Incremental Parsing with TAG

Miriam Käshammer

Course: Incremental Processing
Department of Computational Linguistics, Universität des Saarlandes

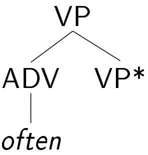
May 28, 2011

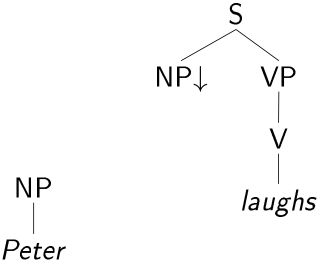
Tree Adjoining Grammar (TAG)

- tree-rewriting formalism
- a set of (elementary) trees with two operations

initial trees

auxiliary trees



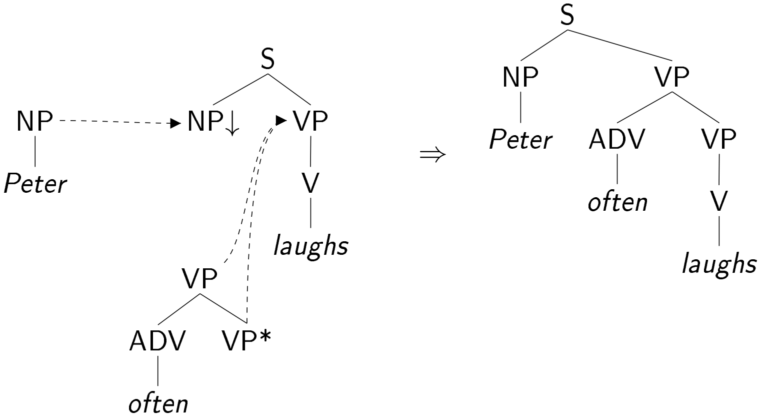


Tree Adjoining Grammar

Operations

Substitution Replacing a leaf with an initial tree

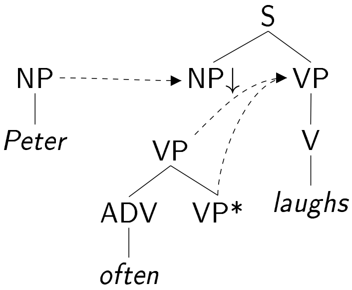
Adjunction Replacing an internal node with an auxiliary tree



Tree Adjoining Grammar

Some linguistic principles:

- Lexicalization: Each elementary tree has at least one non-empty lexical item, its anchor. (LTAG)
- Predicate argument co-occurence: Elementary trees of predicates contain slots for the arguments they subcategorize for.



Why is TAG interesting?

- **Mildly context-sensitive** formalism
  1. It generates (at least) all context-free languages.
  2. It captures a limited amount of cross-serial dependencies, e.g. the copy language $\{ww|w \in \{a,b\}^*\}$.
  3. It can be parsed in polynomial time. $(O(n^6))$
  4. It has constant growth property.

  $\Rightarrow$ appropriate to describe natural languages

- Important characteristics
  1. **Extended domain of locality** - elementary trees can be arbitrarily large.
  2. **Factoring of recursion** - adjunction operations allows to put recursive structures into separate elementary trees.

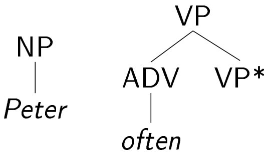
Why incremental parsing?

- Psycholinguistic evidence
  - Humans build up semantic representation before reaching the end of the sentence.
  - Interpretation is based on fast, left-to-write construction of syntactic relations.
- Boost in speed

Incremental TAG parsing

- [Sturt and Lombardo, 2005] argue that models of human parsing incorporate an operation similar to adjunction in TAG.
- Traditional LTAG does not allow full connectedness.

Peter often ...



Where is incrementality encoded?

Components of a parser:

- a (competence) grammar
- a parsing strategy
- a memory organizing strategy
- an oracle

Where is incrementality encoded?

Components of a parser:

- a (competence) grammar
- ⇒ a parsing strategy
- a memory organizing strategy
- an oracle

Where is incrementality encoded?

