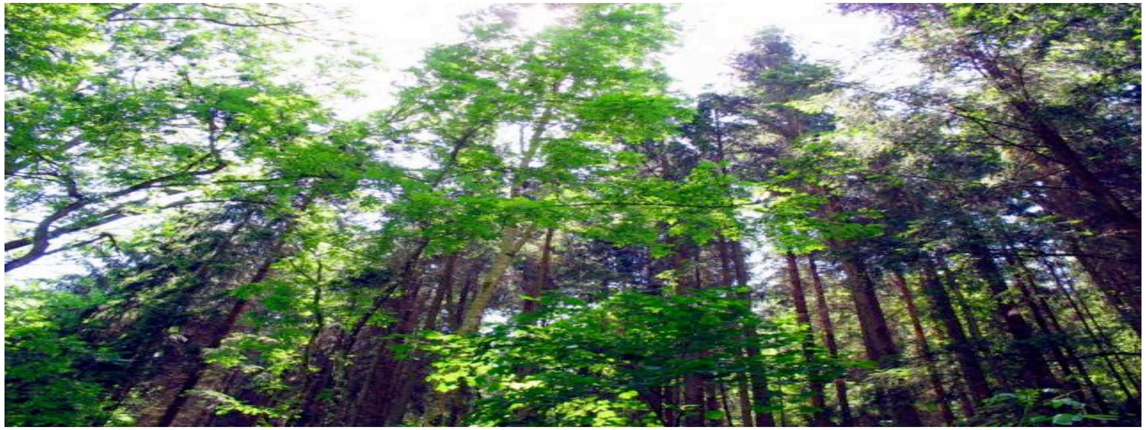
Forest-based Algorithms in Natural Language Processing



Liang Huang

overview of Ph.D. work done at Penn (and ISI, ICT)

includes joint work with David Chiang, Kevin Knight, Aravind Joshi, Haitao Mi and Qun Liu

CMU LTI Seminar, Pittsburgh, PA, May 14, 2009

NLP is all about ambiguities

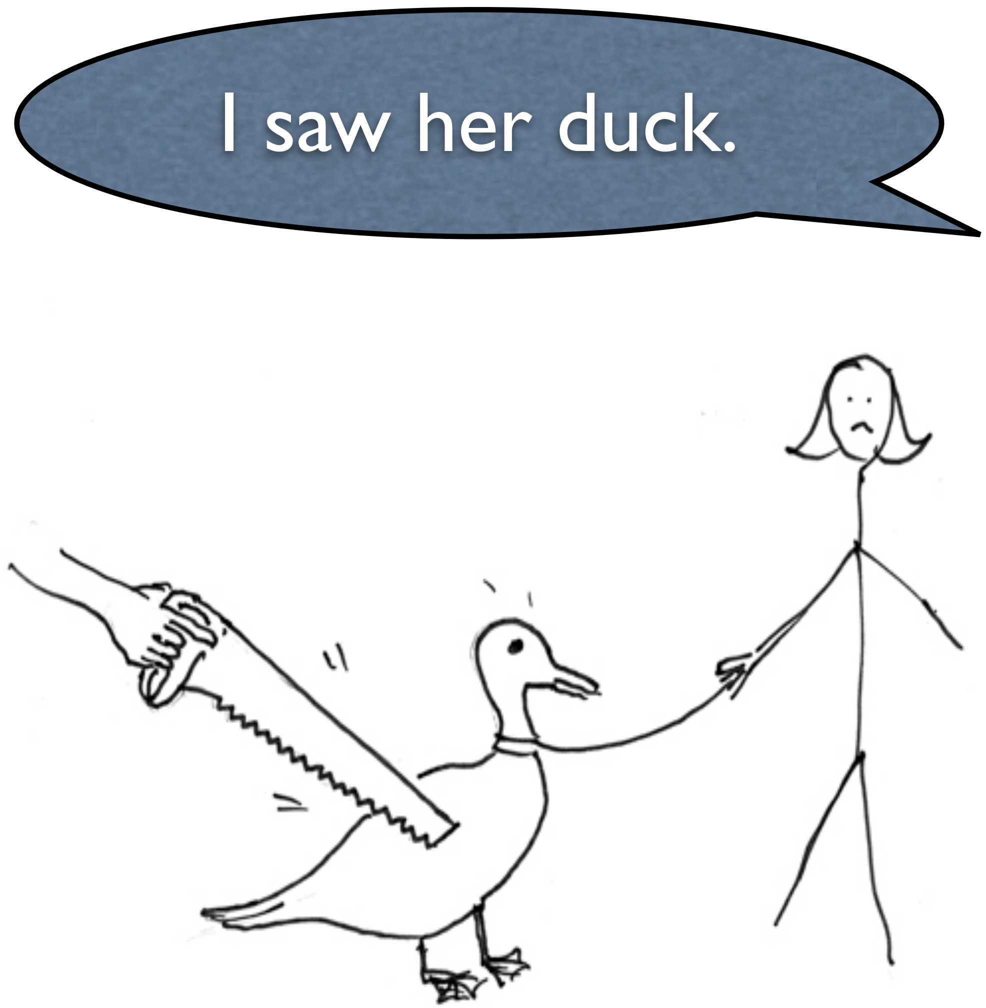
- to middle school kids: what does this sentence mean?

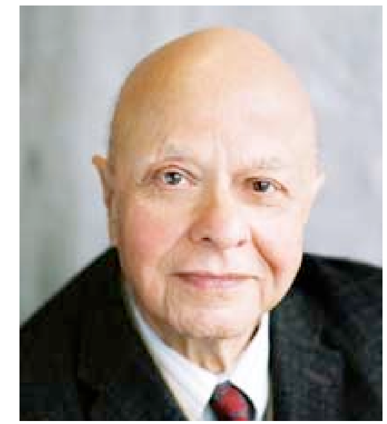
I saw her duck.

Aravind Joshi

NLP is all about ambiguities

- to middle school kids: what does this sentence mean?





Aravind Joshi

NLP is all about ambiguities

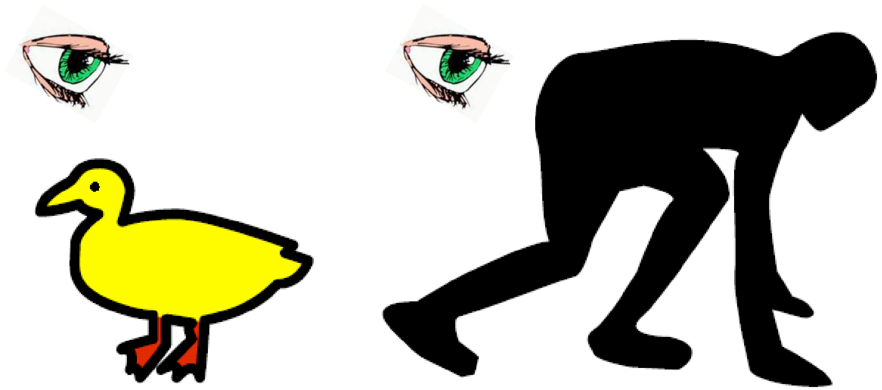
- to middle school kids: what does this sentence mean?

I eat sushi with tuna.

Aravind Joshi

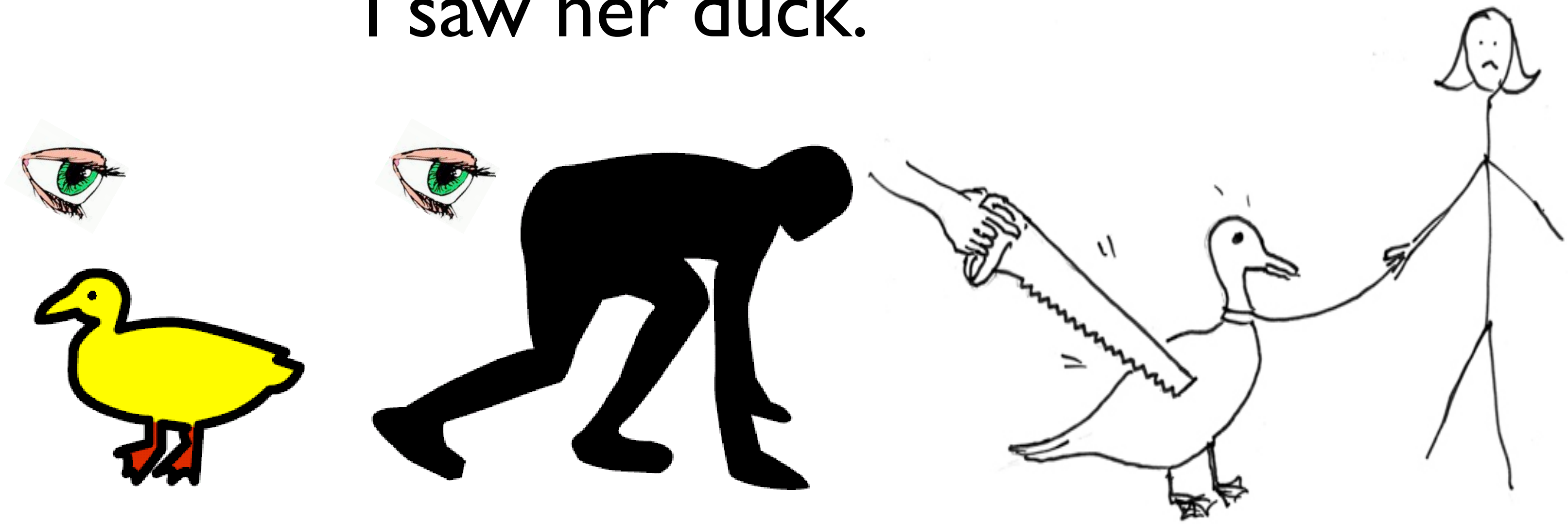
NLP is all about ambiguities

I saw her duck.



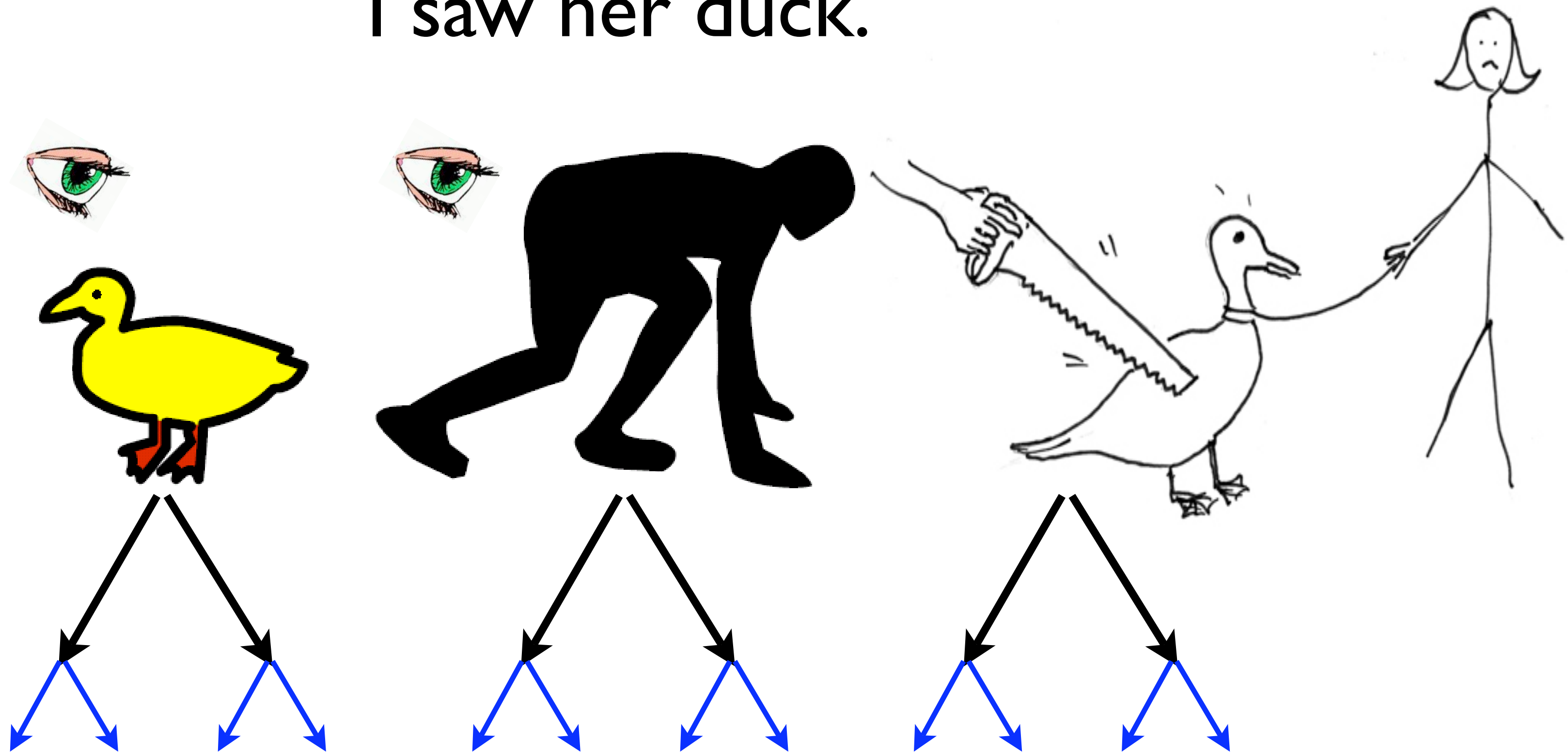
NLP is all about ambiguities

I saw her duck.



NLP is all about ambiguities

I saw her duck.

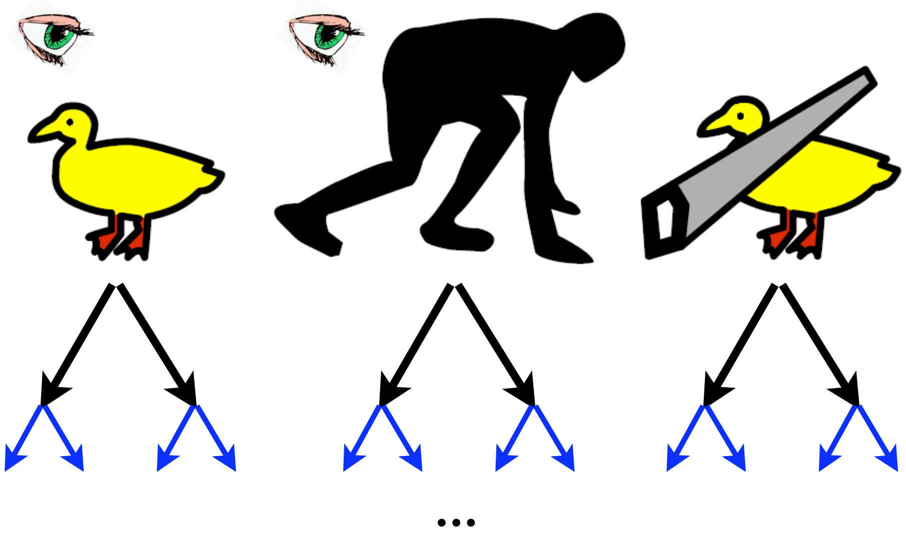


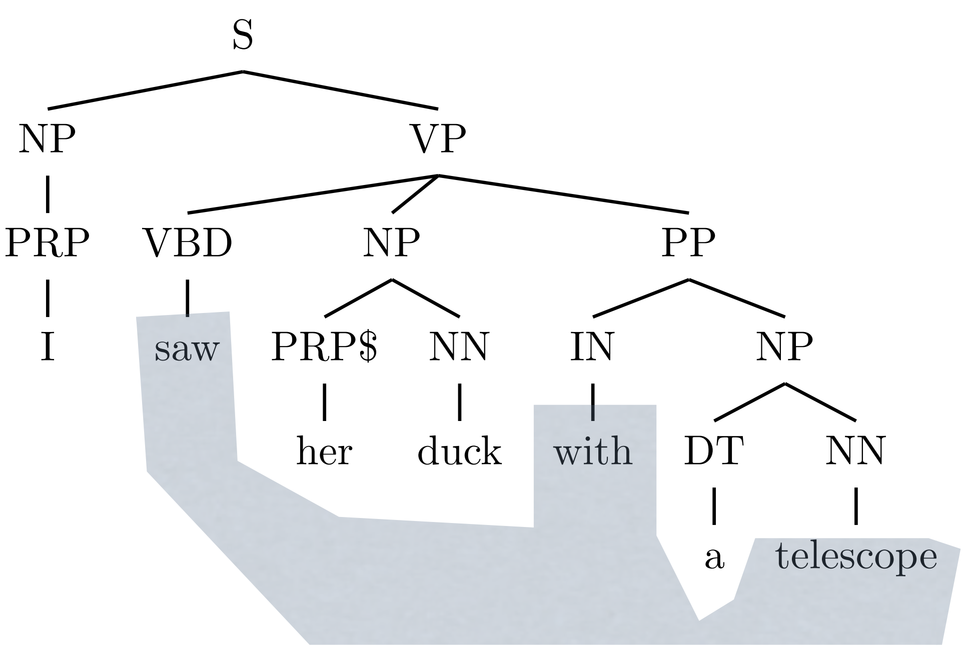
...

- how about...
  - I saw her duck with a telescope.
  - I saw her duck with a telescope in the garden...

NLP is HARD!

- exponential explosion of the search space
- non-local dependencies (context)





Ambiguities in Translation

zi　zhu　zhong　duan
自　助　终　端

self help terminal device

needs context to disambiguate!

Evil Rubbish;   Safety Export

needs context for fluency!

Key Problem

Key Problem

- How to efficiently incorporate non-local information?

Key Problem

- How to efficiently incorporate non-local information?
- Solution 1: pipelined reranking / rescoring
  - postpone disambiguation by propagating *k*-best lists
  - examples: tagging => parsing => semantics
  - (open) need efficient algorithms for *k*-best search

Key Problem

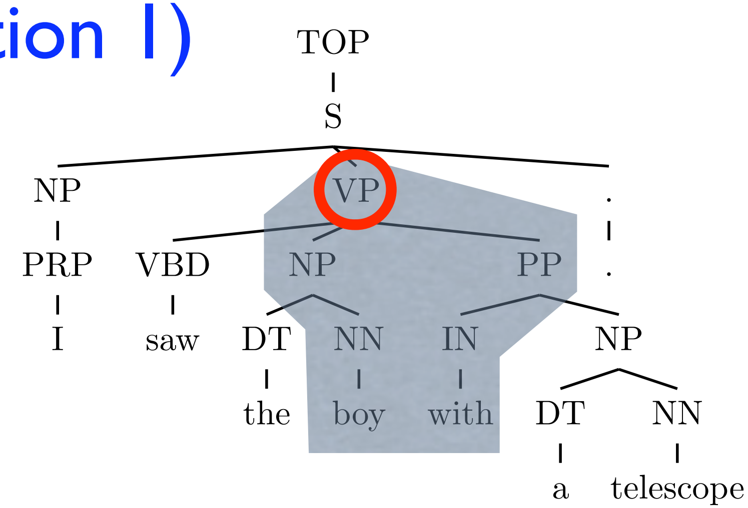
- How to efficiently incorporate non-local information?
- Solution 1: pipelined reranking / rescoring
  - postpone disambiguation by propagating *k*-best lists
  - examples: tagging => parsing => semantics
  - (open) need efficient algorithms for *k*-best search
- Solution 2: exact joint search on a much larger space
  - examples: head/parent annotations; often intractable

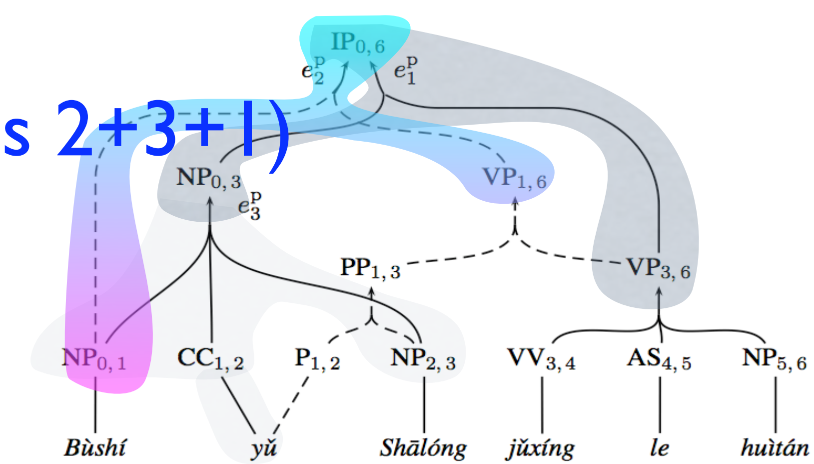
Key Problem

- How to efficiently incorporate non-local information?
- Solution 1: pipelined reranking / rescoring
  - postpone disambiguation by propagating *k*-best lists
  - examples: tagging => parsing => semantics
  - (open) need efficient algorithms for *k*-best search
- Solution 2: exact joint search on a much larger space
  - examples: head/parent annotations; often intractable
- Solution 3: approximate joint search (focus of this talk)
  - (open) integrate non-local information on the fly

Outline

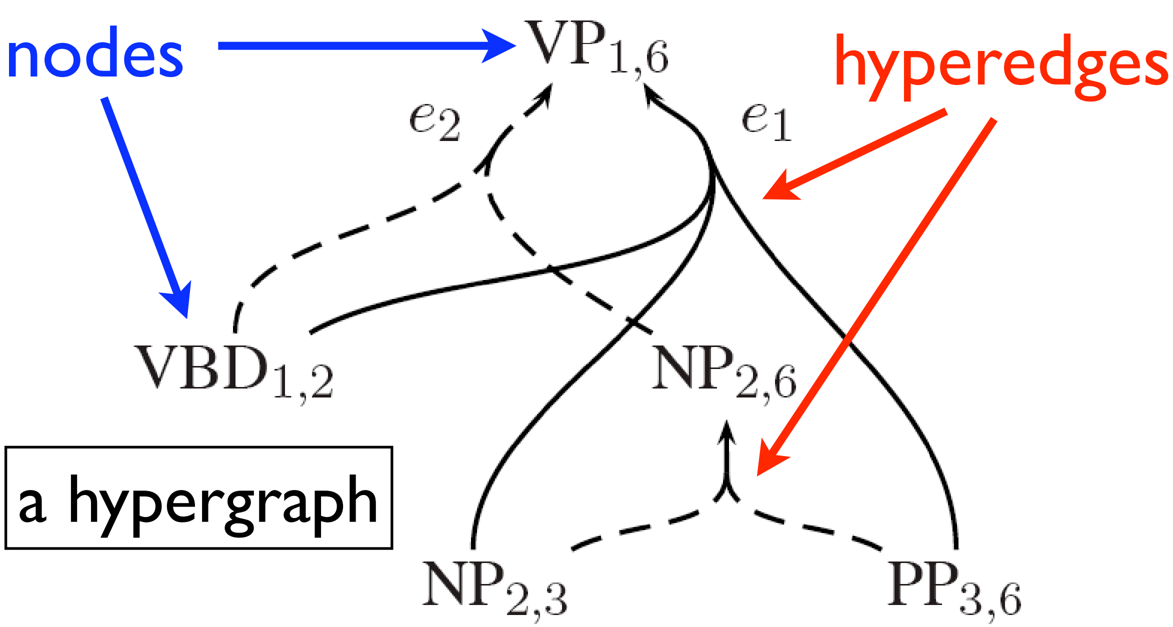
- Forest: Packing Exponential Ambiguities
- Exact *k*-best Search in Forest (Solution 1)
- Approximate Joint Search with Non-Local Features (Solution 3)
  - Forest Reranking
  - Forest Rescoring
- Forest-based Translation (Solutions 2+3+1)
  - Tree-based Translation
  - Forest-based Decoding





Packed Forests

- a compact representation of many parses
  - by sharing common sub-derivations
  - polynomial-space encoding of exponentially large set



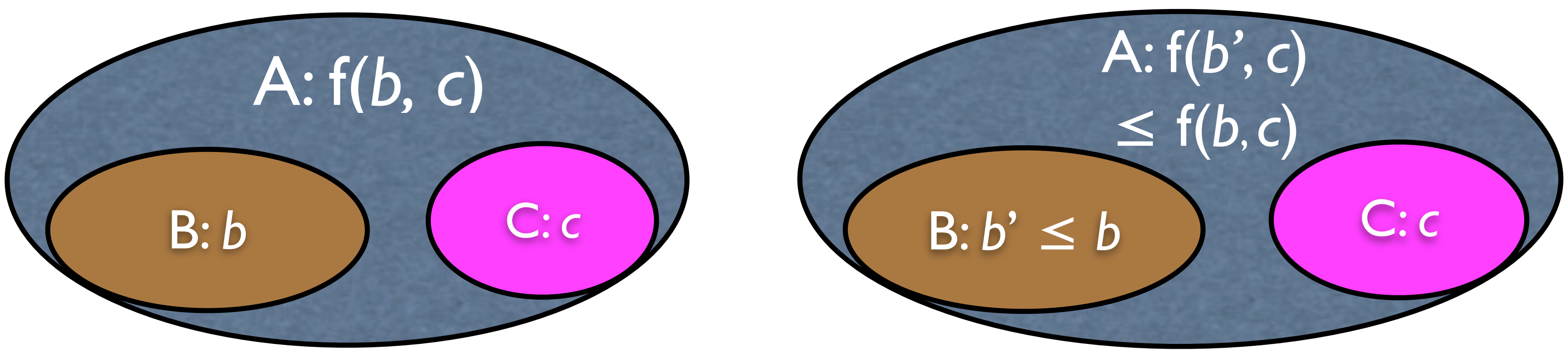
${}_0$ I ${}_1$ saw ${}_2$ him ${}_3$ with ${}_4$ a ${}_5$ mirror ${}_6$

$$e_1 \quad \frac{\mathrm{VBD}_{1,2} \quad \mathrm{NP}_{2,3} \quad \mathrm{PP}_{3,6}}{\mathrm{VP}_{1,6}}$$

(Klein and Manning, 2001; Huang and Chiang, 2005)

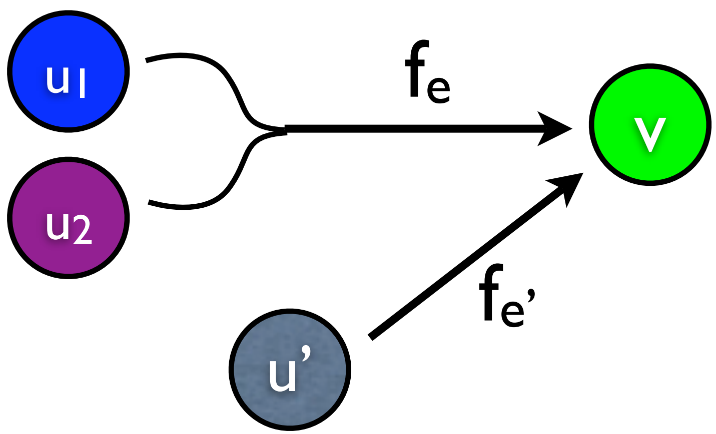
Weight Functions

- Each hyperedge e has a weight function f_e
  - monotonic in each argument
  - e.g. in CKY, $f_e(a, b) = a \times b \times \Pr(\text{rule})$
- optimal subproblem property in dynamic programming
  - optimal solutions include optimal sub-solutions



1-best Viterbi on Forest

1. topological sort (assumes acyclicity)
2. visit each node v in sorted order and do updates
   - for each incoming hyperedge $e = ((u_1, .., u_{|e|}), v, f_e)$
   - use $d(u_i)$'s to update $d(v)$
   - key observation: $d(u_i)$'s are fixed to optimal at this time

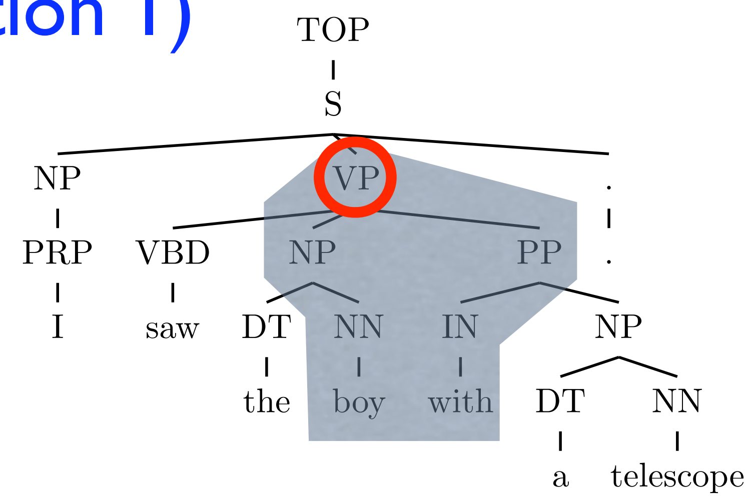


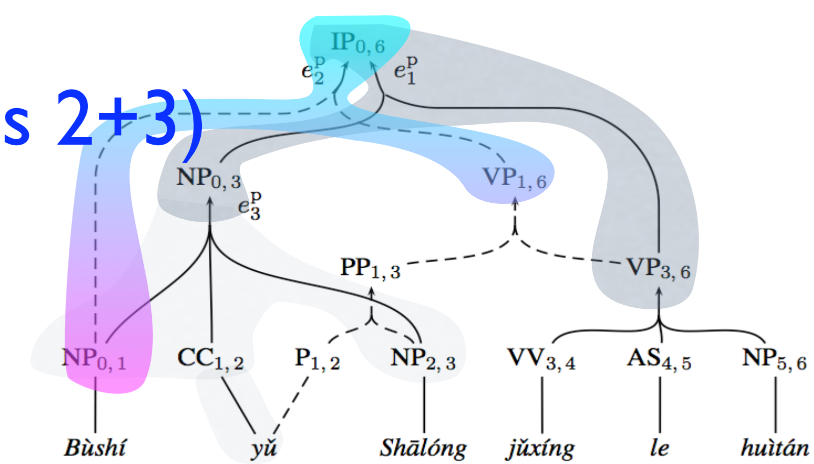
$$d(v) \oplus = f_e(d(u_1), \cdots, d(u_{|e|}))$$

- time complexity: O(V+E) = O(E)   for CKY: $O(n^3)$

Outline

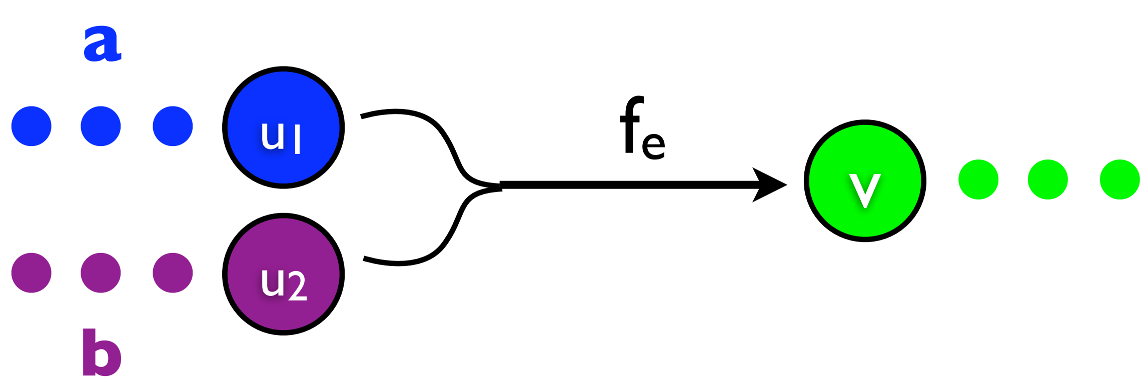
- Forest: Packing Exponential Ambiguities
- Exact *k*-best Search in Forest (Solution 1)
- Approximate Joint Search with Non-Local Features (Solution 3)
  - Forest Reranking
  - Forest Rescoring
- Forest-based Translation (Solutions 2+3)
  - Tree-based Translation
  - Forest-based Decoding

