Transforming Projective Bilexical Dependency Grammars into efficiently-parsable CFGs with Unfold-Fold

Mark Johnson
Microsoft Research   Brown University

ACL 2007

Motivation and summary

- What's the relationship between CKY parsing and the Eisner/Satta $O(n^3)$ PBDG parsing algorithm? (c.f., McAllester 1999)
  - *split-head encoding*, collecting left and right dependents separately
  - *unfold-fold transform* reorganizes grammar for efficient CKY parsing
- Approach generalizes to 2nd-order dependencies
  - predict argument given governor and sibling (McDonald 2006)
  - predict argument given governor and governor's governor
- *In principle* can use any CFG parsing or estimation algorithm for PBDGs
  - transformed grammars typically too large to enumerate
  - my CKY implementations transform grammar on the fly

Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

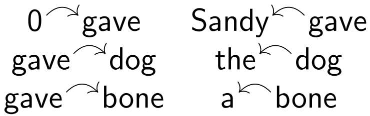
$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

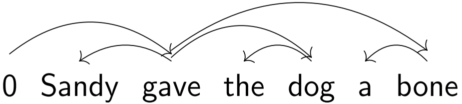
Conclusion

Projective Bilexical Dependency Grammars

- Projective Bilexical Dependency Grammar (PBDG)



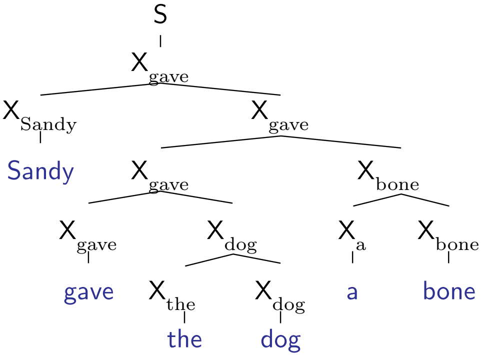
- A dependency parse generated by the PBDG



- Weights can be attached to dependencies (and preserved in CFG transforms)

A naive encoding of PBDGs as CFGs

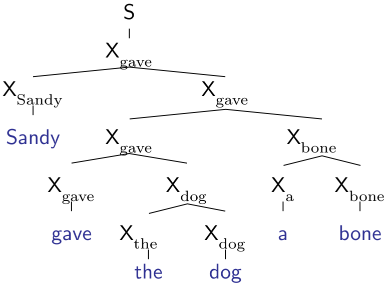
$$
\begin{aligned}
\mathsf{S} &\to \mathsf{X}_u && \text{where } 0 \curvearrowright u \\
\mathsf{X}_u &\to u && \\
\mathsf{X}_u &\to \mathsf{X}_v\ \mathsf{X}_u && \text{where } v \curvearrowleft u \\
\mathsf{X}_u &\to \mathsf{X}_u\ \mathsf{X}_v && \text{where } u \curvearrowright v
\end{aligned}
$$

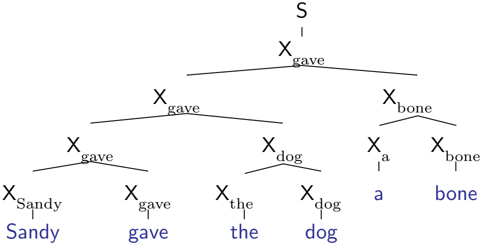


Spurious ambiguity in naive encoding

- Naive encoding allows dependencies on different sides of head to be freely reordered

⇒ Spurious ambiguity in CFG parses (not present in PBDG parses)



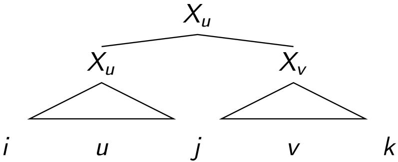


Parsing naive CFG encoding takes $O(n^5)$ time

- A production schema such as

$$X_u \rightarrow X_u X_v$$

has 5 variables, and so can match input in $O(n^5)$ different ways



Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion

Simple split-head encoding

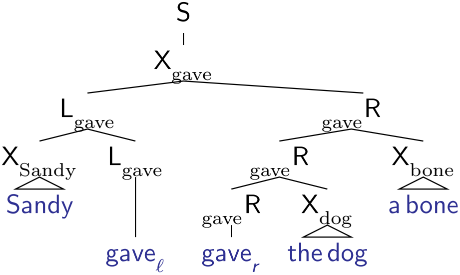
- Replace input word u with a *left variant* u_ℓ and a *right variant* u_r (can be avoided in practice with fancy book-keeping)