Components of a parser:

- ⇒ a (competence) grammar
- a parsing strategy
- a memory organizing strategy
- an oracle

Approaches

LTAG-spinal - "Incremental" parser

Incremental LTAG Parsing
L. Shen and A. K. Joshi (2005)

DVTAG - Strictly incremental parser

Dynamic TAG and lexical dependencies
A. Mazzei, V. Lombardo and P. Sturt (2007)

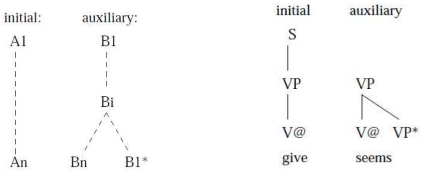
Outline

Outline

LTAG-spinal

Variant of LTAG

- initial tree only contains its *spine*
- auxiliary tree only contains its *spine* and a foot node directly linked to the spine



Definition: Spine

The *spine* of an elementary tree is the path from the root node to the anchor of the tree.

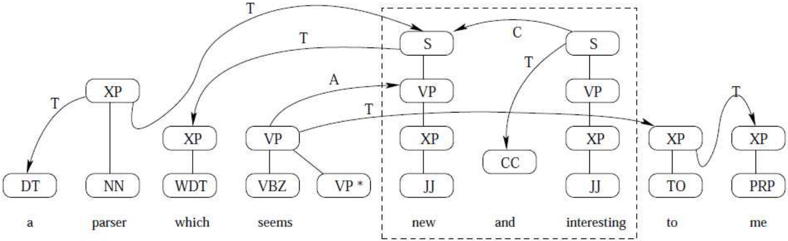
LTAG-spinal formalism

Operations

Adjunction (A) Same as in LTAG.

Attachment (T) Attachment of an initial tree α to a node n of another tree α': add the root of α to n as a new child.

Conjunction (C) Special operation to build coordination structures.



LTAG-spinal formalism

Relation to LTAG

- LTAG-spinal is more powerful than CFG. [Shen and Joshi, 2005a]
- LTAG-spinal with attachment constraints is weakly equivalent to traditional LTAG. [Shen et al., 2007]

LTAG-spinal formalism

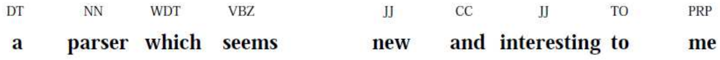
Relation to LTAG

- LTAG-spinal is more powerful than CFG. [Shen and Joshi, 2005a]
- LTAG-spinal with attachment constraints is weakly equivalent to traditional LTAG. [Shen et al., 2007]
- LTAG-spinal trees generalize over predicates with different subcategorization frames.

The Parsing Algorithm

- Four types of **parser operations**:
  - Attachment, adjunction, conjunction
  - Generation: generate a possible spine for a given word according to the context and the lexicon (Supertagging)
- Variant of the shift-reduce algorithm, using a **stack of disconnected treelets** to represent the left context
  - **Shift**: Read a word, generate a list of possible elementary trees for this word. For each elementary tree, push it into the stack.
  - **Reduce**: Pop the top two treelets from the stack, combine them by attachment, adjunction or conjunction and push the combined tree into the stack.
- Beam-search to prune the search space

Example



G: generate T: attach A: adjoin C: conjoin

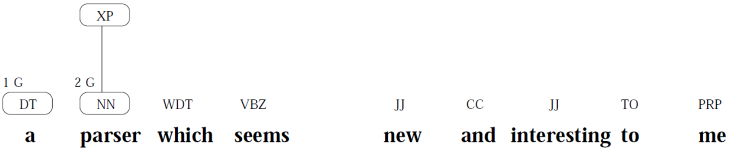
Graph taken from `http://libinshen.net/Documents/ijc04_slides.ps`

Example

1 G								
$\boxed{\text{DT}}$	NN	WDT	VBZ	JJ	CC	JJ	TO	PRP
a	**parser**	**which**	**seems**	**new**	**and**	**interesting**	**to**	**me**

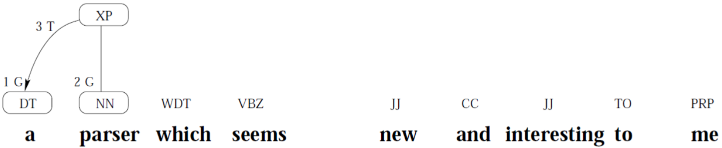
G: generate T: attach A: adjoin C: conjoin