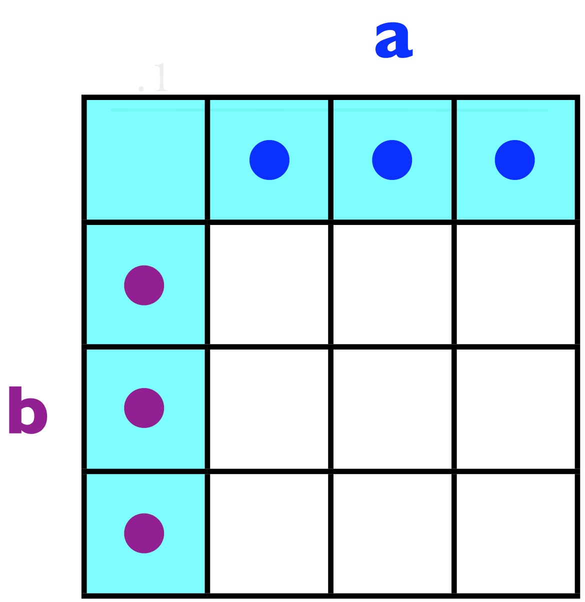




k-best Viterbi Algorithm 0

- straightforward k-best extension
  - a vector of k (sorted) values for each node
  - now what's the result of f_e (**a**, **b**) ?
    - $k \times k = k^2$ possibilities! => then choose top k

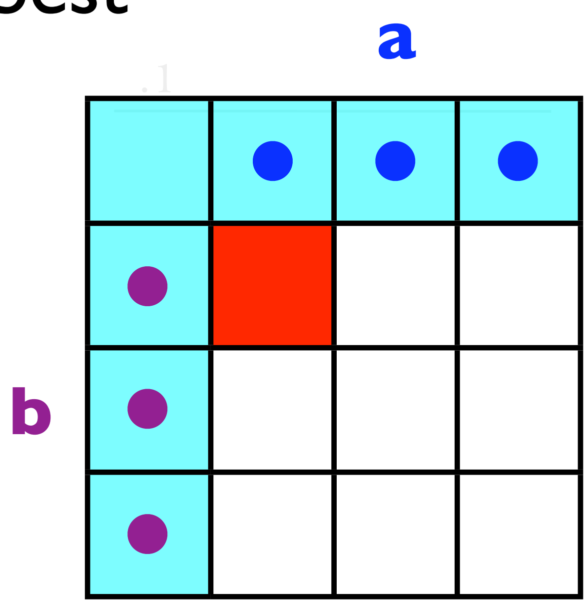




- time complexity: $O(k^2 E)$

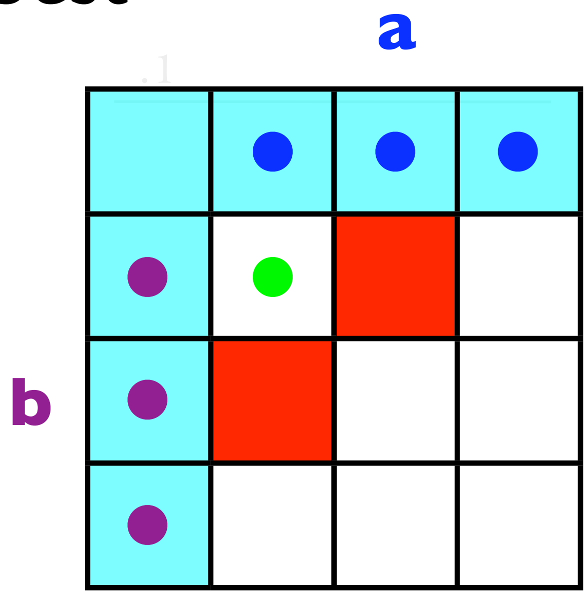
k-best Viterbi Algorithm I

- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: $O(k \log k\ E)$



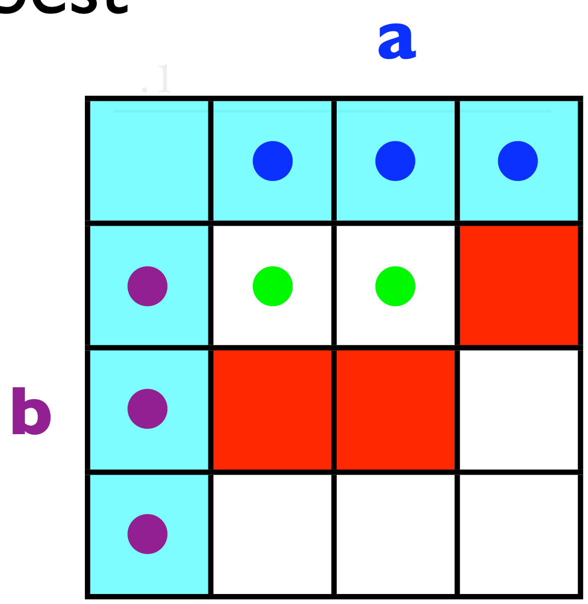
k-best Viterbi Algorithm 1

- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: $O(k \log k\, E)$



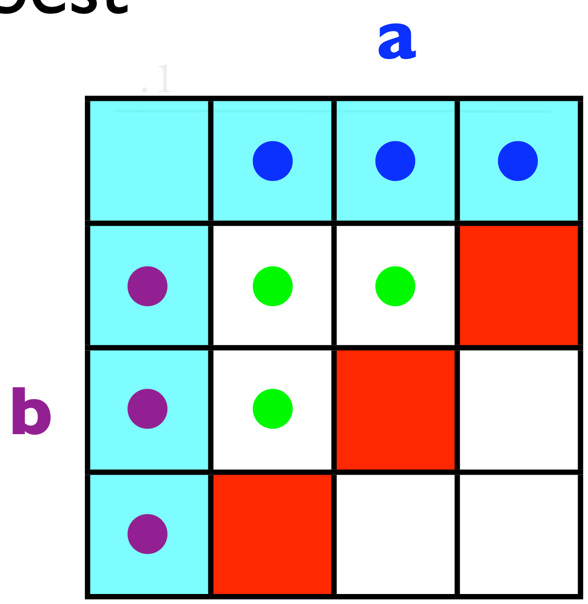
k-best Viterbi Algorithm 1

- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: $O(k \log k\ E)$



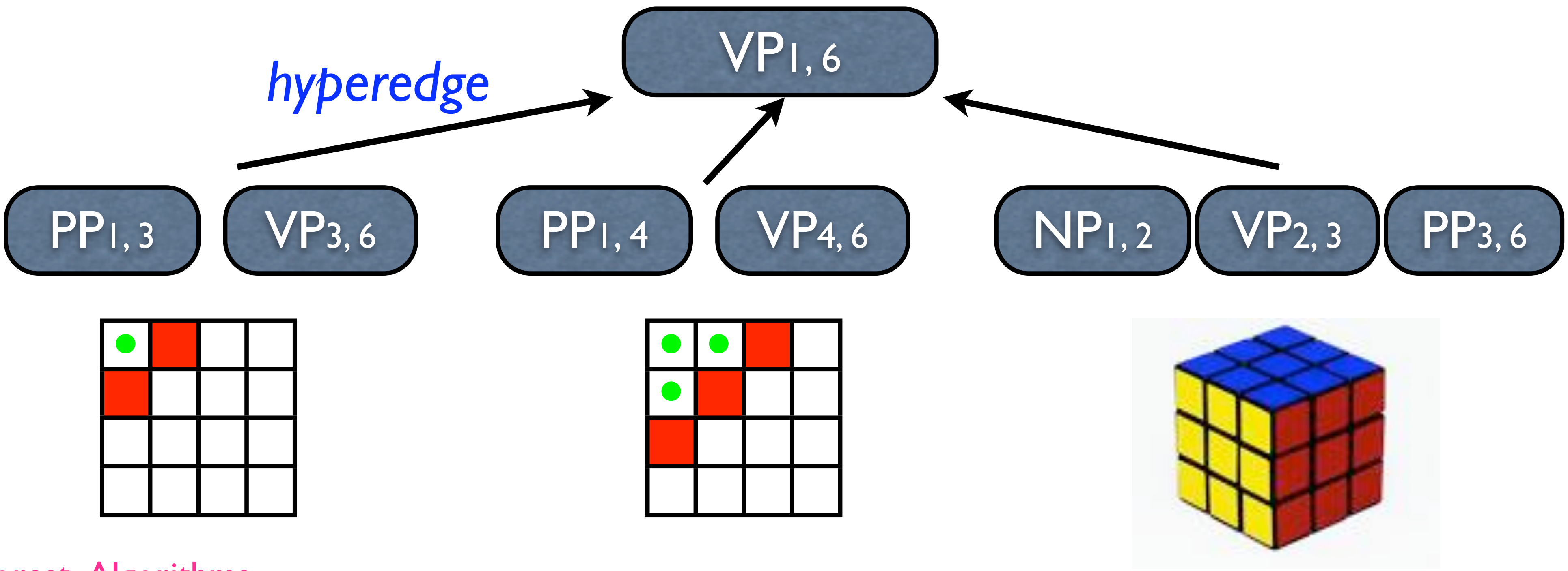
k-best Viterbi Algorithm 1

- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: $O(k \log k\ E)$



k-best Viterbi Algorithm 2

- Algorithm 1 works on each hyperedge sequentially
  - O(*k* log *k* E) is still too slow for big *k*
- Algorithm 2 processes all hyperedges in parallel
  - dramatic speed-up: O(E + V *k* log *k*)



k-best Viterbi Algorithm 3

- Algorithm 2 computes k-best for each node
  - but we are only interested in k-best of the root node
- Algorithm 3 computes as many as really needed
  - forward-phase
    - same as 1-best Viterbi, but stores the forest (keeping alternative hyperedges)
  - backward-phase
    - recursively asking "what's your 2^{nd}-best" top-down
    - asks for more when need more

Summary of Algorithms

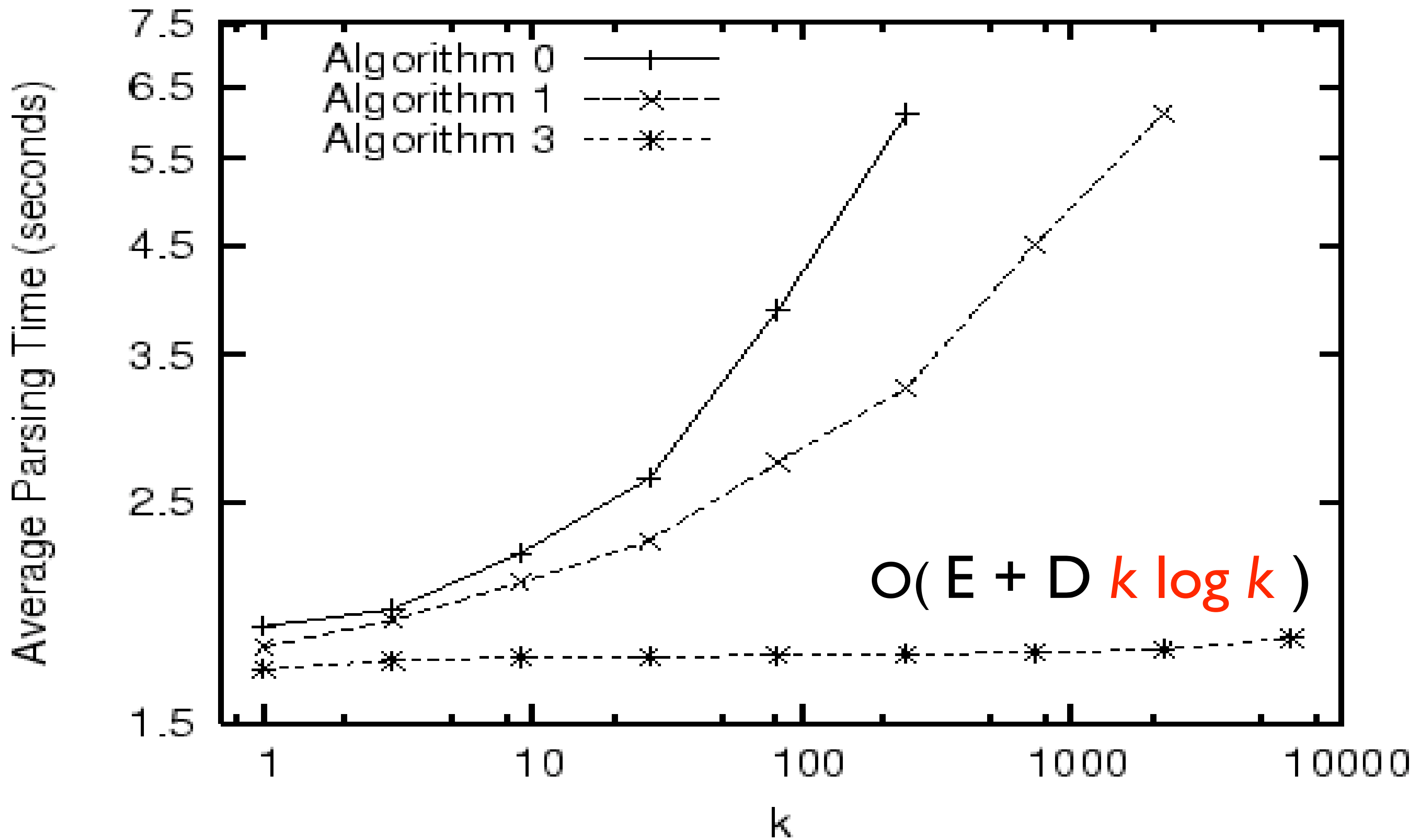
- Algorithms 1 => 2 => 3
  - lazier and lazier (computation on demand)
  - larger and larger locality
  - Algorithm 3 is very fast, but requires storing forest

	locality	time	space
Algorithm 1	hyperedge	$O(E\, k \log k)$	$O(k V)$
Algorithm 2	node	$O(E + V\, k \log k)$	$O(k V)$
Algorithm 3	global	$O(E + D\, k \log k)$	$O(E + k D)$

E - hyperedges: $O(n^3)$; V - nodes: $O(n^2)$; D - derivation: $O(n)$

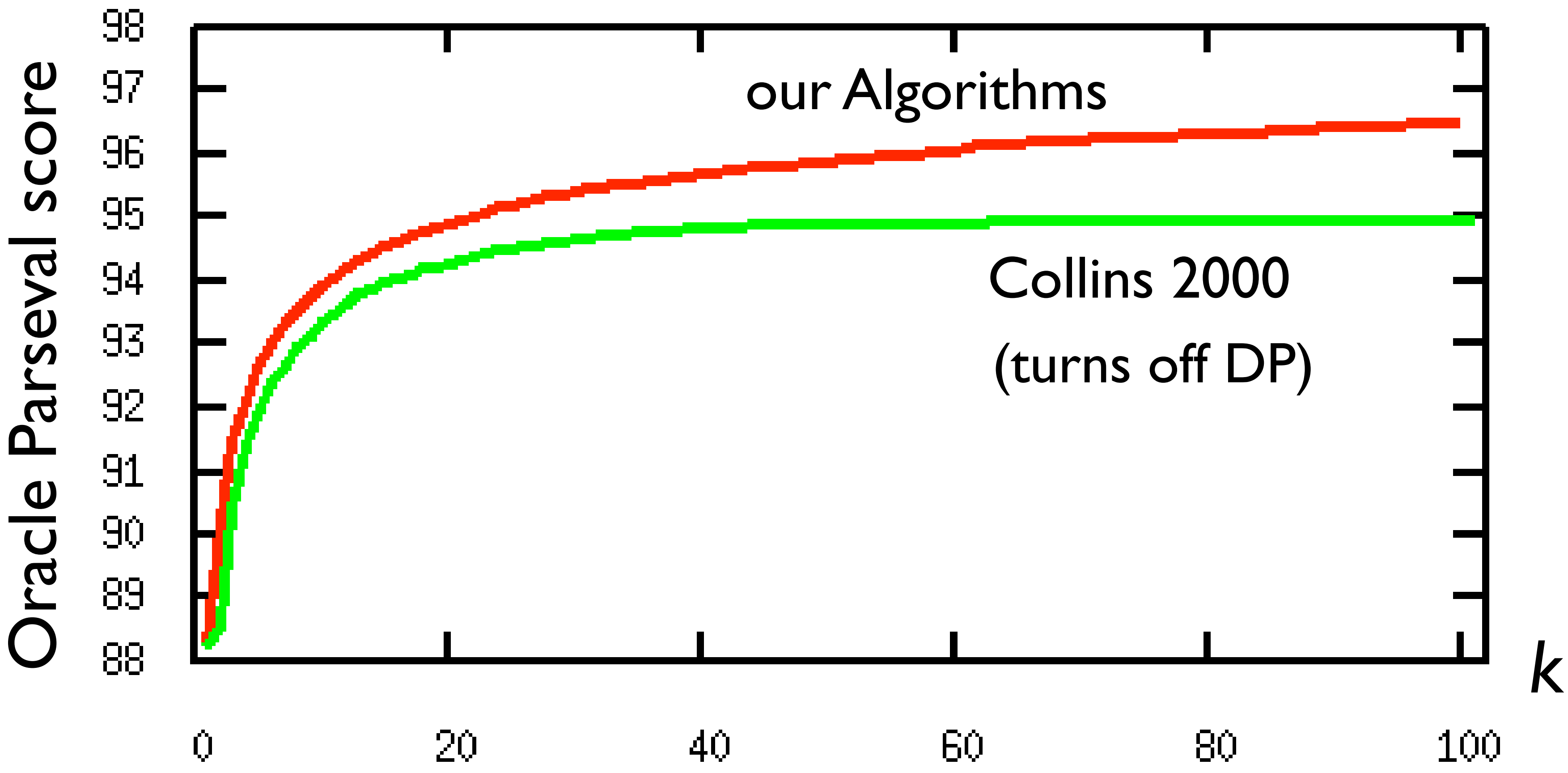
Experiments - Efficiency

- on state-of-the-art Collins/Bikel parser (Bikel, 2004)
- average parsing time per sentence using Algs. 0, 1, 3



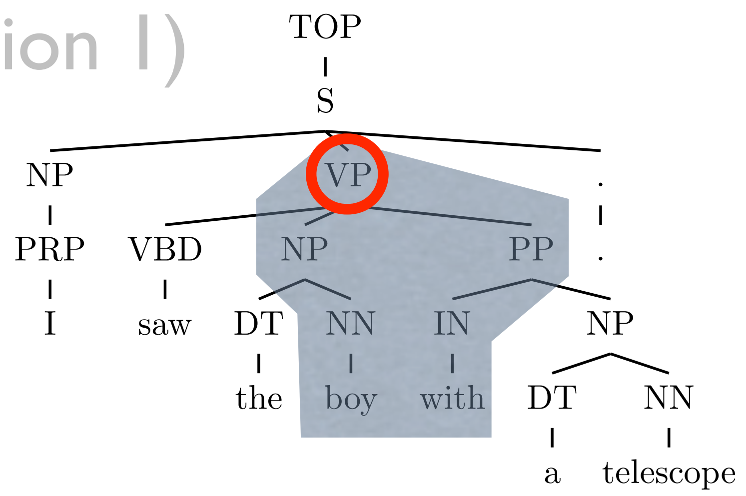
Reranking and Oracles

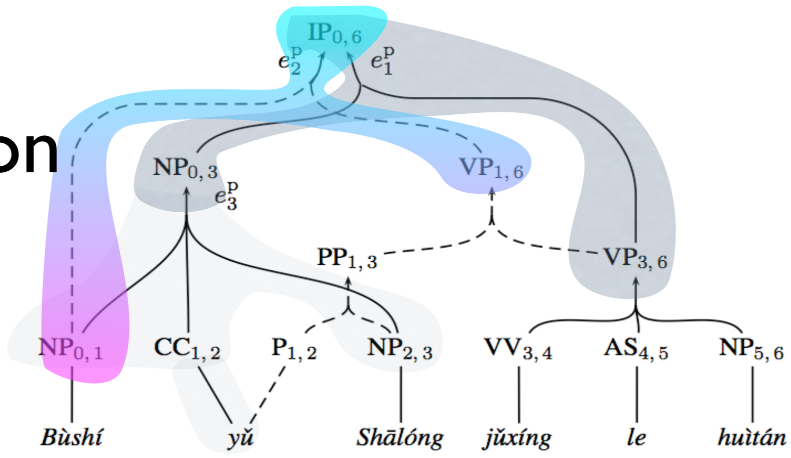
- oracle - the candidate closest to the correct parse among the *k*-best candidates
- measures the potential of real reranking



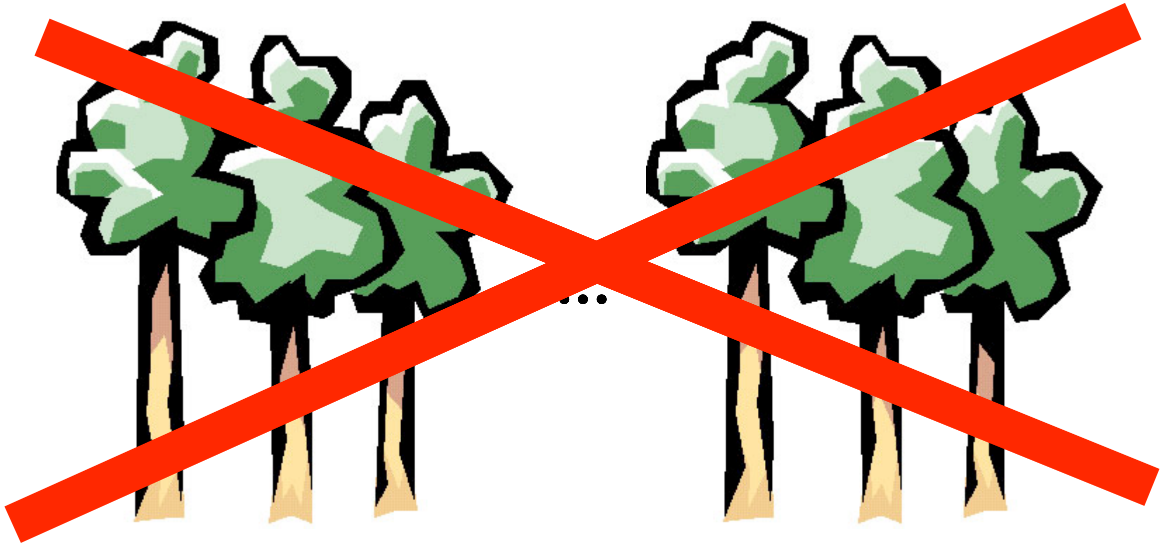
Outline

- Packed Forests and Hypergraph Framework
- Exact k-best Search in Forest (Solution 1)
- Approximate Joint Search with Non-Local Features (Solution 3)
    - Forest Reranking
    - Forest Rescoring
- Application: Forest-based Translation
    - Tree-based Translation
    - Forest-based Decoding





Why not *k*-best reranking?



- too few variations (limited scope)
  - 41% correct parses are not in ~30-best (Collins, 2000)
  - worse for longer sentences
- too many redundancies
  - 50-best usually encodes 5-6 binary decisions ($2^5<50<2^6$)

Redundancies in n-best lists

Not all those who wrote oppose the changes.

(TOP (S (NP (NP (RB Not) (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (RB Not) (NP (NP (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

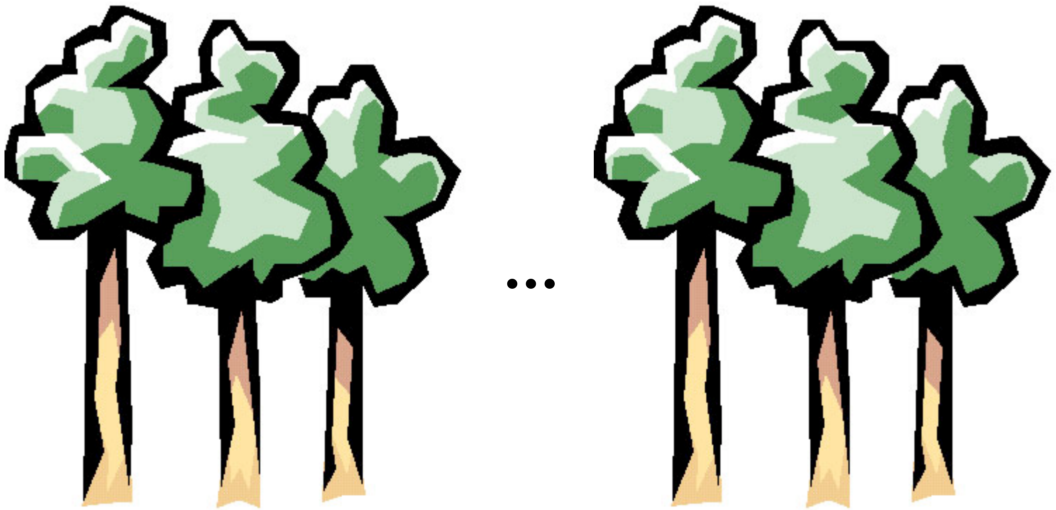
(TOP (S (NP (NP (RB Not) (DT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (RB Not) (NP (NP (DT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (NP (NP (RB Not) (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VB oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (NP (NP (RB Not) (RB all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (RB Not) (NP (NP (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VB oppose) (NP (DT the) (NNS changes))) (. .)))



Redundancies in n-best lists

Not all those who wrote oppose the changes.

(TOP (S (NP (NP (RB Not) (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (RB Not) (NP (NP (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

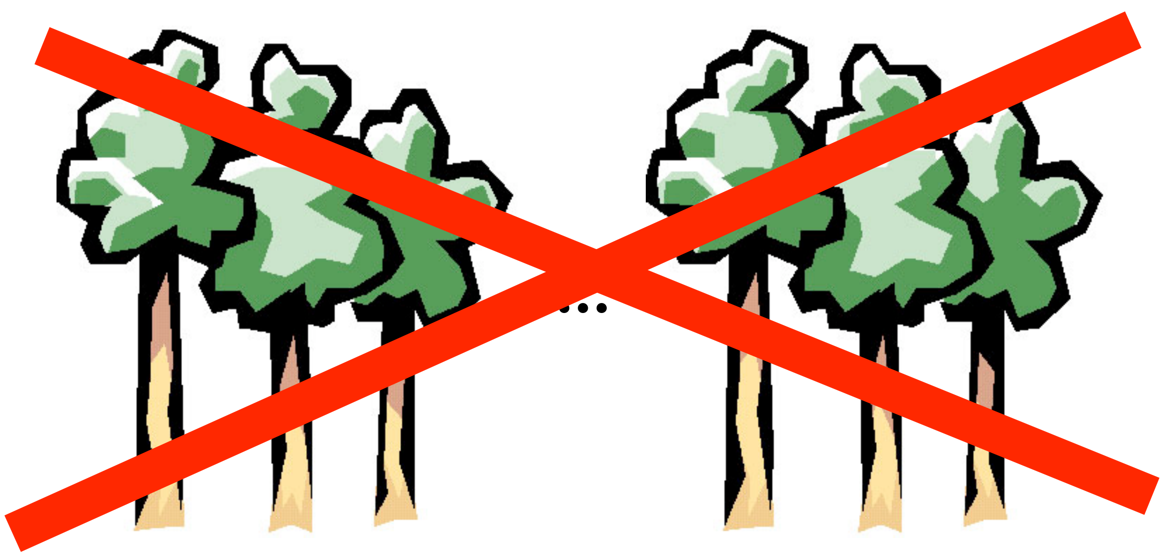
(TOP (S (NP (NP (RB Not) (DT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (RB Not) (NP (NP (DT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (NP (NP (RB Not) (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VB oppose) (NP (DT the) (NNS changes))) (. .)))

(TOP (S (NP (NP (RB Not) (RB all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VBP oppose) (NP (DT the) (NNS changes))) (. .)))

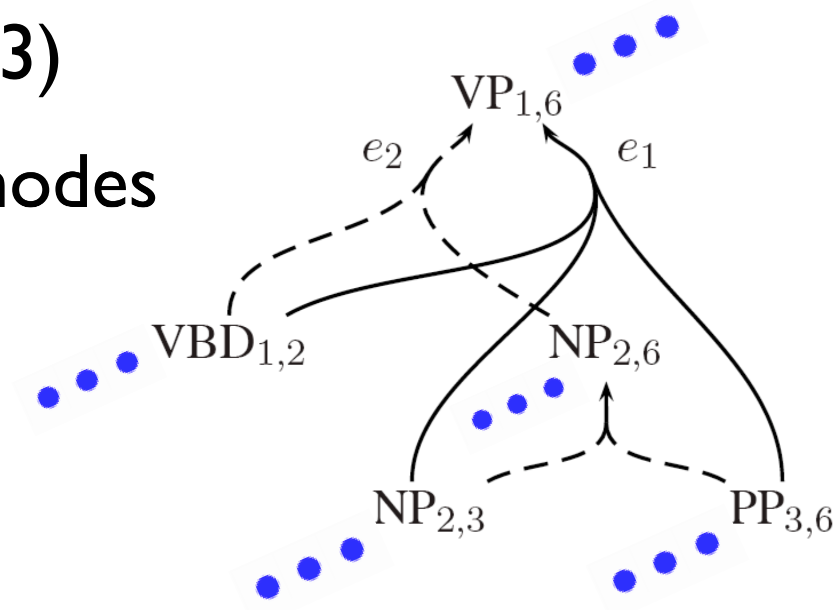
(TOP (S (RB Not) (NP (NP (PDT all) (DT those)) (SBAR (WHNP (WP who)) (S (VP (VBD wrote))))) (VP (VB oppose) (NP (DT the) (NNS changes))) (. .)))



packed forest

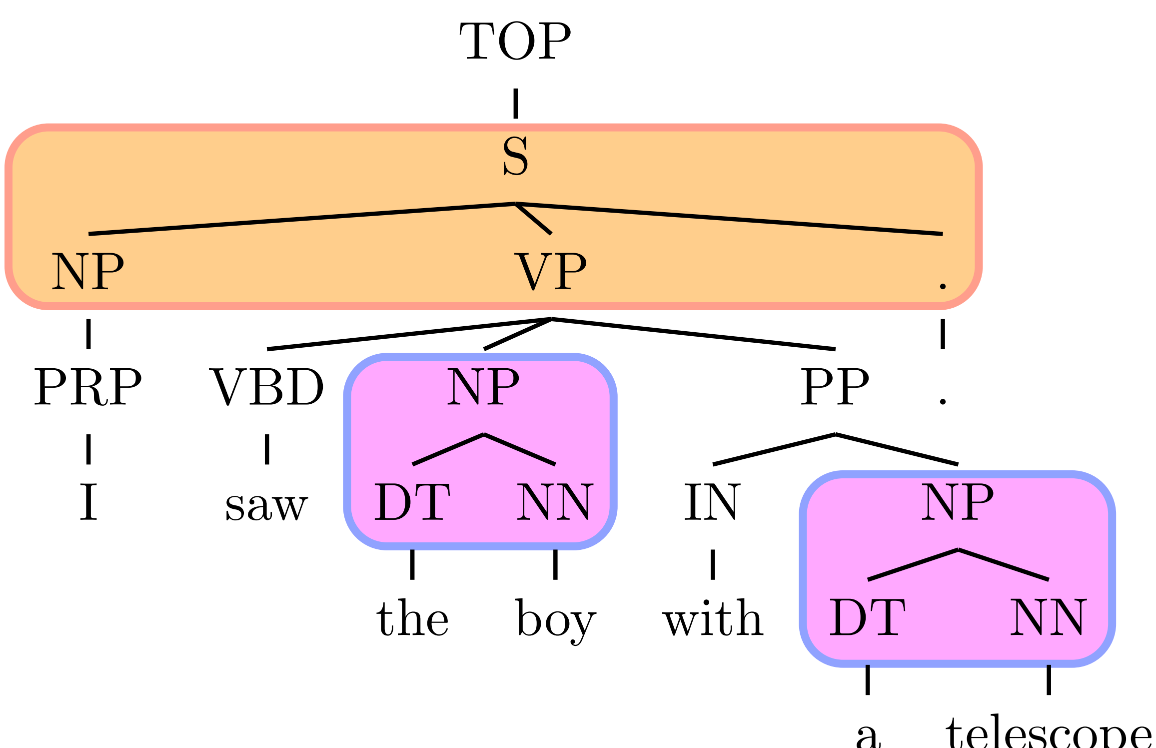
Reranking on a Forest?

