Constituent Parsing (3/9)

CS 690N, Spring 2017

Advanced Natural Language Processing

http://people.cs.umass.edu/~brenocon/anlp2017/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

- PCFG and CRF-CFG models:
    - Only allow interactions between parents and direct children
    - Key enhancement: state splitting to propagate information from above and below
- Extension: use unlabeled data
- Alternatives
    - Whole-tree models
    - History-based models

Semi-supervised training

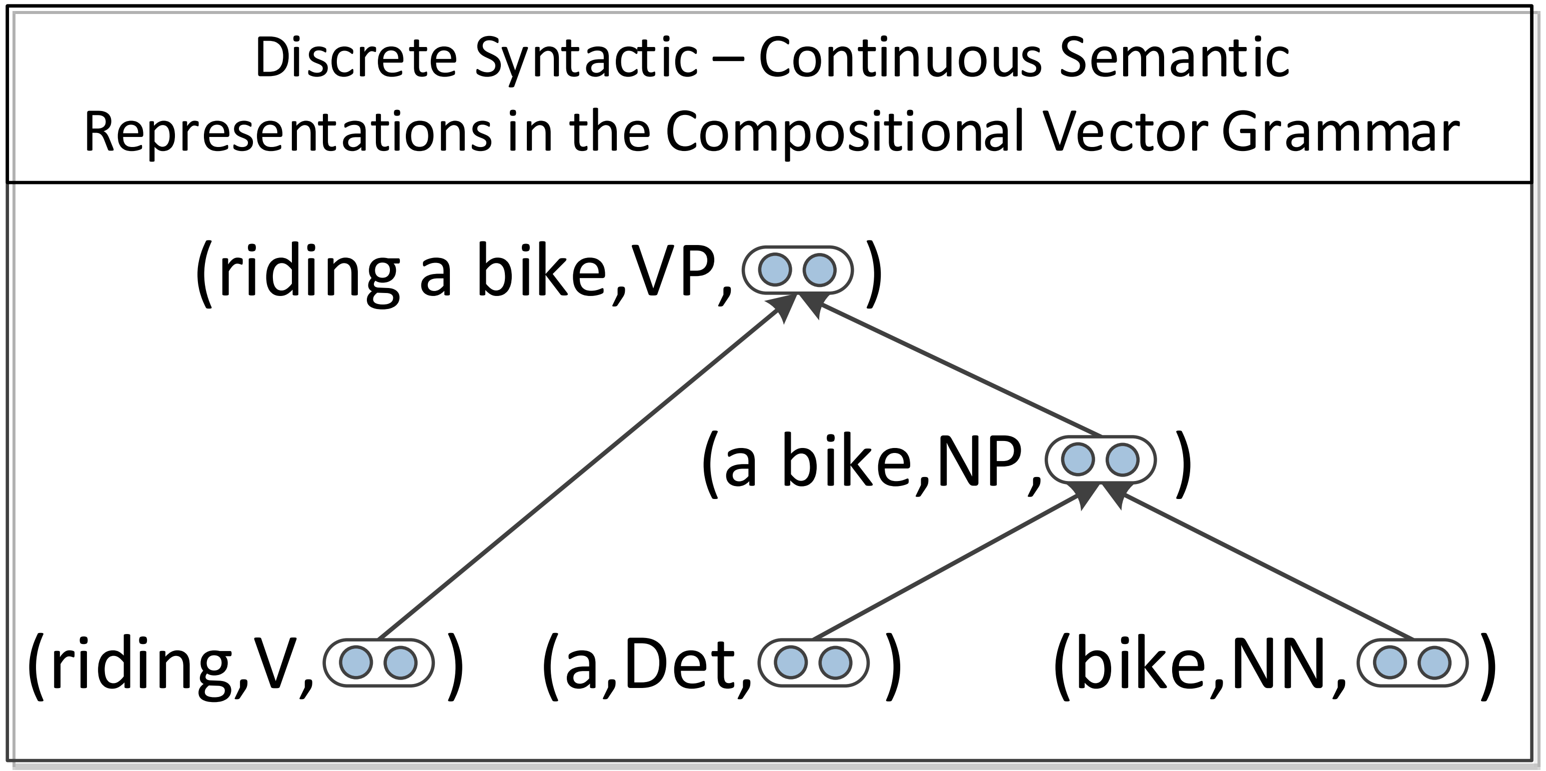
- Data
  - Labeled data (x,y) pairs +
  - Unlabeled data (x)
- EM for semi-supervised learning
  - Initialize with supervised model on labeled data
  - Assume latent y for unlabeled data; optimize total log-likelihood
  - Well-defined only for generative models
  - Trickiness with objective balancing
- Self-training: just use 1-best inferences on unlabeled data ("hard EM")
  - Variants: only use high-confidence predictions... etc.
  - "Bootstrapping" / "Bootstrapped learning"
- Improves performance *[McClosky et al. 2006]*

Discriminative re-ranking

- No more PCFG: Why not use a log-linear model with *whole-tree* features?
    - Now CKY is no longer possible. Why?
- Make it fast with **re-ranking**:
    - Take top-K trees from a PCFG.
    - Extract features for each, and re-rank them.
- Re-ranking is a very powerful general technique in NLP
    - Simple, fast model generates candidates
    - Slow, more accurate model decides the best one

Whole-tree discrim. models

- Log-linear features *[Johnson and Charniak 2005]*
  - Does this NP contain 15-20 words? Right-branching tendencies?
- Tree-structured recursive NNs *[Socher et al. 2013]*
  - Compare to *head rules* for lexicalization
  - Alternate application: hierarchical phrase sentiment analysis



Shift-reduce parsing

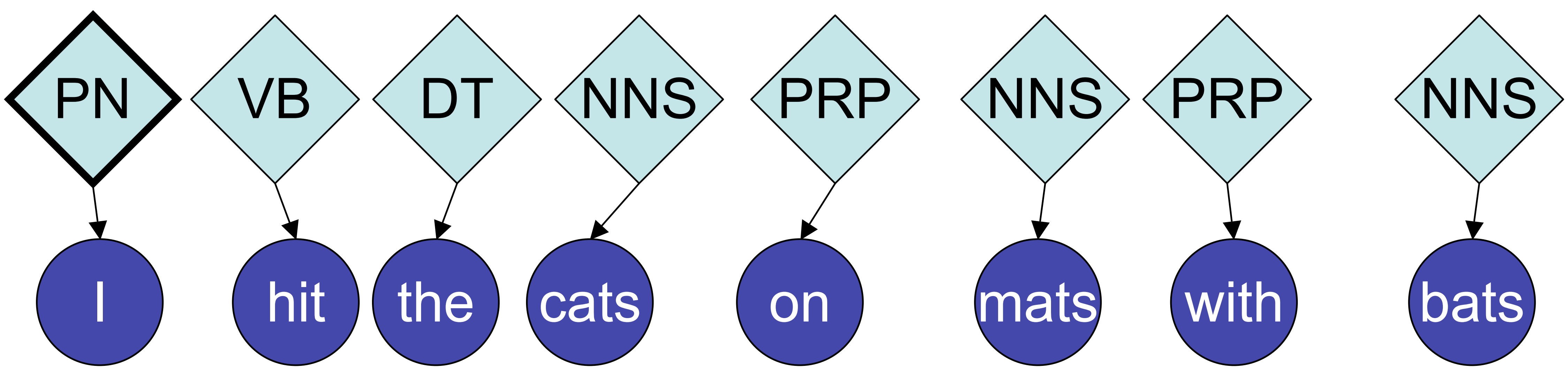
- One form of left-to-right / top-down parsing
- Incrementally build up the parse tree, scanning words left-to-right.
  - Parser as a state machine
- No dynamic programming! O(n) runtime!
- Potentially related to cognitive processing?
- Most practically efficient for constituent parsing -- e.g. *zpar* and *CoreNLP* implementations

Example from a similar incremental parser (slightly different than current work)

Ratnaparkhi (1998)

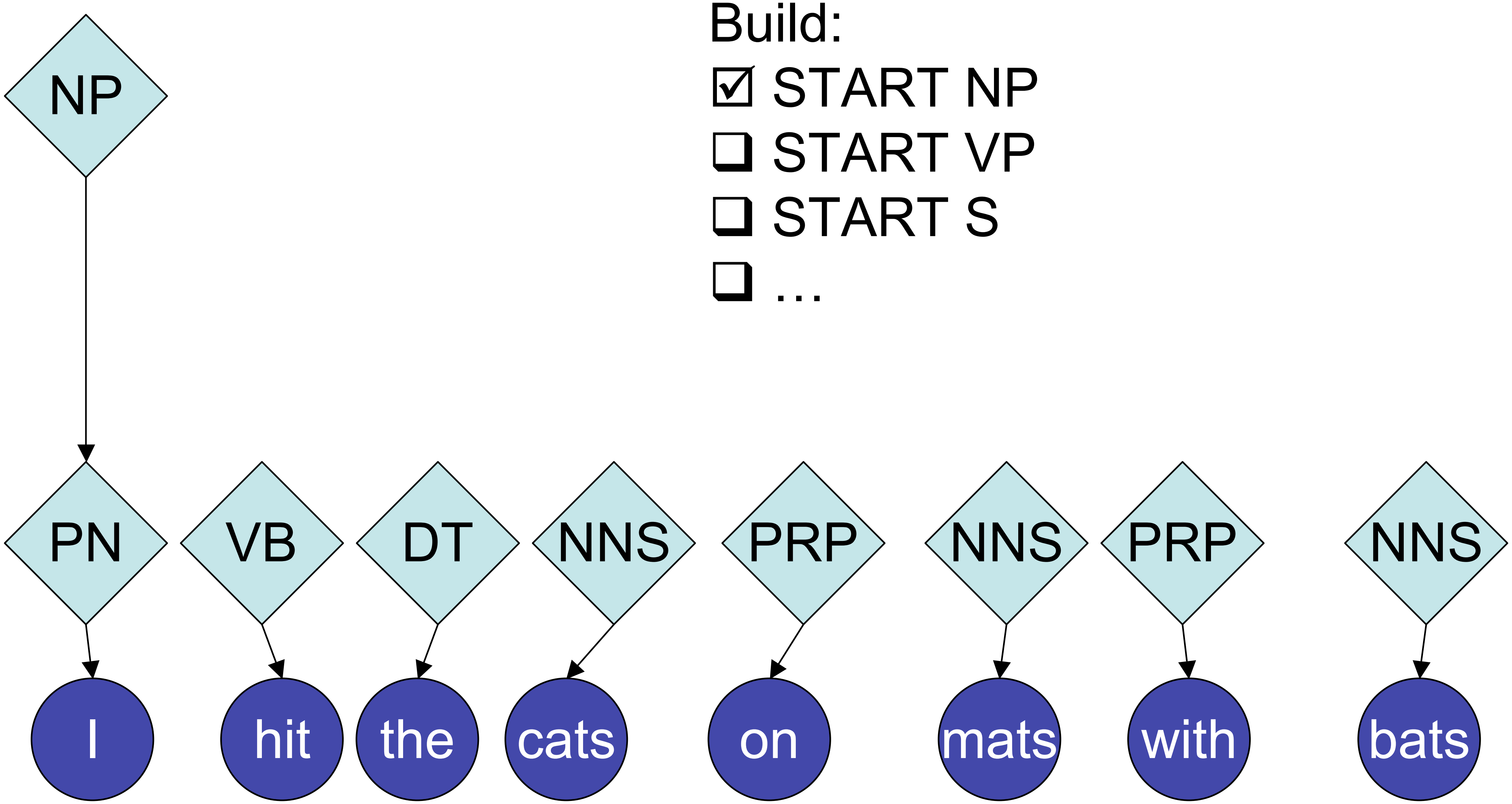
Build:

- ❑ START NP
- ❑ START VP
- ❑ START S
- ❑ …



[Slides: Noah Smith]

Ratnaparkhi (1998)



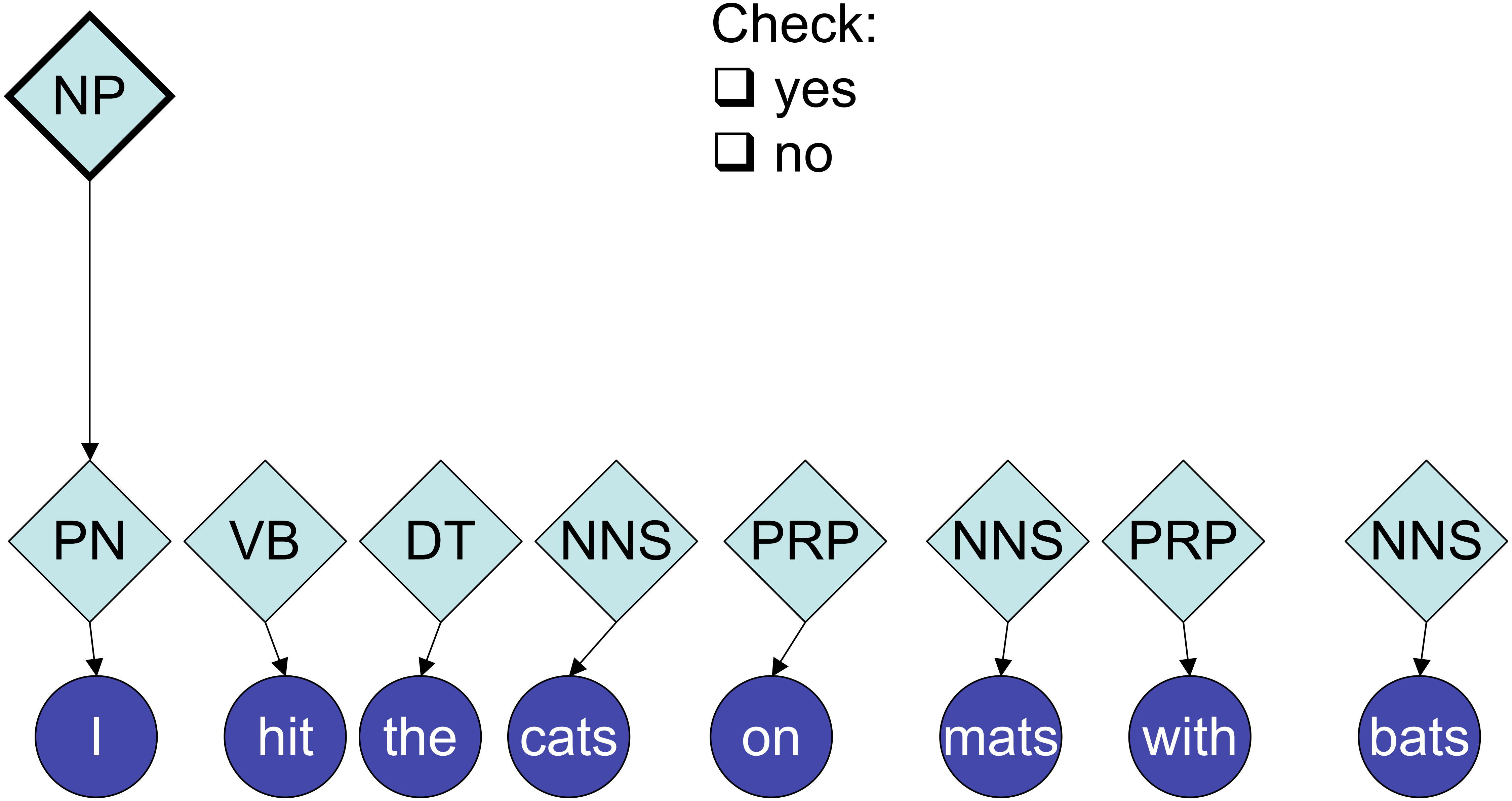
Build:

- ☑ START NP
- ❑ START VP
- ❑ START S
- ❑ …

[Slides: Noah Smith]

Ratnaparkhi (1998)

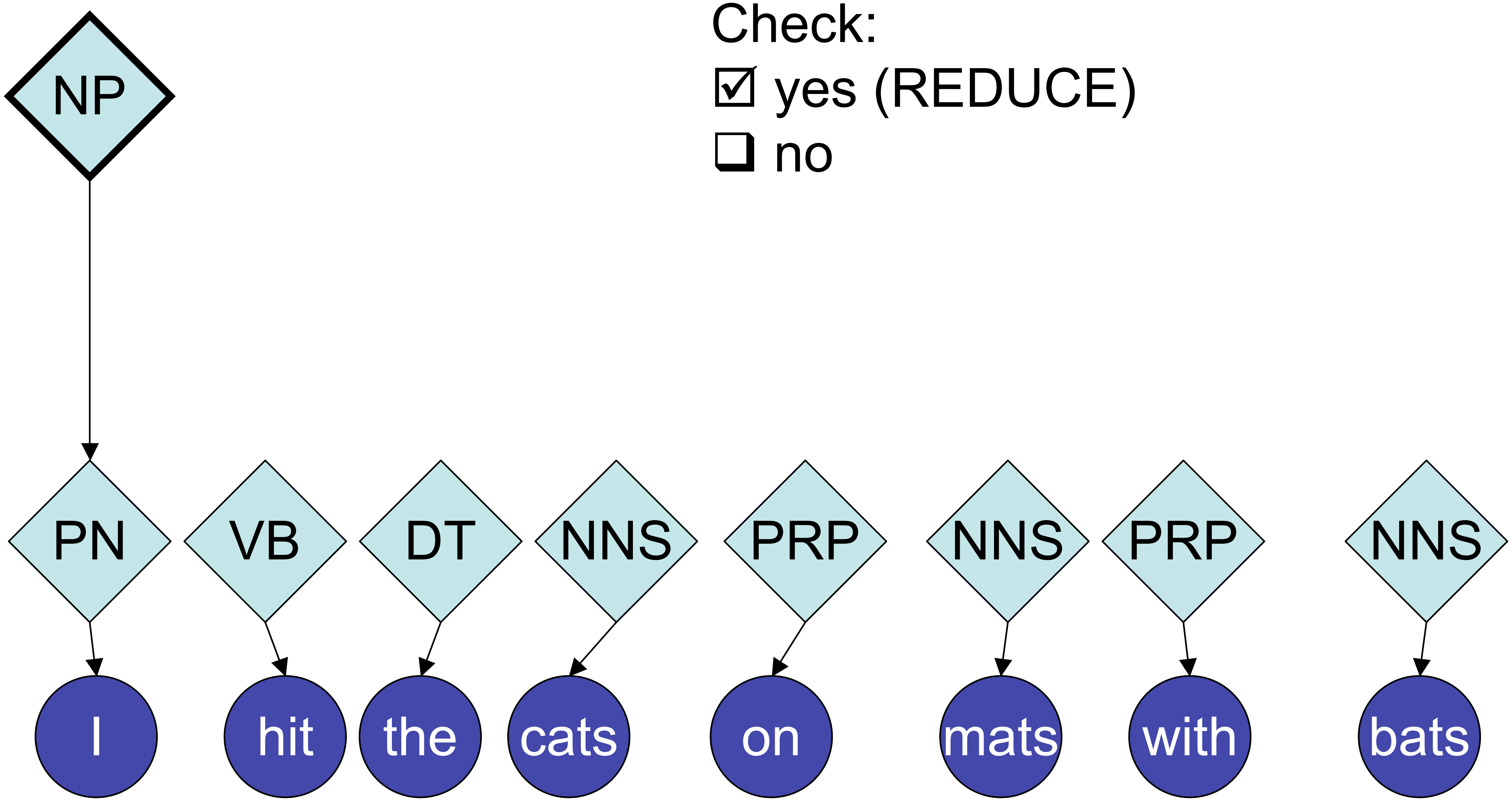
Check:
❑ yes
❑ no



[Slides: Noah Smith]