Example



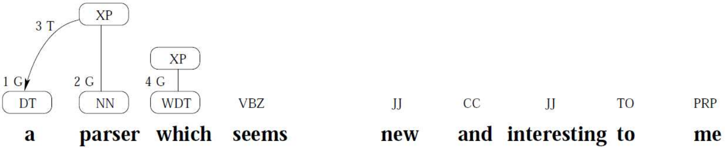
G: generate T: attach A: adjoin C: conjoin

Example



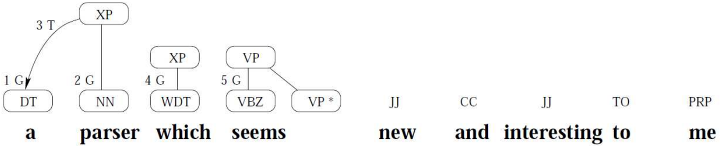
G: generate T: attach A: adjoin C: conjoin

Example



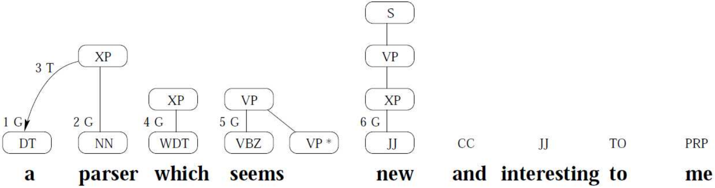
G: generate T: attach A: adjoin C: conjoin

Example



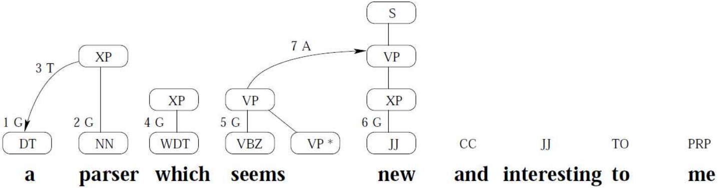
G: generate T: attach A: adjoin C: conjoin

Example



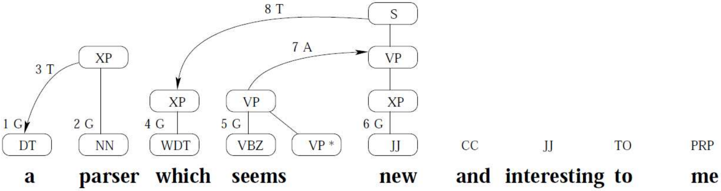
G: generate T: attach A: adjoin C: conjoin

Example



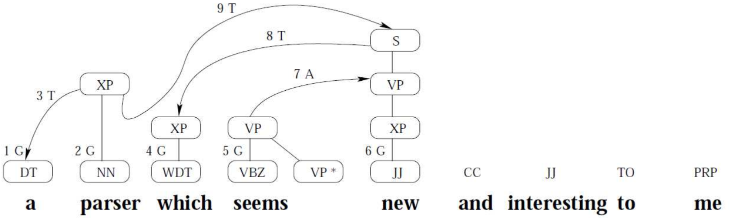
G: generate T: attach A: adjoin C: conjoin

Example



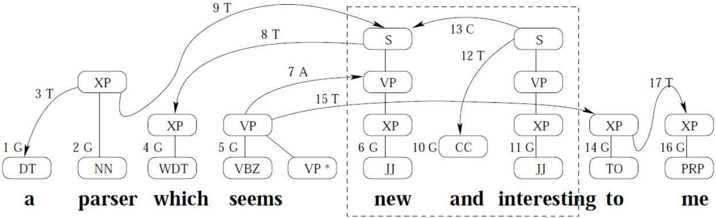
G: generate T: attach A: adjoin C: conjoin

Example



G: generate T: attach A: adjoin C: conjoin

Example



G: generate T: attach A: adjoin C: conjoin

Graph taken from `http://libinshen.net/Documents/ijc04_slides.ps`

Flex Model vs. Eager Model

Pseudo-ambiguity in the shift-reduce derivation:
A adjoins to B, B adjoins to C

- $((A \rightarrow B) \rightarrow C)$
- $(A \rightarrow (B \rightarrow C))$

Flex Model vs. Eager Model

Pseudo-ambiguity in the shift-reduce derivation:
A adjoins to B, B adjoins to C

- $((A \rightarrow B) \rightarrow C)$
- $(A \rightarrow (B \rightarrow C))$

Flex Model

- Both derivations are allowed.

Eager Model

- Only $((A \rightarrow B) \rightarrow C)$ is allowed.