- with only local features (Solution 2)
  - dynamic programming, exact, tractable (Taskar et al. 2004; McDonald et al., 2005)
- with non-local features (Solution 3)
  - on-the-fly reranking at internal nodes
  - top *k* derivations at each node
  - use as many non-local features as possible at each node
  - chart parsing + discriminative reranking
- we use perceptron for simplicity



Features

- a feature f is a function from tree y to a real number
  - $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser
  - every other feature *counts* the number of times a particular configuration occurs in y



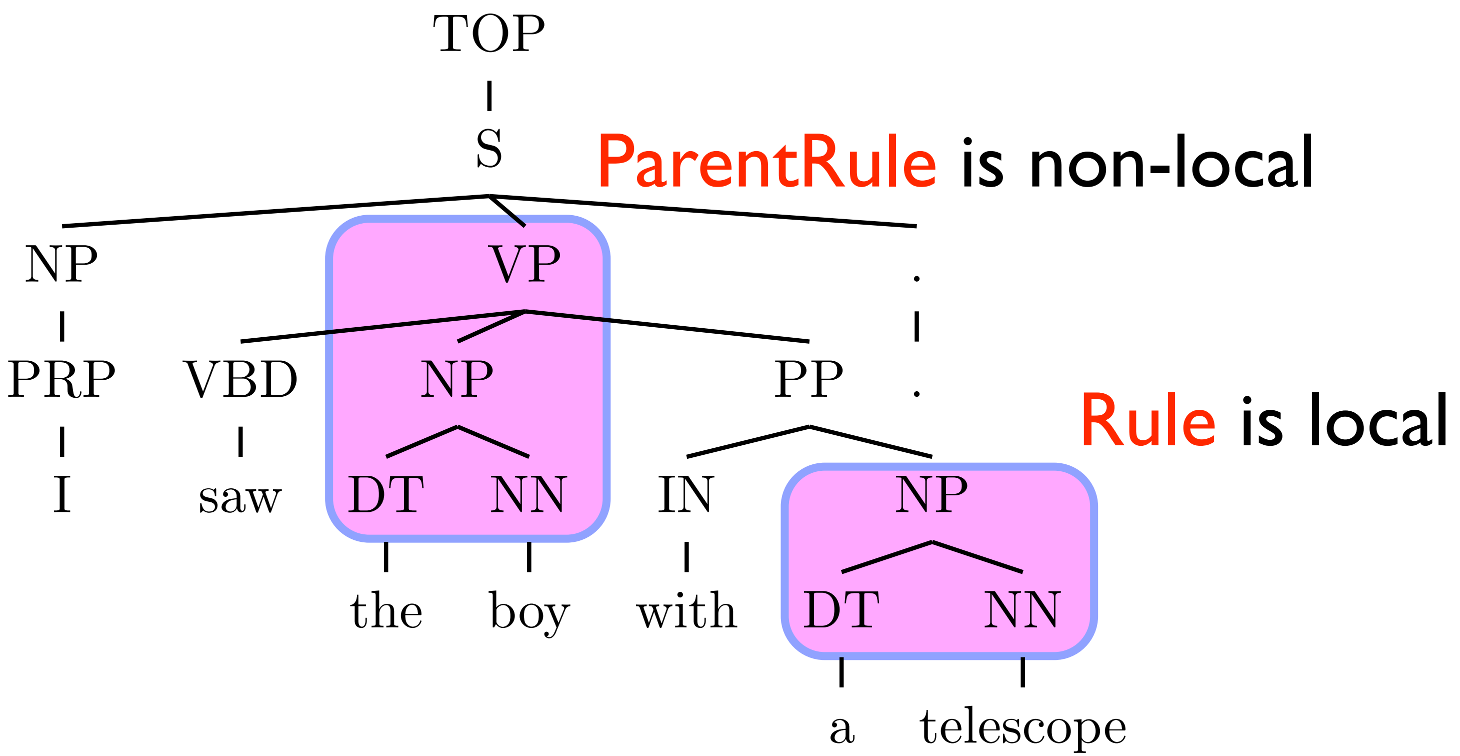
our features are from
(Charniak & Johnson, 2005)
(Collins, 2000)

instances of Rule feature

$$f_{100}(y) = f_{S \to NP\ VP\ .}(y) = 1$$
$$f_{200}(y) = f_{NP \to DT\ NN}(y) = 2$$

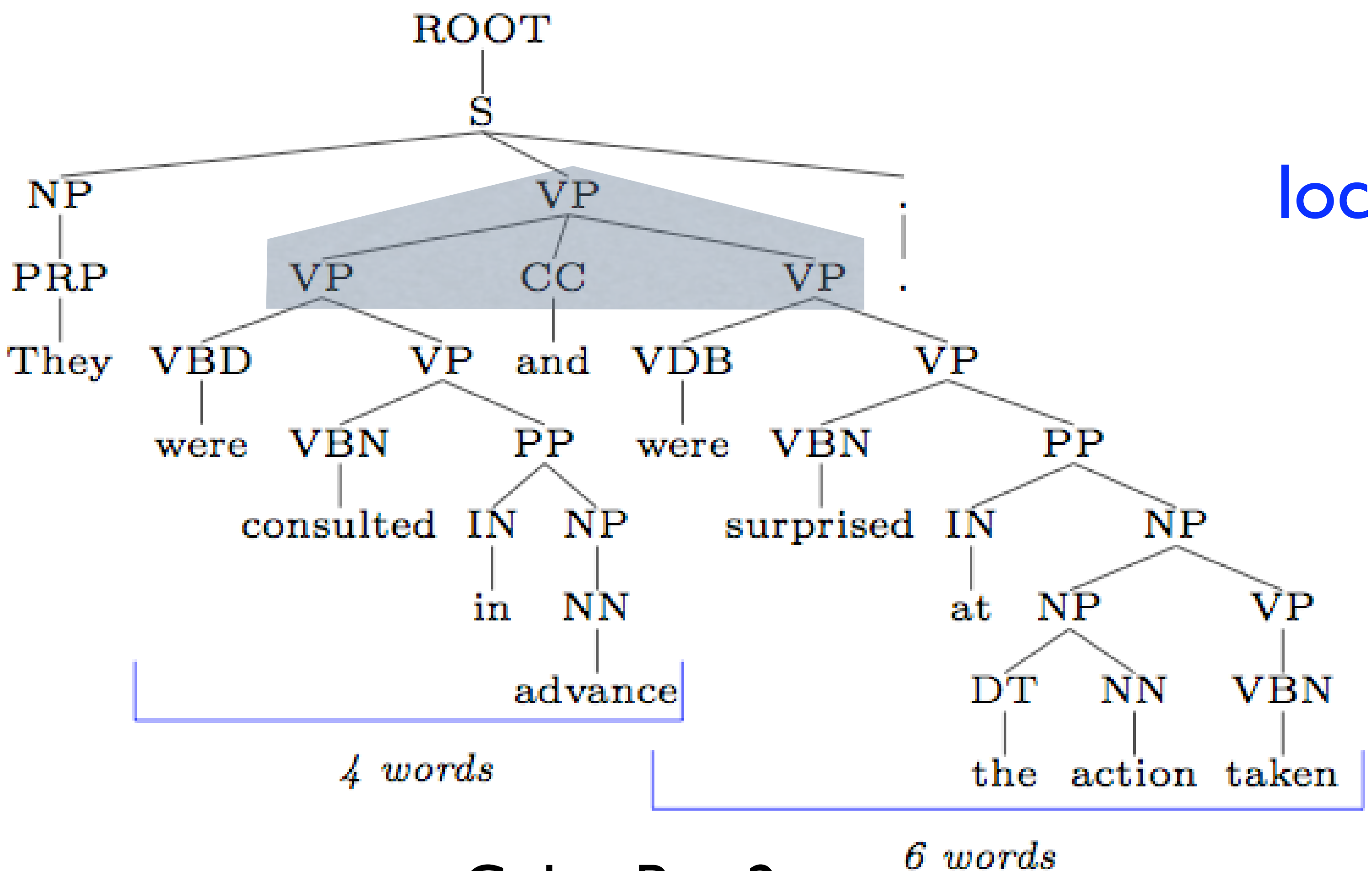
Local vs. Non-Local Features

- a feature is local iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest)
- local features can be pre-computed on each hyperedge in the forest; non-locals can not



Local vs. Non-Local: Examples

- CoLenPar feature captures the difference in lengths of adjacent conjuncts (Charniak and Johnson, 2005)

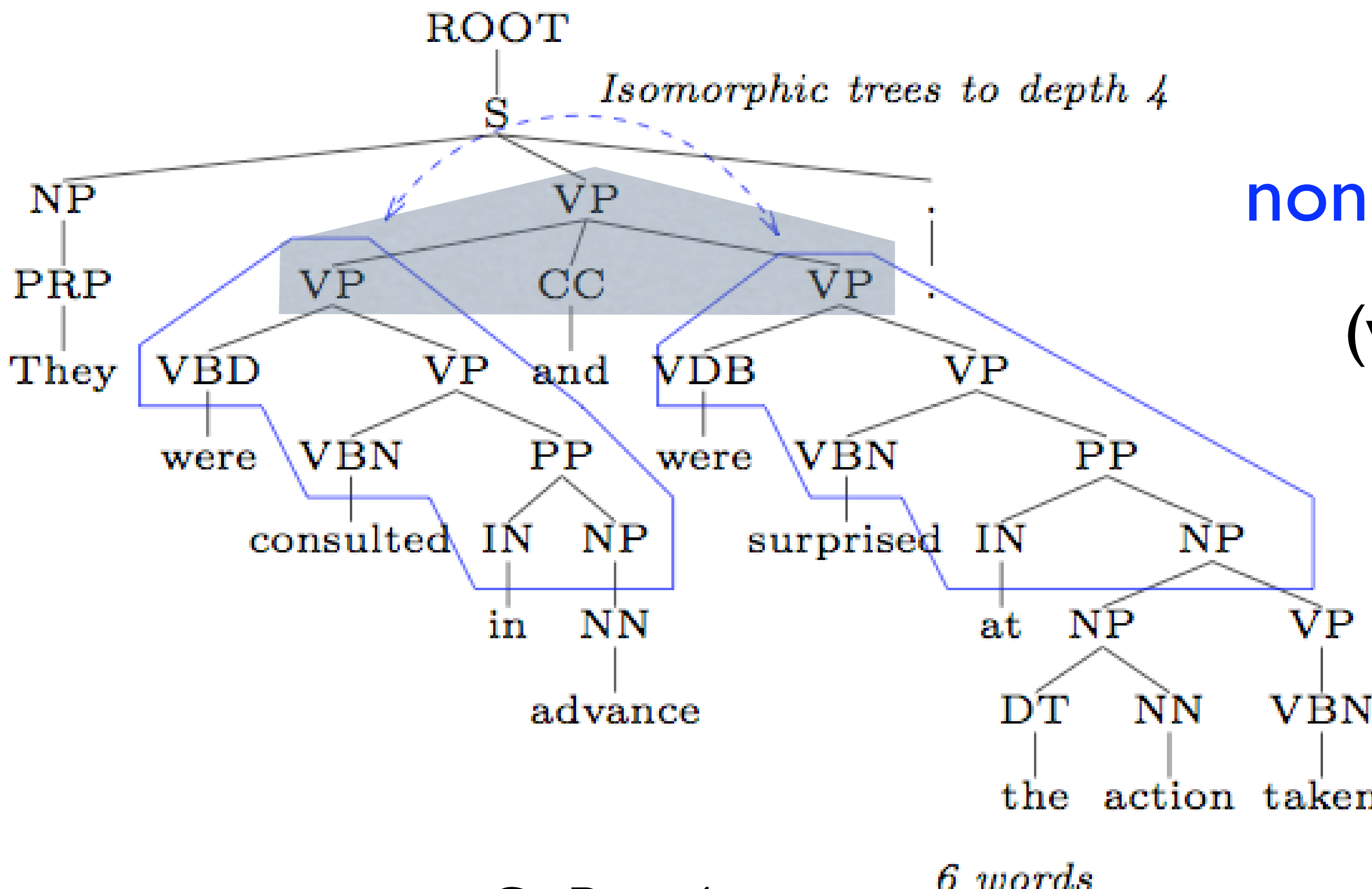


local!

CoLenPar: 2

Local vs. Non-Local: Examples

- CoPar feature captures the depth to which adjacent conjuncts are isomorphic (Charniak and Johnson, 2005)



non-local!

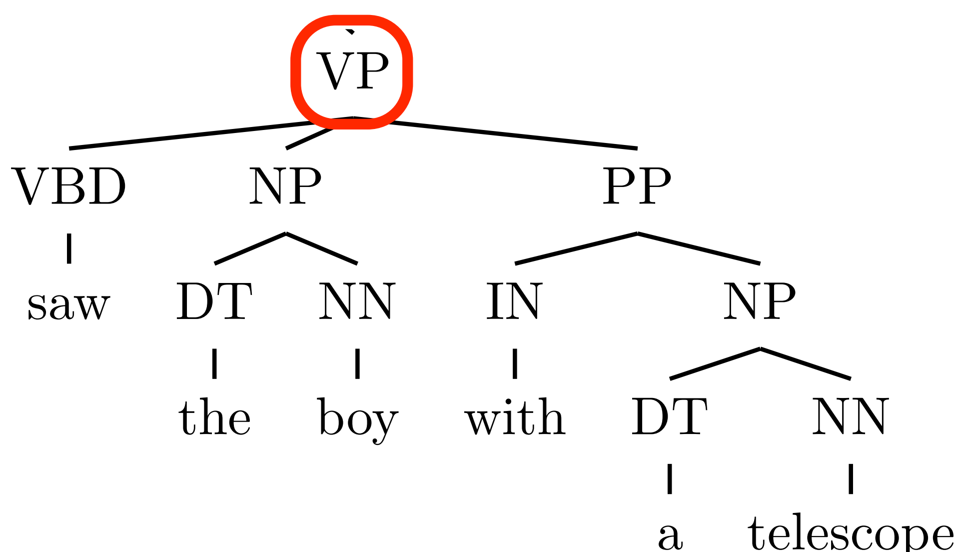
(violates DP principle)

CoPar: 4

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors

unit instance of ParentRule feature at VP node

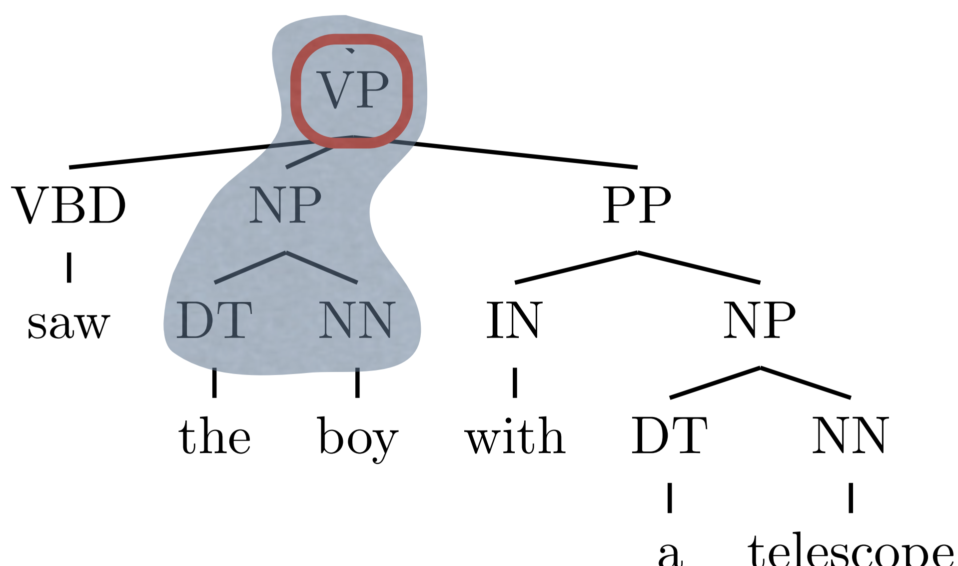


local features factor across hyperedges *statically*

non-local features factor across nodes *dynamically*

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



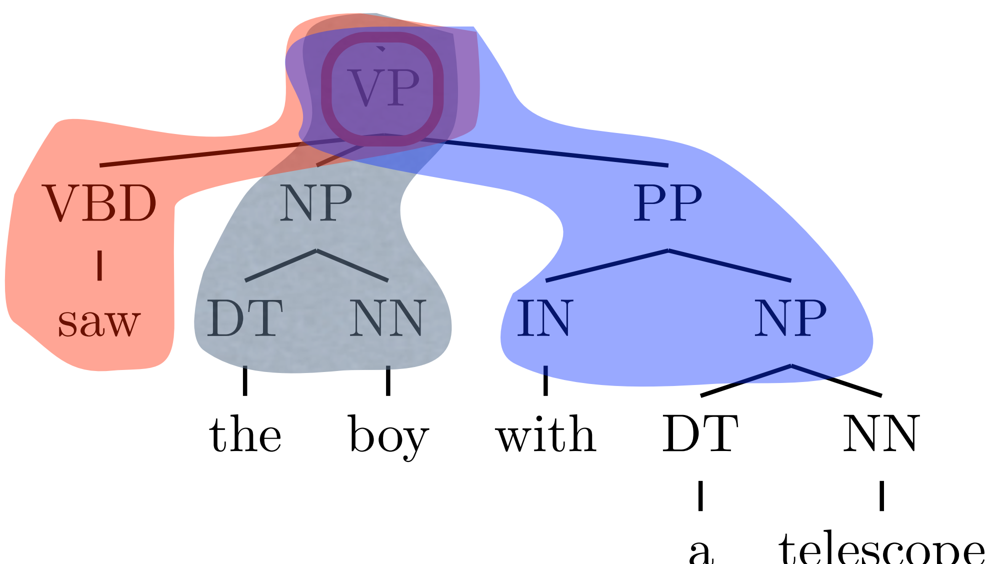
unit instance of ParentRule feature at VP node

local features factor across hyperedges *statically*

non-local features factor across nodes *dynamically*

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



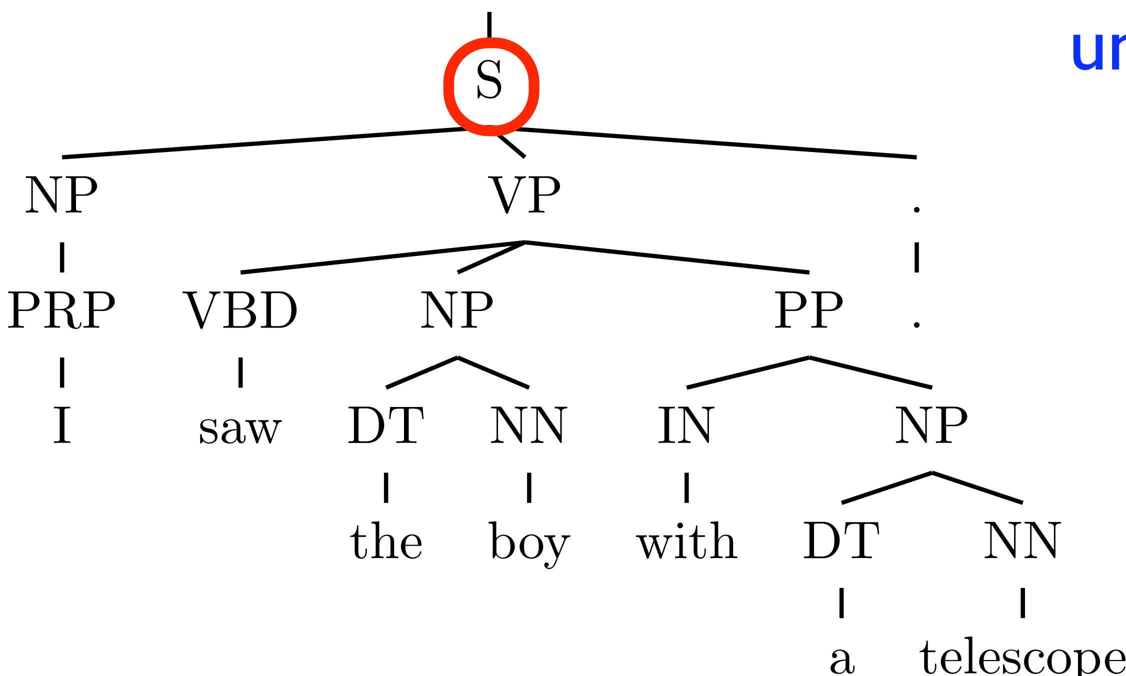
unit instance of ParentRule feature at VP node

local features factor across hyperedges *statically*

non-local features factor across nodes *dynamically*

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



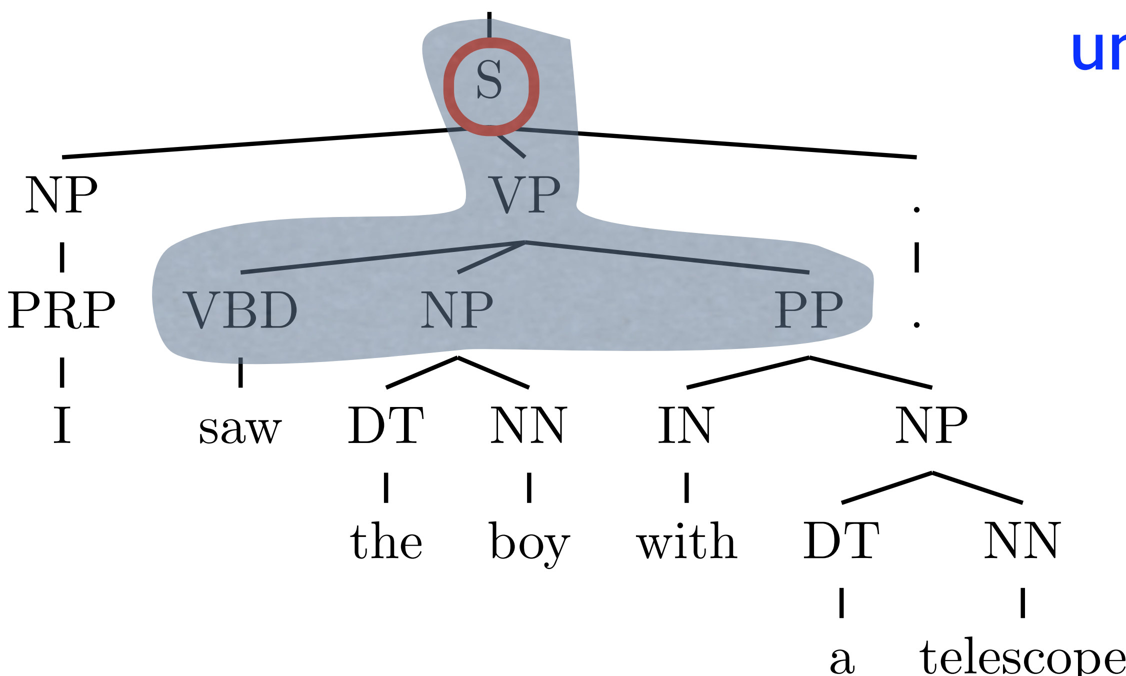
unit instance of ParentRule feature at S node

local features factor across hyperedges *statically*

non-local features factor across nodes *dynamically*

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



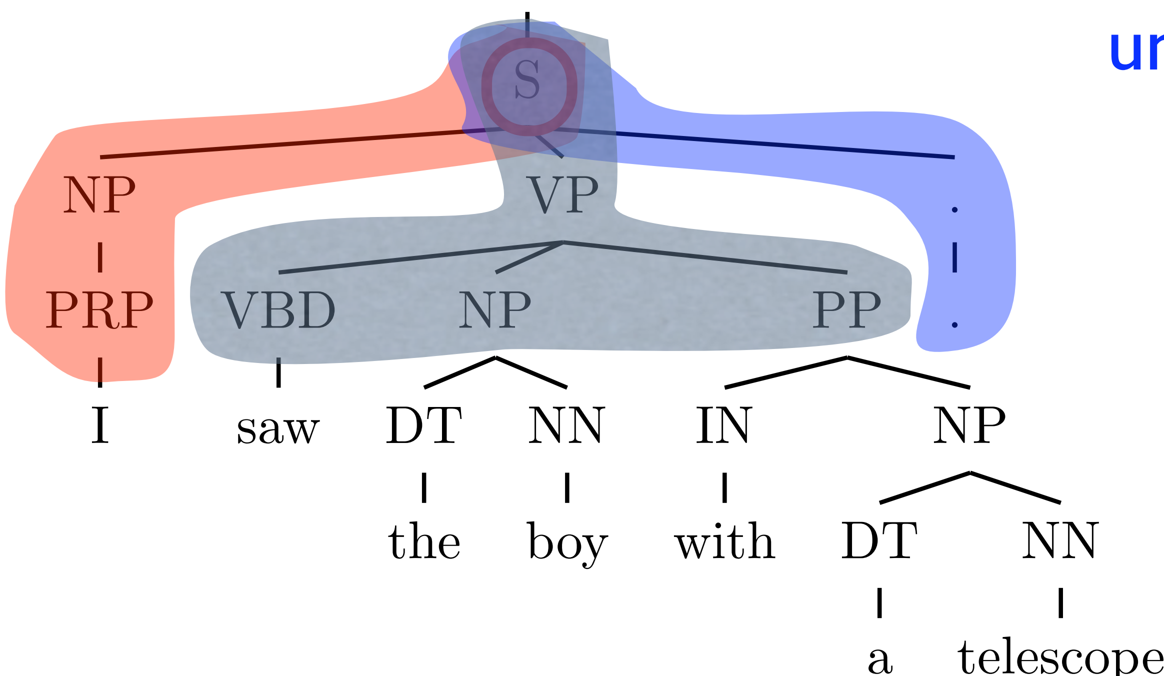
unit instance of ParentRule
feature at S node

local features factor across hyperedges *statically*

non-local features factor across nodes *dynamically*

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



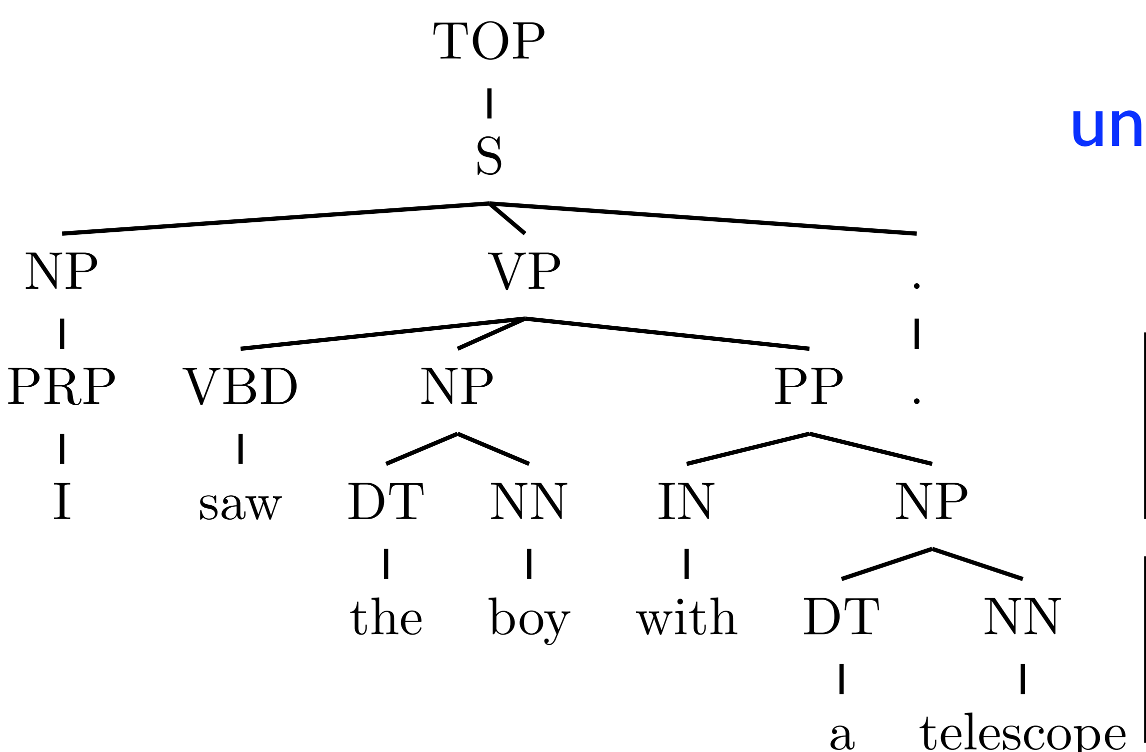
unit instance of ParentRule feature at S node

local features factor across hyperedges *statically*

non-local features factor across nodes *dynamically*

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



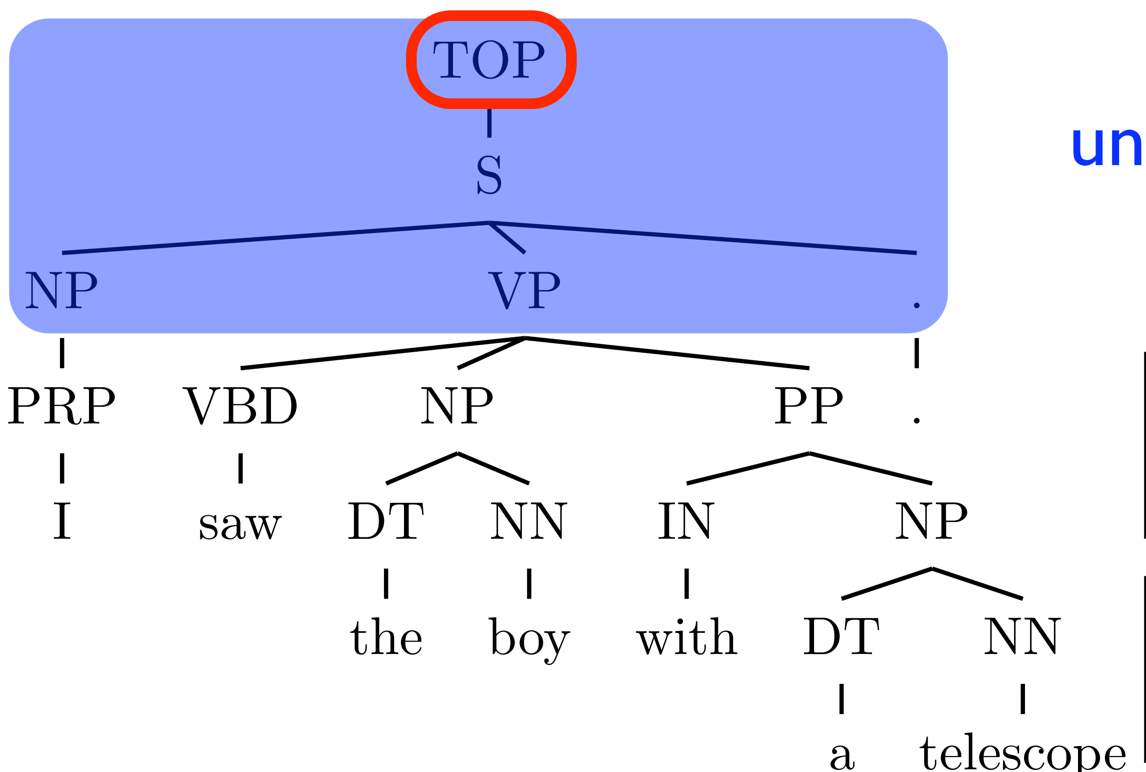
unit instance of ParentRule feature at TOP node

non-local features factor across nodes *dynamically*

local features factor across hyperedges *statically*

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



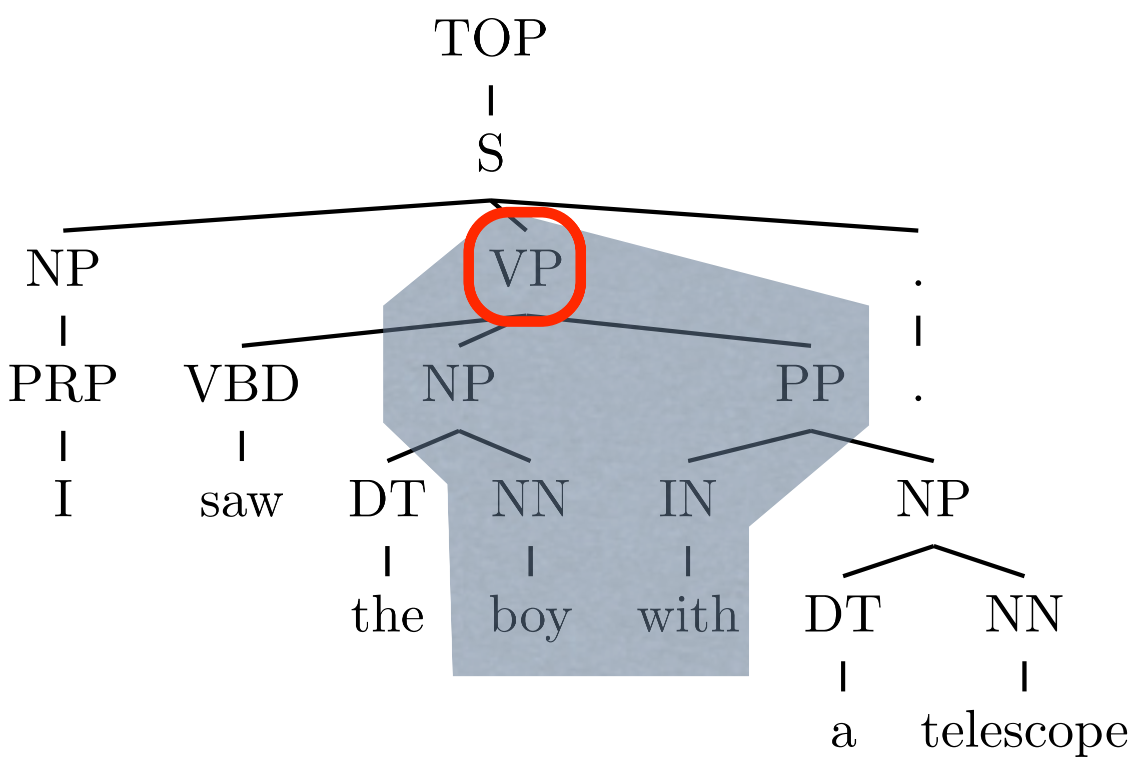
unit instance of ParentRule feature at TOP node