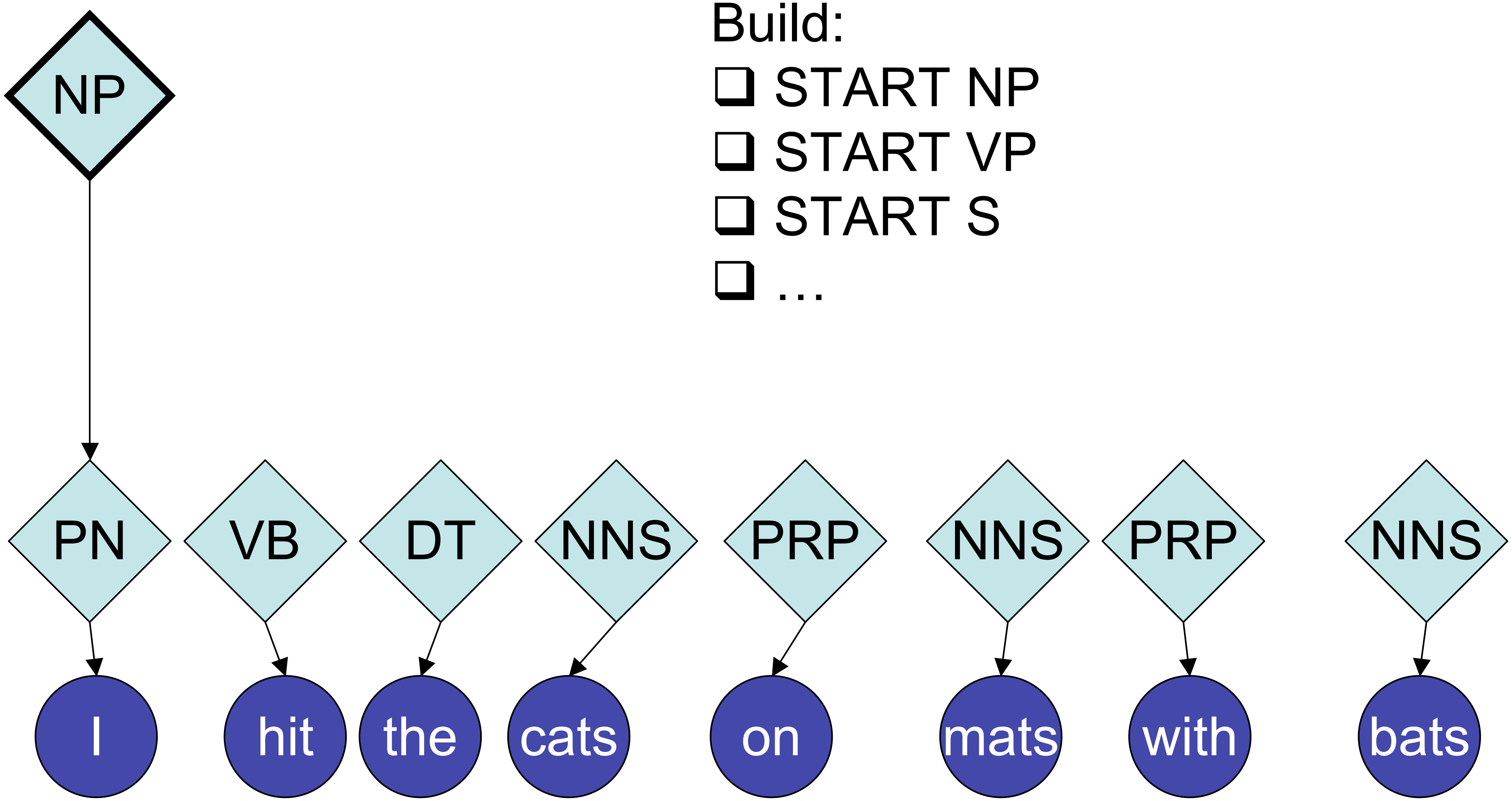
Ratnaparkhi (1998)

Check:
☑ yes (REDUCE)
❑ no



[Slides: Noah Smith]

Ratnaparkhi (1998)

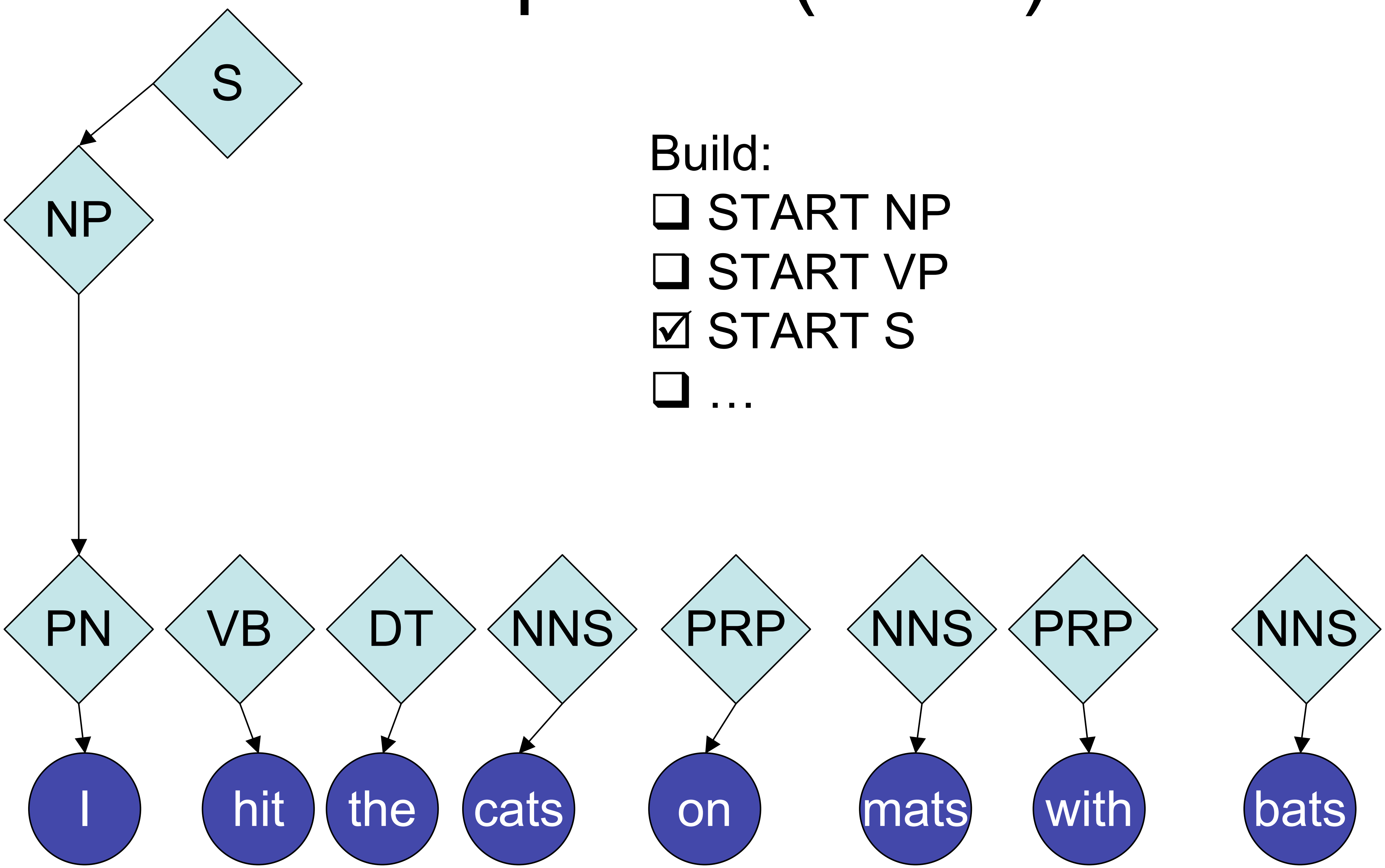


Build:

- ❑ START NP
- ❑ START VP
- ❑ START S
- ❑ …

[Slides: Noah Smith]

Ratnaparkhi (1998)

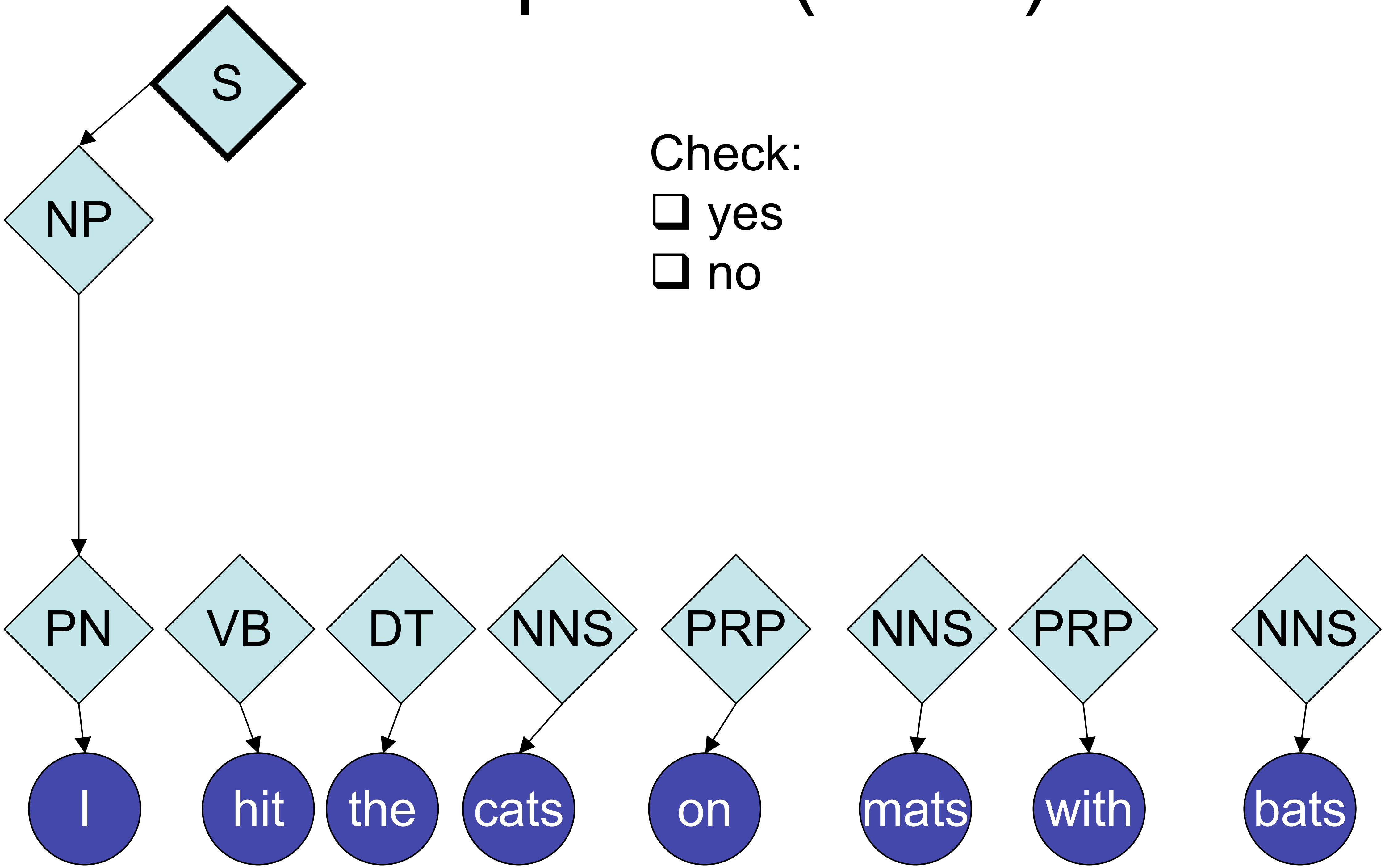


Build:

- ❑ START NP
- ❑ START VP
- ☑ START S
- ❑ …

[Slides: Noah Smith]

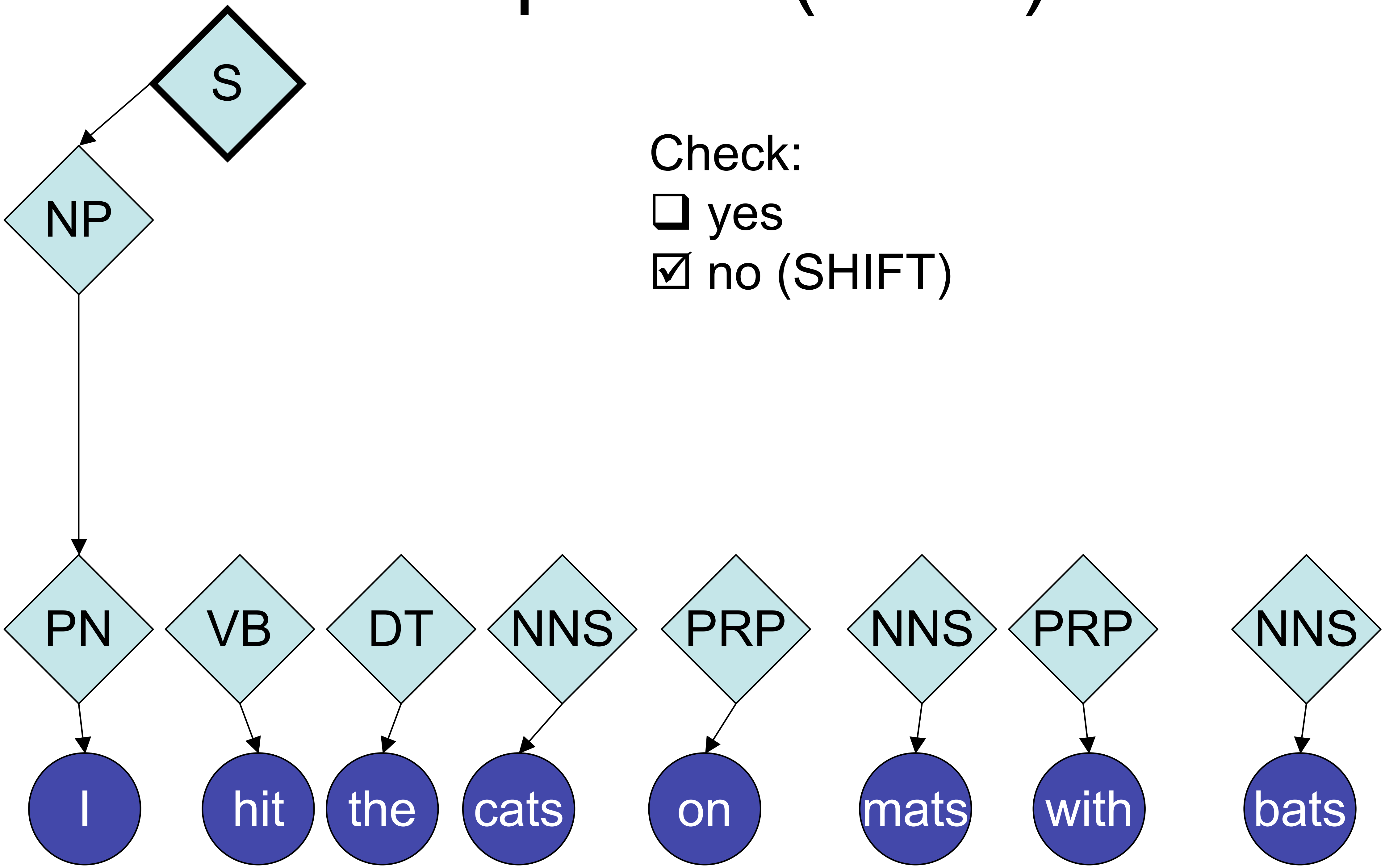
Ratnaparkhi (1998)



Check:
- ❑ yes
- ❑ no

[Slides: Noah Smith]

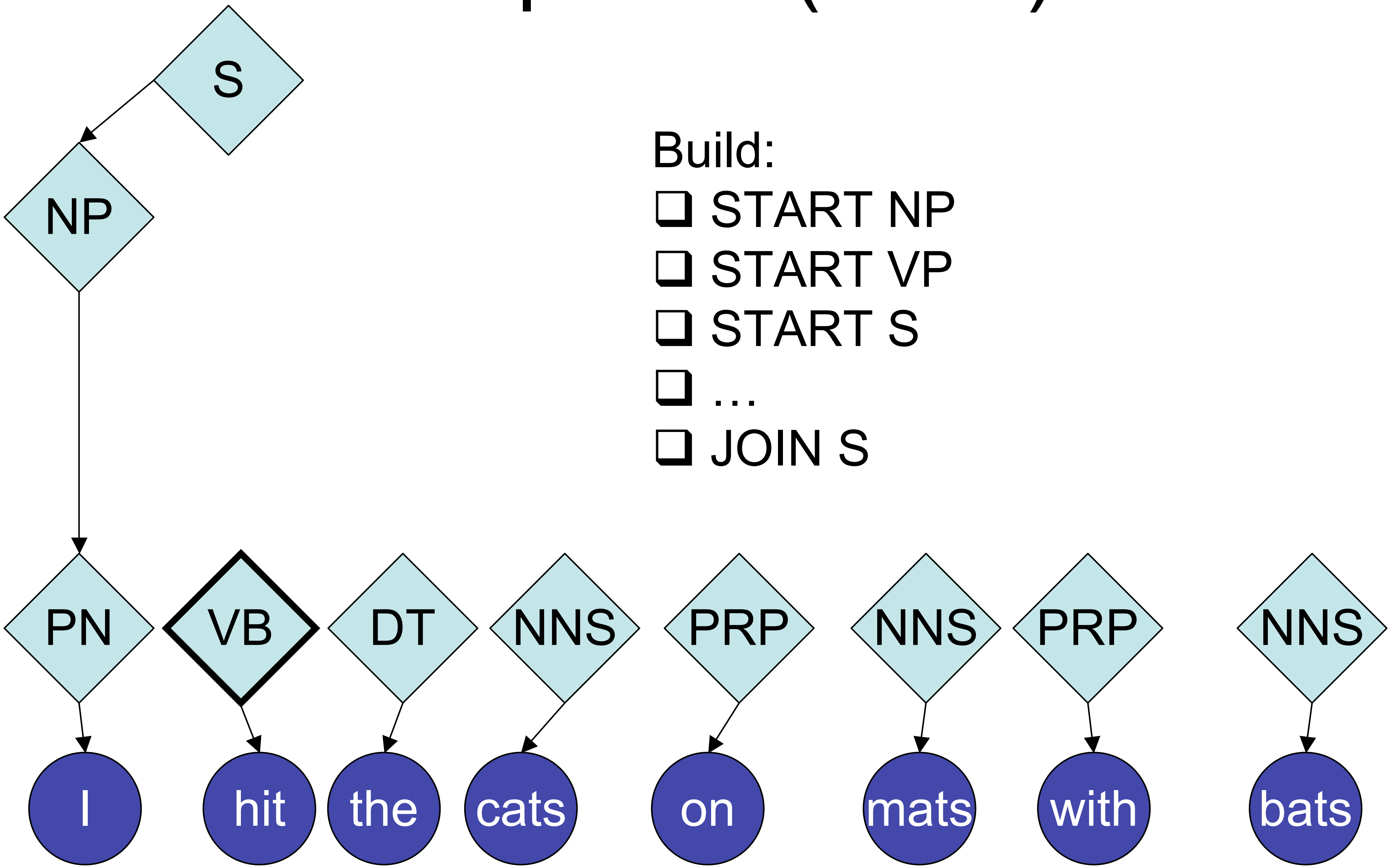
Ratnaparkhi (1998)



Check:
❑ yes
☑ no (SHIFT)

[Slides: Noah Smith]

Ratnaparkhi (1998)