Features and Learning

Features extracted from gold-standard parses have the following format:

(`operation`, `main_spine`, `child_spine`, `spine_node`, `context`)

- `spine_node`: node on the `main_spine` onto which the `child_spine` is attached/adjoined/conjoined
- `context`: dependent on the type of operation; includes amongst others the (0,2) window in the sentence

Weights for the features are learned using a perceptron-like algorithm as proposed in [Collins, 2002].

Evaluation

- LTAG-spinal treebank (see [Shen and Joshi, 2005b])
- Training, development and test data from WSJ
- Syntactic dependency for evaluation against PTB

model	beam	sen/sec	f-score %
Flex	10	0.37	89.3
Eager	10	0.79	88.7

With an extension (Combined Parses) and beam=100: 94.2%

What we have seen so far

Parser for LTAG-spinal

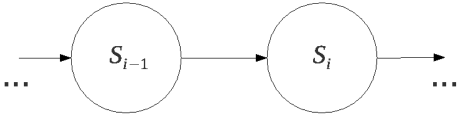
- Incremental?
  - Input is processed incrementally, but only partially
  - Structure is not fully connected (usage of a stack)
  - Look-ahead of 2 words
- Implemented: efficient statistical parsing
- Generative power of grammar is stronger than CFG
- Motivation?

Outline

Dynamics

Dynamic Grammar

- The syntactic analysis is viewed as a dynamic process
  $\Rightarrow$ A sequence of **transitions** between adjacent syntactic **states** S_{i-1} and S_i.
- A syntactic state contains all the syntactic information about the fragment already processed.



Dynamics and Incrementality

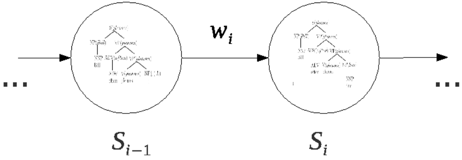
Incremental processing:

- Each input word w_i defines a transition from S_{i-1} (left-context) to S_i.
- States as partial syntactic structures

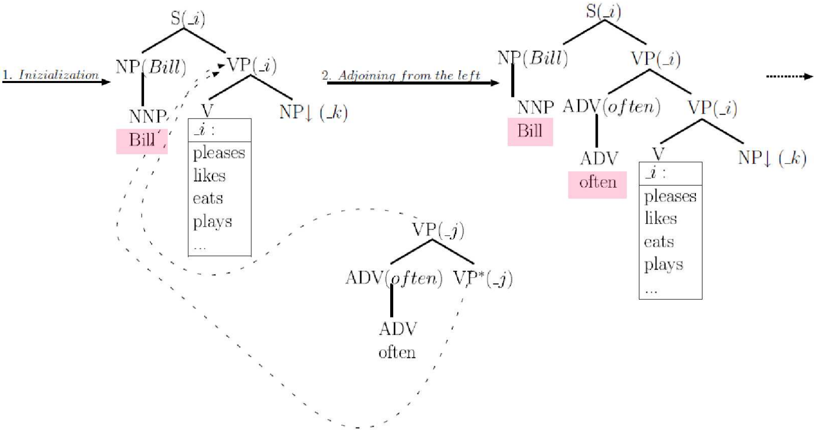
Strong connectivity:

- Impose that transitions do not produce disconnected trees

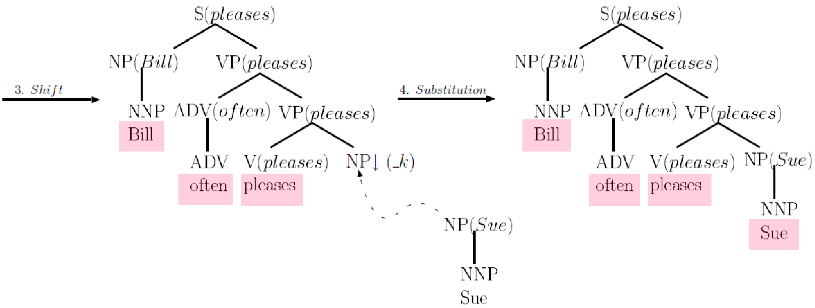
Parsing strategy is part of the grammar
$\Rightarrow$ Incrementality-in-compentence



Dynamics in TAG - Intuition (I)



Dynamics in TAG - Intuition (II)



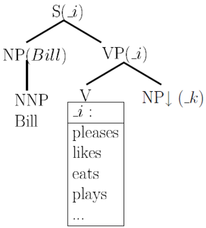
At step i, the elementary tree anchored in w_i is combined with the partial structure spanning the words $w_1 \ldots w_{i-1}$.