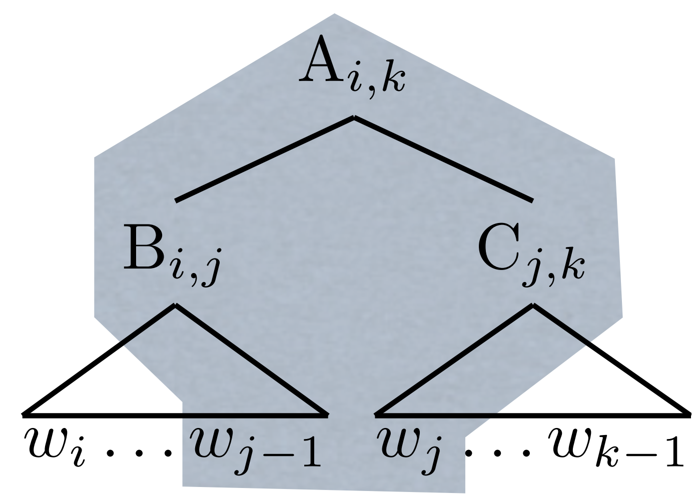
non-local features factor across nodes *dynamically*

local features factor across hyperedges *statically*

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

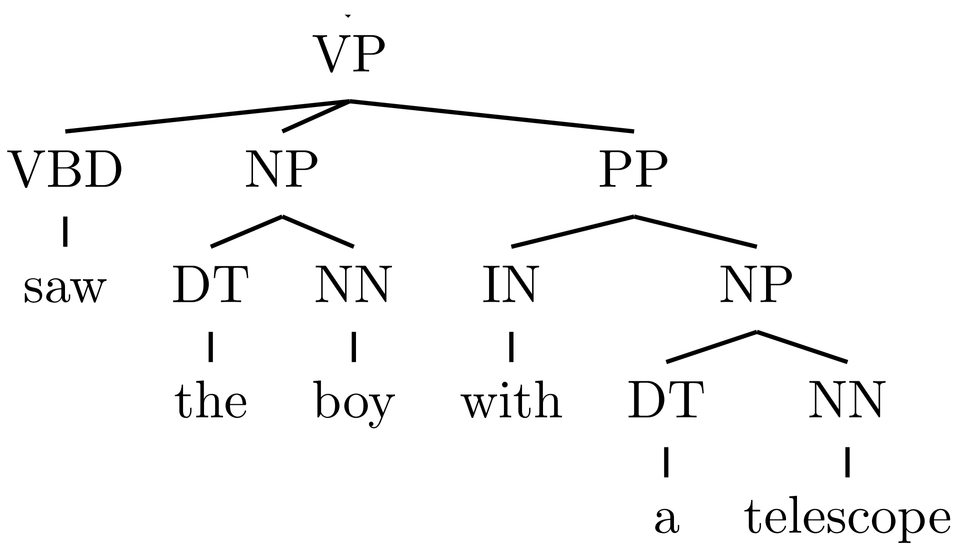


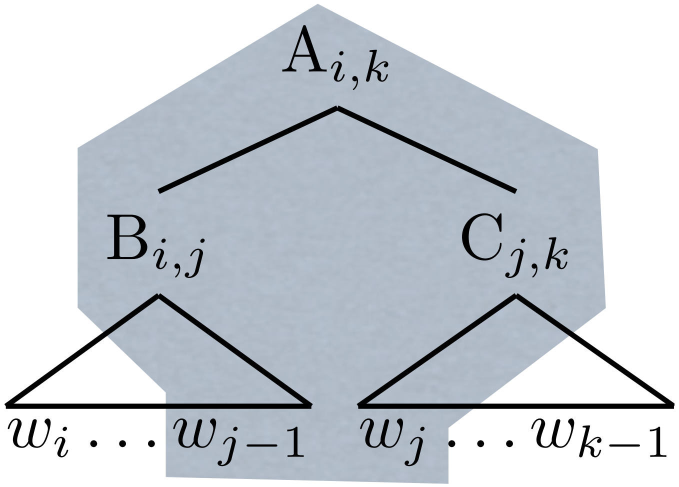


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

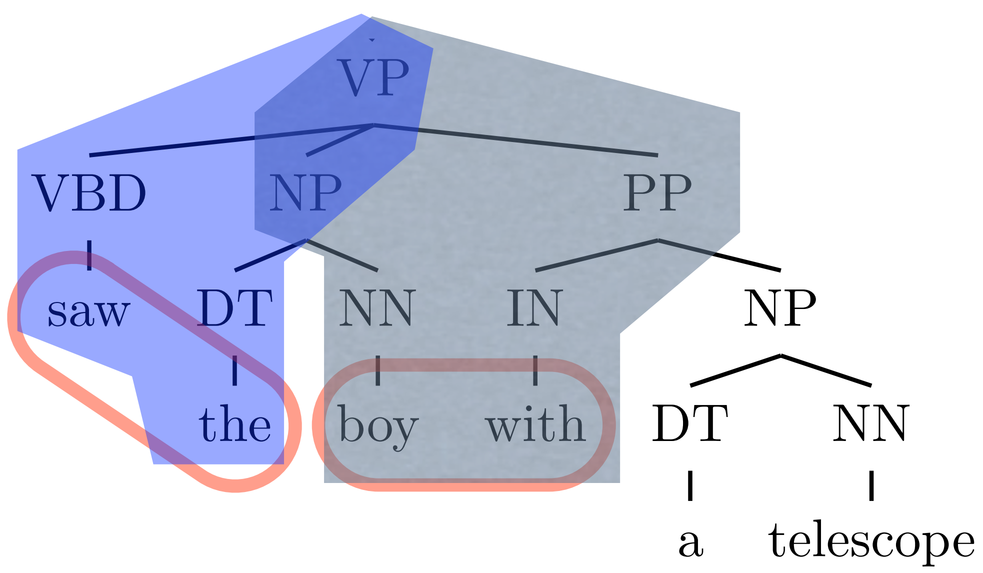


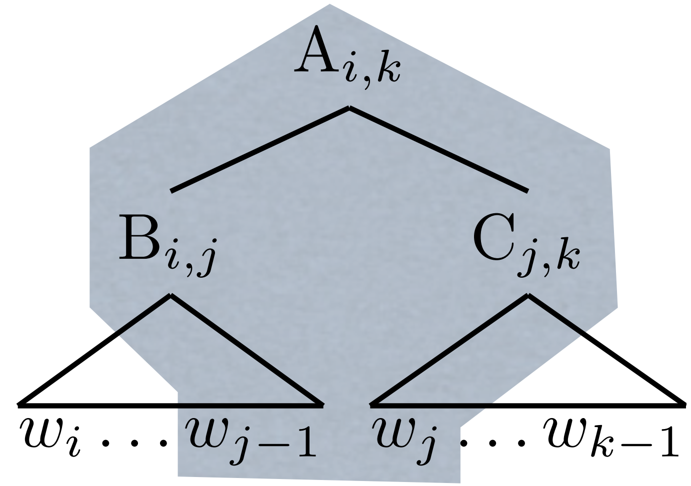


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

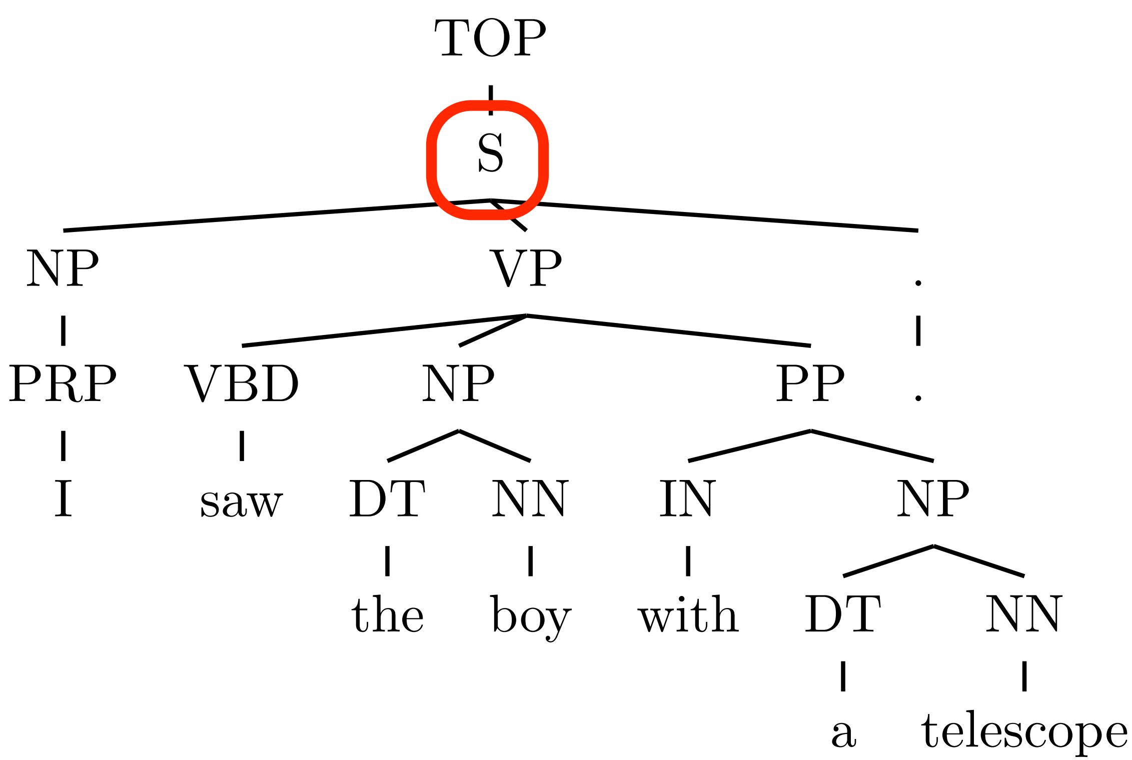


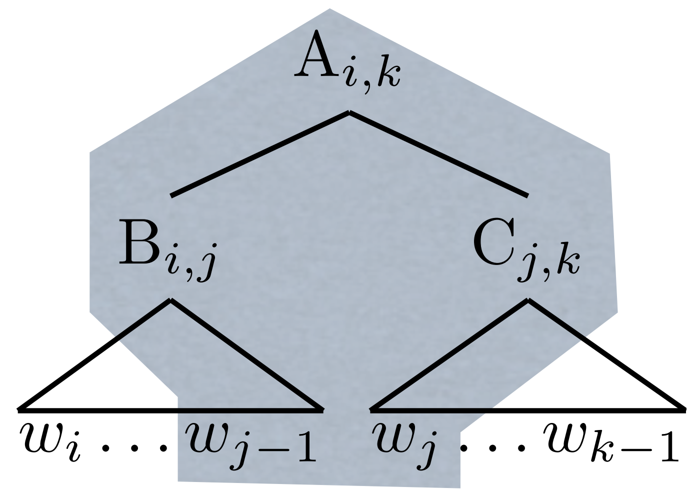


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

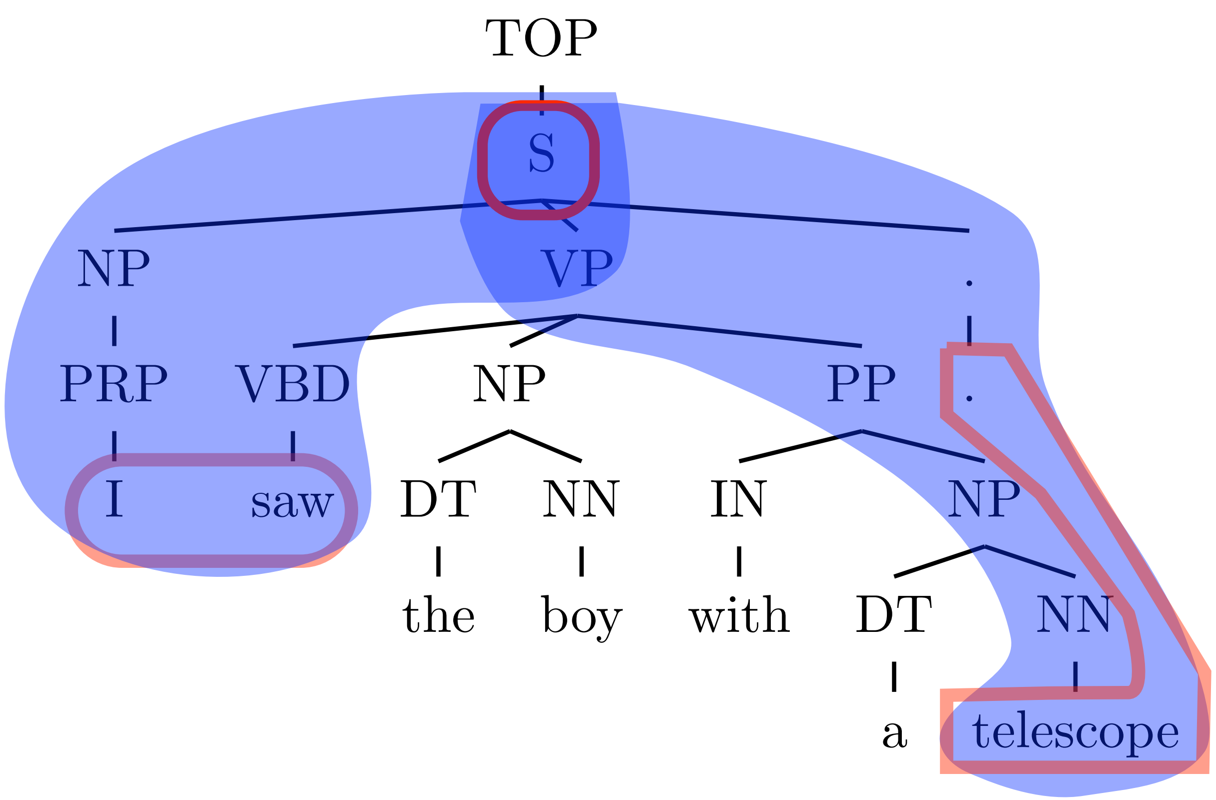


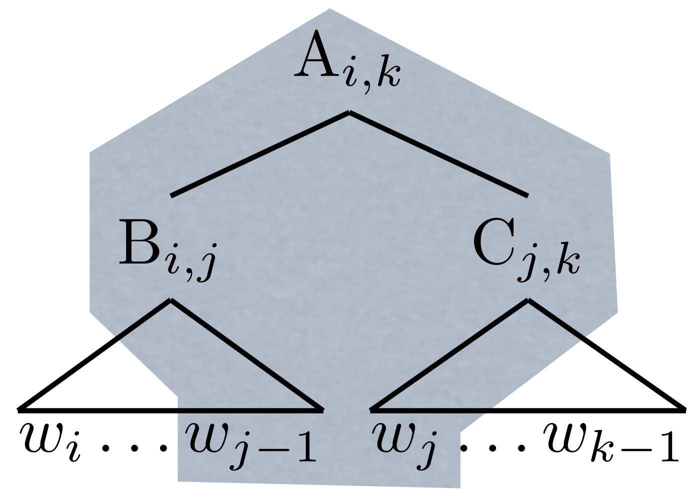


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

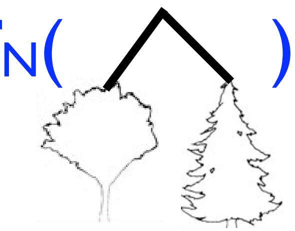


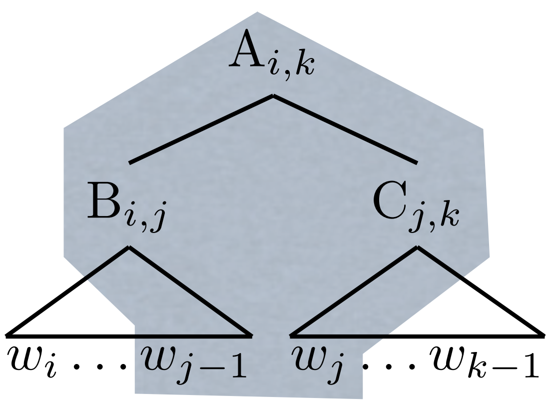


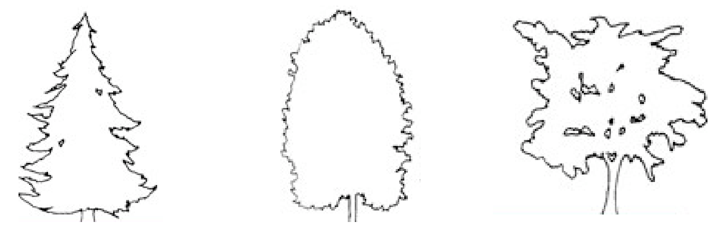
unit instance of node A

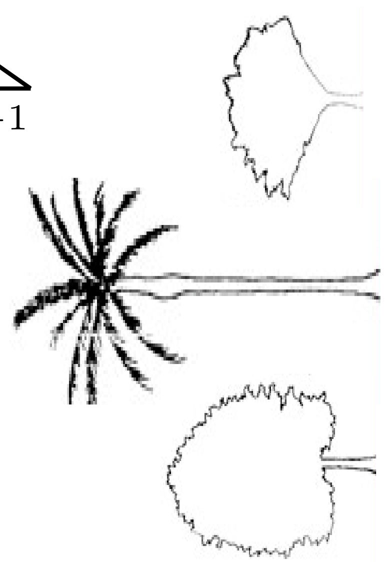
Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$



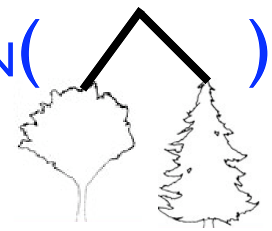


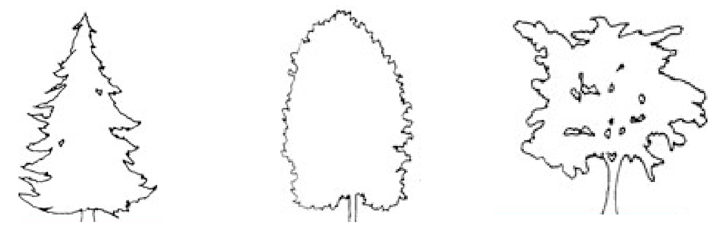


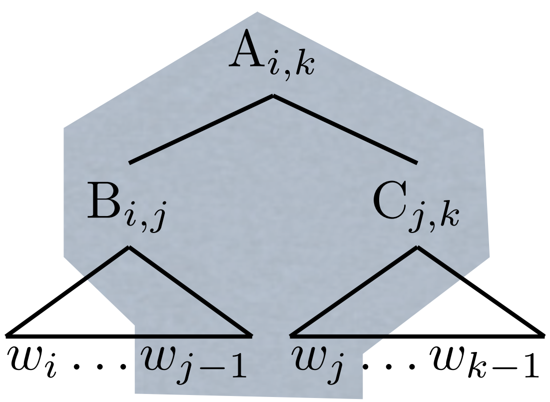
	1.0	3.0	8.0
1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
3.5	4.5 + 0.6	6.5 +10.5	11.5 + 0.6

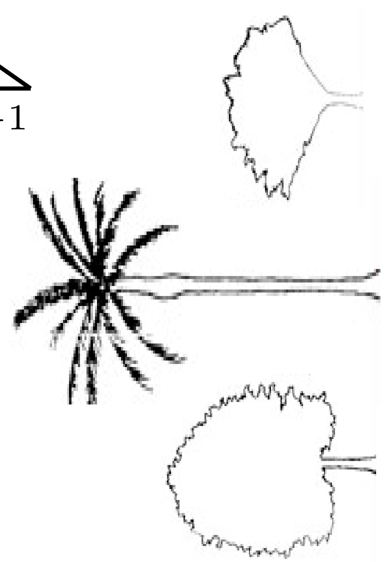
Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$



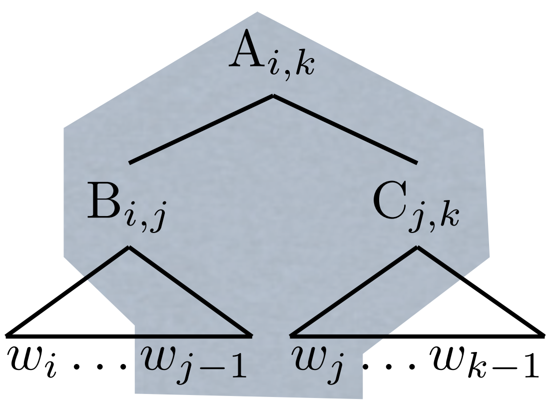


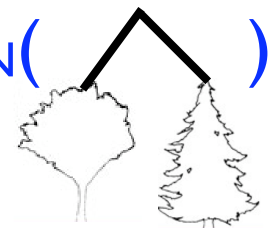


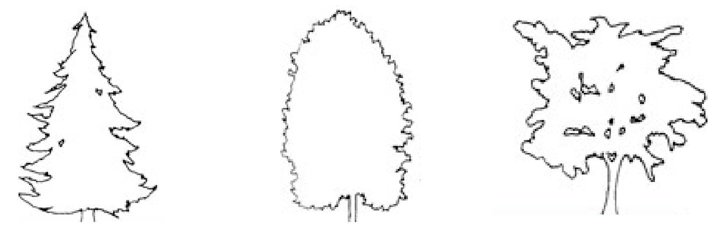
	1.0	3.0	8.0
1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
3.5	4.5 + 0.6	6.5 +10.5	11.5 + 0.6

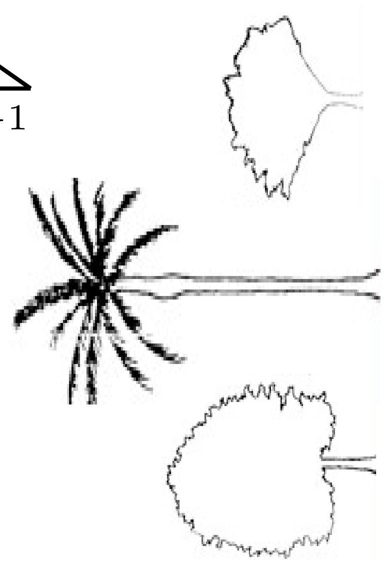
Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features



$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$

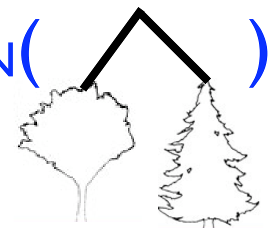


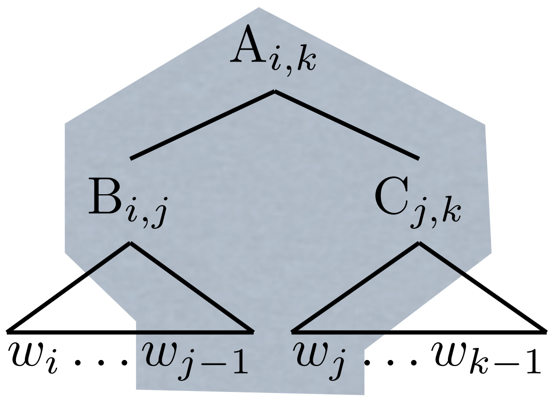


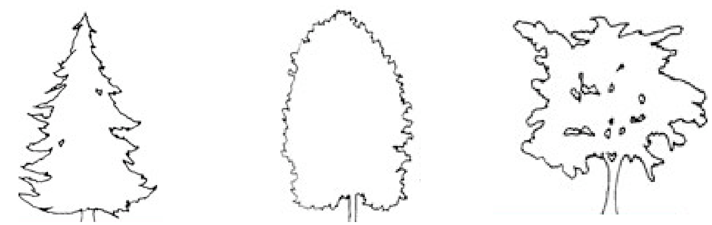
	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

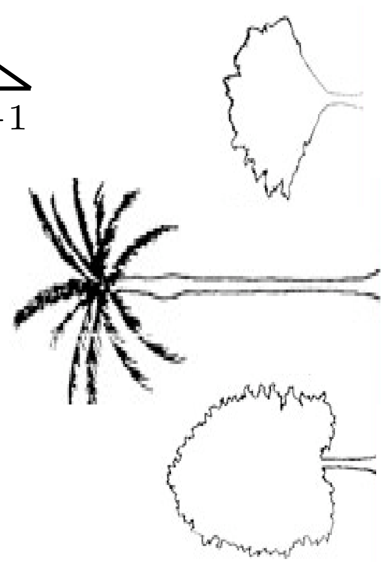
Algorithm 2 => Cube Pruning

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$



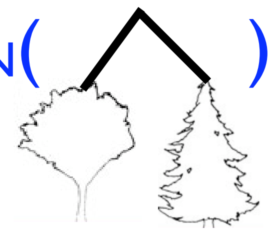


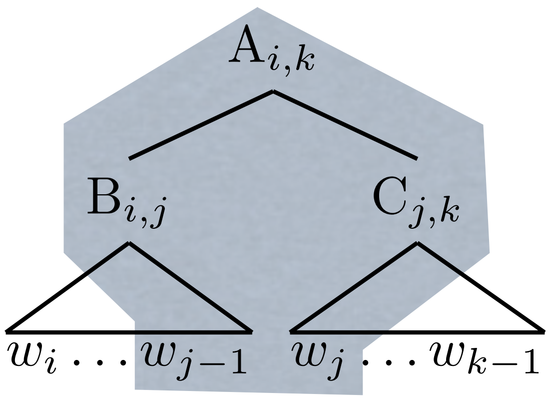


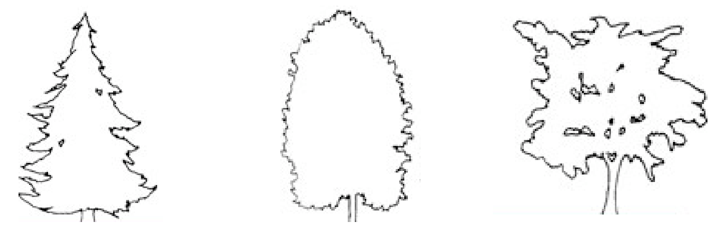
	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

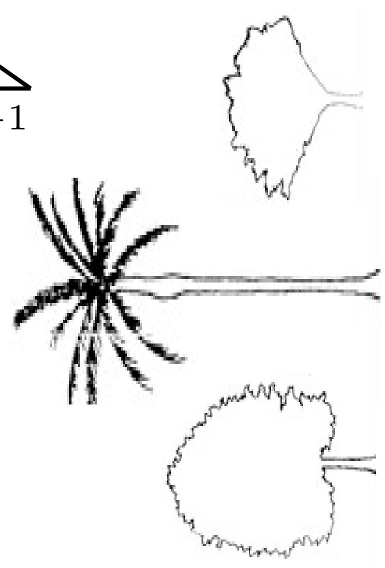
Algorithm 2 => Cube Pruning

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$



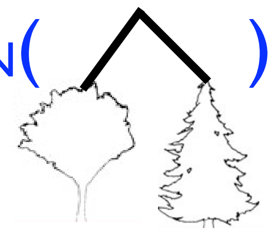


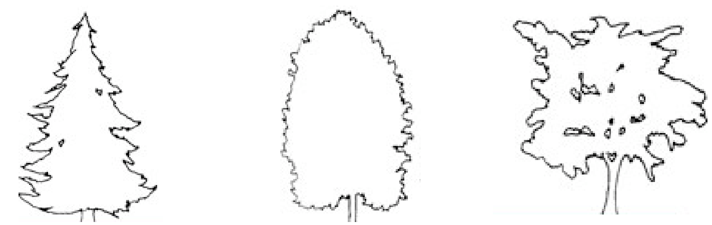


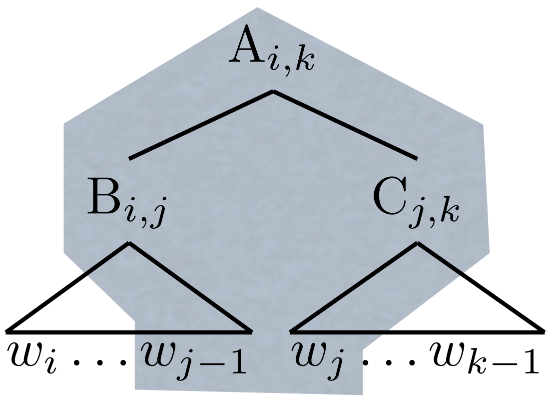
	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

