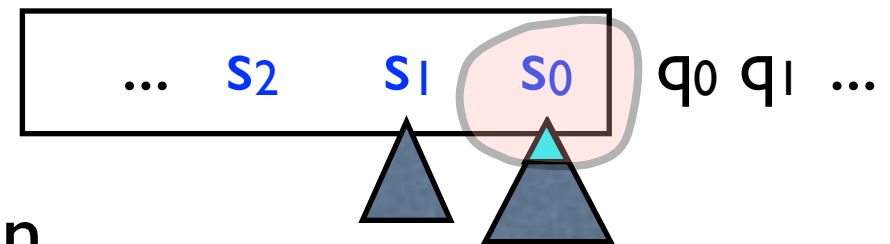


Merging Equivalent States

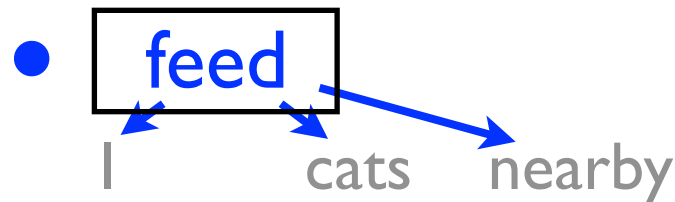
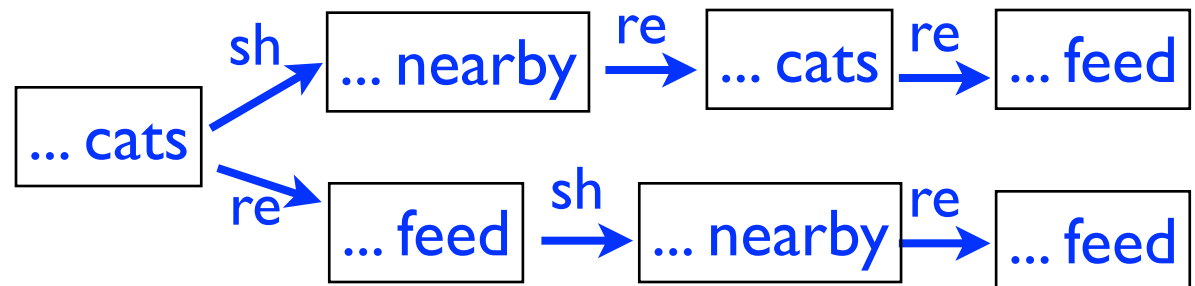
- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

← stack queue →



in the garden



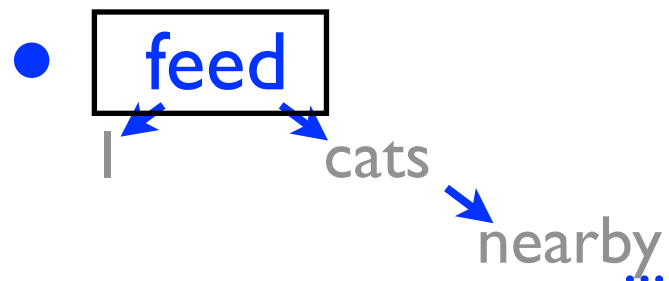
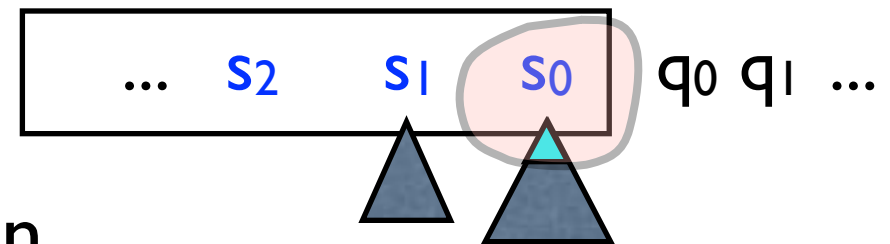
in the garden

Merging Equivalent States

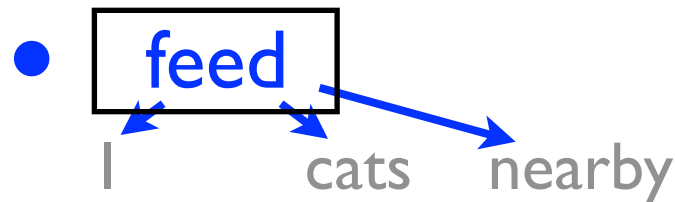
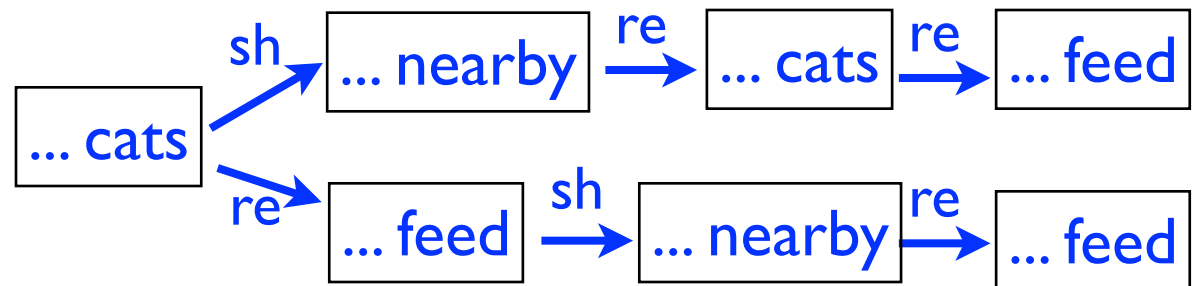
- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

← stack queue →



in the garden



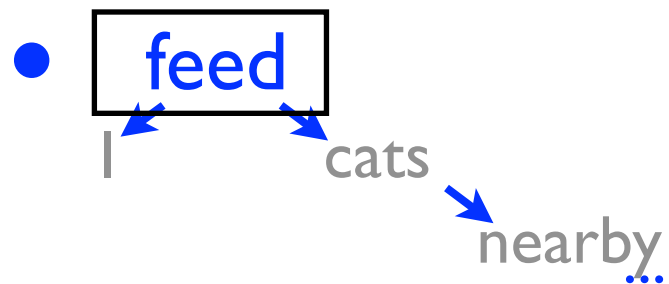
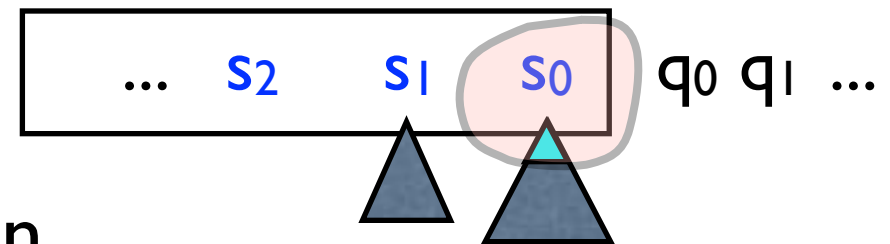
in the garden

Merging Equivalent States

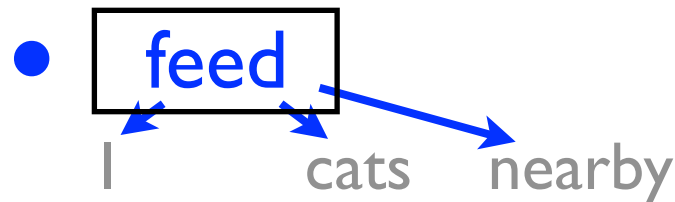
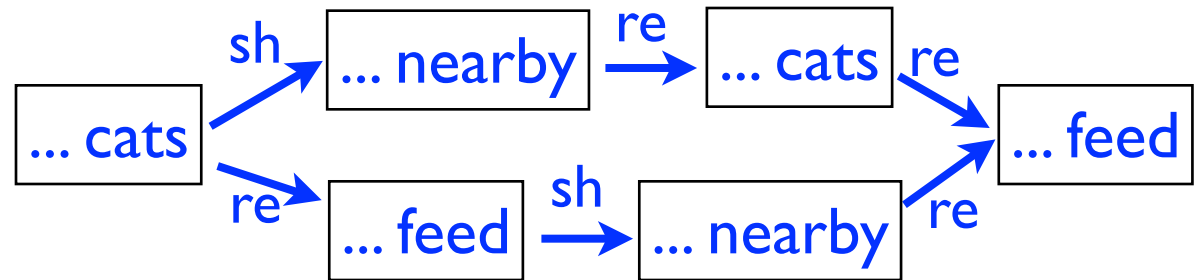
- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:

← stack queue →



in the garden



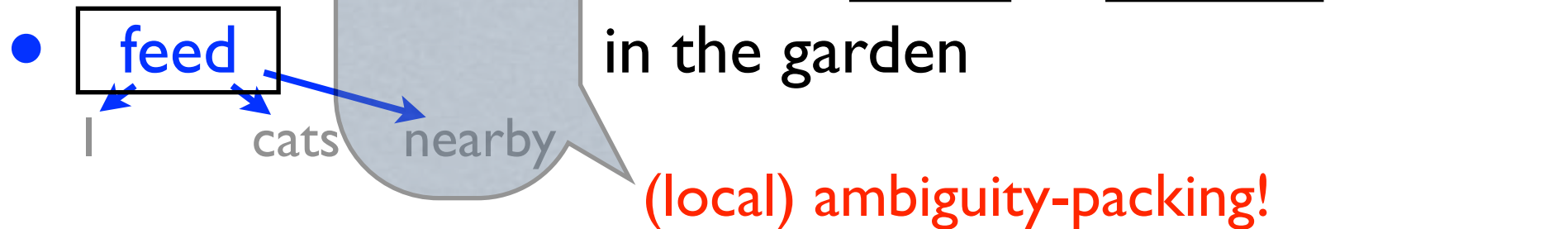
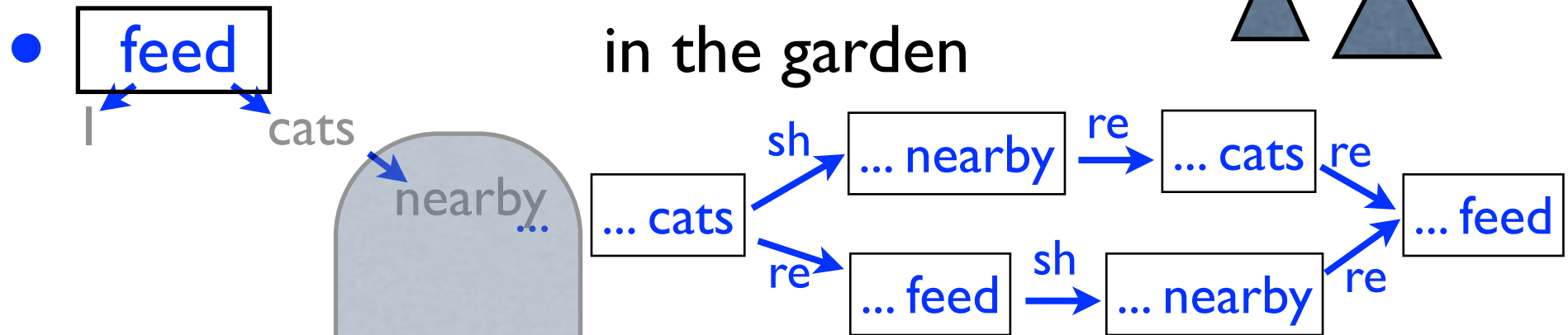
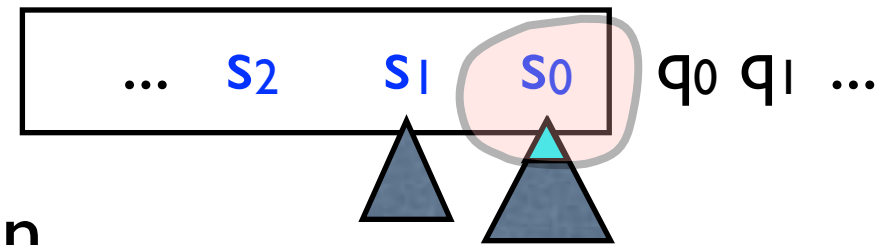
in the garden

Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

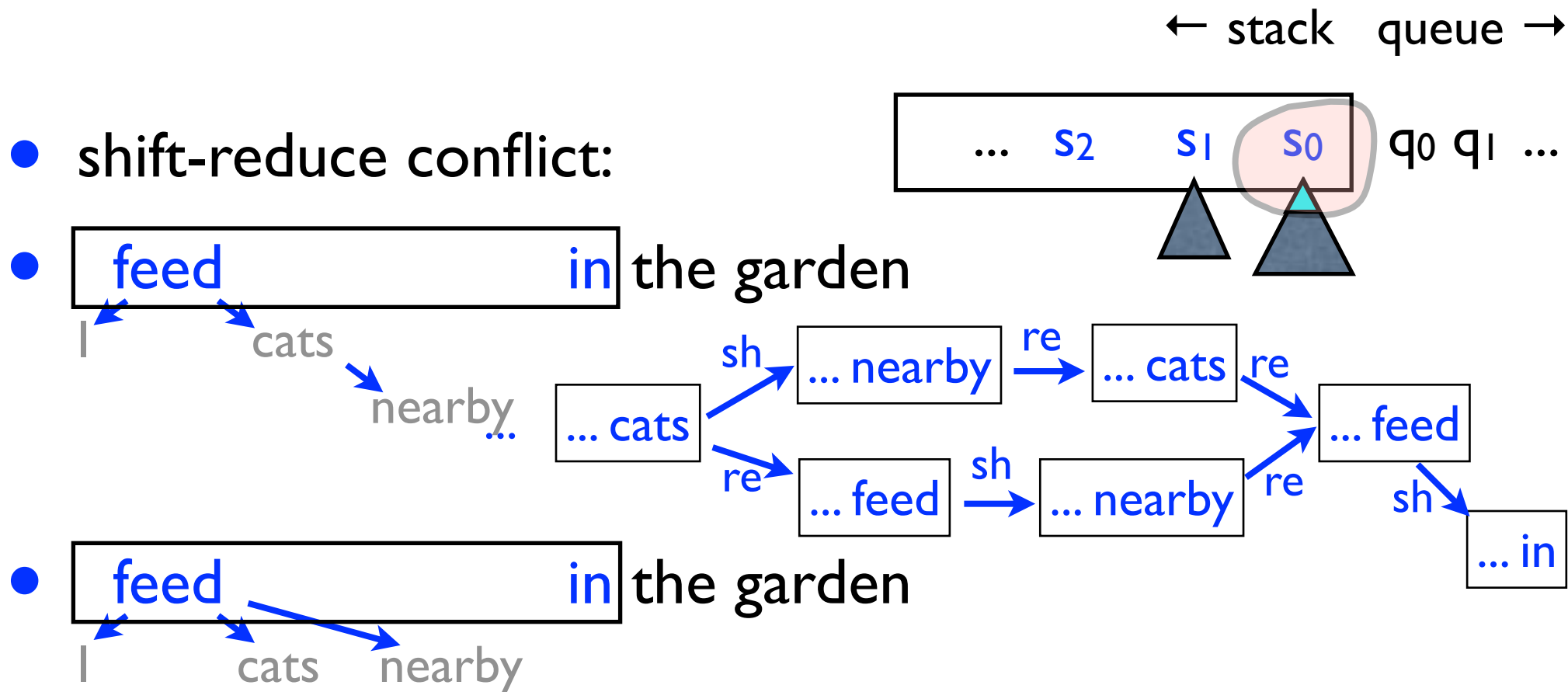
- shift-reduce conflict:

← stack queue →



Merging Equivalent States

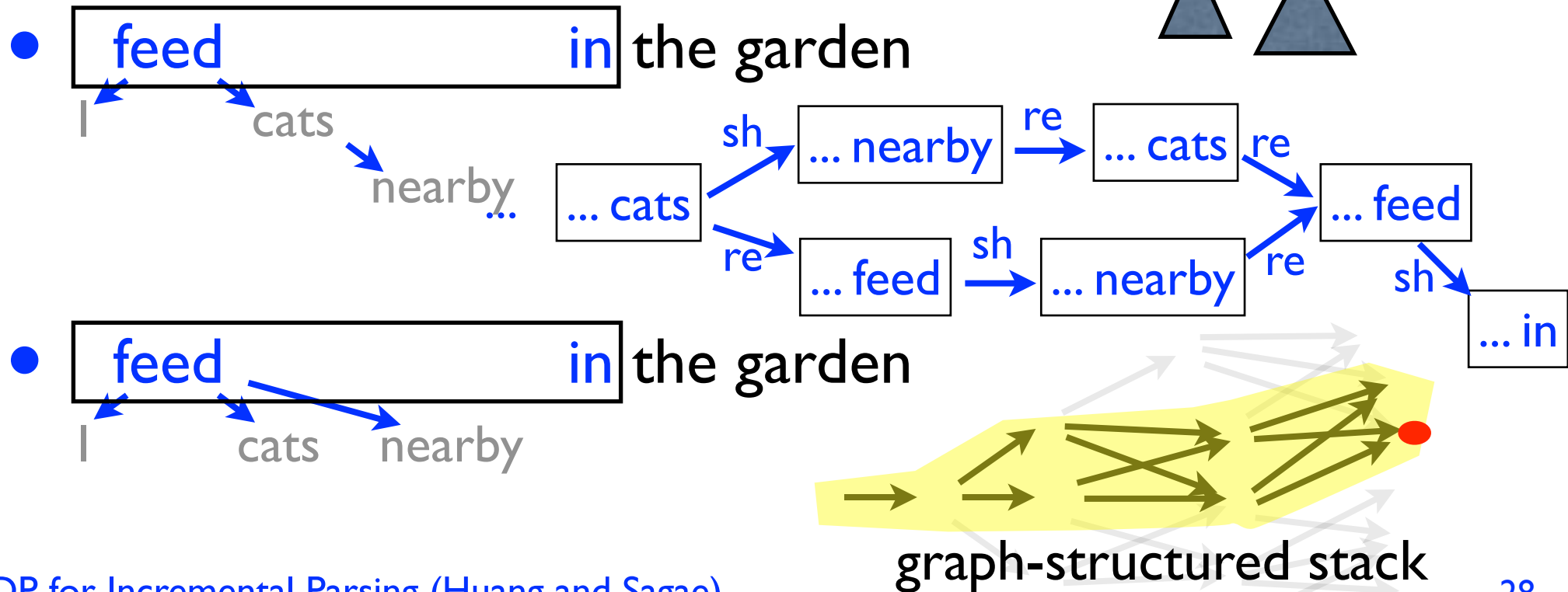
- two states are equivalent if they agree on features
- because same features guarantee same cost



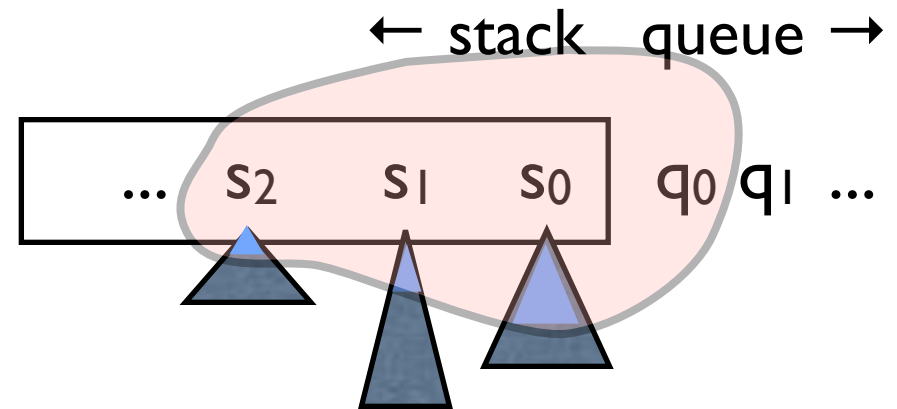
Merging Equivalent States

- two states are equivalent if they agree on features
- because same features guarantee same cost

- shift-reduce conflict:



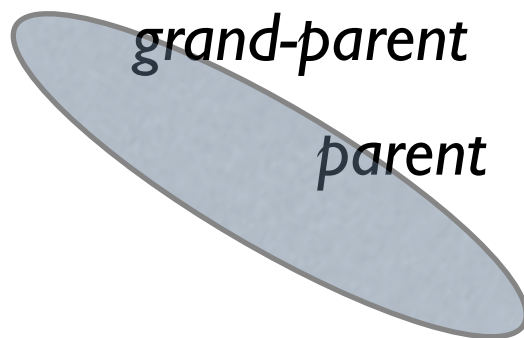
Theory: Polynomial-Time DP



- this DP is *exact* and *polynomial-time* if features are:
 - a) **bounded** -- for polynomial time
 - features can only look at a **local window**
 - b) **monotonic** -- for correctness (optimal substructure)
 - features should draw no more info from trees farther away from stack top than from trees closer to top
- both are intuitive: a) always true; b) *almost* always true

Theory: Monotonic History

- related: grammar refinement by annotation (Johnson, 1998)
 - annotate vertical context history (e.g., parent)
 - monotonicity: can't annotate grand-parent without annotating the parent (**otherwise DP would fail**)
- our features: **left-context** history instead of vertical-context
 - similarly, can't annotate s_2 without annotating s_1
 - but we can always design “minimum monotonic superset”



Related Work

- Graph-Structured Stack (Tomita 88): Generalized LR
 - GSS is just a chart viewed from left to right (e.g. Earley 70)
 - this line of work started w/ Lang (1974); **stuck since 1990**
 - b/c explicit LR table is impossible with modern grammars
 - general idea: *compile* CFG parse chart to FSAs (e.g. our beam)

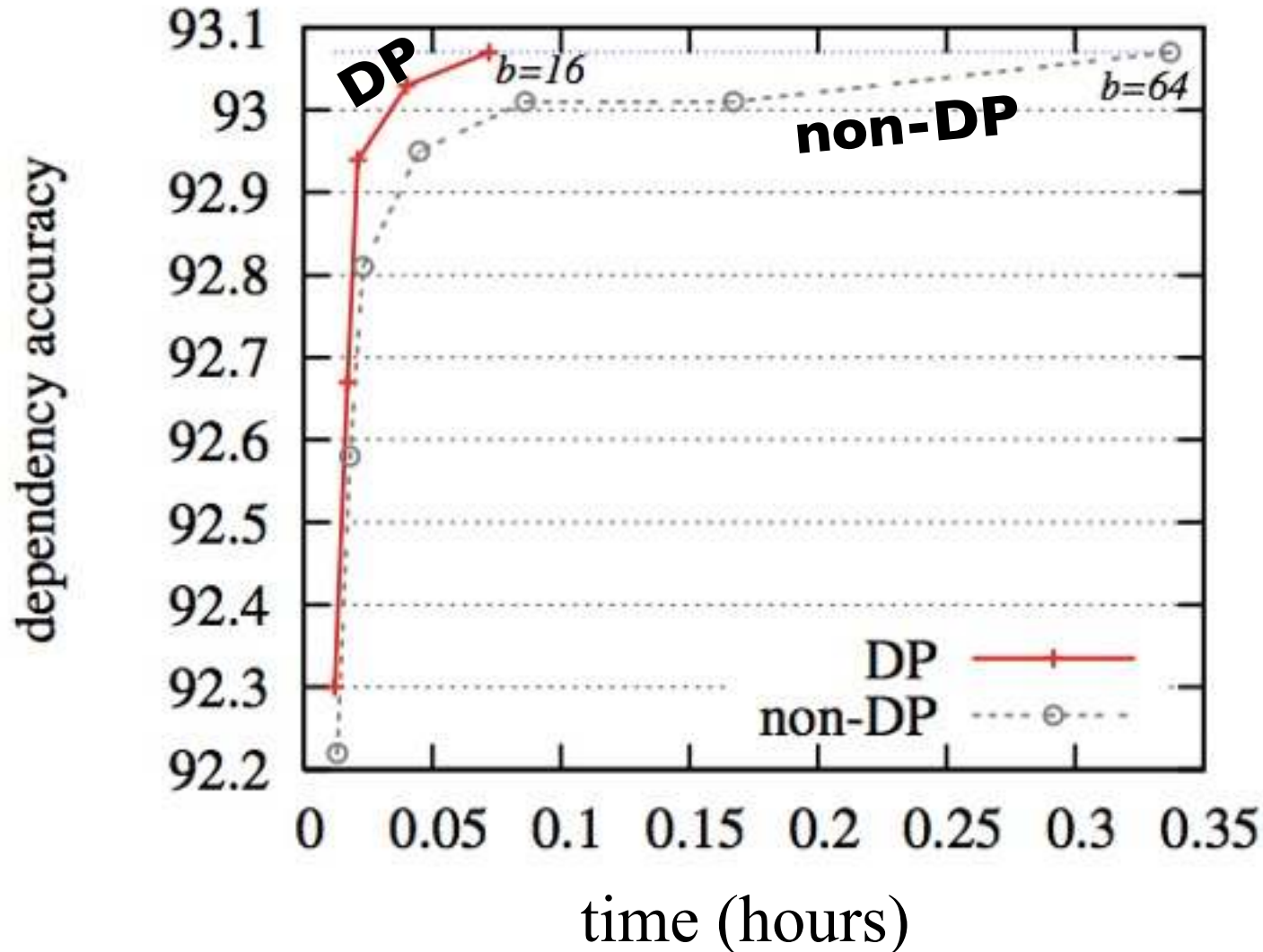
Related Work

- Graph-Structured Stack (Tomita 88): Generalized LR
 - GSS is just a chart viewed from left to right (e.g. Earley 70)
 - this line of work started w/ Lang (1974); **stuck since 1990**
 - b/c explicit LR table is impossible with modern grammars
 - general idea: *compile* CFG parse chart to FSAs (e.g. our beam)
- We revived and advanced this line of work in two aspects
 - theoretical: **implicit** LR table based on features
 - merge and split on-the-fly; no pre-compilation needed
 - monotonic feature functions guarantee correctness (**new**)
 - practical: achieved linear-time performance with pruning