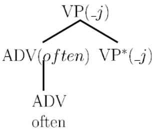
$\Rightarrow$ Updated left-context spanning the words $w_1 \ldots w_i$

DVTAG

Dynamic Version of TAG

Elementary trees similar to LTAG trees, BUT



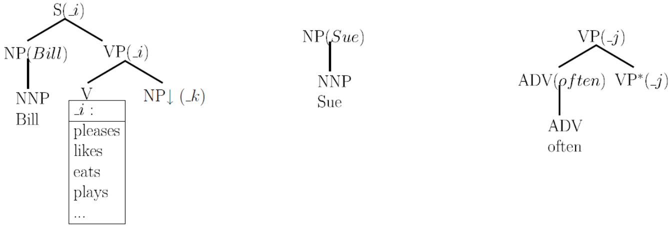


DVTAG

Dynamic Version of TAG

Elementary trees similar to LTAG trees, BUT

- Lexical items (but not the left-anchor) can be **underspecified**. The preterminal category is paired with a finite list of values. $\Rightarrow$ **predicted nodes** to fulfill full connectivity

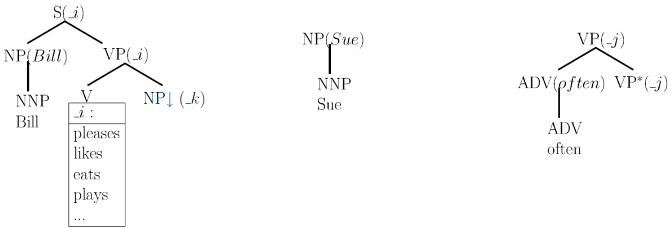


DVTAG

Dynamic Version of TAG

Elementary trees similar to LTAG trees, BUT

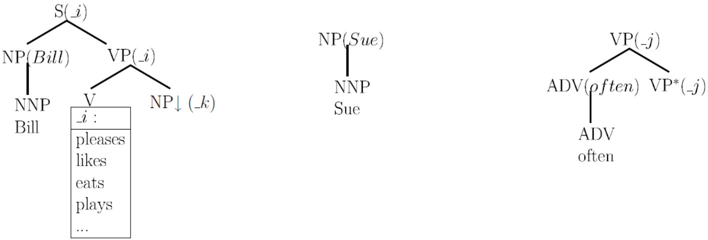
- Lexical items (but not the left-anchor) can be **underspecified**. The preterminal category is paired with a finite list of values. $\Rightarrow$ **predicted nodes** to fulfill full connectivity
- Distinction between left/right auxiliary trees



DVTAG

Head feature

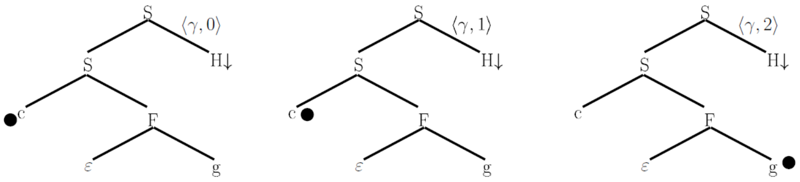
- Feature that indicates the lexical head of each node in the elementary tree
- Needed to compute a *dependency tree*



Some DVTAG terminology

Dotted tree

A pair $\langle \gamma, i \rangle$ where γ is a tree and i is an integer such that $i \in 0 \ldots |YIELD(\gamma)|$.
Moreover, all the symbols on the yield that are on the left of the dot are terminal symbols.



Fringe of $\langle \gamma, i \rangle$
Set of nodes that are accessible for operations

DVTAG Operations

Combination of a left-context $\langle \Lambda, i \rangle$ with an (unanalysed) elementary tree $\langle \gamma, 0 \rangle$

- 2 substitution operations
- 4 adjoining operations
- a shift operation

"Normal" operations

- Substitution $Sub^{\rightarrow}$
- Adjoining from the left $\nabla_L^{\rightarrow}$
- Adjoining from the right $\nabla_R^{\rightarrow}$
- Shift Shi