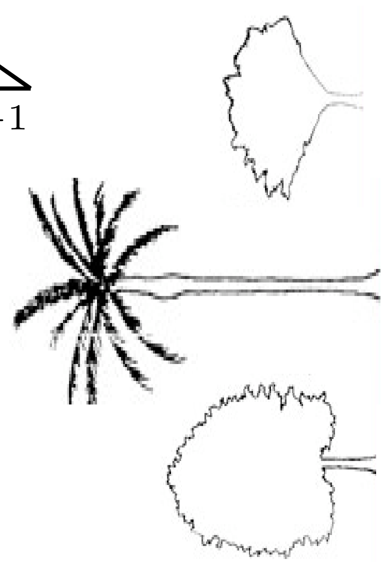
Algorithm 2 => Cube Pruning

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$



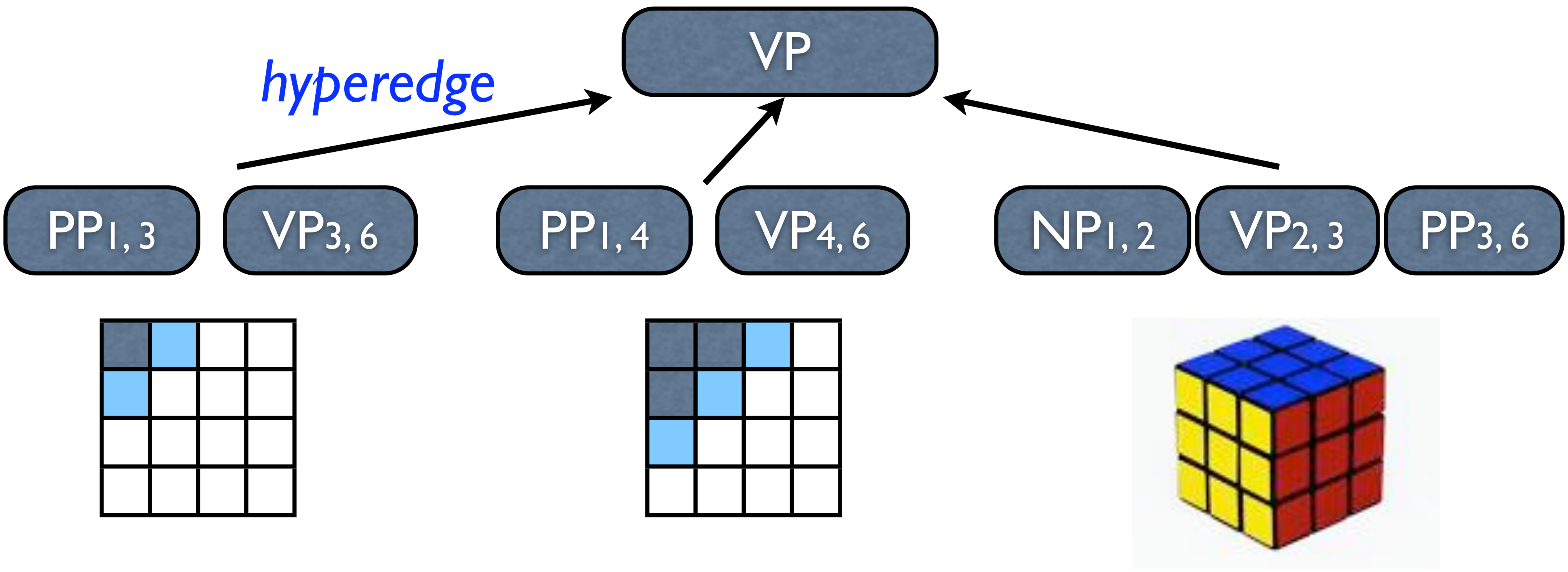




	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

Algorithm 2 => Cube Pruning

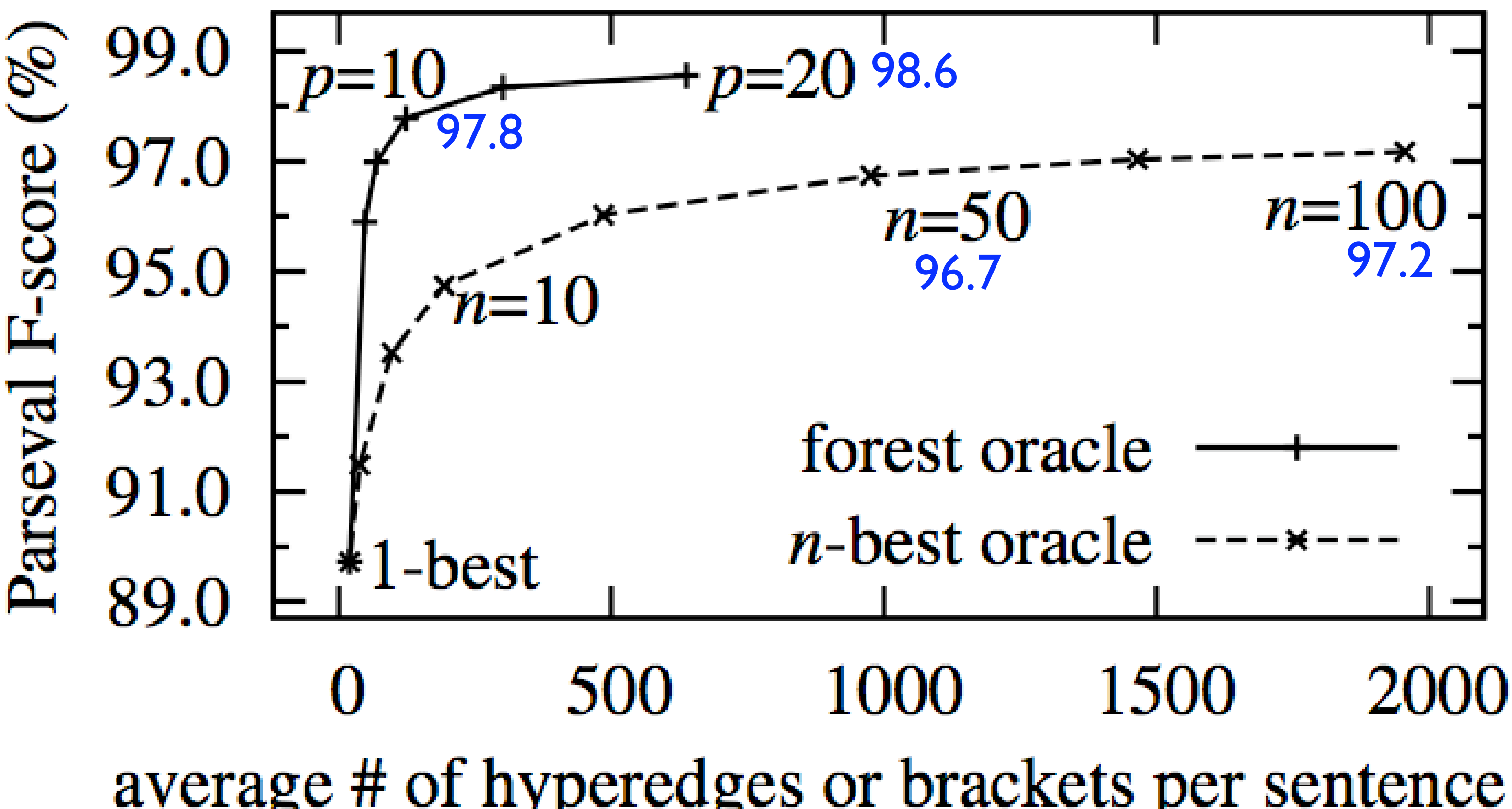
- process all hyperedges simultaneously!
  significant savings of computation



there are search errors, but the trade-off is favorable.

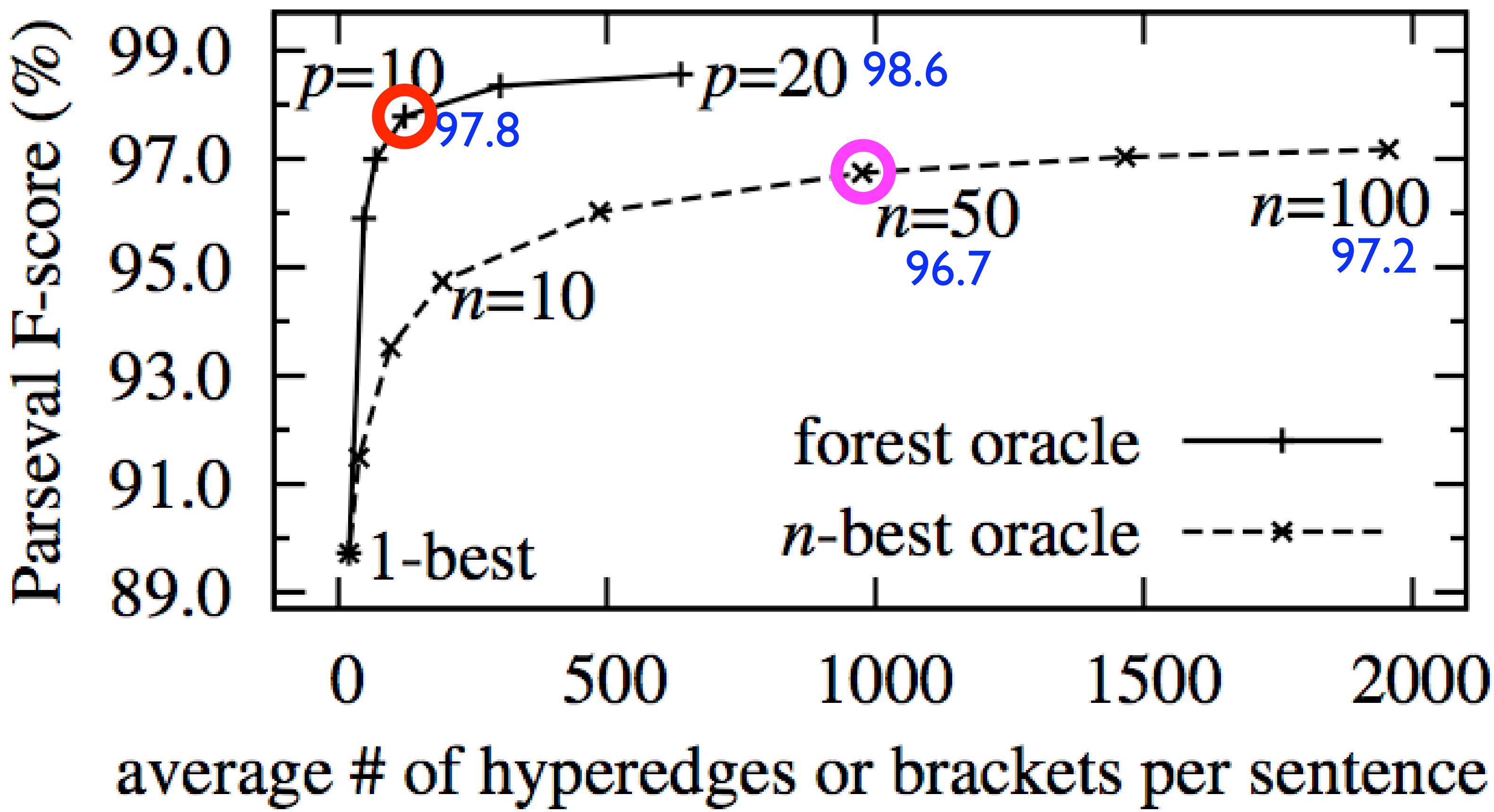
Forest vs. *k*-best Oracles

- on top of Charniak parser (modified to dump forest)
- forests enjoy higher oracle scores than *k*-best lists
  - with much smaller sizes



Forest vs. *k*-best Oracles

- on top of Charniak parser (modified to dump forest)
- forests enjoy higher oracle scores than *k*-best lists
  - with much smaller sizes



Main Results

- forest reranking beats 50-best & 100-best reranking
- can be trained on the whole treebank in ~1 day even with a pure Python implementation!
  - most previous work only scaled to short sentences (<=15 words) and local features

baseline: 1-best Charniak parser		89.72
approach	training time	F1%
50-best reranking	4 x 0.3h	91.43
100-best reranking	4 x 0.7h	91.49
forest reranking	4 x 6.1h	91.69

Main Results

- forest reranking beats 50-best & 100-best reranking
- can be trained on the whole treebank in ~1 day even with a pure Python implementation!
    - most previous work only scaled to short sentences (<=15 words) and local features

baseline: 1-best Charniak parser		89.72	feature extract	
approach	training time	F1%	space	time
50-best reranking	4 x 0.3h	91.43	2.4G	19h
100-best reranking	4 x 0.7h	91.49	5.3G	44h
forest reranking	4 x 6.1h	91.69	1.2G	2.9h

Comparison with Others

type	system	F_1%
D	Collins (2000)	89.7
	Charniak and Johnson (2005)	91.0
	updated (2006)	91.4
	Petrov and Klein (2008)	88.3
	this work	91.7
	Carreras et al. (2008)	91.1
G	Bod (2000)	90.7
	Petrov and Klein (2007)	90.1
S	McClosky et al. (2006)	92.1

n-best reranking

dynamic programming

semi-supervised

best accuracy to date on the Penn Treebank, and fast training

on to Machine Translation...

applying the same ideas of non-locality...

Translate Server Error

Translate Server Error

clear evidence that MT is used in real life.

Context in Translation

Context in Translation

Algorithm 2 => cube pruning

fluency problem (*n*-gram)

Context in Translation

Algorithm 2 => cube pruning

fluency problem (*n*-gram)

xiaoxin

小心 X <=> be careful not to X

syntax problem (SCFG)

Context in Translation

xiaoxin gou
小心 狗 <=> be aware of dog

Algorithm 2 => cube pruning

fluency problem (*n*-gram)

xiaoxin
小心 X <=> be careful not to X

syntax problem (SCFG)

Context in Translation

xiaoxin gou
小心 狗 <=> be aware of dog

Algorithm 2 => cube pruning

fluency problem (*n*-gram)

小心 VP <=> be careful not to VP

小心 NP <=> be careful of NP

xiaoxin
小心 X <=> be careful not to X

syntax problem (SCFG)

How do people translate?

1. understand the source language sentence
2. generate the target language translation

布什	与	沙龙	举行	了	会谈
Bùshí	*yu*	*Shalóng*	*juxíng*	*le*	*huìtán*
Bush	and/ with	Sharon	hold	[*past.*]	meeting

How do people translate?

1. understand the source language sentence
2. generate the target language translation

布什	与	沙龙	举行	了	会谈
Bùshí	*yu*	*Shalóng*	*juxíng*	*le*	*huìtán*
Bush	and/ with	Sharon	hold	[*past.*]	meeting

How do people translate?

1. understand the source language sentence
2. generate the target language translation

布什	与	沙龙	举行	了	会谈
Bùshí	*yu*	*Shalóng*	*juxíng*	*le*	*huìtán*
Bush	and/ with	Sharon	hold	[*past.*]	meeting

"Bush held a meeting with Sharon"

How do compilers translate?

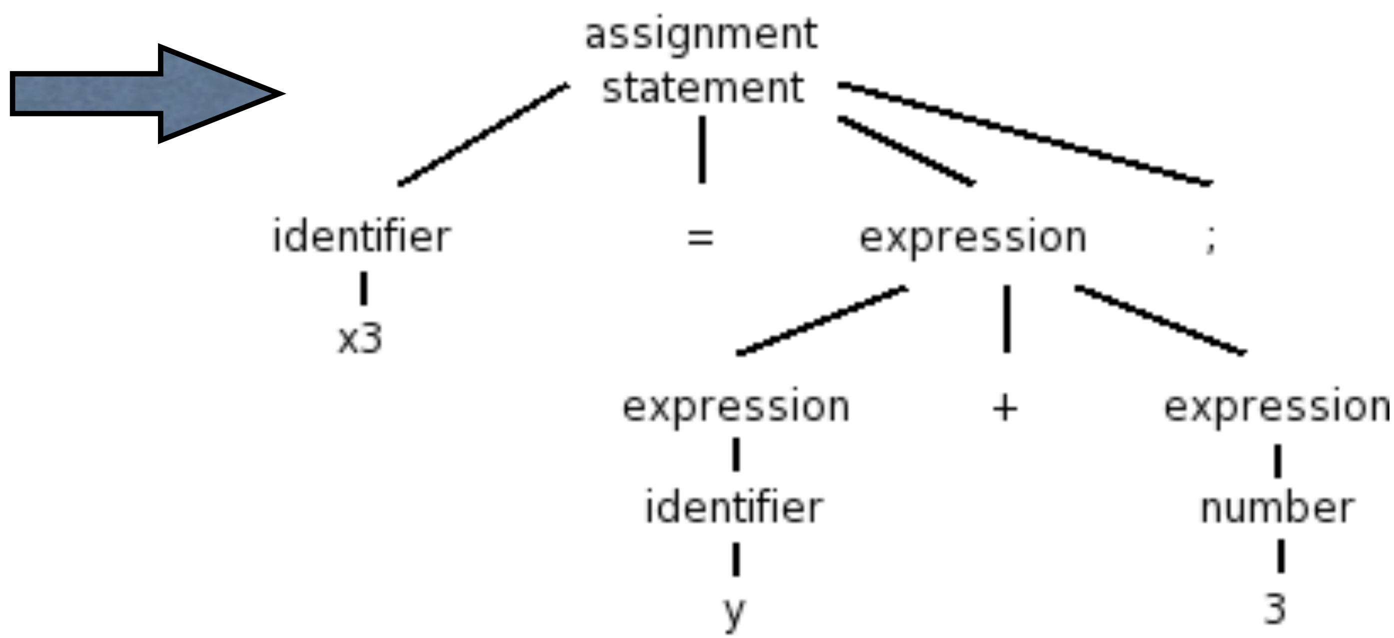
1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

```
x3 = y + 3;
```

How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

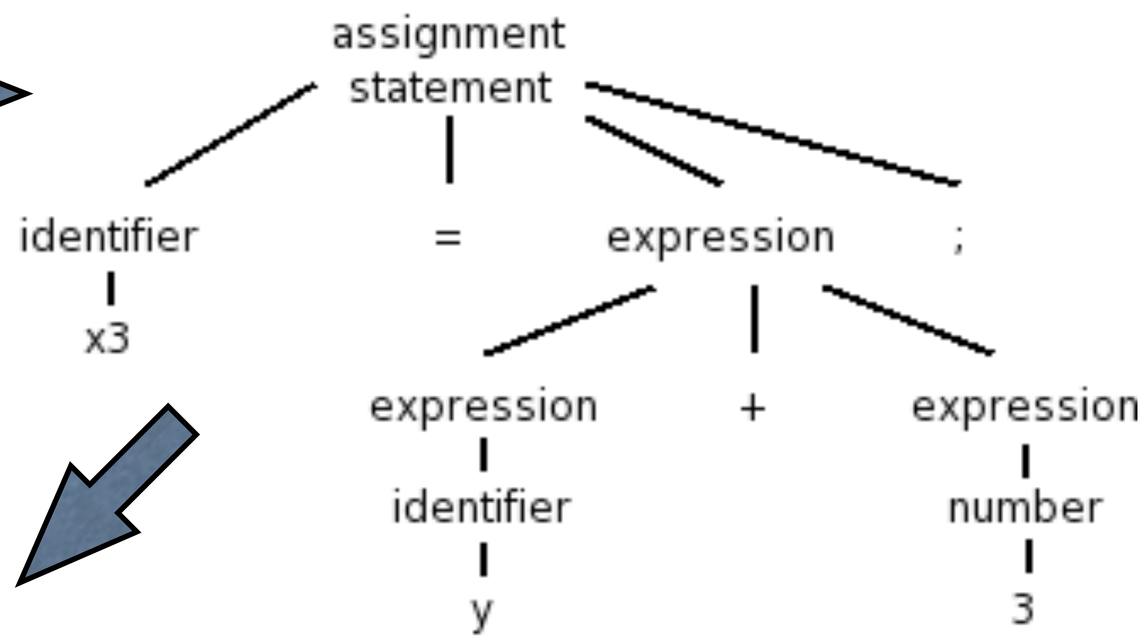
```
x3 = y + 3;
```



How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

```
x3 = y + 3;
```

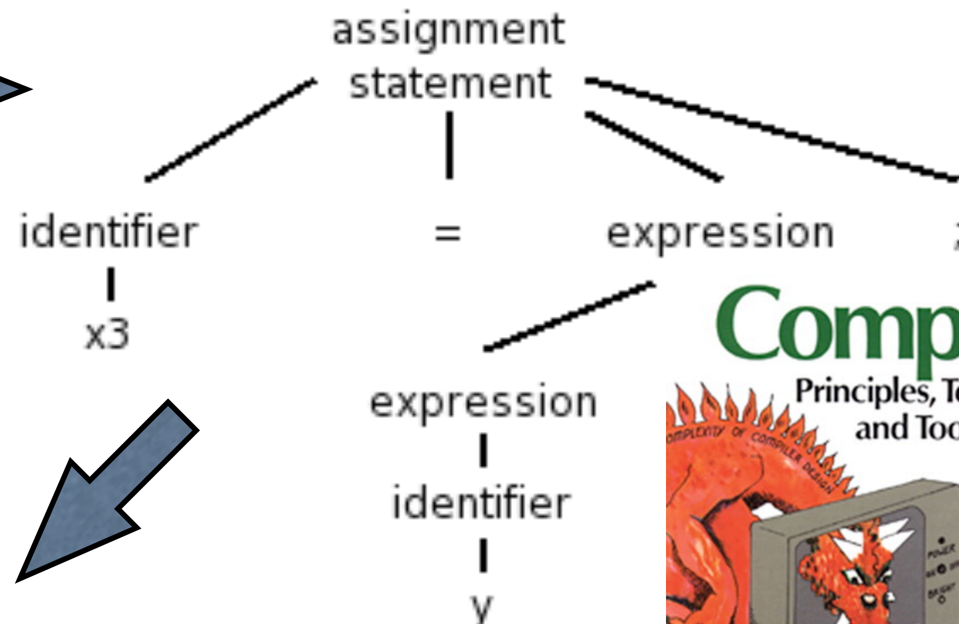



```
LD      R1,  id2
ADDF    R1,  R1, #3.0   // add float
RTOI    R2,  R1         // real to int
ST      id1, R2
```

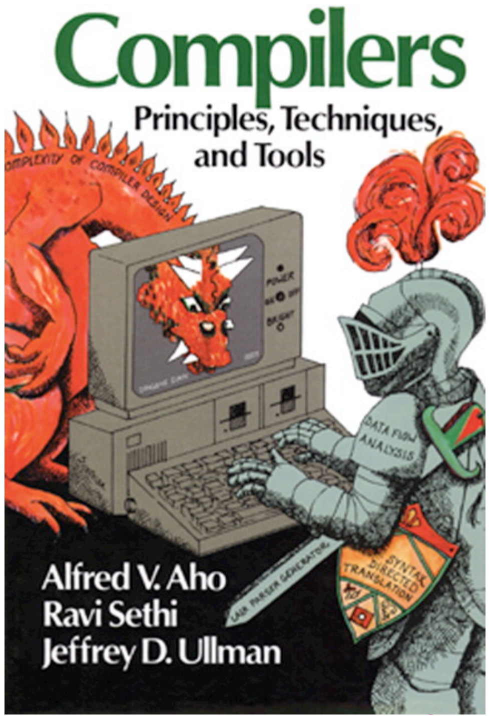
How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

```
x3 = y + 3;
```



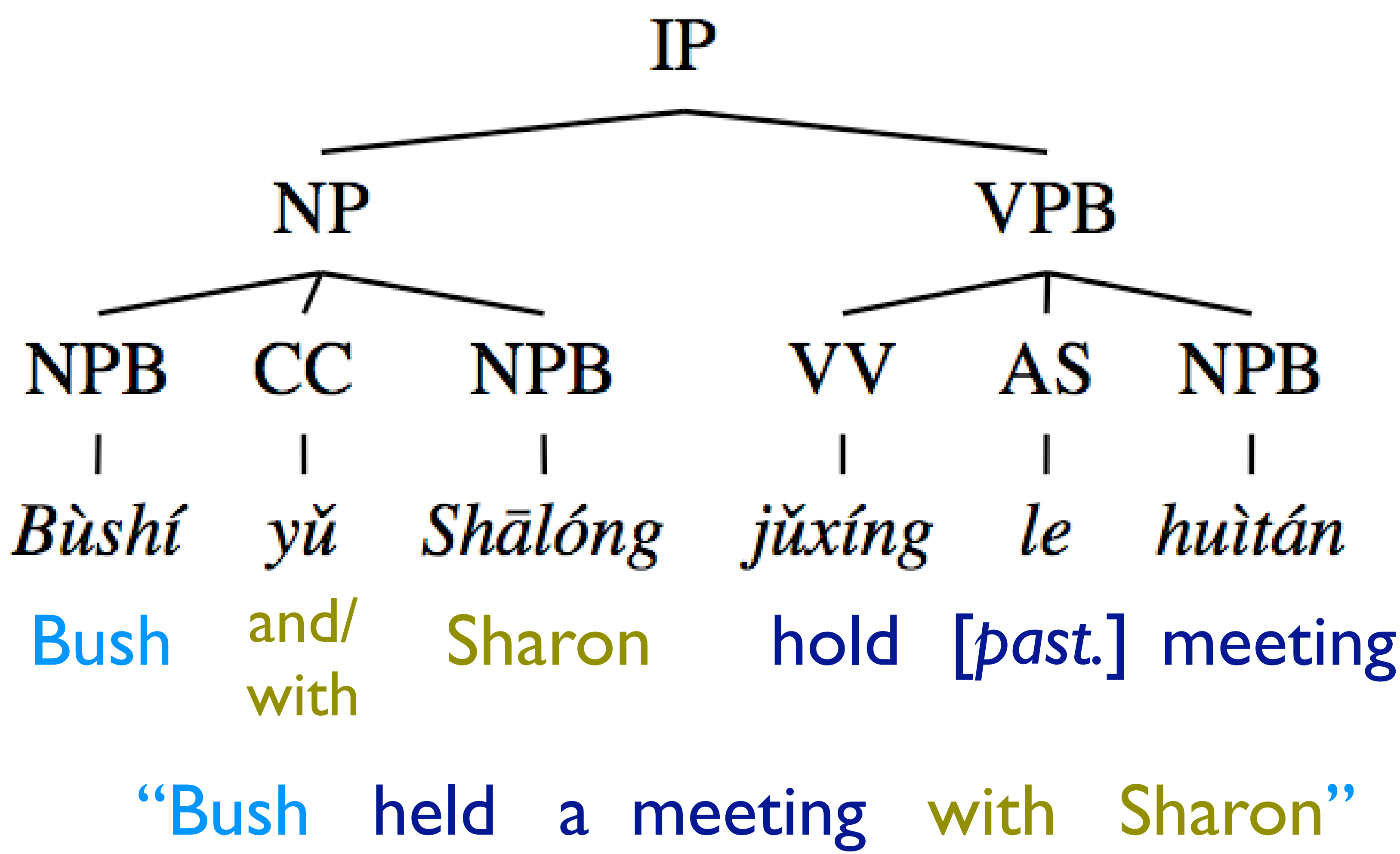

```
LD      R1,  id2
ADDF    R1,  R1, #3.0   // add float
RTOI    R2,  R1         // real to int
ST      id1, R2
```



syntax-directed translation (~1960)

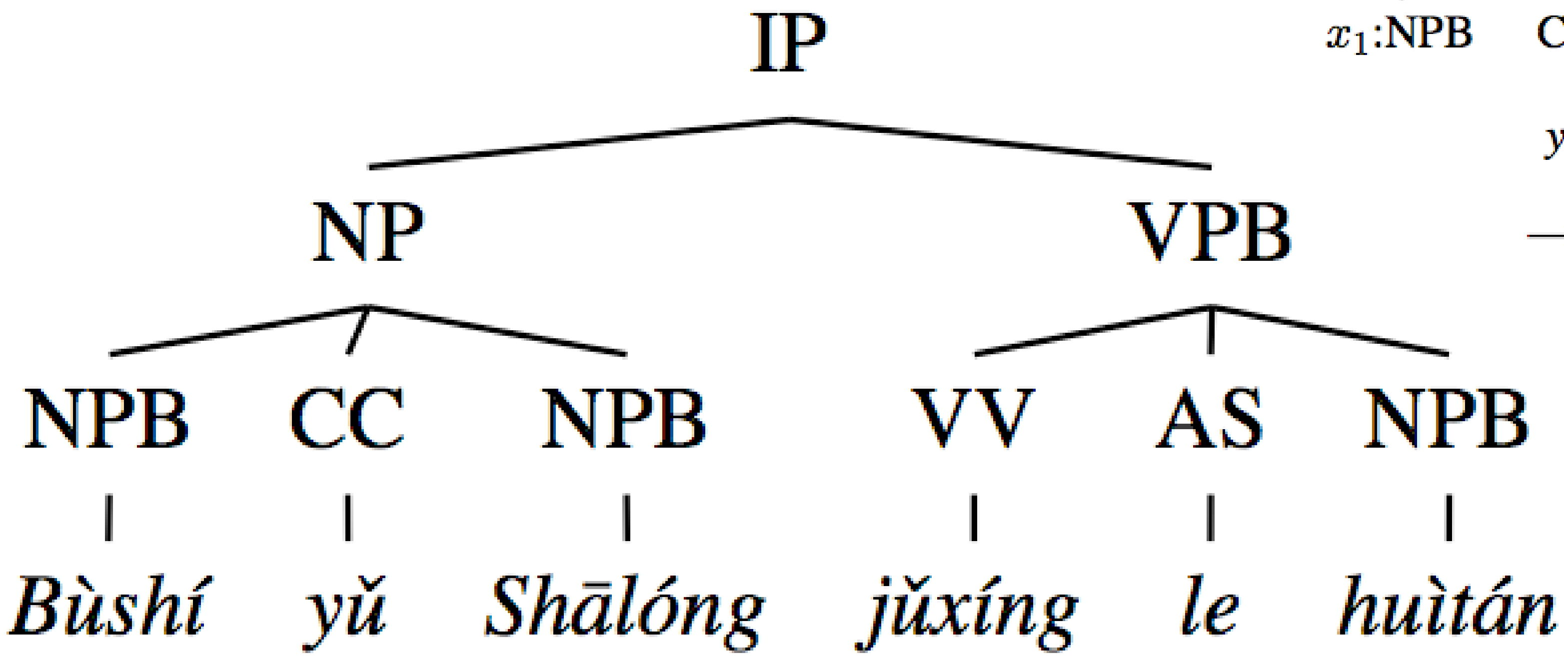
Syntax-Directed Machine Translation

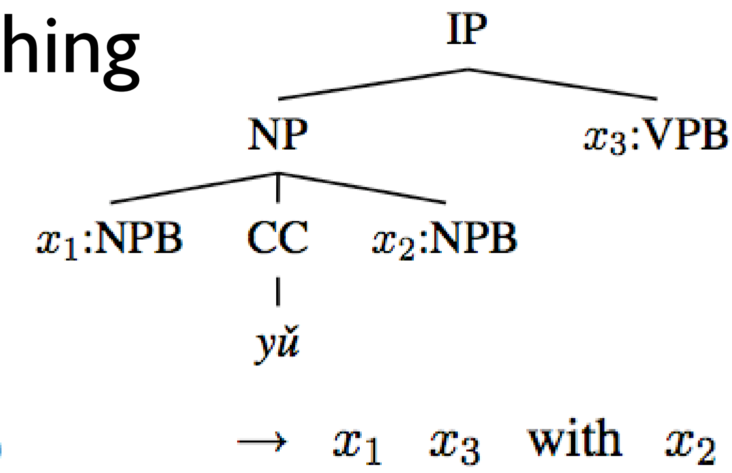
- get 1-best parse tree; then convert to English