Experiments

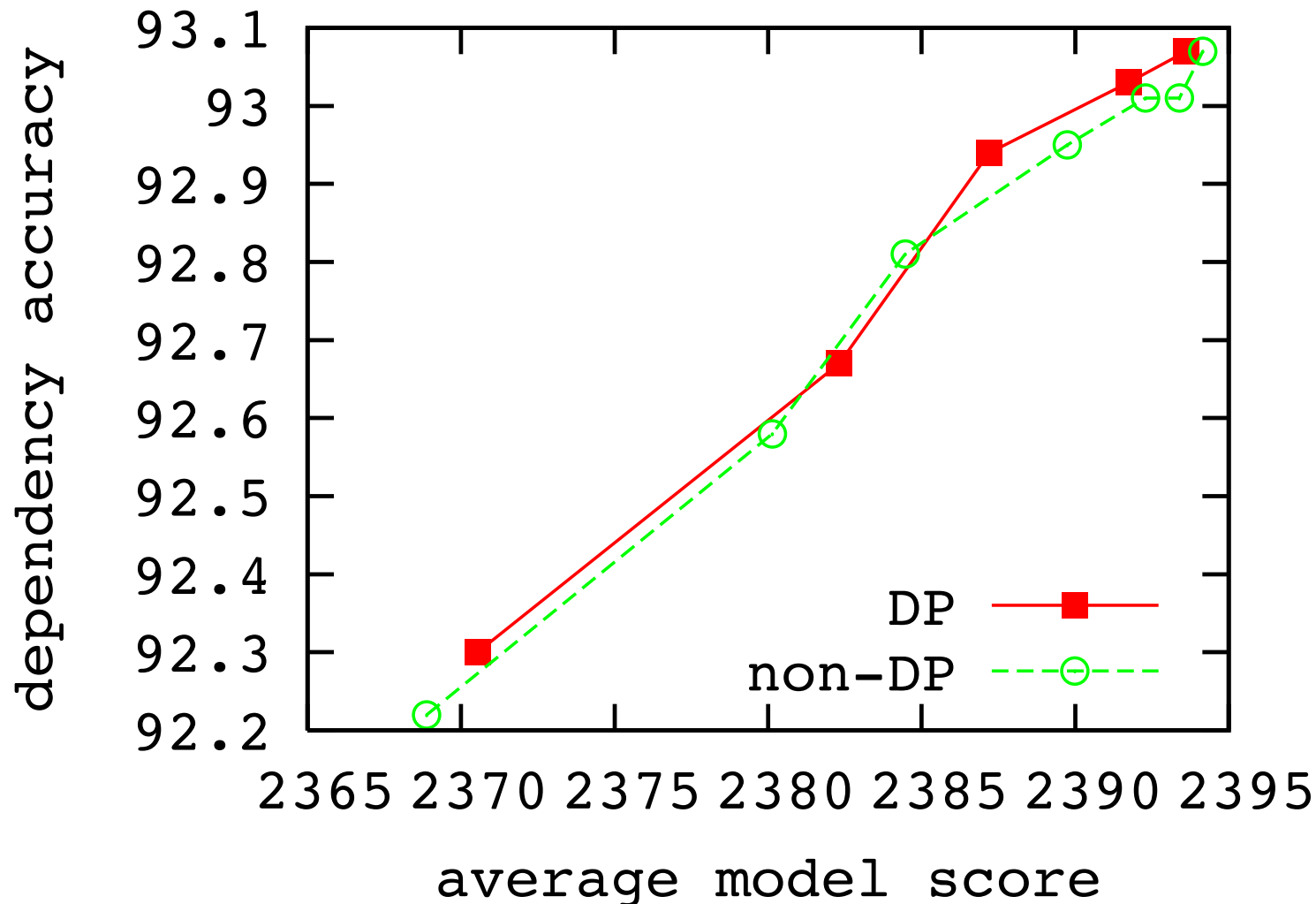
Speed Comparison

- 5 times faster with the same parsing accuracy

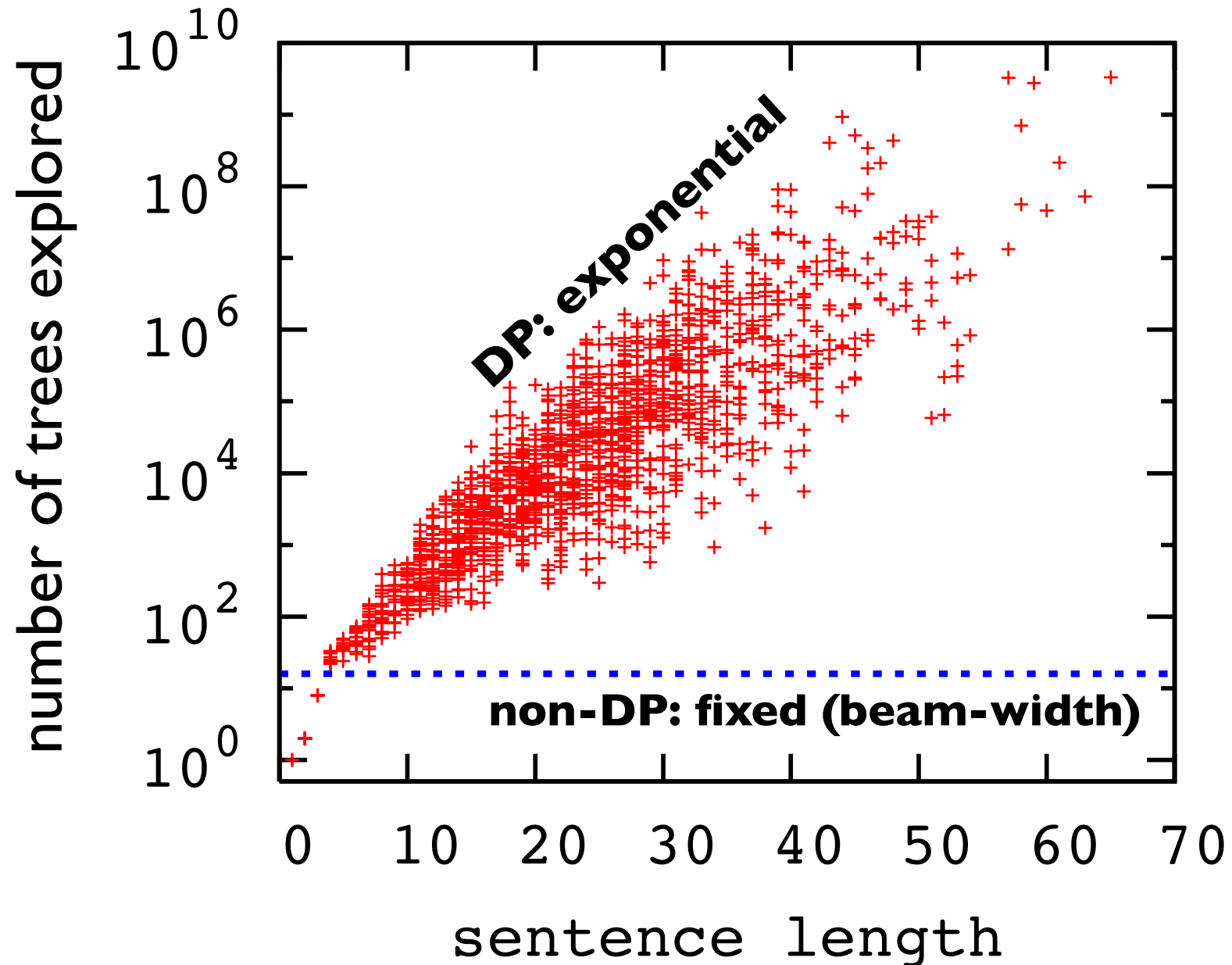


Correlation of Search and Parsing

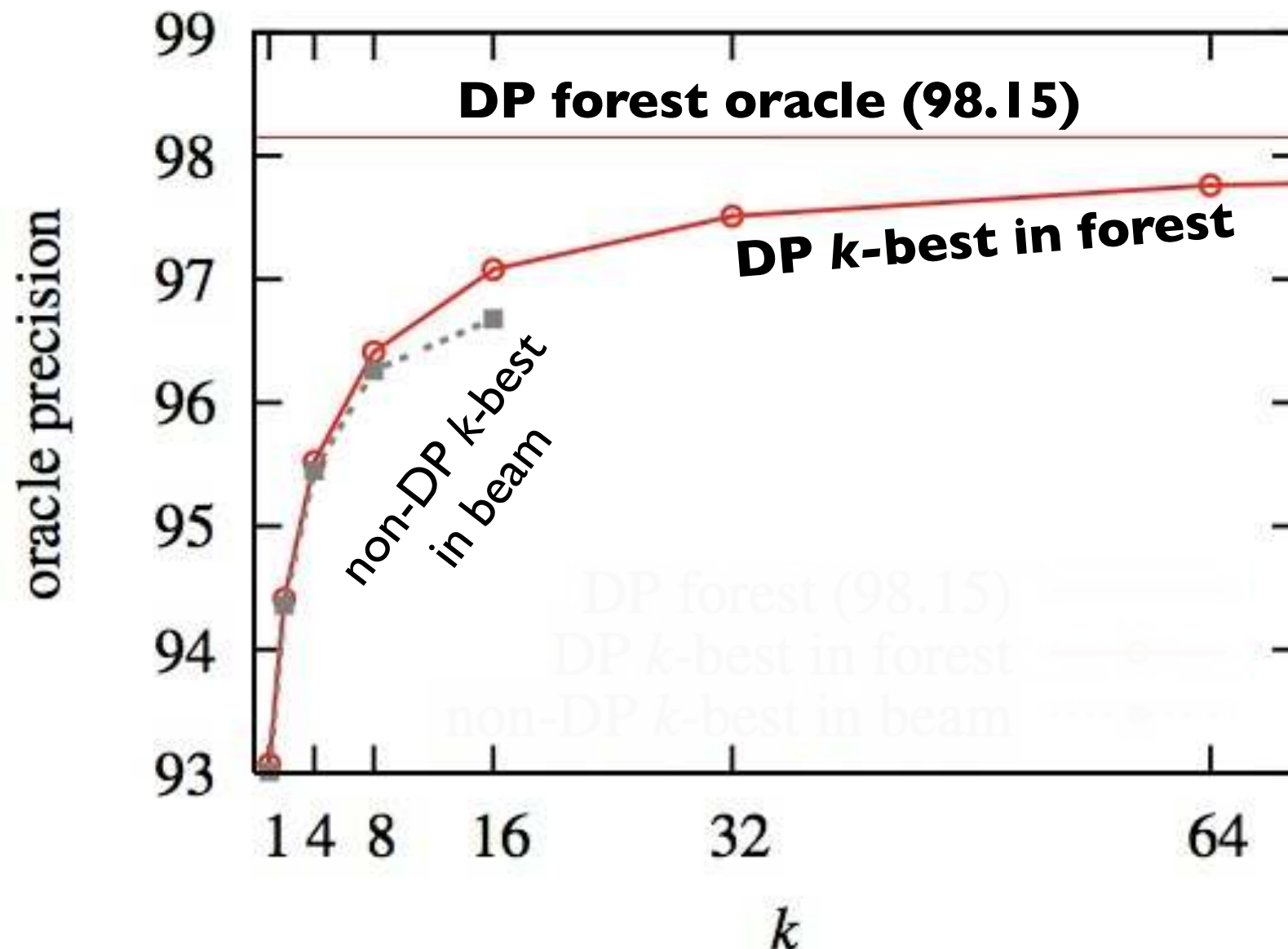
- better search quality \Leftrightarrow better parsing accuracy