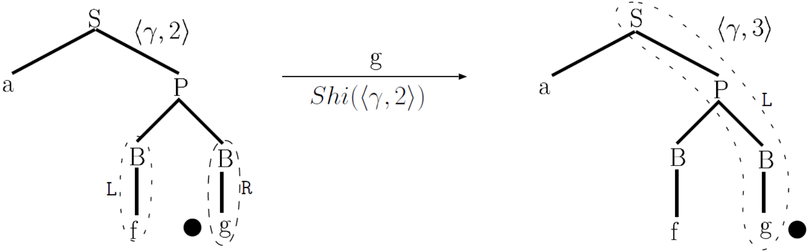
Inverse operations

- Inv. substitution $Sub^{\leftarrow}$
- Inv. adj. from the left $\nabla_L^{\leftarrow}$
- Inv. adj. from the right $\nabla_R^{\leftarrow}$

DVTAG Operations

Shift

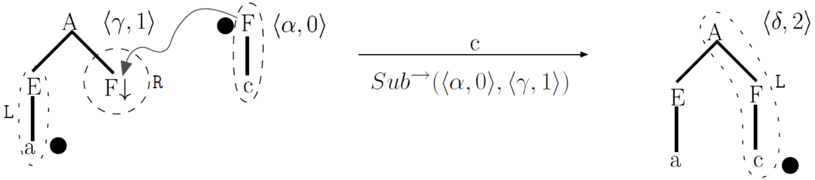
$Shi(\langle\gamma, i\rangle)$ takes as input a single dotted tree $\langle\gamma, i\rangle$ and returns the dotted tree $\langle\gamma, i+1\rangle$. It can be applied only if a terminal symbol belongs to the fringe of $\langle\gamma, i\rangle$.



DVTAG Operations

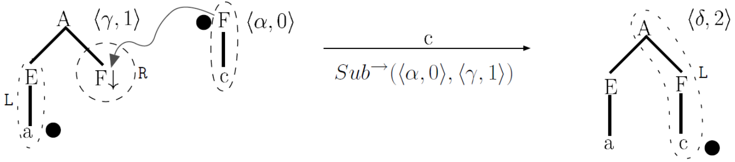
Substitution

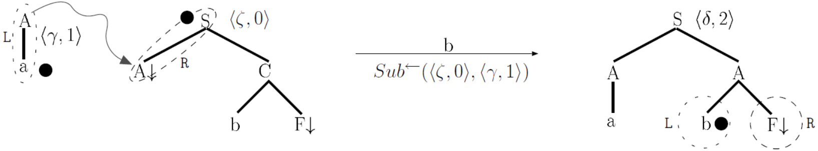
$Sub^{\rightarrow}(\langle\alpha,0\rangle,\langle\gamma,i\rangle)$:
If there is a substitution node N in the fringe of $\langle\gamma,i\rangle$ such that $label(N)=label(root(\alpha))$, the operation returns a new dotted tree $\langle\delta,i+1\rangle$ such that δ is obtained by grafting α into N.