Sandy gave the dog a bone

$$\Downarrow$$

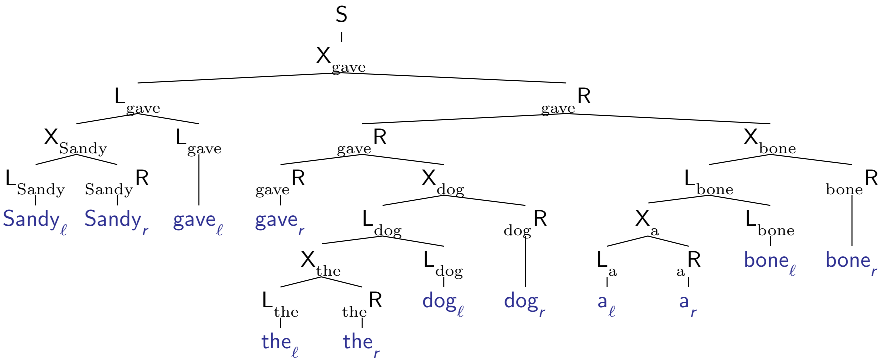
$\text{Sandy}_\ell\ \text{Sandy}_r\ \text{gave}_\ell\ \text{gave}_r\ \text{the}_\ell\ \text{the}_r\ \text{dog}_\ell\ \text{dog}_r\ \text{a}_\ell\ \text{a}_r\ \text{bone}_\ell\ \text{bone}_r$

- PCFG separately collects left dependencies and right dependencies



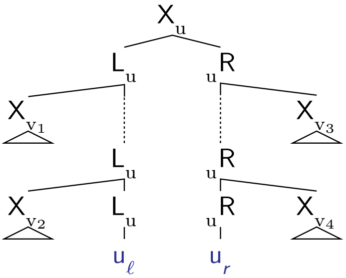
$$
\begin{array}{rcll}
S & \to & X_u & \text{where } 0 \curvearrowright u \\
X_u & \to & L_u \ {}_uR & \text{where } u \in \Sigma \\
L_u & \to & u_l & \\
L_u & \to & X_v \ L_u & \text{where } v \curvearrowleft u \\
{}_uR & \to & u_r & \\
{}_uR & \to & {}_uR \ X_v & \text{where } u \curvearrowright v
\end{array}
$$

Simple split-head CFG parse



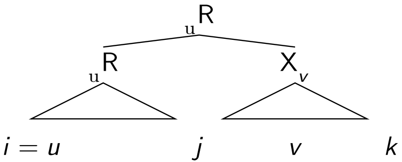
L_u and ${}_uR$ heads are phrase-peripheral $\Rightarrow O(n^4)$

- Heads of L_u and ${}_uR$ are always at right (left) edge



$$
\begin{array}{lcll}
S & \rightarrow & X_u & \text{where } 0 \curvearrowright u \\
X_u & \rightarrow & L_u \ {}_uR & \text{where } u \in \Sigma \\
L_u & \rightarrow & u_l & \\
L_u & \rightarrow & X_v \ L_u & \text{where } v \curvearrowleft u \\
{}_uR & \rightarrow & u_r & \\
{}_uR & \rightarrow & {}_uR \ X_v & \text{where } u \curvearrowright v
\end{array}
$$

- $X_u \rightarrow L_u \ {}_uR$ take $O(n^3)$
- ${}_uR \rightarrow {}_uR \ X_v$ take $O(n^4)$



Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion

The Unfold-Fold transform

- Unfold-fold originally proposed for transforming recursive programs; used here to transform CFGs into new CFGs
- *Unfolding* a nonterminal replaces it with its expansion

$$\begin{array}{l} A \rightarrow \alpha\, B\, \gamma \\ B \rightarrow \beta_1 \\ B \rightarrow \beta_2 \\ \quad\ldots \end{array} \quad \Rightarrow \quad \begin{array}{l} A \rightarrow \alpha\, \beta_1\, \gamma \\ A \rightarrow \alpha\, \beta_2\, \gamma \\ B \rightarrow \beta_1 \\ B \rightarrow \beta_2 \\ \quad\ldots \end{array}$$

- *Folding* is the inverse of unfolding (replace RHS with nonterminal)

$$\begin{array}{l} A \rightarrow \alpha\, \beta\, \gamma \\ B \rightarrow \beta \\ \quad\ldots \end{array} \quad \Rightarrow \quad \begin{array}{l} A \rightarrow \alpha\, B\, \gamma \\ B \rightarrow \beta \\ \quad\ldots \end{array}$$

- Transformed grammar generates same language (Sato 1992)

Unfold-fold converts $O(n^4)$ to $O(n^3)$ grammar

- Unfold X_v responsible for $O(n^4)$ parse time

$$\begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{X}_v \ \mathrm{L}_u \\ \mathrm{X}_v & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{R} \end{array} \quad \Rightarrow \quad \begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{R} \ \mathrm{L}_u \end{array}$$

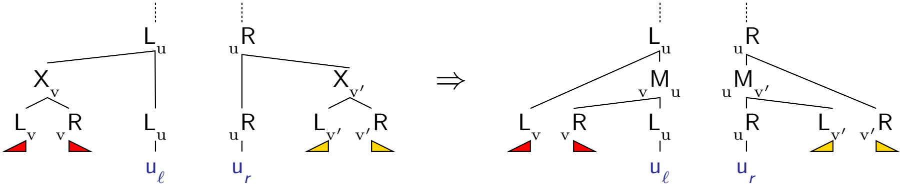
- Introduce new non-terminals ${}_x\mathrm{M}_y$ (doesn't change language)

$${}_x\mathrm{M}_y \rightarrow {}_x\mathrm{R} \ \mathrm{L}_y$$

- Fold two children of L_u into ${}_x\mathrm{M}_y$

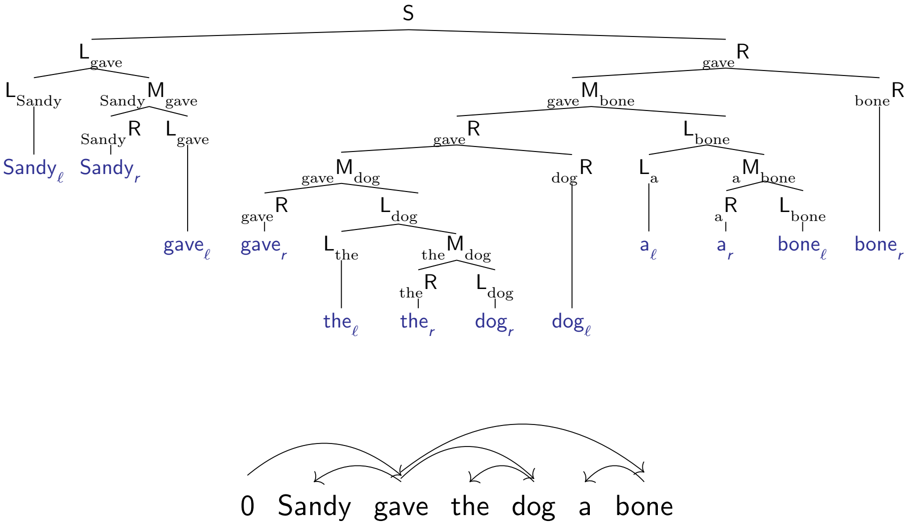
$$\begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{R} \ \mathrm{L}_u \\ {}_x\mathrm{M}_y & \rightarrow & {}_x\mathrm{R} \ \mathrm{L}_y \end{array} \quad \Rightarrow \quad \begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{M}_u \\ {}_x\mathrm{M}_y & \rightarrow & {}_x\mathrm{R} \ \mathrm{L}_y \end{array}$$