Search Space: Exponential



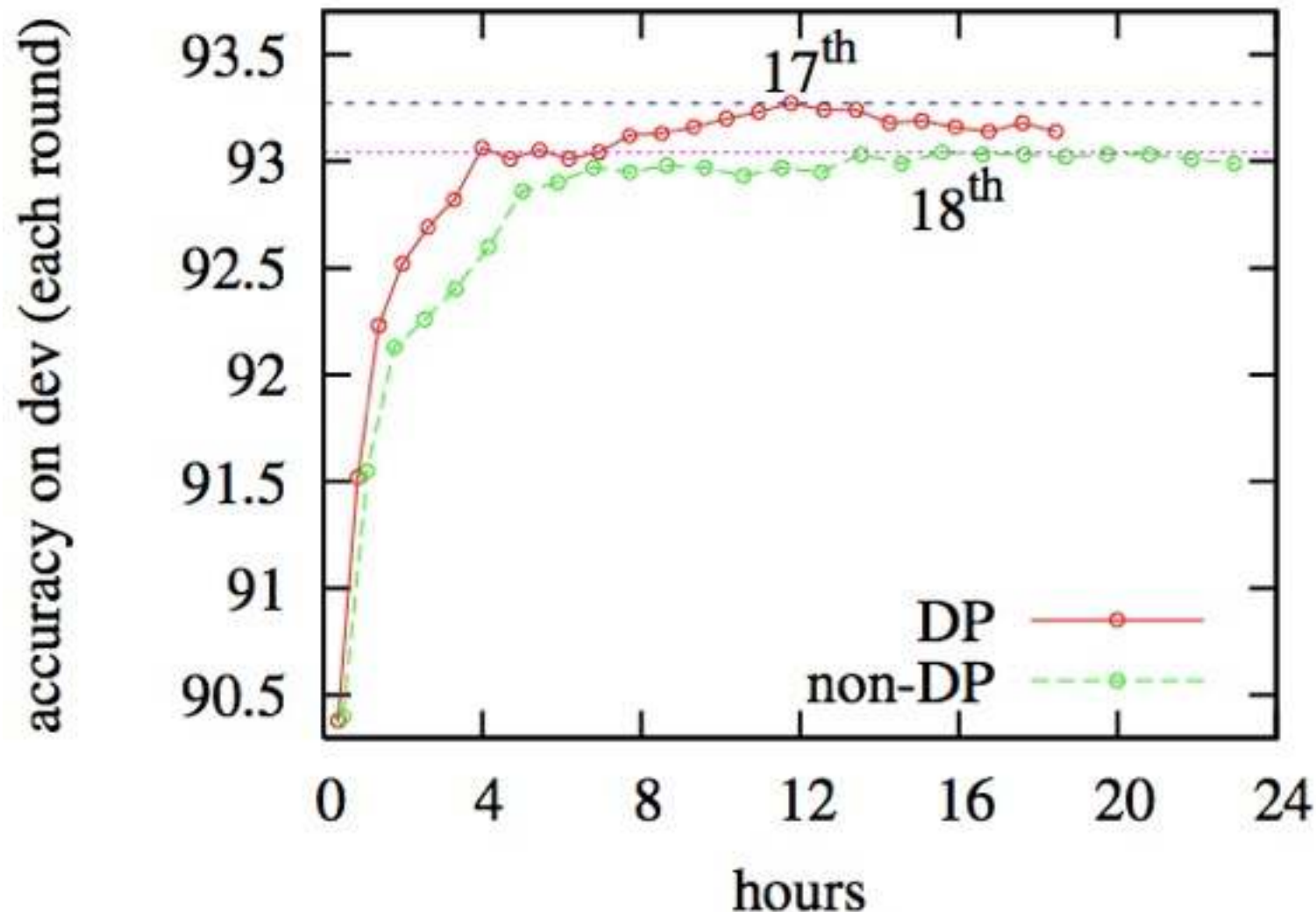
N-Best / Forest Oracles



(b) oracle precision on dev

Better Search => Better Learning

- DP leads to faster and better learning w/ perceptron



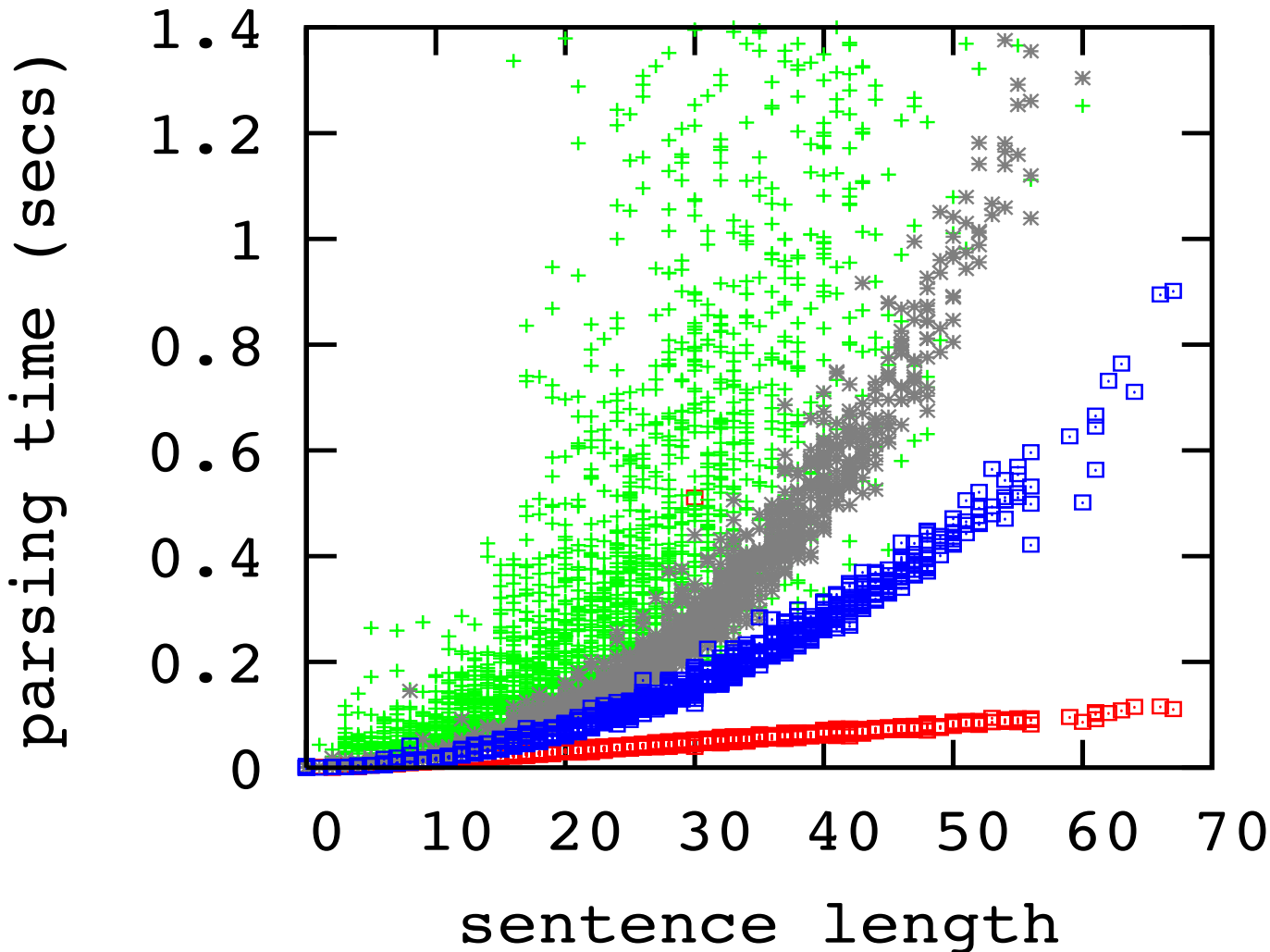
Learning Details: Early Updates

- greedy search: update at first error (Collins/Roark 04)
- beam search: update when gold is pruned (Zhang/Clark 08)