Syntax-Directed Machine Translation

- recursive rewrite by pattern-matching

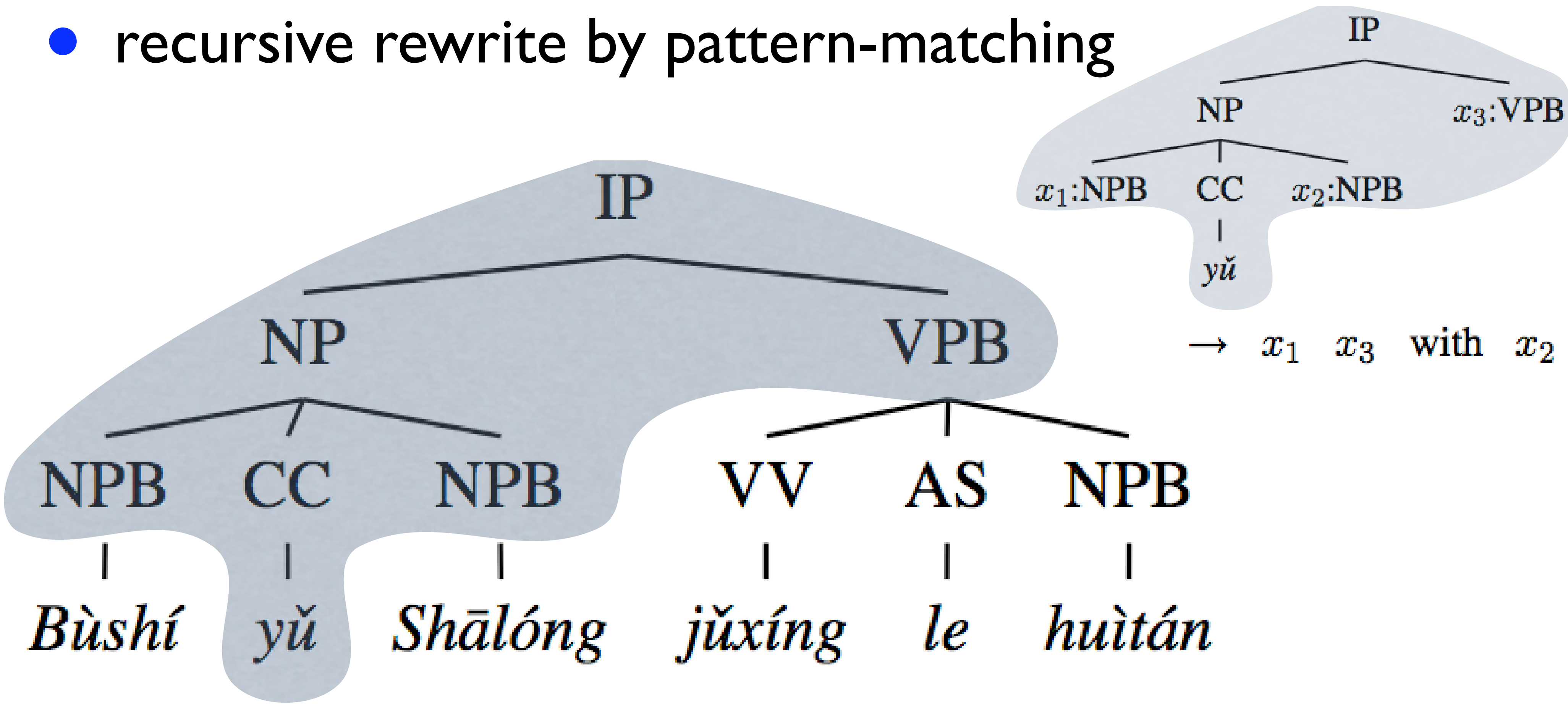




(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

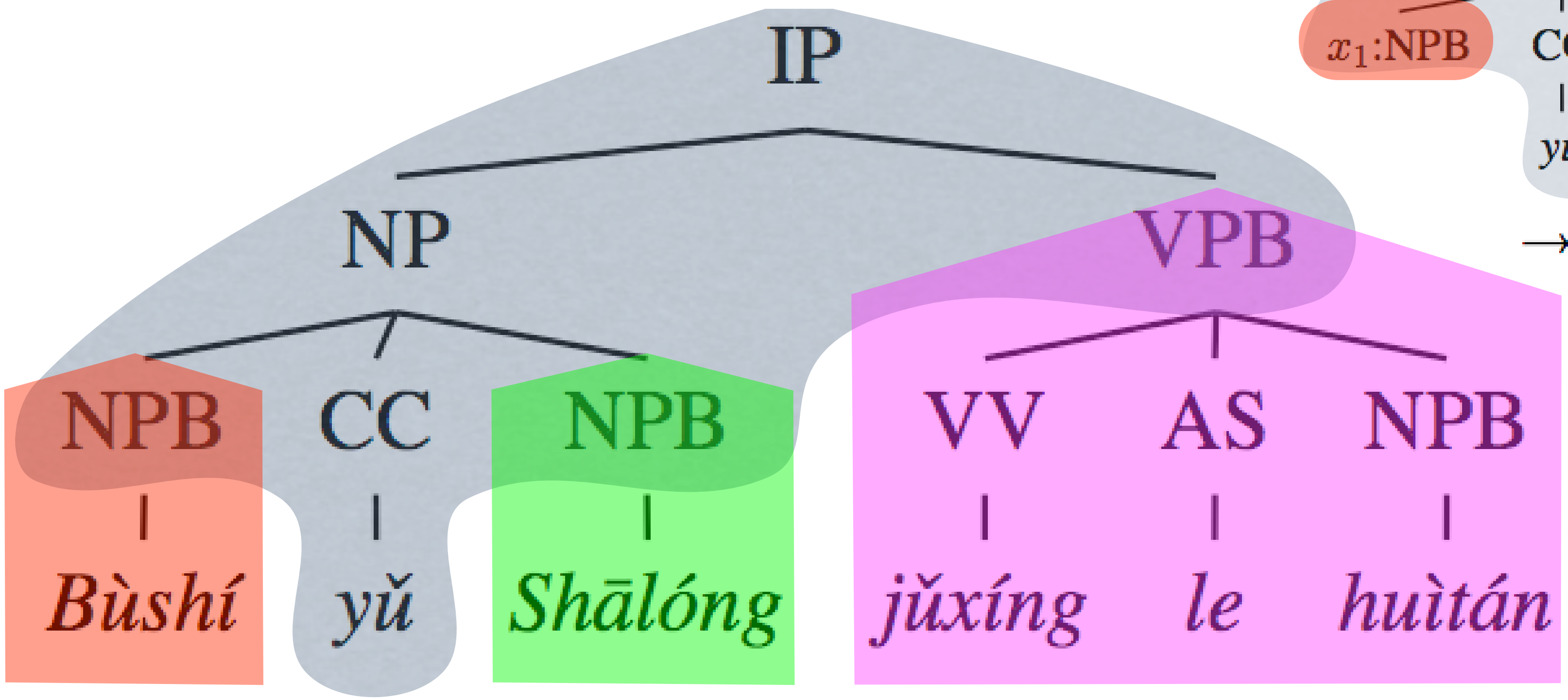
- recursive rewrite by pattern-matching

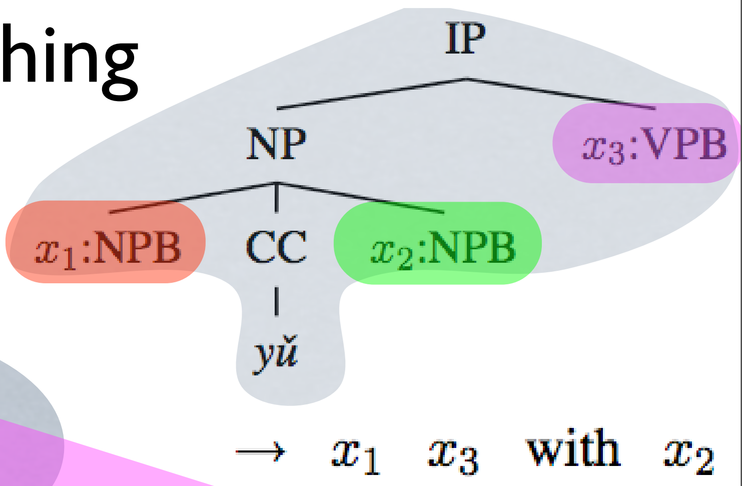


(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

- recursive rewrite by pattern-matching

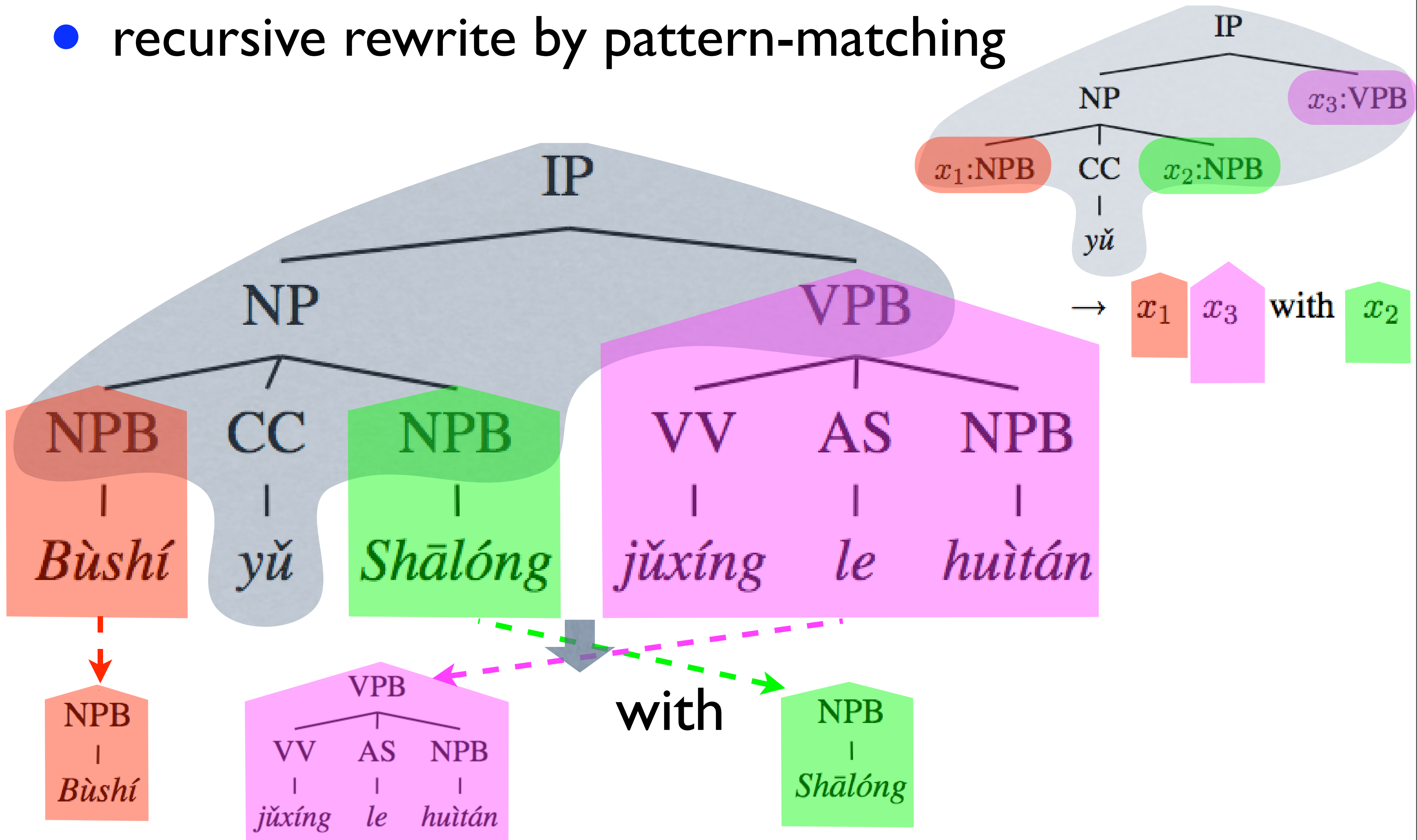




(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

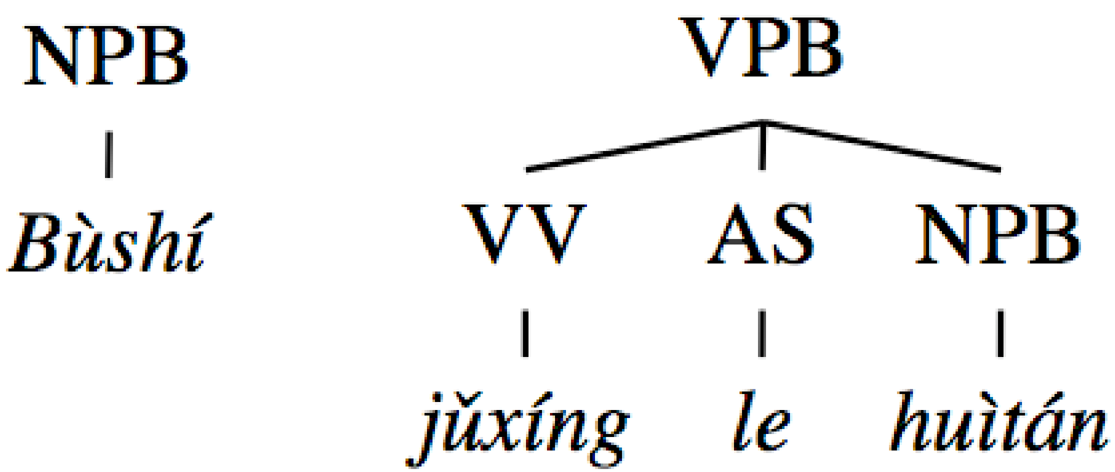
- recursive rewrite by pattern-matching



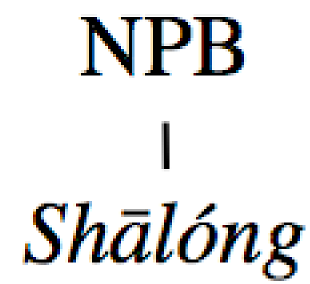
(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

- recursively solve unfinished subproblems



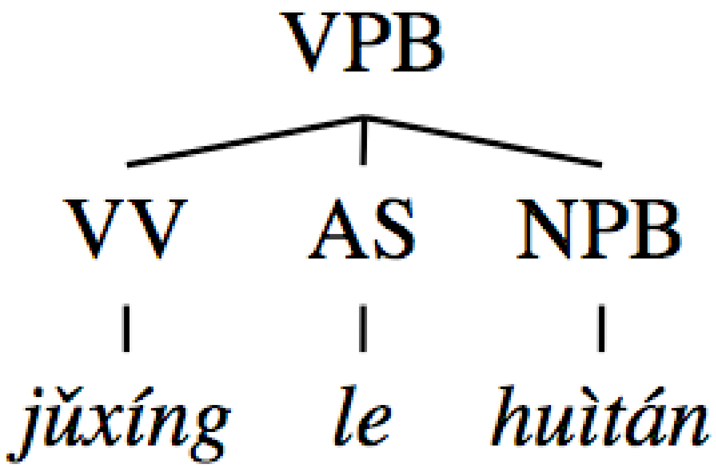
with



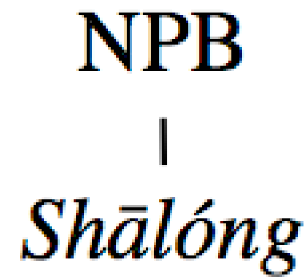
(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

- recursively solve unfinished subproblems



with

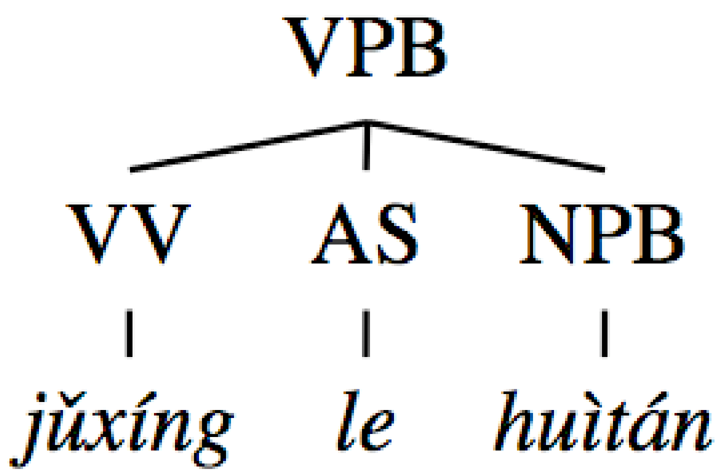


(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

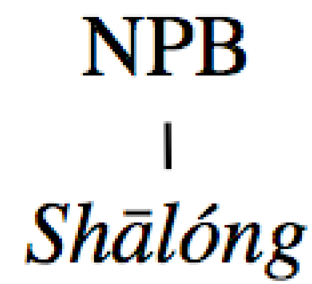
Syntax-Directed Machine Translation

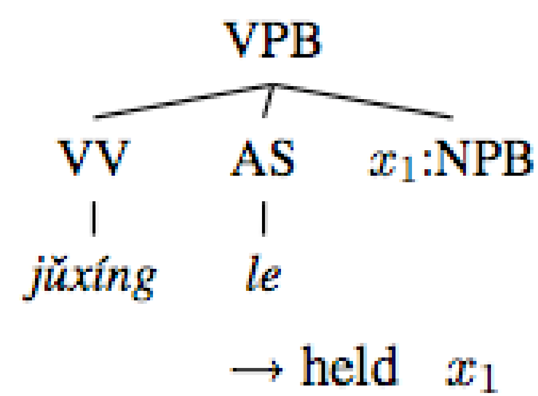
- recursively solve unfinished subproblems

Bush



with

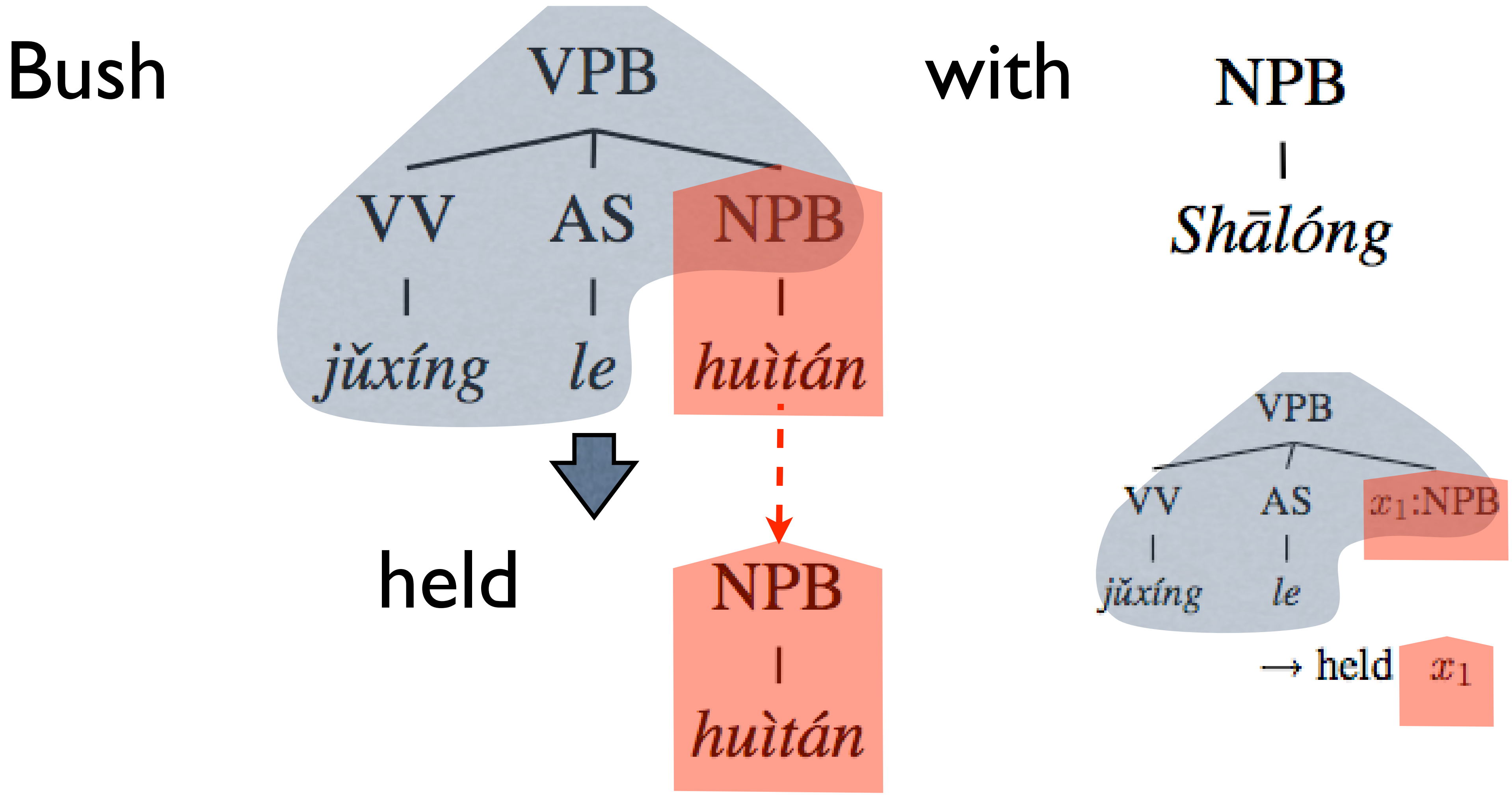




 (Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

- recursively solve unfinished subproblems



 (Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

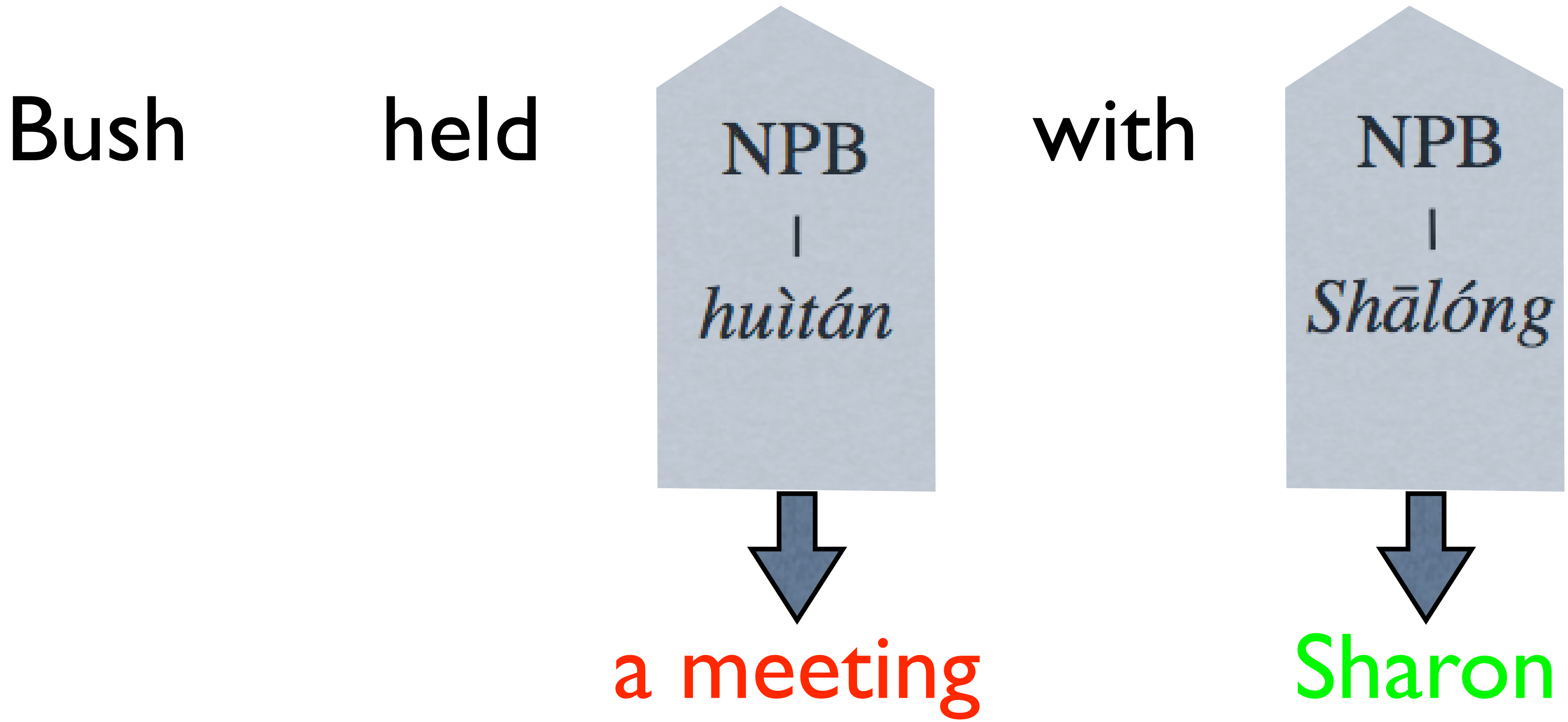
- continue pattern-matching

Bush	held	NPB	with	NPB
		|		|
		huìtán		*Shālóng*

(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

- continue pattern-matching



(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)

Syntax-Directed Machine Translation

- continue pattern-matching

Bush held a meeting with Sharon

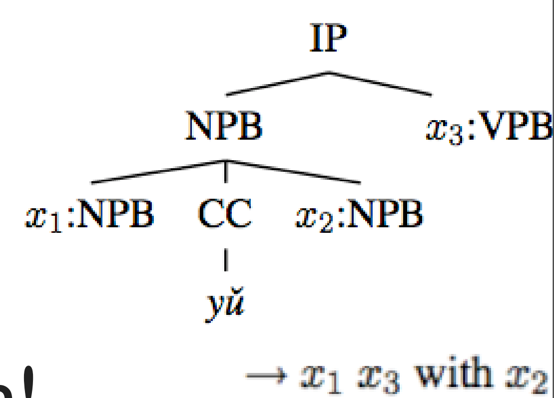
- continue pattern-matching

Bush held a meeting with Sharon

this method is simple, fast, and expressive.

but... crucial difference between PL and NL:

ambiguity!

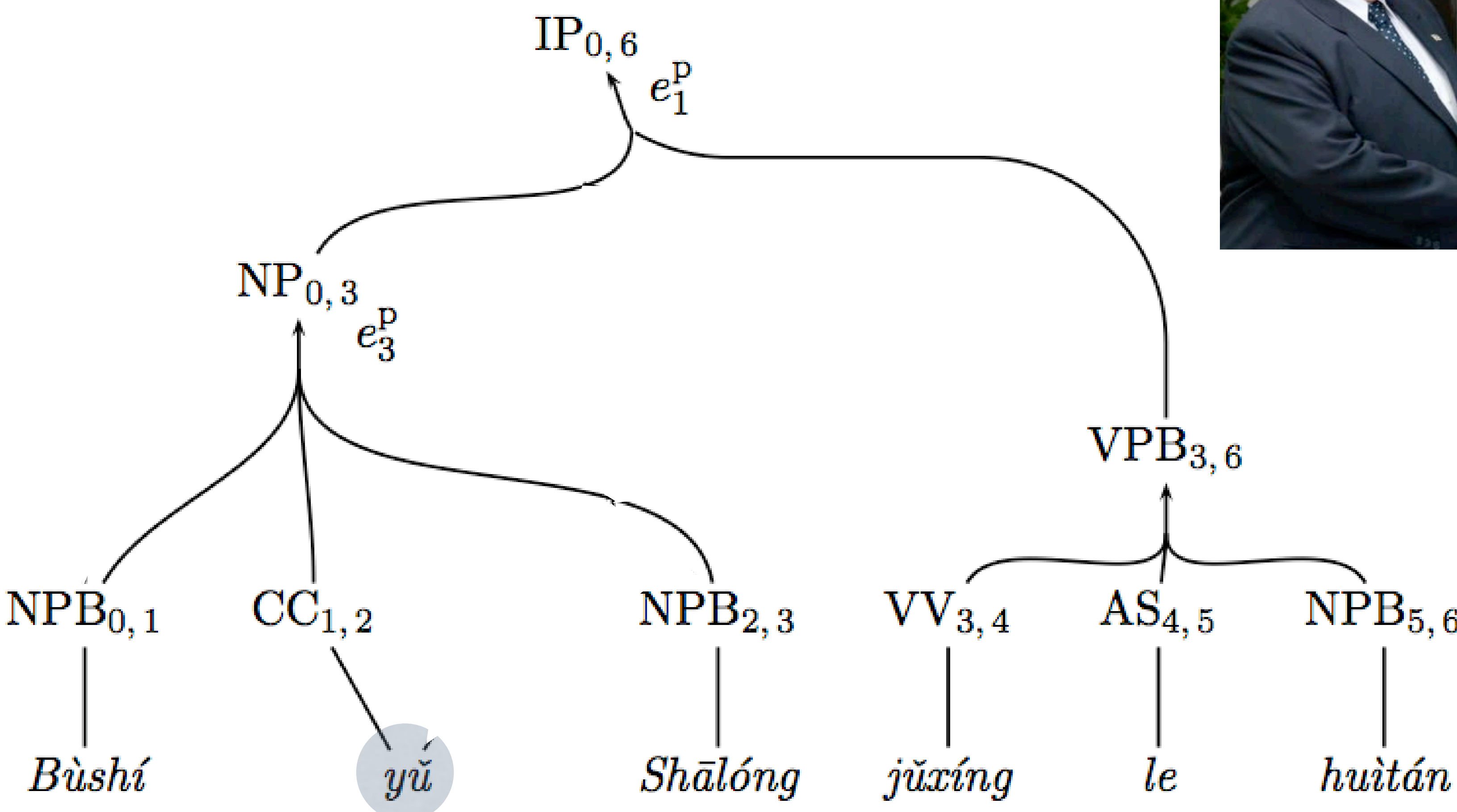


using 1-best parse causes error propagation!

idea: use *k*-best parses?

use a parse forest!