DVTAG Operations

Substitution and inverse Substitution

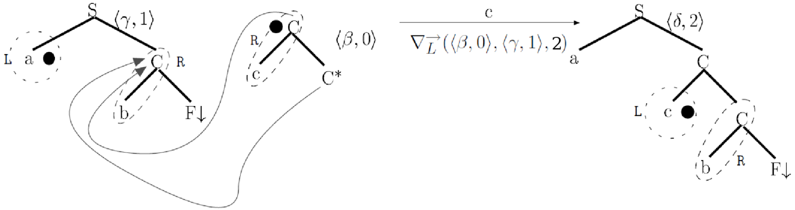




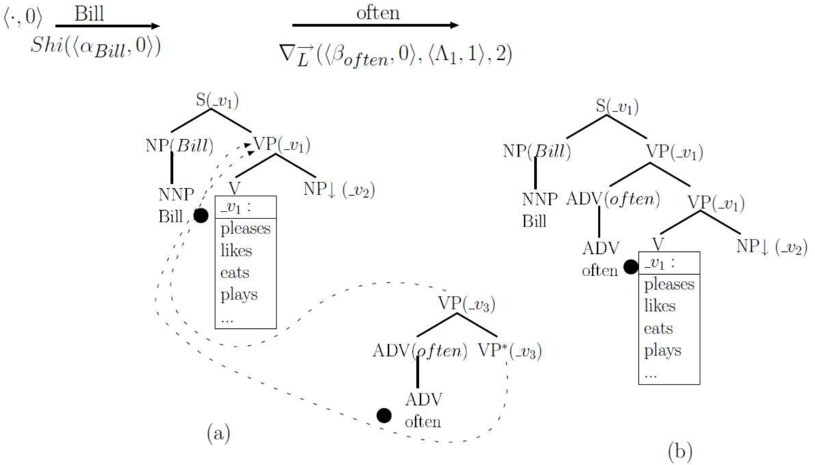
DVTAG Operations

Adjoining from the left

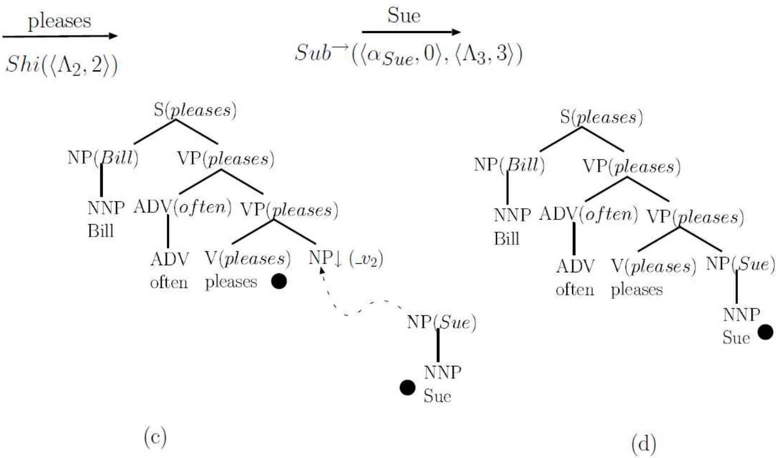
$\nabla_L^{\rightarrow}(\langle\beta,0\rangle,\langle\gamma,i\rangle,add)$ where β is a left auxiliary tree:
If there is a non-terminal node N at position add in the fringe of $\langle\gamma,i\rangle$ such that $label(N) = label(root(\beta))$, the operation returns a new dotted tree $\langle\delta,i+1\rangle$ such that δ is obtained by grafting β into N.



Example derivation



Example derivation



A wide-coverage DVTAG

Ways to build a grammar:

1. Manually write it (XTAG, FTAG)
2. Automatically extract it from treebanks

Anticipated problem: Size of grammar because of predicted nodes

Grammar seizes in comparison

	# of tree templates
XTAG	1,200
DVTAG from XTAG	6,000,000

Grammar seizes in comparison

	# of tree templates
XTAG	1,200
DVTAG from XTAG	6,000,000
DVTAG from PTB	12,000

Grammar seizes in comparison

	# of tree templates
XTAG	1,200
DVTAG from XTAG	6,000,000
DVTAG from PTB	12,000
LTAG-spinal	1,200

Numbers are taken from [Mazzei, 2005] and [Shen et al., 2007], and rounded.

Outline

Conclusion

LTAG-spinal

- Parsing strategy specifies the "incremental" nature.
- In fact, not very incremental (stack, look-ahead)
- Efficient, implemented parser available

DVTAG

- The (competence) grammar determines the parsing strategy
- Natively fulfills a strict version of incrementality
- Resembles left-corner strategy ($\Rightarrow$ center embeddings)
- Grammars grow very large in size

References I

Collins, M. (2002).
Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms.
EMNLP.

Mazzei, A. (2005).
Formal and empirical issues of applying dynamics to Tree Adjoining Grammars.
PhD thesis, Dipartimento di Informatica, Universita degli studi di Torino.

Mazzei, A., Lombardo, V., and Sturt, P. (2007).
Dynamic TAG and lexical dependencies.
Research on Language and Computation.

Shen, L. and Joshi, A. K. (2005a).
Building an LTAG Treebank.
Technical Report MS-CIS-05-15, CIS, University of Pennsylvania, Philadelphia, PA.

References II

Shen, L. and Joshi, A. K. (2005b).
Incremental LTAG Parsing.
Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing.

Shen, L., Joshi, A. K., and Champollion, L. (2007).
LTAG-spinal and the Treebank.
Language Resources and Evaluation.

Sturt, P. and Lombardo, V. (2005).
Processing coordinated structures: Incrementality and connectedness.
Cognitive Science, 29.

Discussion

- Why would we want incrementality in competence?
- For NLP applications the "incremental" parser might be enough.
- Can psycholinguistic findings/memory profiles be explained with these grammers?