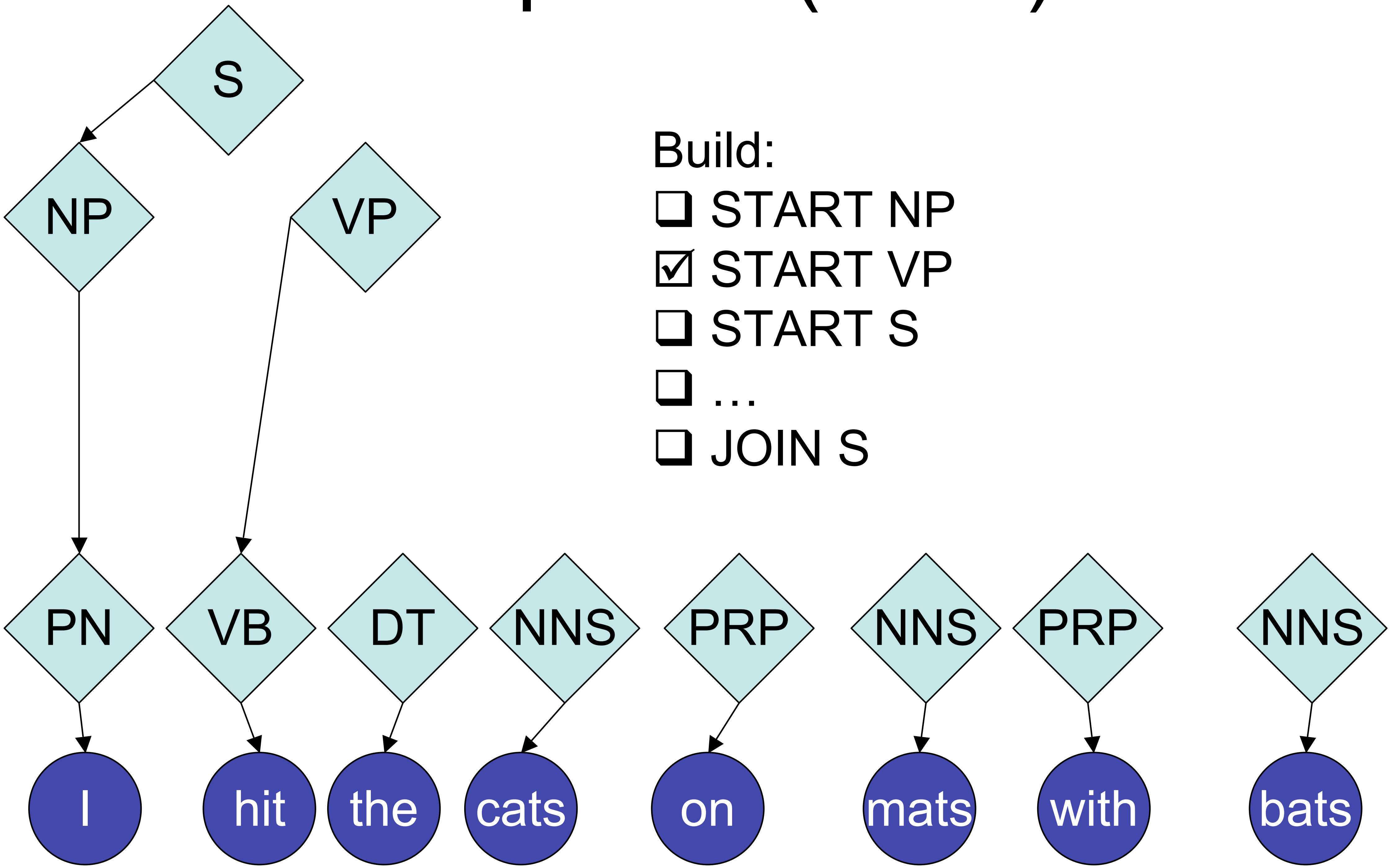


Build:

- ❑ START NP
- ❑ START VP
- ❑ START S
- ❑ …
- ❑ JOIN S

[Slides: Noah Smith]

Ratnaparkhi (1998)

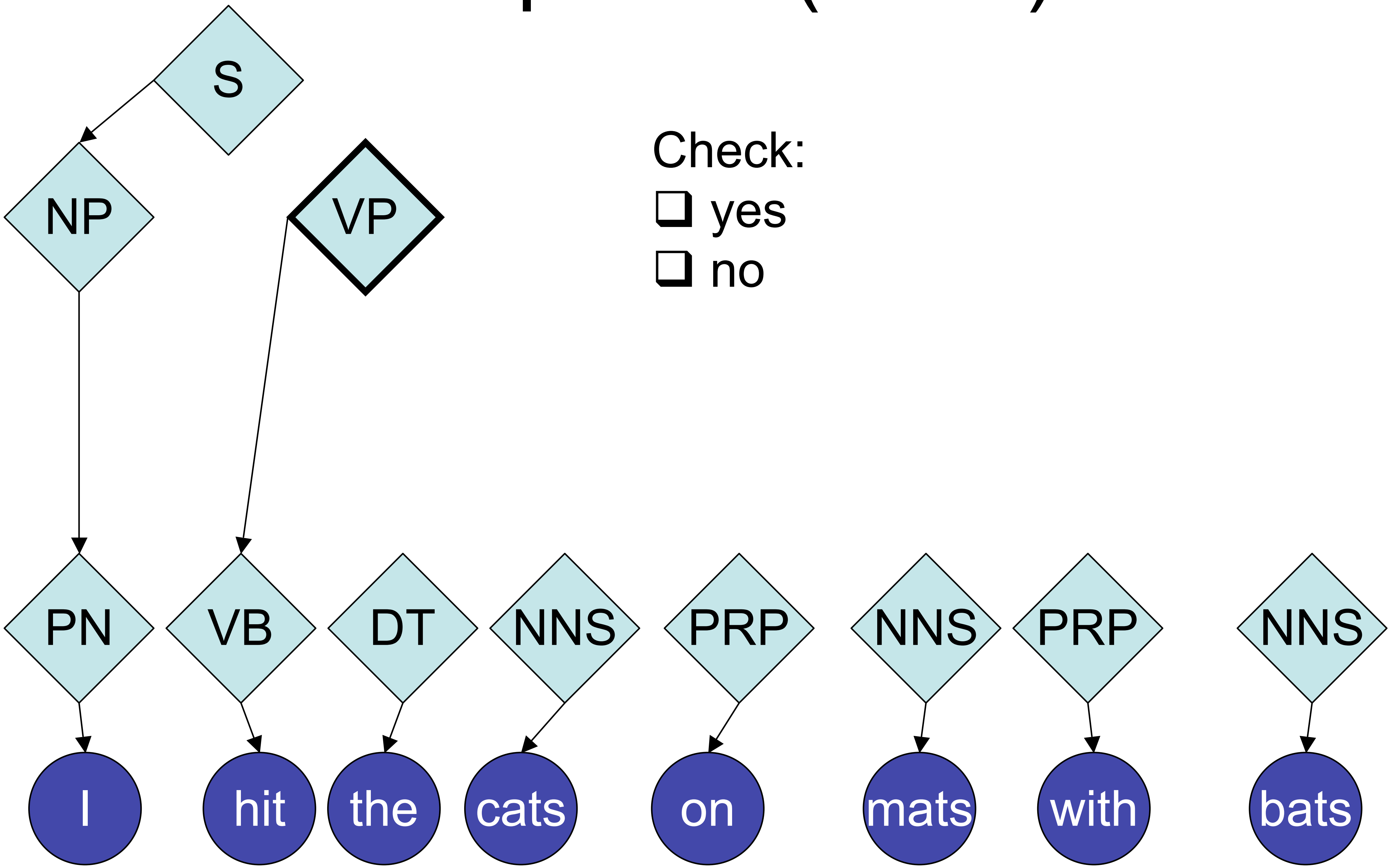


Build:

- ☐ START NP
- ☑ START VP
- ☐ START S
- ☐ …
- ☐ JOIN S

[Slides: Noah Smith]

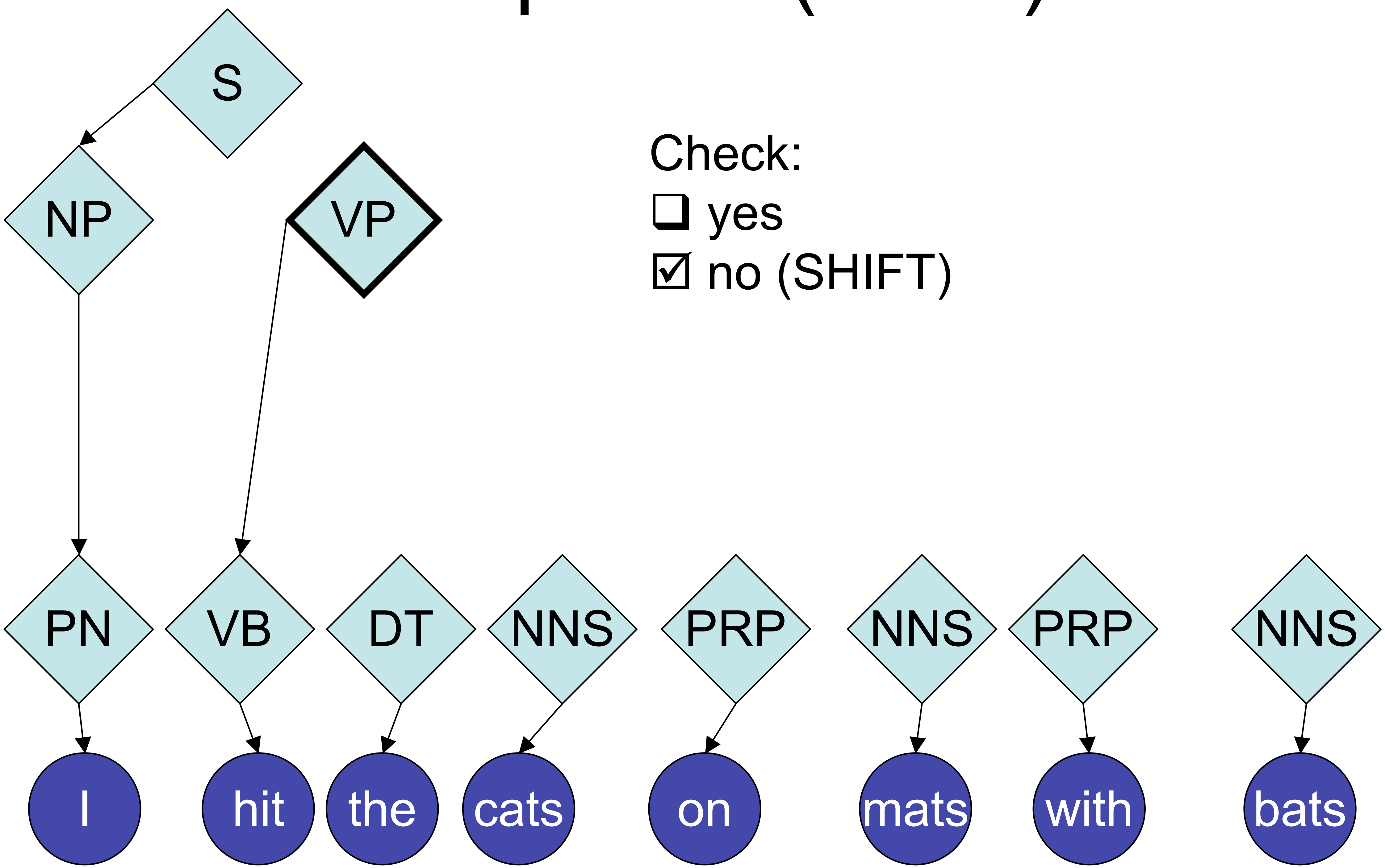
Ratnaparkhi (1998)