Transformed grammar collects left and right dependencies separately



- X_v constituents (which cause $O(n^4)$ parse time) no longer used
- Head annotations now all phrase peripheral ⇒ $O(n^3)$ parse time
- Dependencies can be recovered from parse tree
- Basically same as Eisner and Satta $O(n^3)$ algorithm
  - explains why Inside-Outside sanity check fails for Eisner/Satta
  - two copies of each terminal ⇒ each terminals' Outside probability is *double* the Inside sentence probability

Parse using $O(n^3)$ transformed split-head grammar



Parsing time of CFG encodings of same PBDG

CFG schemata	sentences parsed / second
Naive $O(n^5)$ CFG	45.4
$O(n^4)$ simple split-head CFG	406.2
$O(n^3)$ transformed split-head CFG	3580.0

- Weighted PBDG; all pairs of heads have some dependency weight
- Dependency weights precomputed before parsing begins
- Timing results on a 3.6GHz Pentium 4 machine parsing section 24 of the PTB
- CKY parsers with grammars hard-coded in C (no rule lookup)
- Dependency accuracy of Viterbi parses = 0.8918 for all grammars
- *Feature extraction is much slower than even naive CFG*

Outline

Projective Bilexical Dependency Grammars

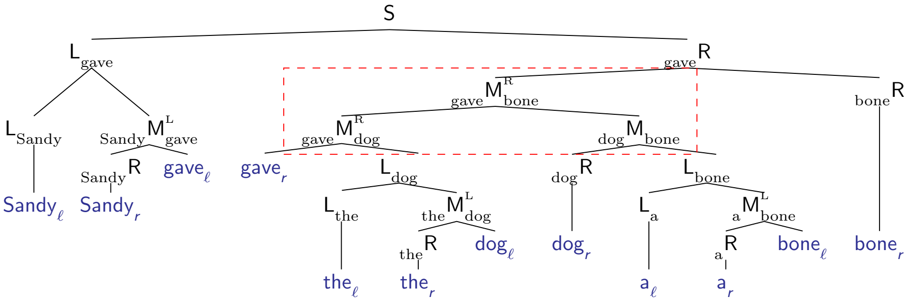
Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

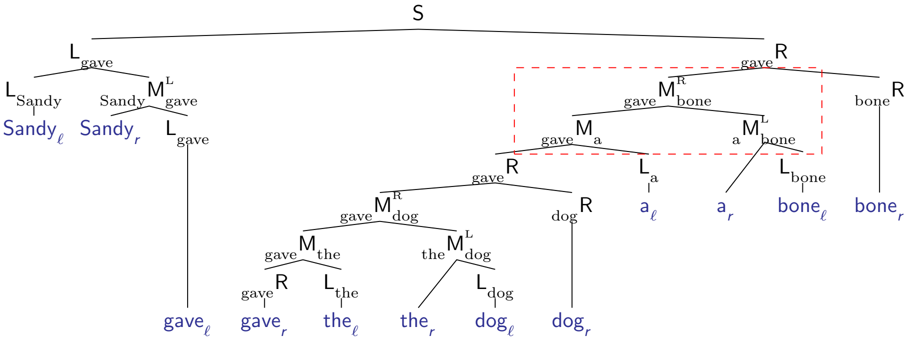
Conclusion

Predict argument based on governor and sibling



- Very similar to second-order algorithm given by McDonald (2006)

Predict argument based on governor and governor's governor



- Because left and right dependencies are assembled separately, only captures 2nd-order dependencies where one dependency is leftward and other is rightward

Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion

Conclusion and future work

- Presented a reduction from PBDGs to $O(n^3)$ parsable CFGs
  - split-head CFG representation of PBDGs
  - Unfold-fold transform
- CKY algorithm on resulting CFG simulates Eisner/Satta algorithm on original PBDG
- Makes CFG techniques applicable to PBDGs
  - max marginal parsing (Goodman 1996)
    and other CFG parsing and estimation algorithms
- Can capture different dependencies, yielding different PDG models
  - 2nd-order "horizontal" dependencies (McDonald 2006)
  - what other combinations of dependencies can we capture?
    (if we permit $O(n^4)$ parse time?)
  - do any of these improve parsing accuracy?