- DP search: *also* update when gold is “merged” (new!)
- b/c we know gold can't make to the top again

<i>it</i>	updates	early%	time	updates	early%	time
1	31943	98.9	22	31189	87.7	29
2	27311	98.8	29	26324	80.9	37
..
5	20236	98.3	38	19027	70.3	47
..
17	8683	97.1	48	7434	49.5	60
..
25	5715	97.2	51	4676	41.2	65

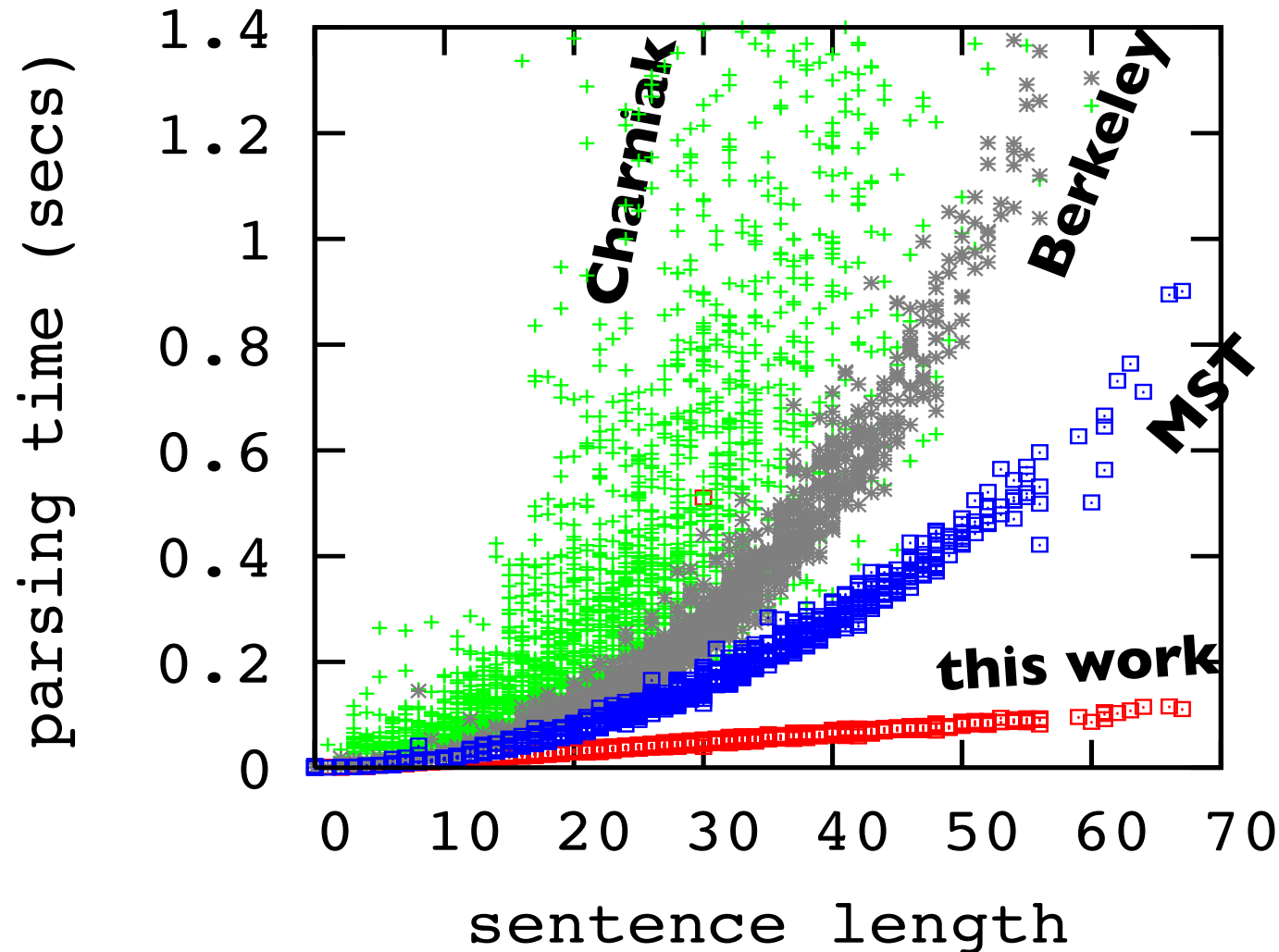
Parsing Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers



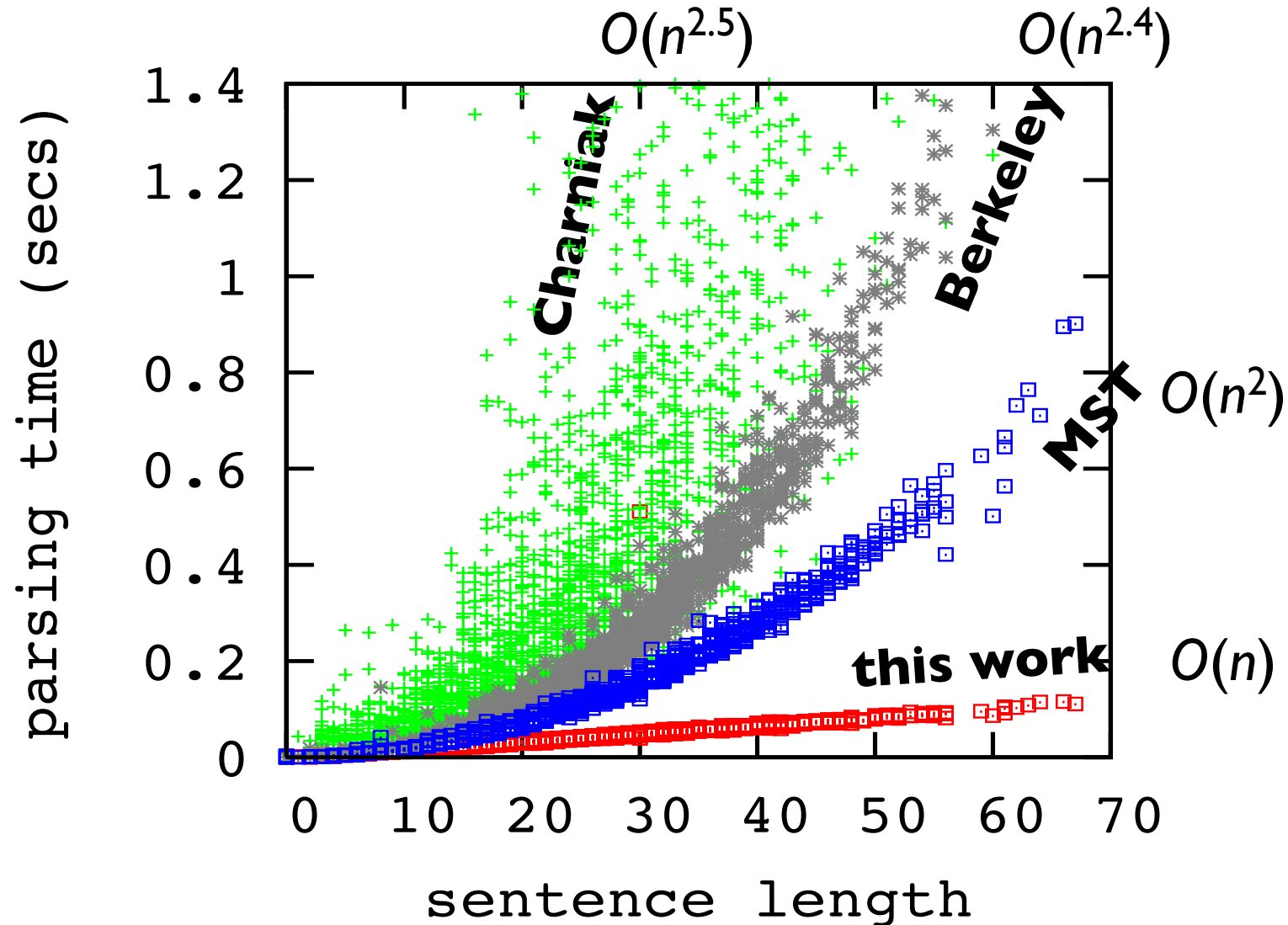
Parsing Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers



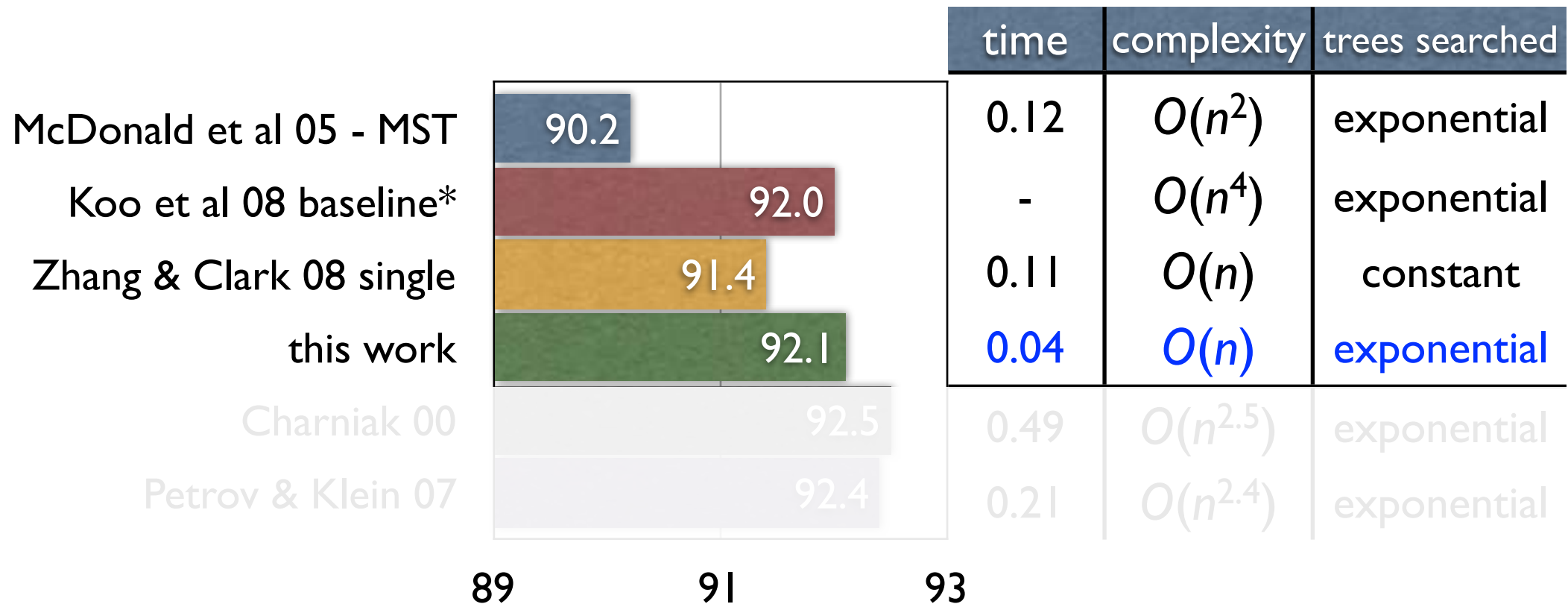
Parsing Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers



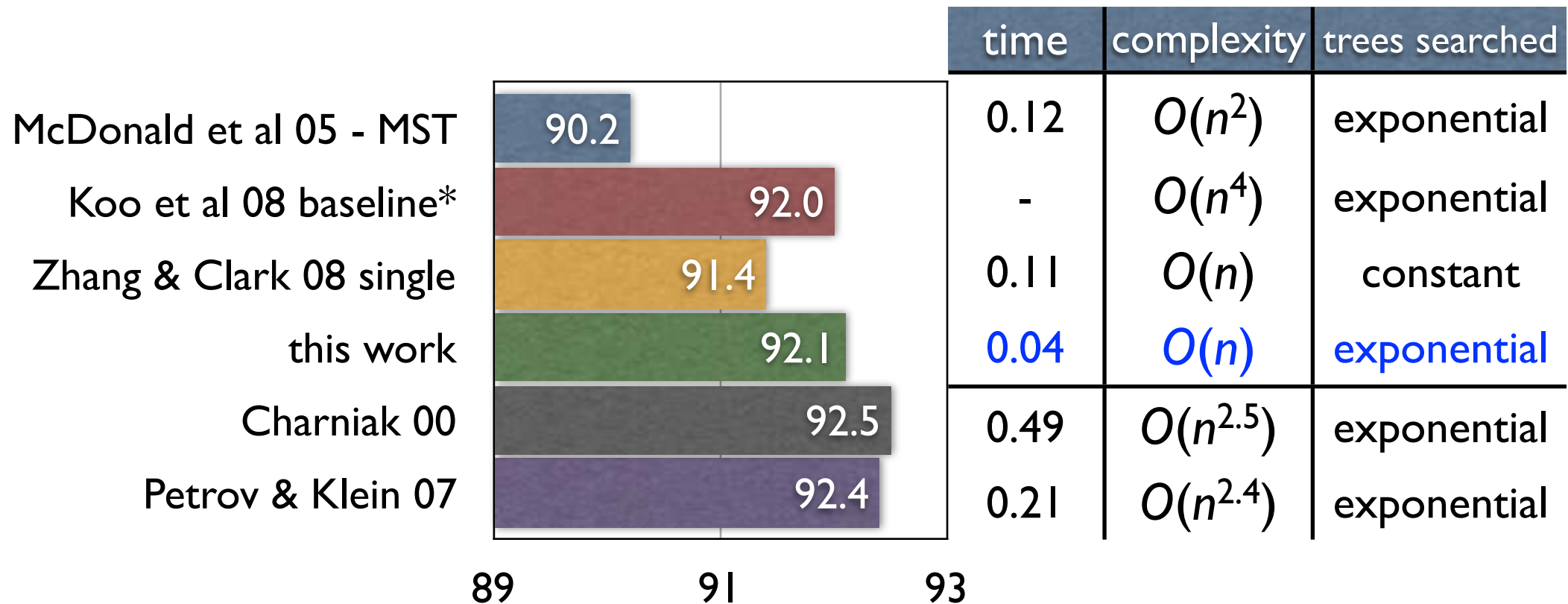
Final Results

- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank



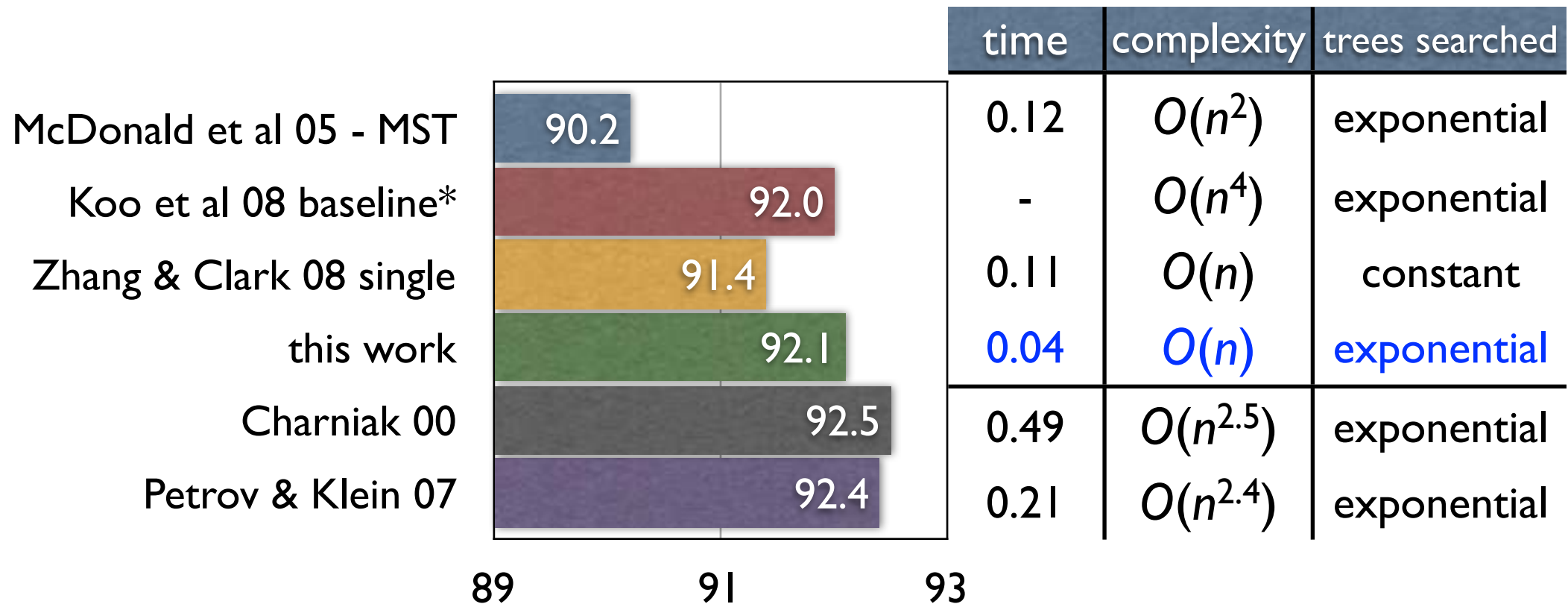
Final Results

- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank



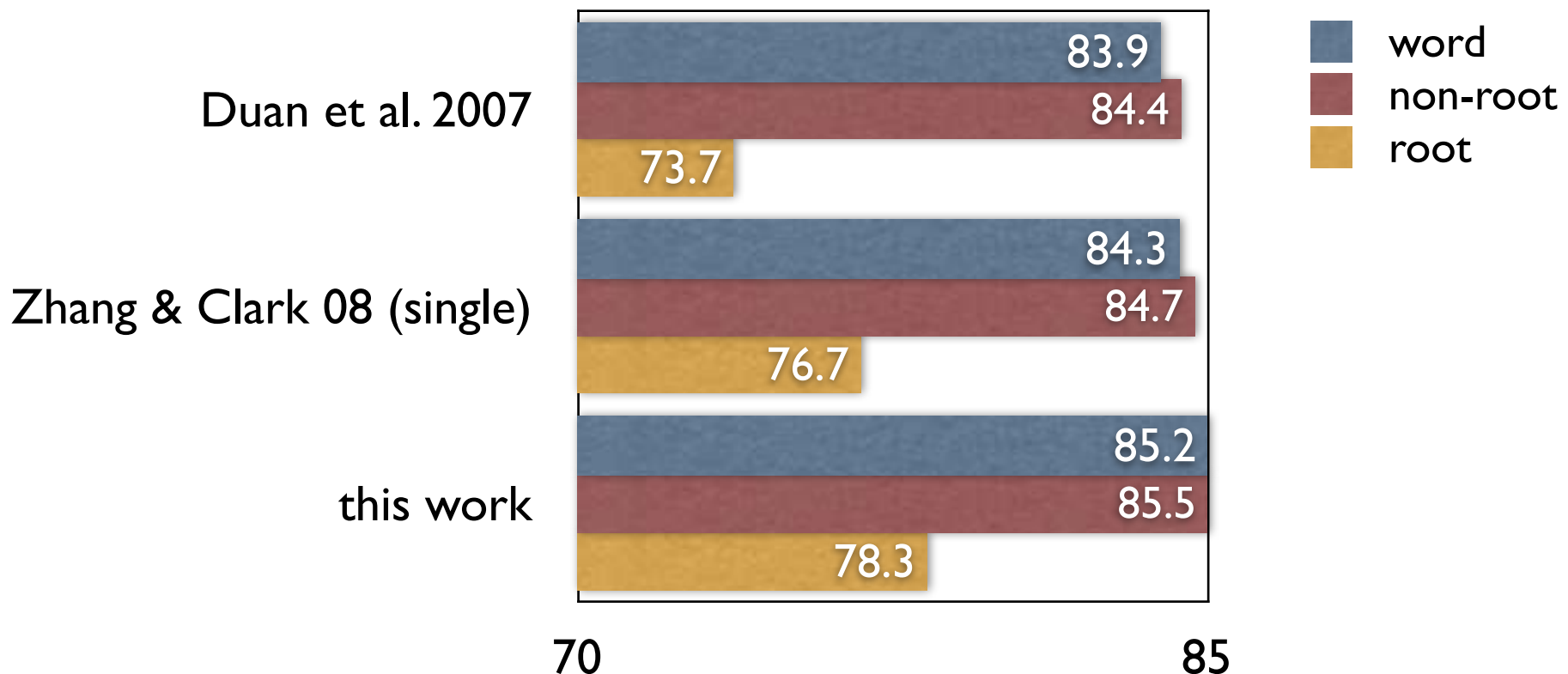
Final Results

- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank






Final Results on Chinese



- also the best parsing accuracy on Chinese
 - Penn Chinese Treebank (CTB 5)
- all numbers below use gold-standard POS tags



Conclusion

greedy search	incremental parsing (e.g. shift-reduce) 	
principled search		full dynamic programming (e.g. CKY) 
	fast (linear-time)	slow (cubic-time)

Conclusion

greedy search	incremental parsing (e.g. shift-reduce) 	
principled search	linear-time shift-reduce parsing w/ dynamic programming	full dynamic programming (e.g. CKY) 
	fast (linear-time)	slow (cubic-time)

Thank You

- a general theory of DP for shift-reduce parsing
 - as long as features are bounded and monotonic
- fast, accurate DP parser release coming soon:
 - <http://www.isi.edu/~lhuang>
 - <http://www.ict.usc.edu/~sagae>
- future work
 - adapt to constituency parsing (straightforward)
 - other grammar formalisms like CCG and TAG
 - integrate POS tagging into the parser