Check:
- ❑ yes
- ❑ no

[Slides: Noah Smith]

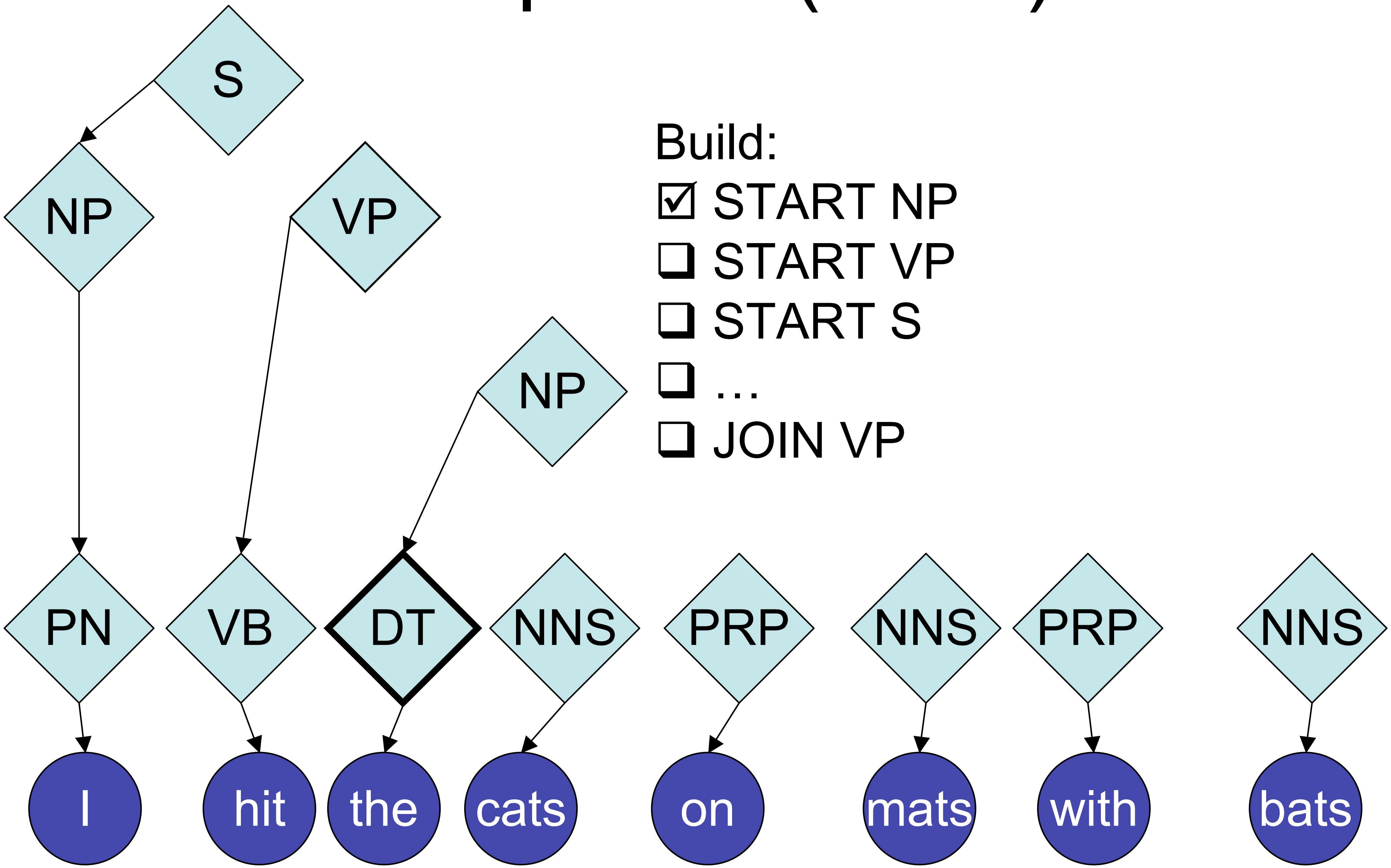
Ratnaparkhi (1998)



Check:
- ❑ yes
- ☑ no (SHIFT)

[Slides: Noah Smith]

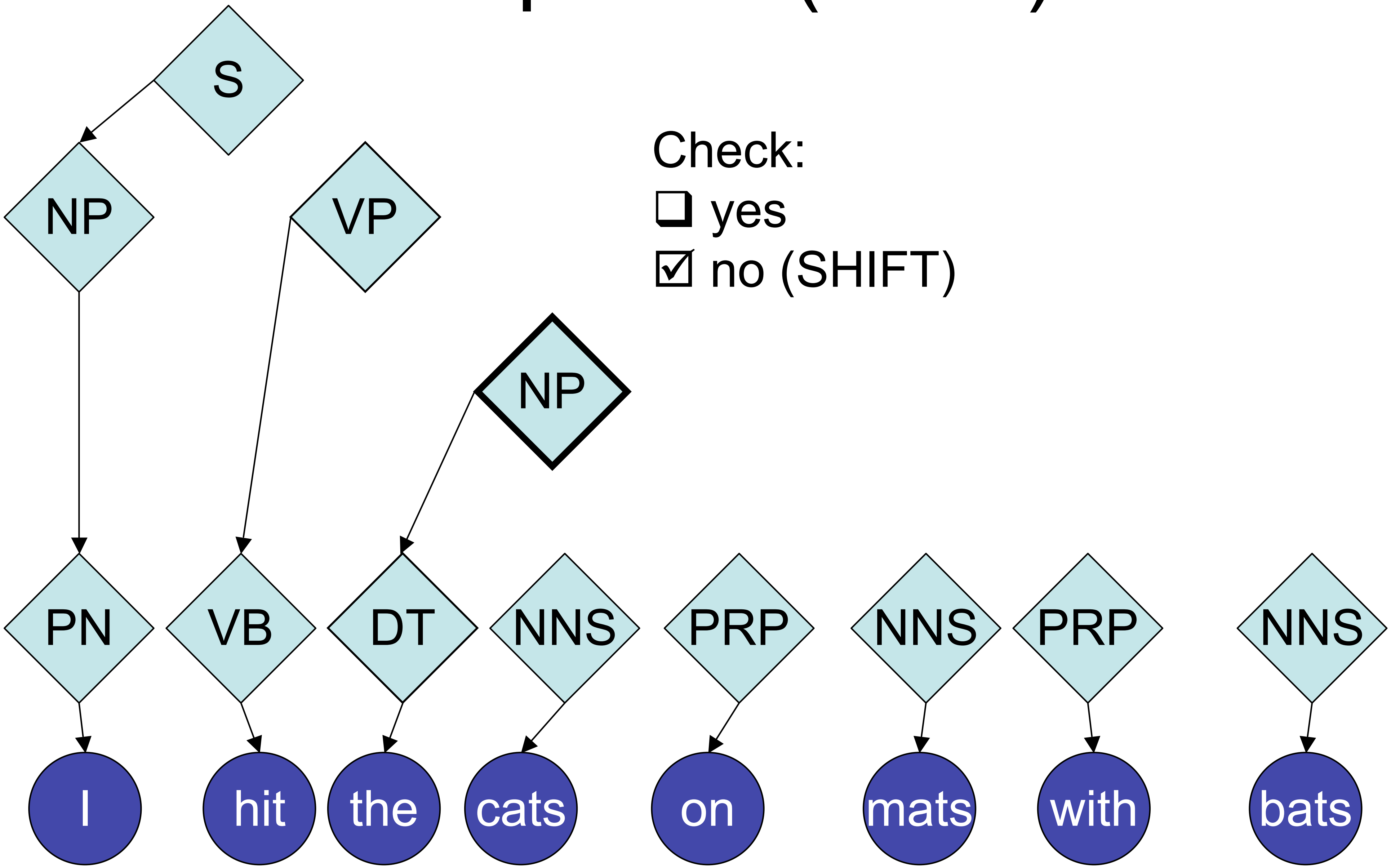
Ratnaparkhi (1998)



Build:
- ☑ START NP
- ❑ START VP
- ❑ START S
- ❑ …
- ❑ JOIN VP

[Slides: Noah Smith]