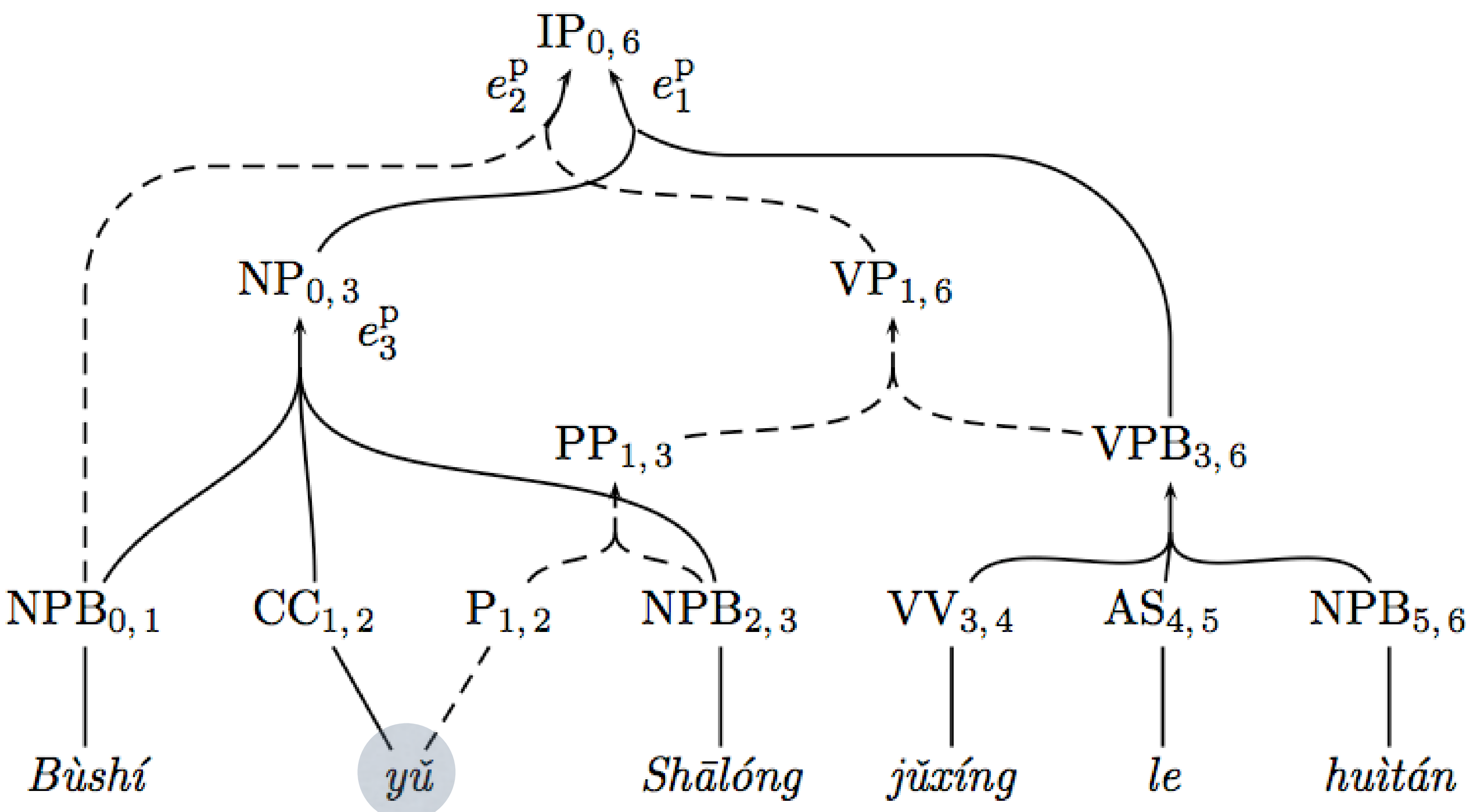
Forest-based Translation



"and" / "with"

Forest-based Translation

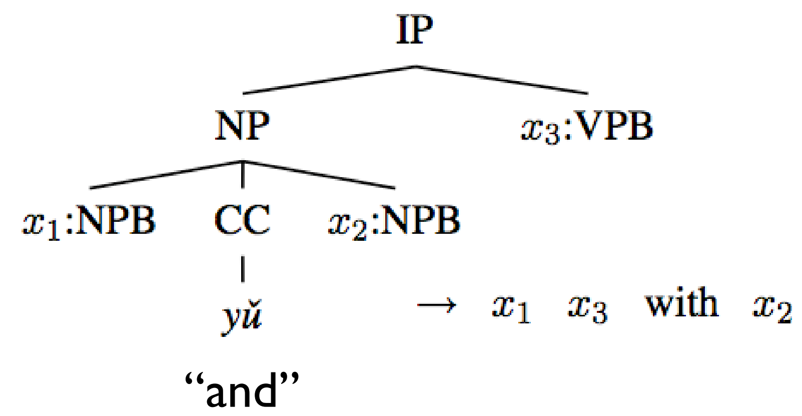
pattern-matching on forest

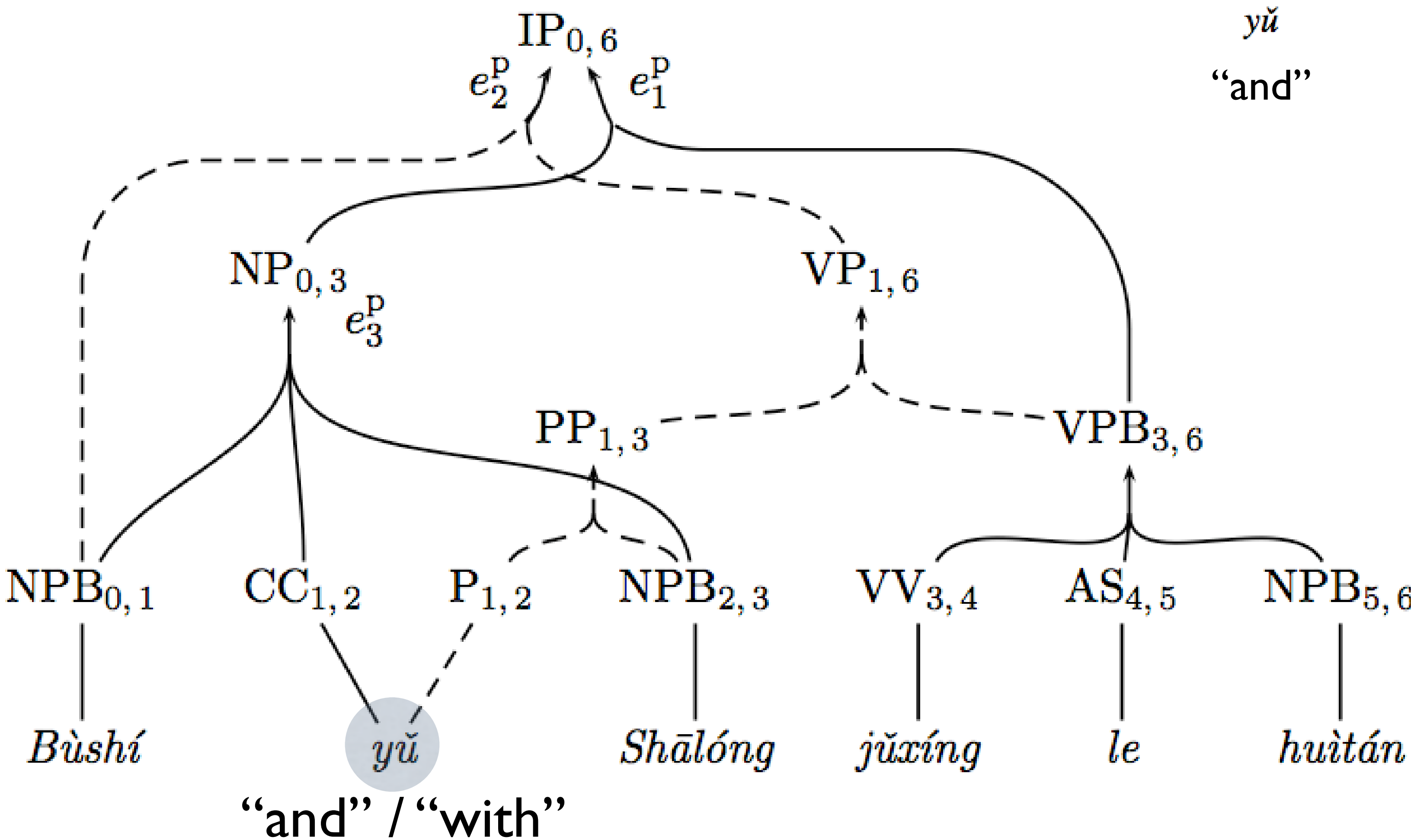


"and" / "with"