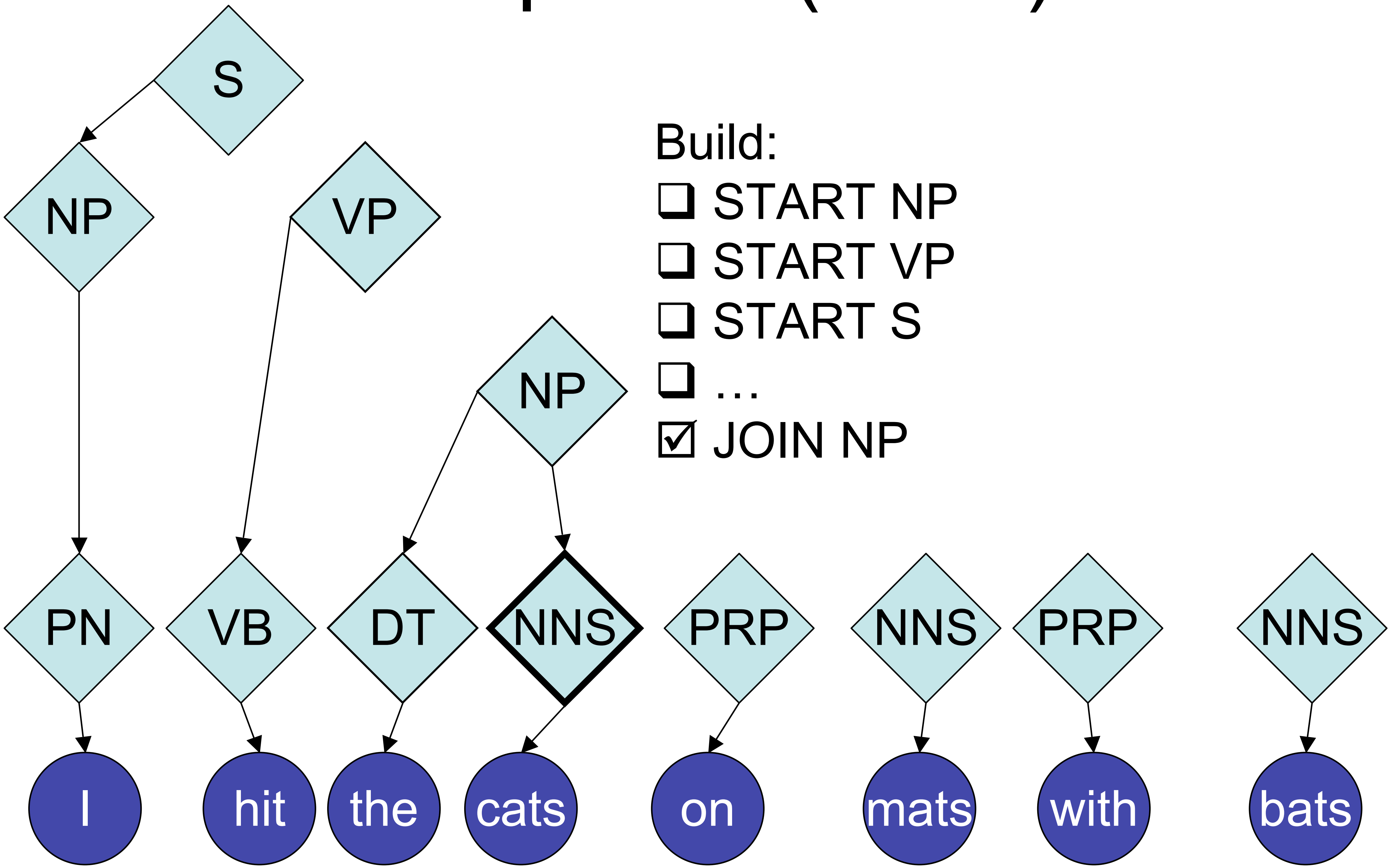
Ratnaparkhi (1998)



Check:
- ❑ yes
- ☑ no (SHIFT)

[Slides: Noah Smith]

Ratnaparkhi (1998)

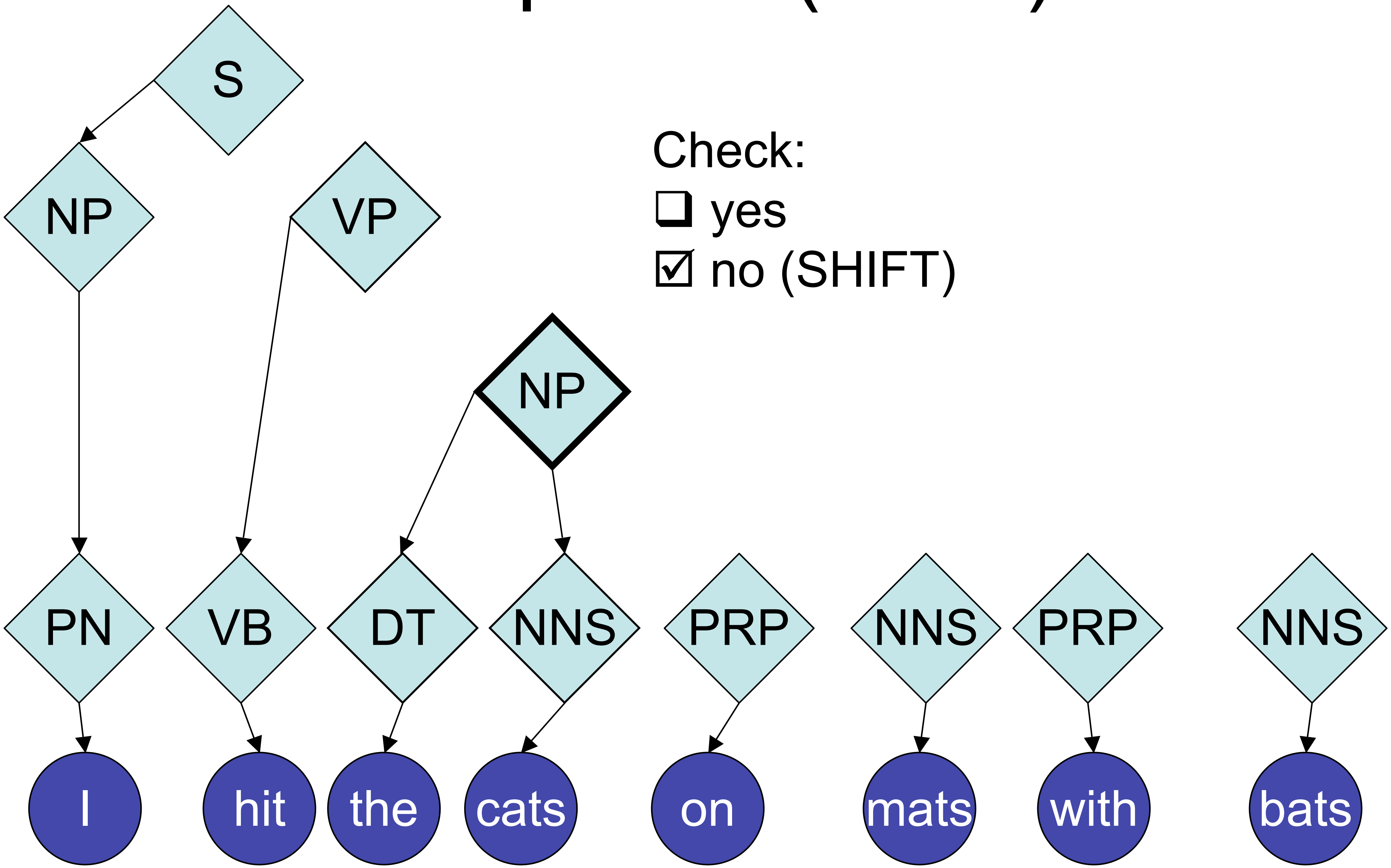


Build:

- ☐ START NP
- ☐ START VP
- ☐ START S
- ☐ …
- ☑ JOIN NP

[Slides: Noah Smith]

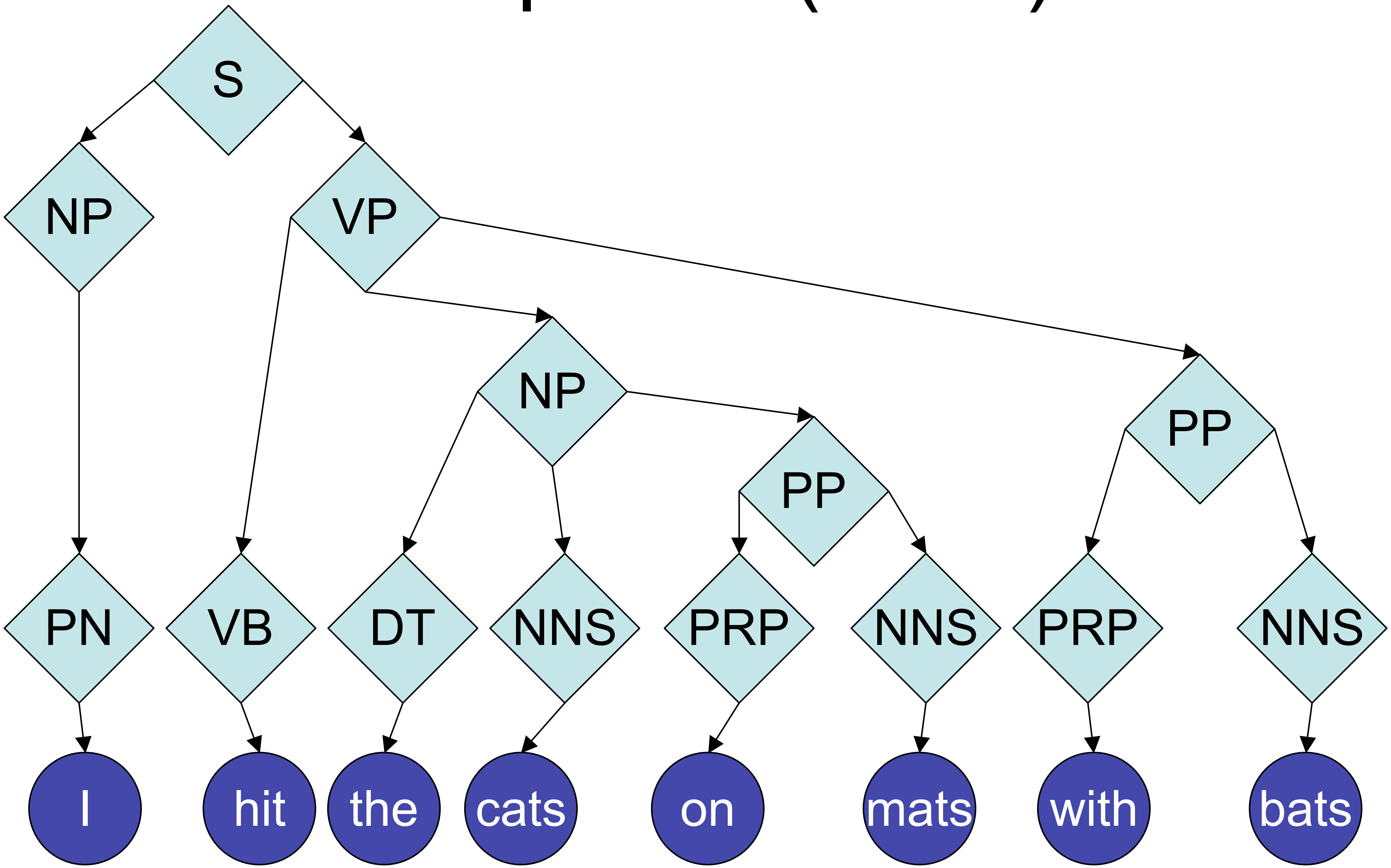
Ratnaparkhi (1998)



Check:
❑ yes
☑ no (SHIFT)

[Slides: Noah Smith]

Ratnaparkhi (1998)



[Slides: Noah Smith]

Shift-reduce parsing

- State machine: *stack* and *buffer*
- Decide on one of 3 *actions*

Stack$_t$	**Buffer**$_t$	**Open NTs**$_t$	**Action**	**Stack**$_{t+1}$	**Buffer**$_{t+1}$	**Open NTs**$_{t+1}$
S	B	n	NT(X)	$S \mid (\mathrm{X}$	B	$n+1$
S	$x \mid B$	n	SHIFT	$S \mid x$	B	n
$S \mid (\mathrm{X} \mid \tau_1 \mid \ldots \mid \tau_\ell$	B	n	REDUCE	$S \mid (\mathrm{X}\ \tau_1\ \ldots\ \tau_\ell)$	B	$n-1$

Input: *The hungry cat meows .*

	Stack	**Buffer**	**Action**
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

[Dyer et al. 2016]

Generation as well

Stack$_t$	**Terms**$_t$	**Open NTs**$_t$	**Action**	**Stack**$_{t+1}$	**Terms**$_{t+1}$	**Open NTs**$_{t+1}$
S	T	n	NT(X)	$S \mid (\mathrm{X}$	T	$n+1$
S	T	n	GEN(x)	$S \mid x$	$T \mid x$	n
$S \mid (\mathrm{X} \mid \tau_1 \mid \ldots \mid \tau_\ell$	T	n	REDUCE	$S \mid (\mathrm{X}\ \tau_1\ \ldots\ \tau_\ell)$	T	$n-1$

Figure 3: Generator transitions. Symbols defined as in Fig. 1 with the addition of T representing the history of generated terminals.

	Stack	**Terminals**	**Action**
0			NT(S)
1	(S		NT(NP)
2	(S | (NP		GEN(*The*)
3	(S | (NP | *The*	*The*	GEN(*hungry*)
4	(S | (NP | *The* | *hungry*	*The* | *hungry*	GEN(*cat*)
5	(S | (NP | *The* | *hungry* | *cat*	*The* | *hungry* | *cat*	REDUCE
6	(S | (NP *The hungry cat*)	*The* | *hungry* | *cat*	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*The* | *hungry* | *cat*	GEN(*meows*)
8	(S | (NP *The hungry cat*) | (VP *meows*	*The* | *hungry* | *cat* | *meows*	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	*The* | *hungry* | *cat* | *meows*	GEN(.)
10	(S | (NP *The hungry cat*) | (VP *meows*) | .	*The* | *hungry* | *cat* | *meows* | .	REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)	*The* | *hungry* | *cat* | *meows* | .	

Figure 4: Joint generation of a parse tree and sentence.

[Dyer et al. 2016]

Shift-reduce parsing

- Models for shift-reduce
  - Any (P)CFG can be parsed in this manner *[Stolcke 1995]*
- History based models: select next action given information about *current state and history*
  - Infinite history, no future (contrast to PCFG assumptions!)
  - ***a***: action
  - ***u***: features/embedding of current state
- Generative form (discriminative also possible):

$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} p(a_t \mid \boldsymbol{a}_{<t})$$

$$= \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} \frac{\exp \mathbf{r}_{a_t}^{\top} \mathbf{u}_t + b_{a_t}}{\sum_{a' \in \mathcal{A}_G(T_t, S_t, n_t)} \exp \mathbf{r}_{a'}^{\top} \mathbf{u}_t + b_{a'}}$$

[Dyer et al. 2016]

- Vector representation of current stack/buffer state
    - Explicit log-linear features over the current stack, buffer etc. *[Ratnaparkhi 1998, Zhang+Clark 2011]*
    - Neural network representation of current state *[e.g. Henderson 2004, Dyer et al. 2016, Bowman et al. 2016]*
- Training: extract oracle decisions paths from labeled data
    - Generative model: use importance sampling to calculate feature expectations

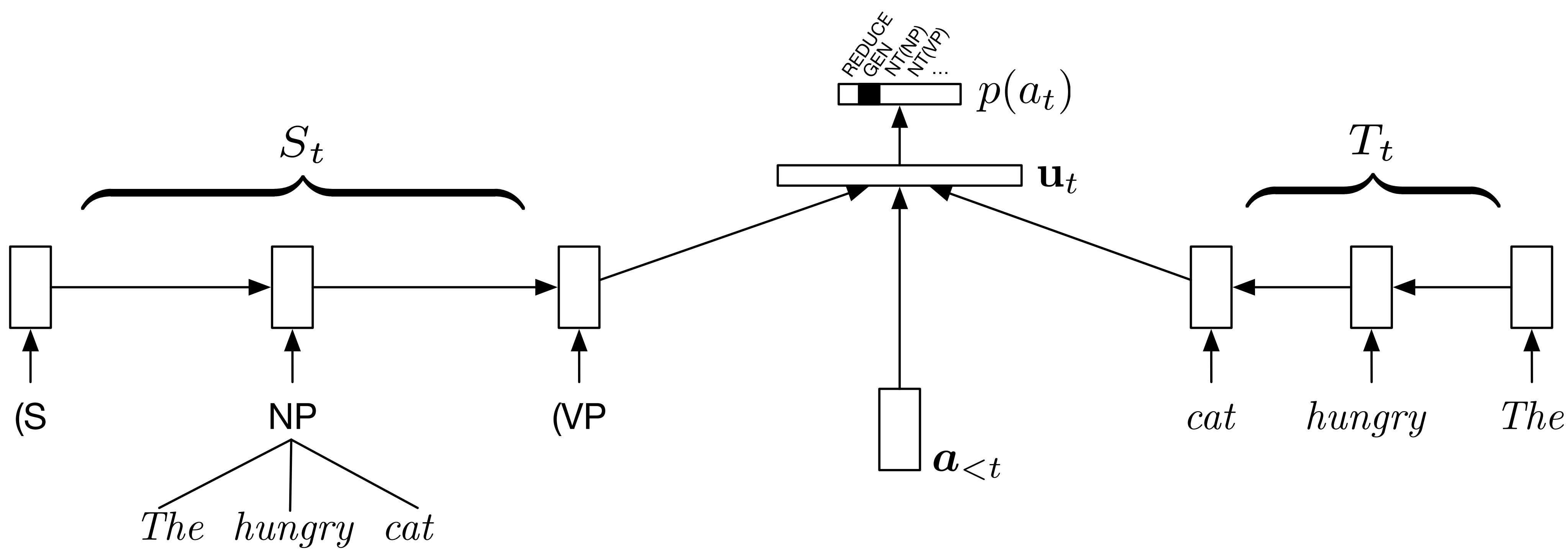


Figure 5: Neural architecture for defining a distribution over a_t given representations of the stack (S_t), output buffer (T_t) and history of actions ($\boldsymbol{a}_{<t}$). Details of the composition architecture of the NP, the action history LSTM, and the other elements of the stack are not shown. This architecture corresponds to the generator state at line 7 of Figure 4.

[Dyer et al. 2016]

Results

Recurs. NN

Latent state-split PCFG (EM training)

Self-training

Model	**type**	$\boldsymbol{F_1}$
Vinyals et al. (2015)* – WSJ only	D	88.3
Henderson (2004)	D	89.4
Socher et al. (2013a)	D	90.4
Zhu et al. (2013)	D	90.4
Petrov and Klein (2007)	G	90.1
Bod (2003)	G	90.7
Shindo et al. (2012) – single	G	91.1
Shindo et al. (2012) – ensemble	G	92.4
Zhu et al. (2013)	S	91.3
McClosky et al. (2006)	S	92.1
Vinyals et al. (2015)	S	92.1
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	D	91.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	G	**93.3**

Chinese parsing results. Chinese parsing results were obtained with the same methodology as in English and show the same pattern (Table 6).

Model	type	$\mathbf{F_1}$
Zhu et al. (2013)	D	82.6
Wang et al. (2015)	D	83.2
Huang and Harper (2009)	D	84.2
Charniak (2000)	G	80.8
Bikel (2004)	G	80.6
Petrov and Klein (2007)	G	83.3
Zhu et al. (2013)	S	85.6
Wang and Xue (2014)	S	86.3
Wang et al. (2015)	S	86.6
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ - buggy	D	80.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ - buggy	G	82.7
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	D	84.6
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	G	**86.9**

Table 6: Parsing results on CTB 5.1 including results with the buggy composition function implementation (indicated by †) and with the correct implementation.

Look out for bugs.

Due to an implentation bug in the RNNG's recursive composition function, the results reported in Dyer et al. (2016) did not correspond to the model as it was presented. This corri-

- Even the experts have bugs!
- Many, MANY unreported bugs in results are likely out there
- Replication and reimplementation are often good ways of finding them

Model	**type**	$\boldsymbol{F_1}$
Vinyals et al. (2015)* – WSJ only	D	88.3
Henderson (2004)	D	89.4
Socher et al. (2013a)	D	90.4
Zhu et al. (2013)	D	90.4
Petrov and Klein (2007)	G	90.1
Bod (2003)	G	90.7
Shindo et al. (2012) – single	G	91.1
Shindo et al. (2012) – ensemble	G	92.4
Zhu et al. (2013)	S	91.3
McClosky et al. (2006)	S	92.1
Vinyals et al. (2015)	S	92.1
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ – buggy	D	89.8
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ – buggy	G	92.4
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	D	91.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	G	**93.3**

Table 5: Parsing results with fixed composition function on PTB §23 (D=discriminative, G=generative, S=semisupervised). * indicates the (Vinyals et al., 2015) model trained only on the WSJ corpus without ensembling. † indicates RNNG models with the buggy composition function implementation.

- stopped here 3/9

Treebanks

- Know what you're getting!
  - Formalism?
  - Annotation assumptions?

- Penn Treebank (constituents, English)
  - http://www.cis.upenn.edu/~treebank/home.html
  - Recent revisions in Ontonotes
- Chinese Treebank ... many others
- Universal Dependencies
  - http://universaldependencies.org/

- CCG Treebank
- Prague Treebank (syn+sem)
- ...many others...