Forest-based Translation

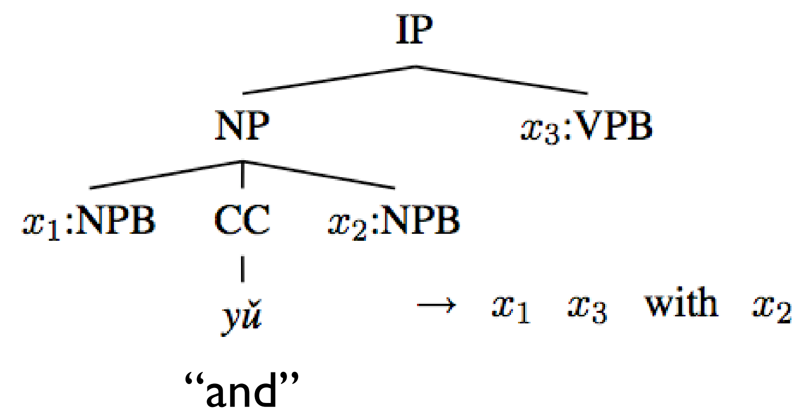
pattern-matching on forest

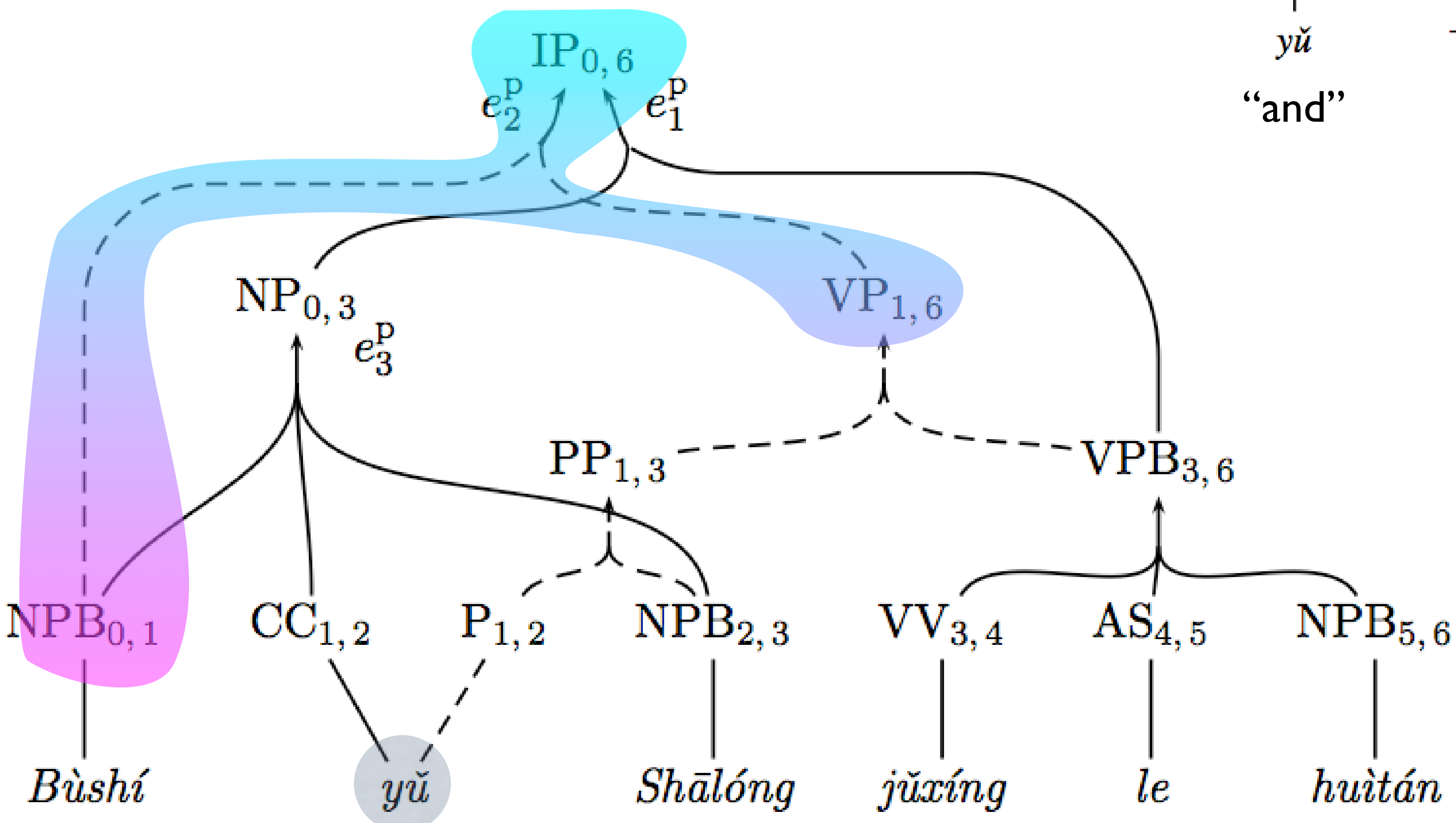




Forest-based Translation

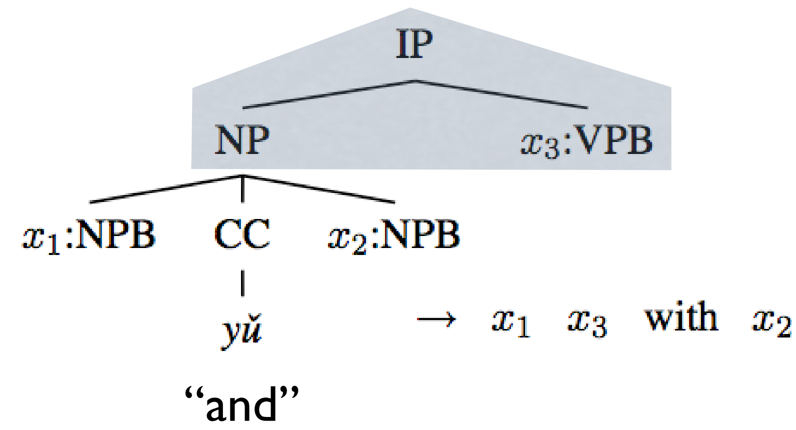
pattern-matching on forest

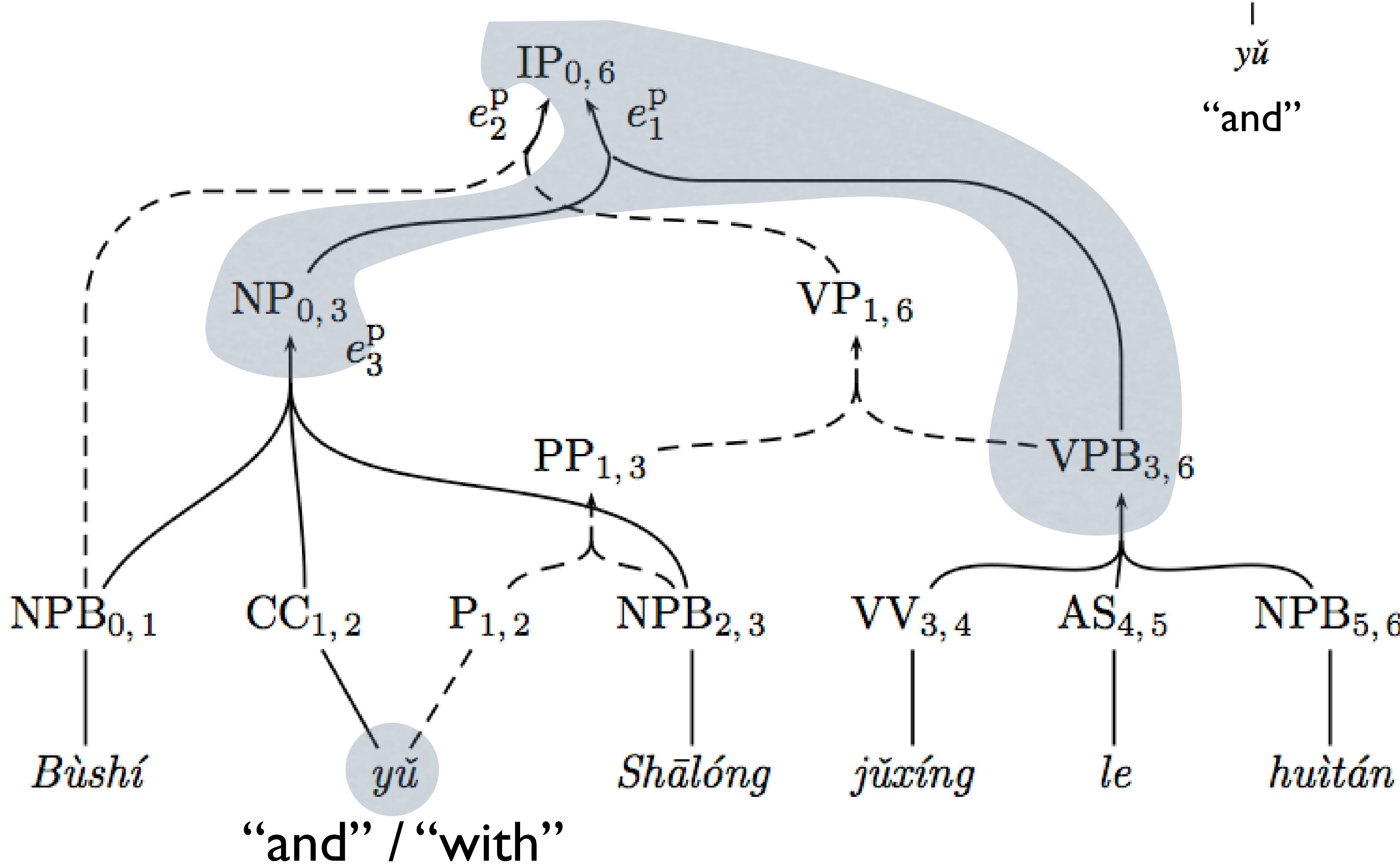




Forest-based Translation

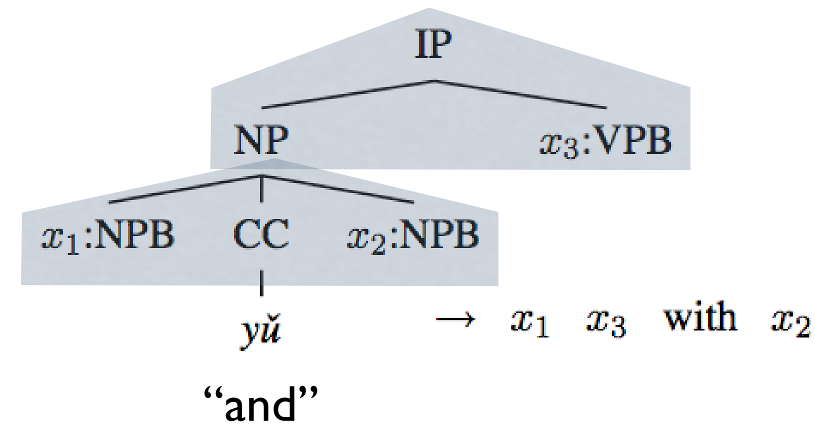
pattern-matching on forest

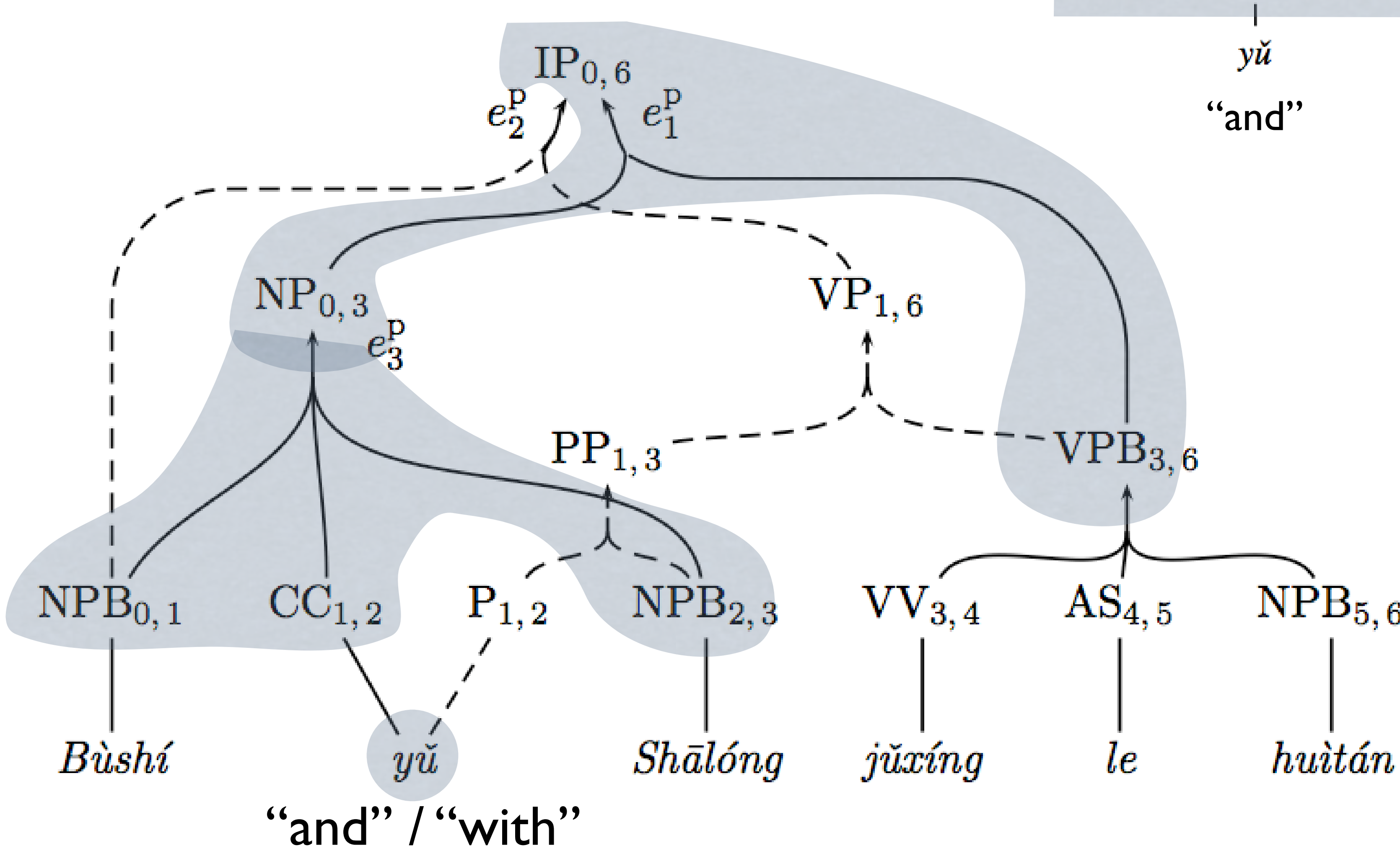




Forest-based Translation

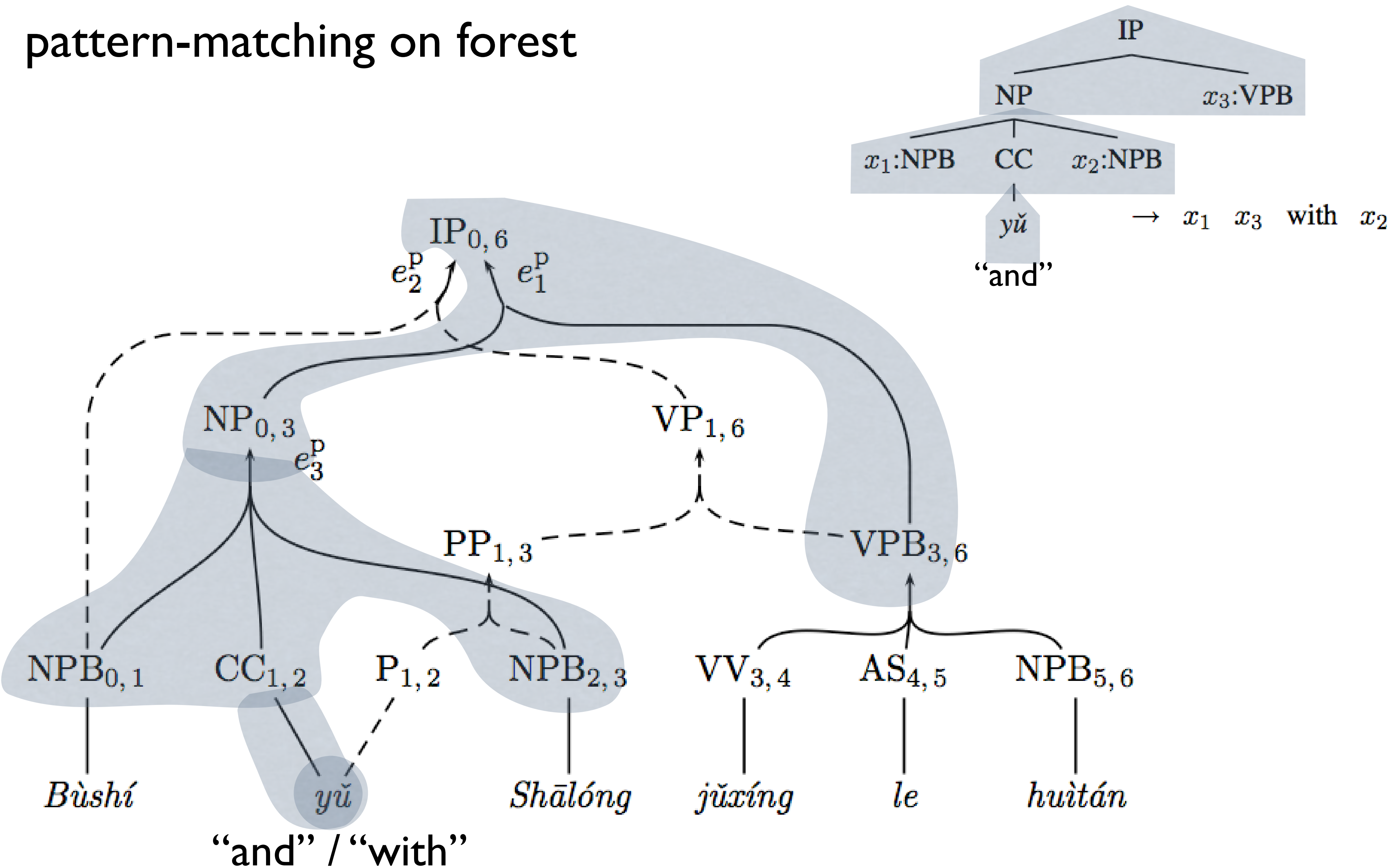
pattern-matching on forest





Forest-based Translation

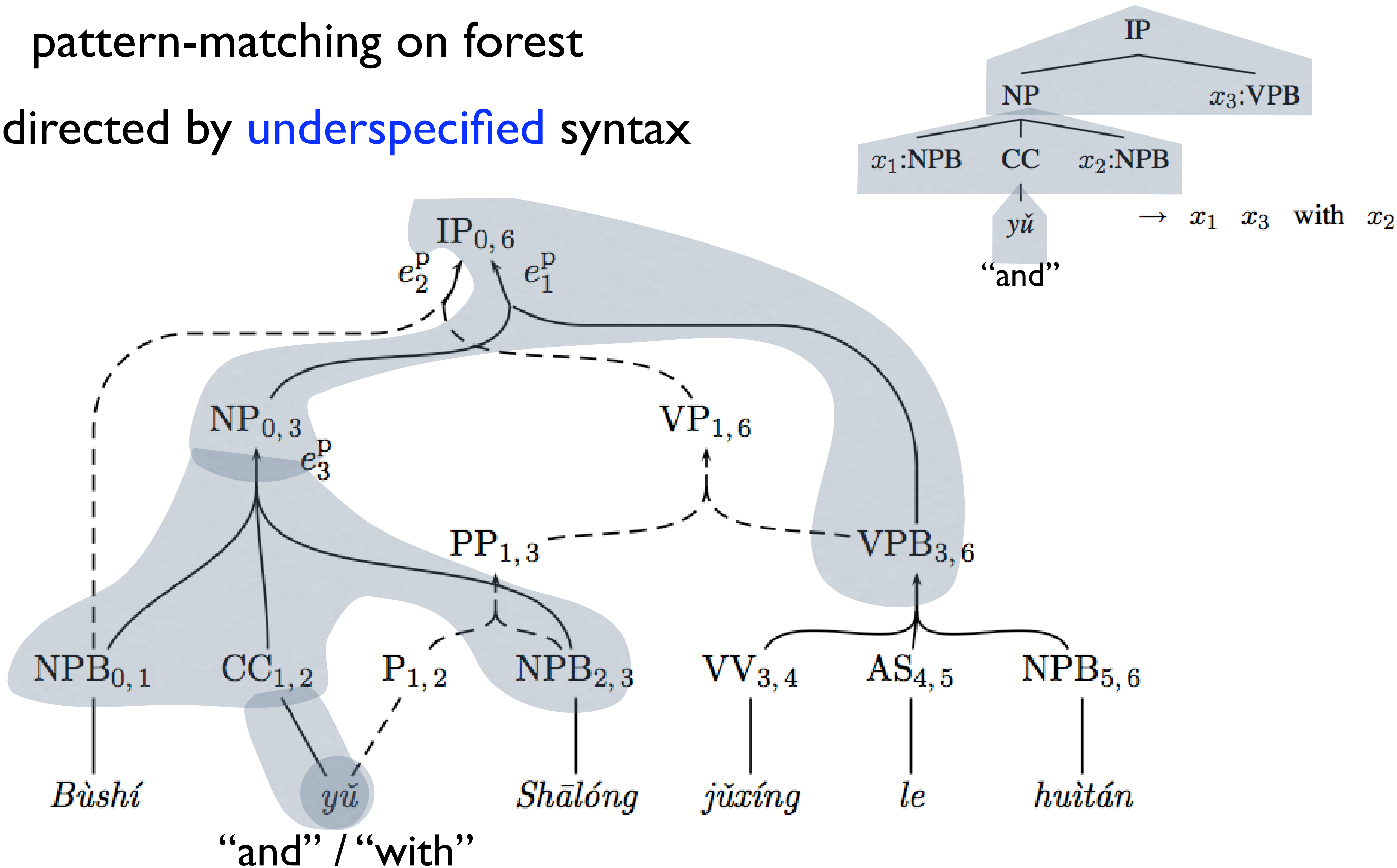
pattern-matching on forest



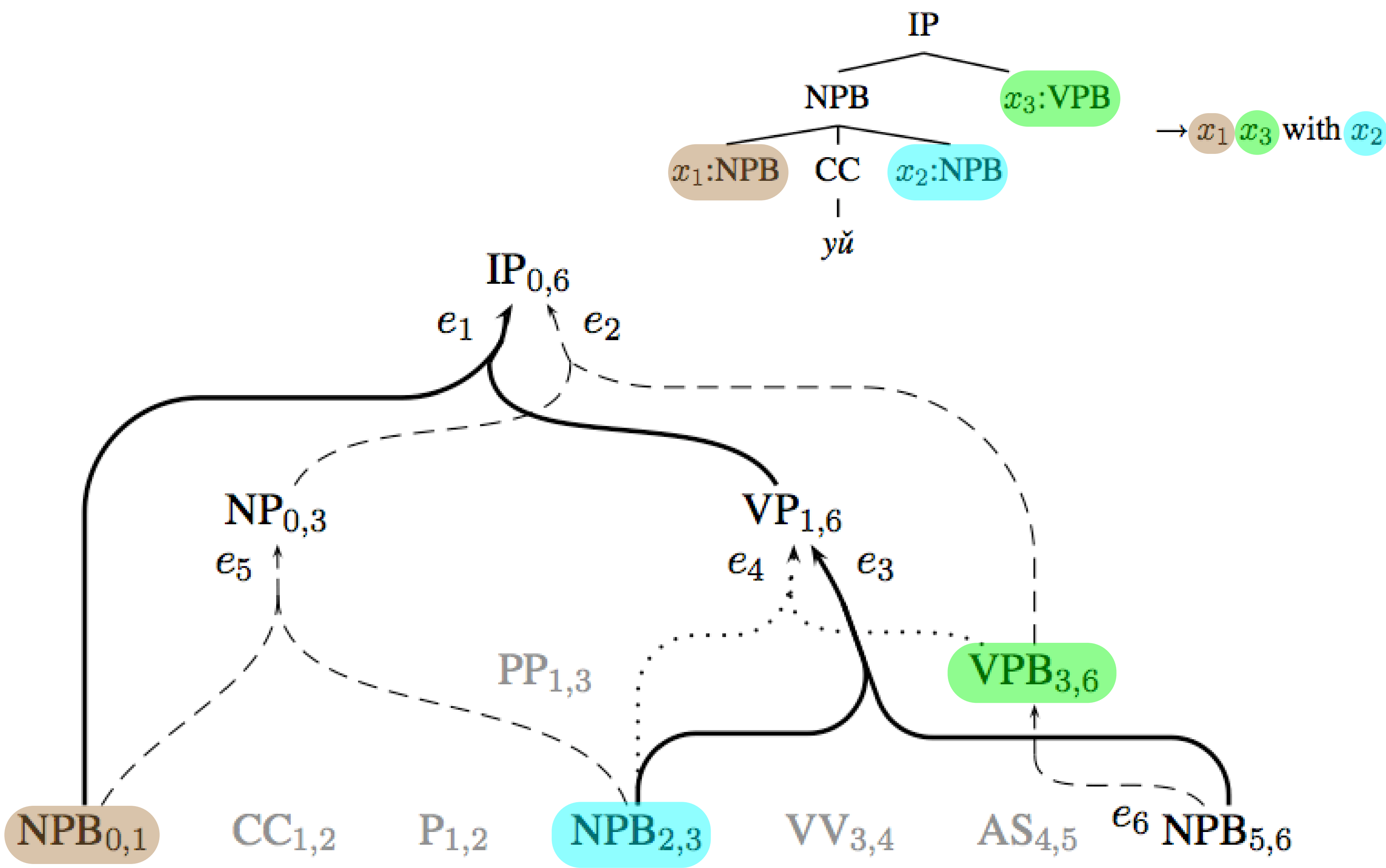
Forest-based Translation

pattern-matching on forest

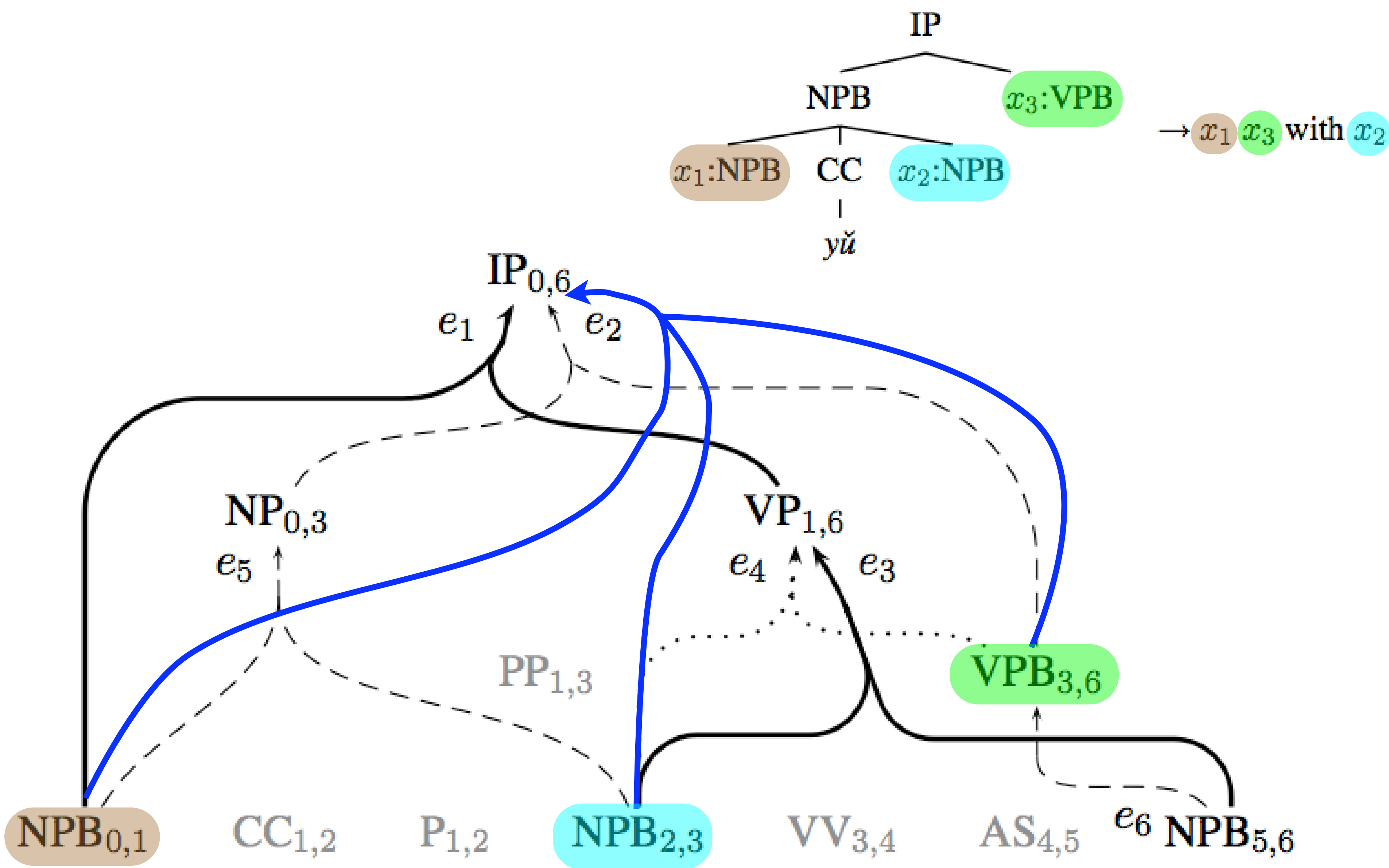
directed by underspecified syntax



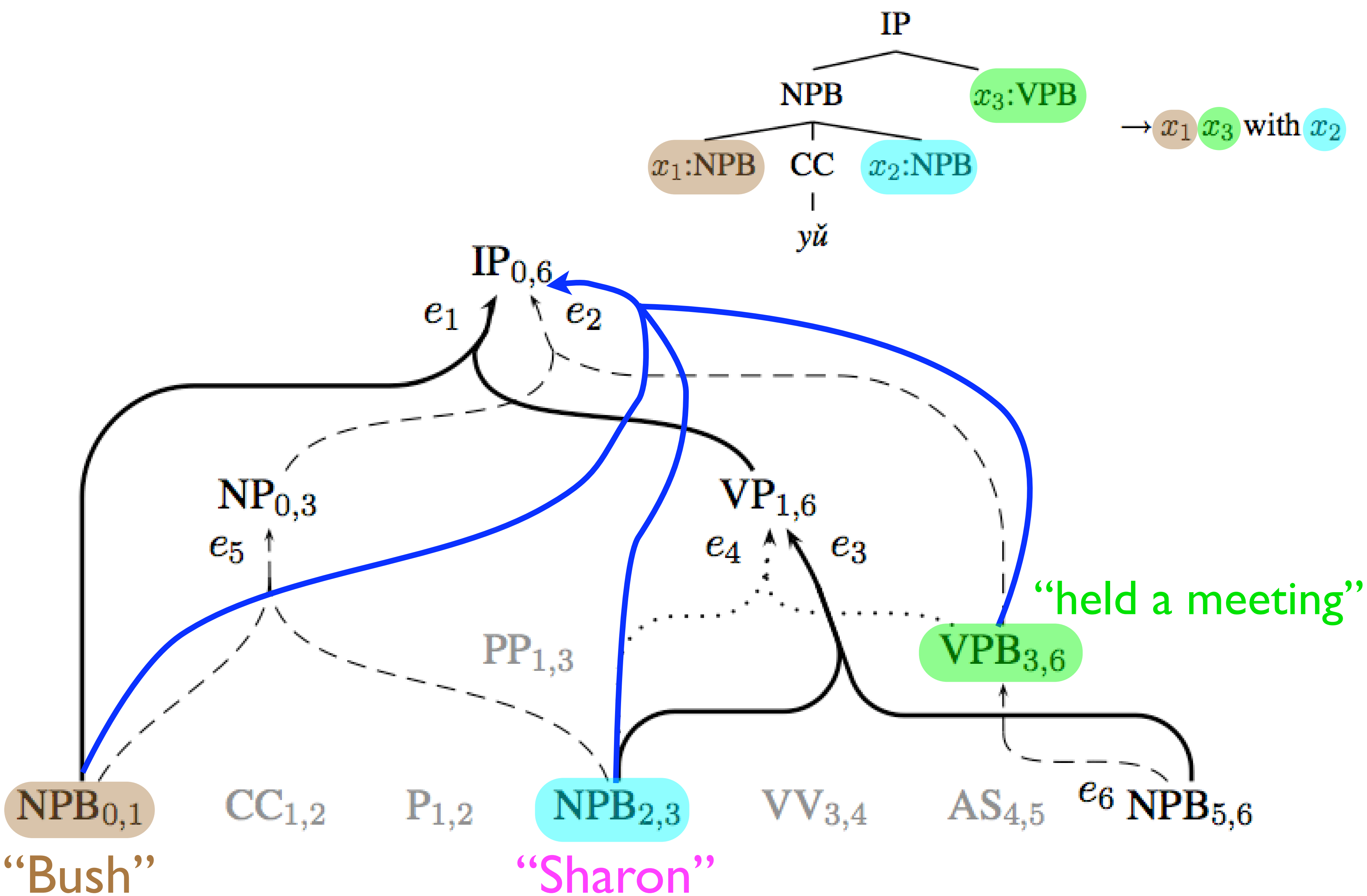
Translation Forest



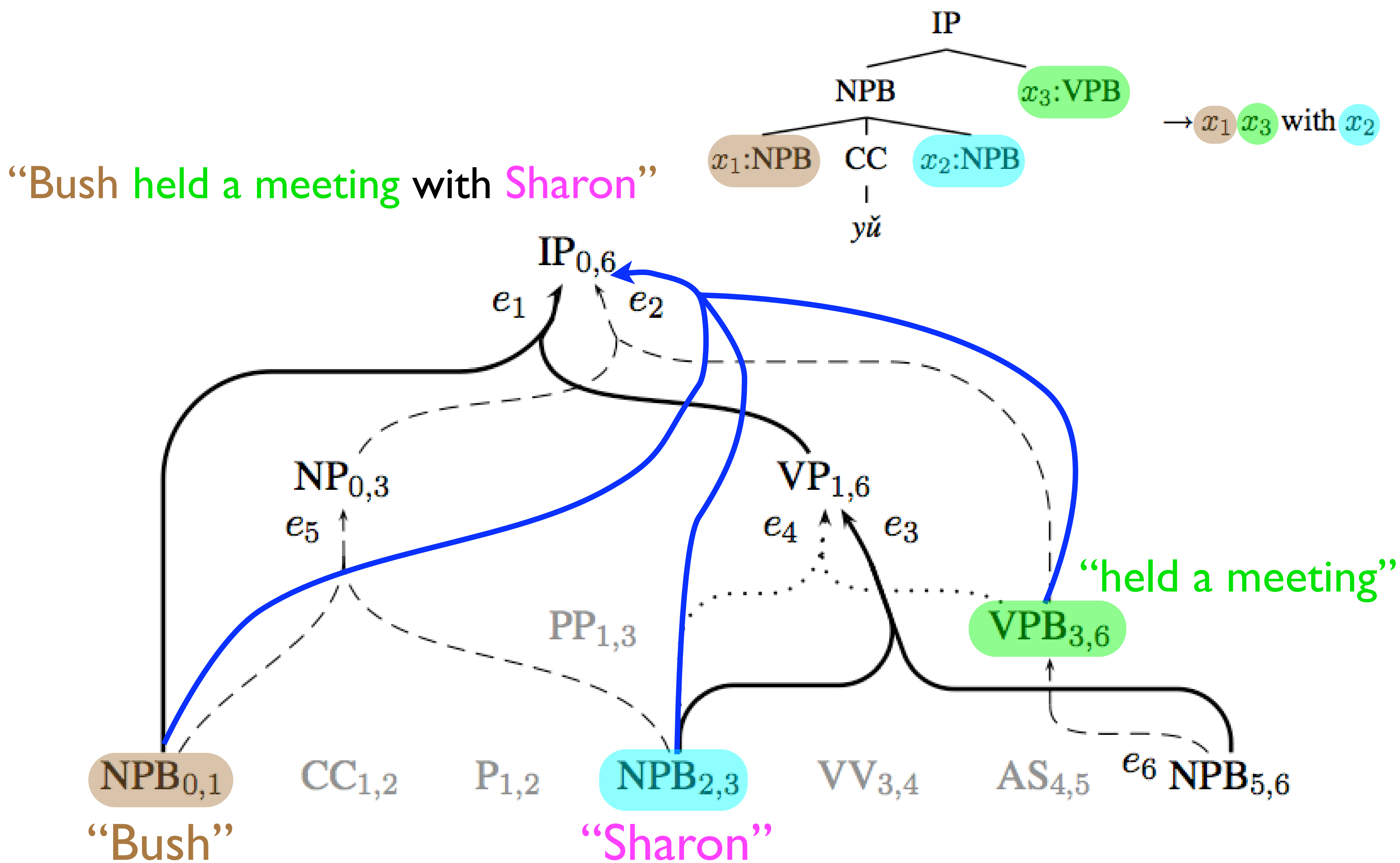
Translation Forest



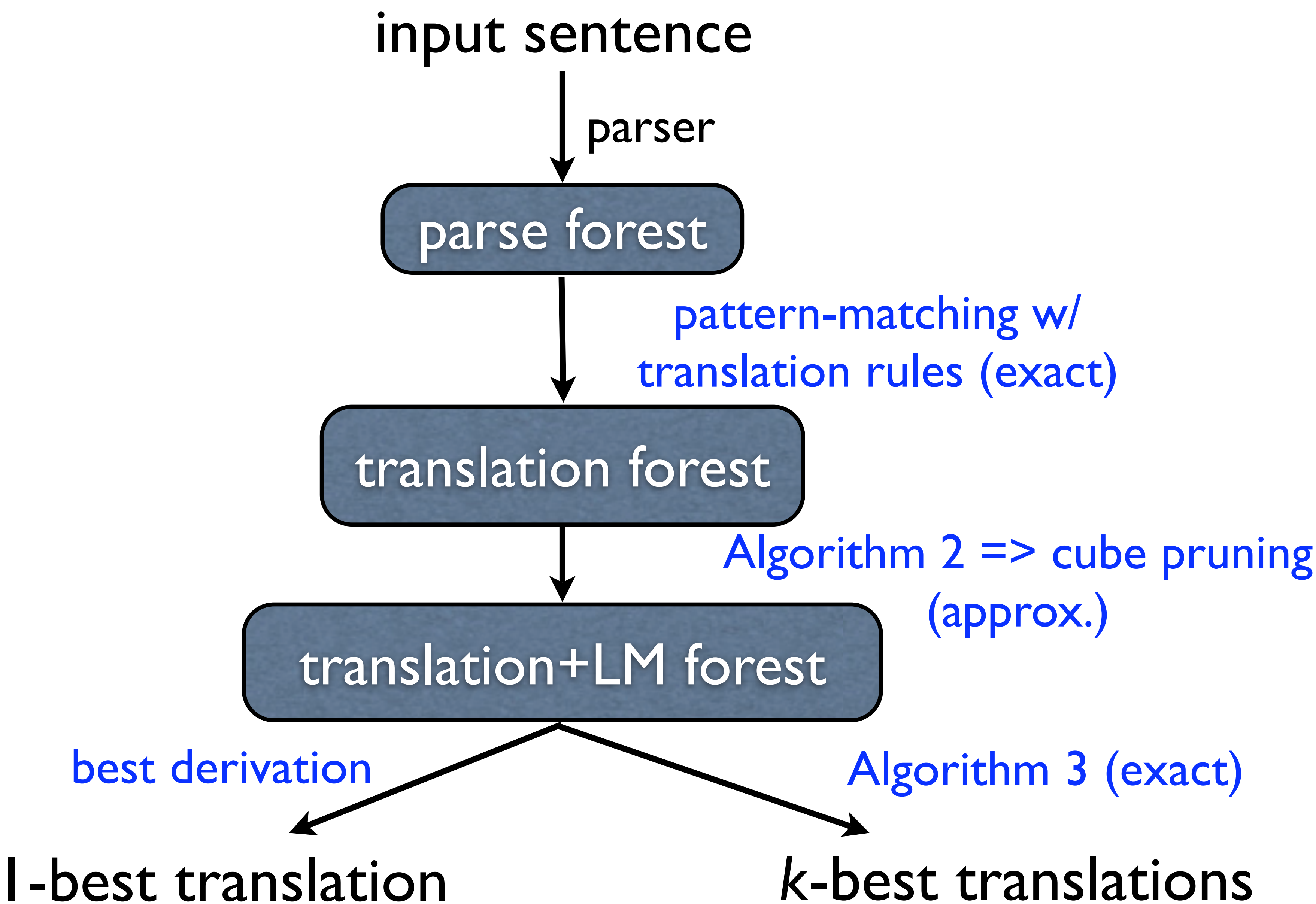
Translation Forest



Translation Forest

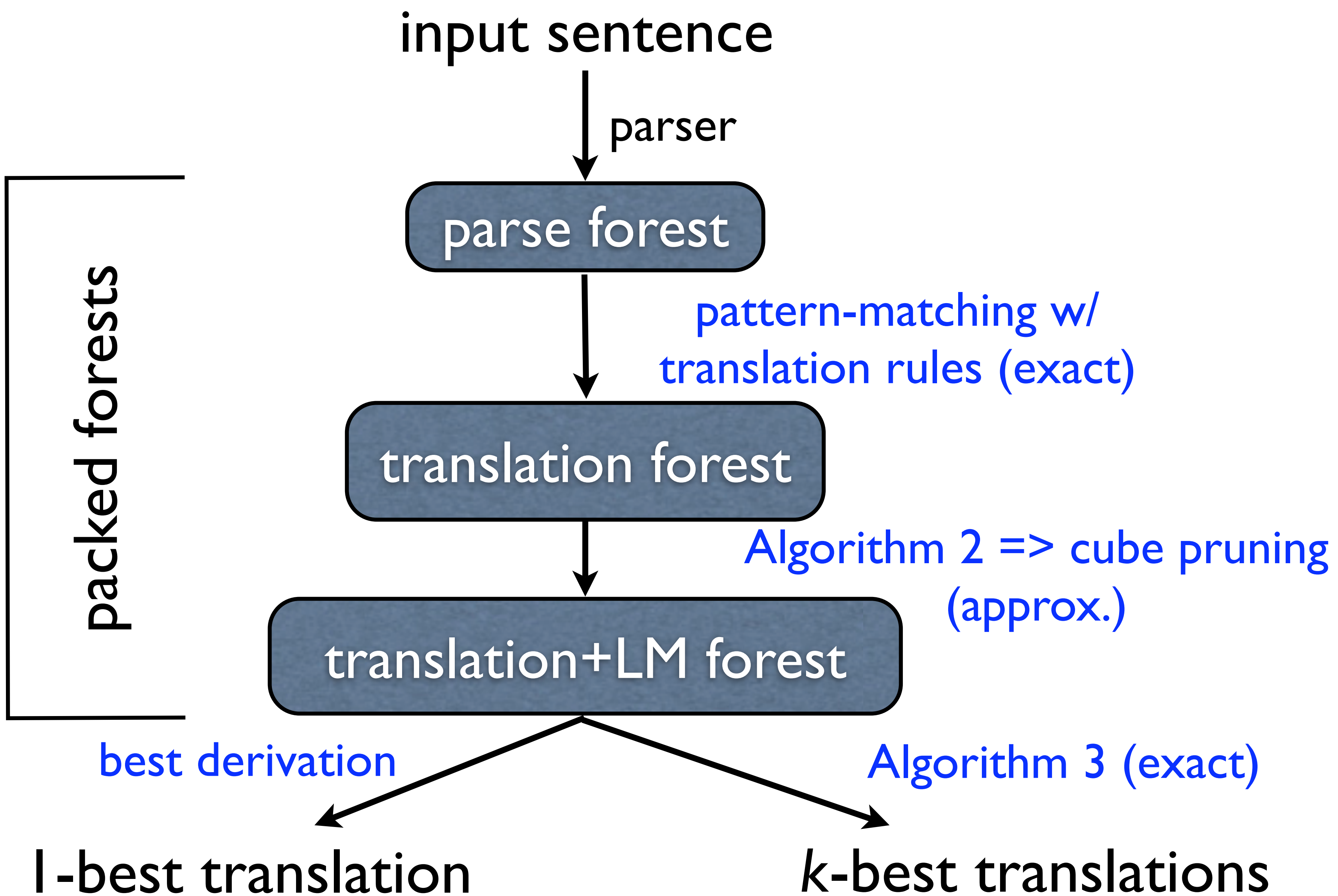


The Whole Pipeline



(Huang and Chiang, 2005; 2007; Chiang, 2007)

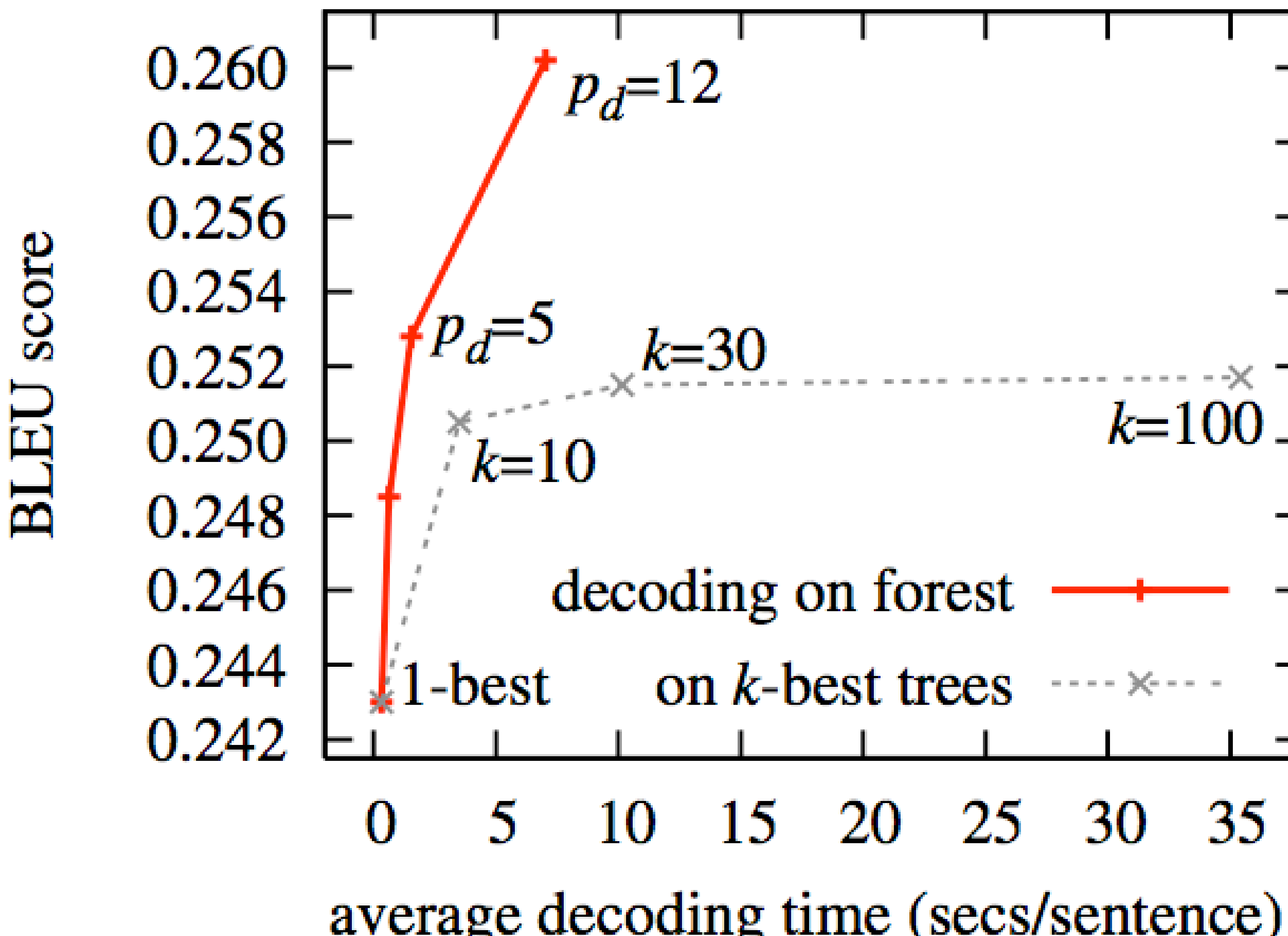
The Whole Pipeline



(Huang and Chiang, 2005; 2007; Chiang, 2007)

k-best trees vs. forest-based

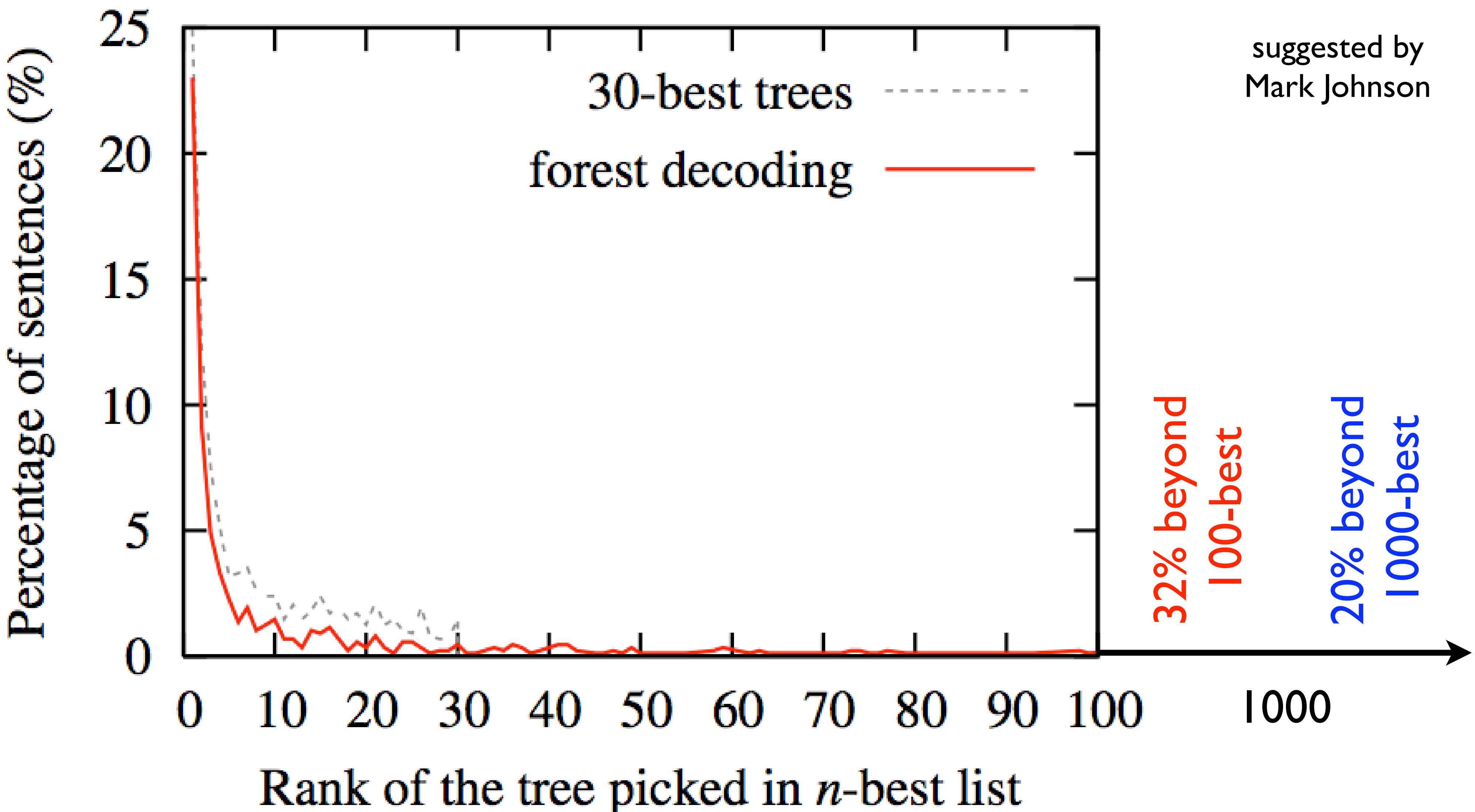
1.7 Bleu improvement over 1-best,
0.8 over 30-best, and even faster!



forest as virtual ∞-best list

- how often is the i^{th}-best tree picked by the decoder?

suggested by
Mark Johnson



Larger Decoding Experiments

- 2.2M sentence pairs (57M Chinese and 62M English words)
- larger trigram models (1/3 of Xinhua Gigaword)
- also use bilingual phrases (BP) as flat translation rules
  - phrases that are consistent with syntactic constituents
- forest enables larger improvement with BP

	T2S	T2S+BP
1-best tree	0.2666	0.2939
30-best trees	0.2755	0.3084
forest	0.2839	0.3149
improvement	1.7	2.1

Conclusions: Dynamic Programming

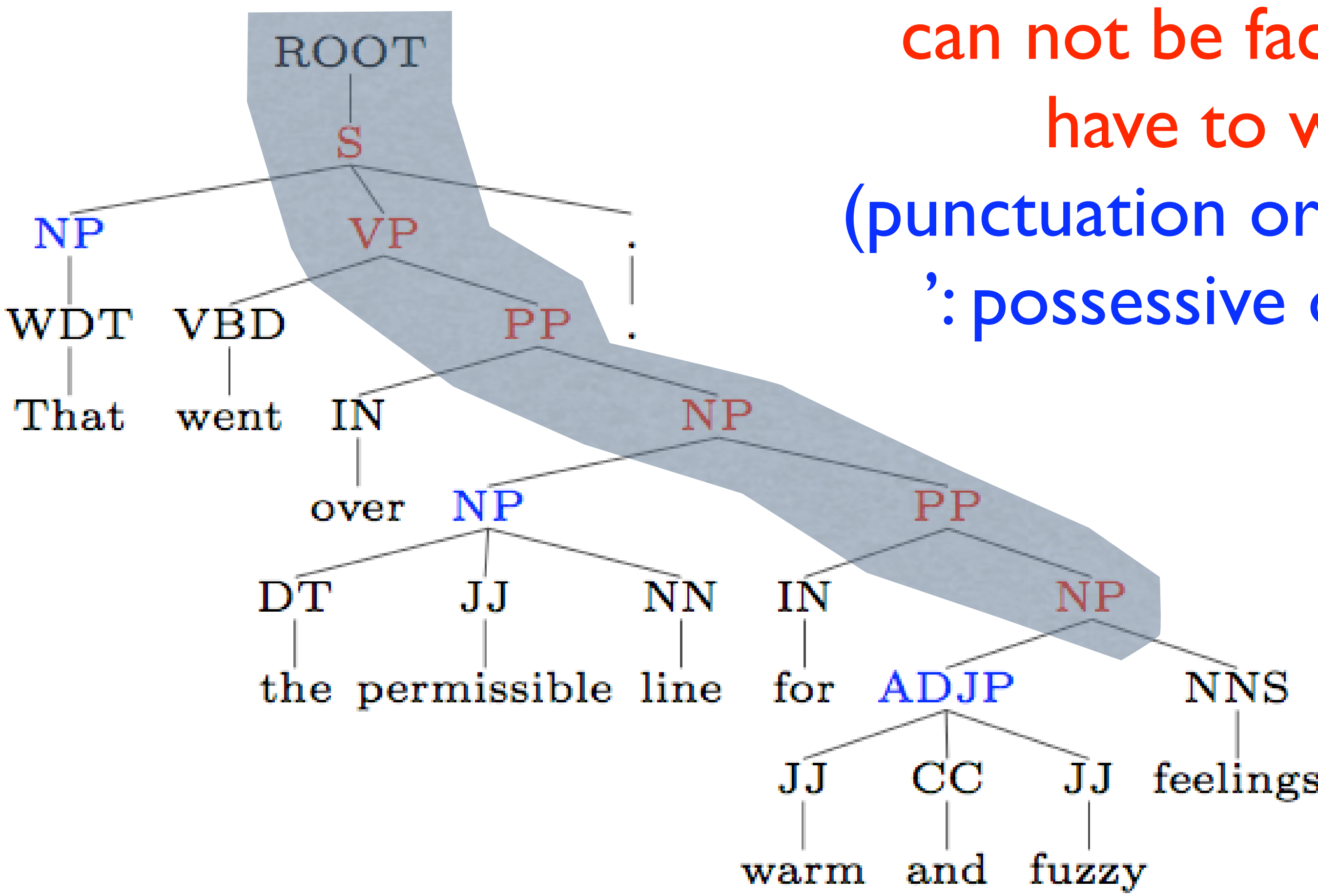
- A general framework of DP on monotonic hypergraphs
- Exact *k*-best DP algorithms (monotonic)
- Approximate DP with non-local features (non-monotonic)
  - Forest Reranking for discriminative parsing
  - Forest Rescoring for MT decoding
- Forest-based Translation
  - translates a parse forest of millions of trees
  - even faster than translating top-30 trees (and better)
- Future Directions: even faster search with richer info...

Forest is your friend.   Save the forest.

Thank you!

Global Feature - RightBranch

- length of rightmost (non-punctuation) path
  - English has a right-branching tendency



can not be factored anywhere
have to wait till root
(punctuation or not is ambiguous:
’: possessive or right quote?)

(Charniak and Johnson, 2005)