On Late Adjunction in Minimalist Grammars

Greg Kobele*

Computation Institute
and Department of Linguistics
University of Chicago

MCFG+ 2010

*Based on joint work with Jens Michaelis

The Big Picture

The Question:

What happens when late adjunction is added to Minimalist Grammars?

The (Short) Answer:

Not sure, but:

- still semi-linear
- describable by 3rd order ACGs
- tree relation definable in MSO, with some extra information

Outline

The idea behind ‘late’ operations

The problem:

Sometimes we have semantic ambiguity without any obvious corresponding difference in the derivation tree.

The idea:

Enrich the derivation tree with information about order.
(not only did I apply this rule, but I applied this rule *before that one*)

Frameworks which have 'late' operations

- Montague Grammar

  (Montague)
- 'Cosubstitution' in TAGs

  (Barker)
- Minimalist Grammars

  (Gärtner & Michaelis;Kobele)

Outline

Syntax

Rules of quantification

S14. If $\alpha \in P_T$ and $\phi \in P_t$, then $F_{10,n}(\alpha, \phi) \in P_t$, where either (i) α does not have the form $\mathbf{he}_k$, and $F_{10,n}(\alpha, \phi)$ comes from ϕ by replacing the first occurrence of $\mathbf{he}_n$ or $\mathbf{him}_n$ by α and all other occurrences of $\mathbf{he}_n$ or $\mathbf{him}_n$ by $\left\{\begin{matrix}\mathbf{he}\\ \mathbf{she}\\ \mathbf{it}\end{matrix}\right\}$ or $\left\{\begin{matrix}\mathbf{him}\\ \mathbf{her}\\ \mathbf{it}\end{matrix}\right\}$ respectively, according as the gender of the first B_{CN} or B_T in α is $\left\{\begin{matrix}\text{masc.}\\ \text{fem.}\\ \text{neuter}\end{matrix}\right\}$, or

(ii) $\alpha = \mathbf{he}_k$, and $F_{10,n}(\alpha, \phi)$ comes from ϕ by replacing all occurrences of $\mathbf{he}_n$ or $\mathbf{him}_n$ by $\mathbf{he}_k$ or $\mathbf{him}_k$ respectively.

S15. If $\alpha \in P_T$ and $\zeta \in P_{CN}$, then $F_{10,n}(\alpha, \zeta) \in P_{CN}$.

S16. If $\alpha \in P_T$ and $\delta \in P_{IV}$, then $F_{10,n}(\alpha, \delta) \in P_{IV}$.

An Example - *De Re* vs *De Dicto* Readings

- John seeks a unicorn.

De Re

There is a particular unicorn that John is looking for.

$\exists y[\text{UNICORN}(y)\ \&\ \text{TRY}(\text{FIND}(y))(\text{J})]$

De Dicto

John will be satisfied with any unicorn.

$\text{TRY}(\exists y[\text{UNICORN}(y)\ \&\ \text{FIND}(y)])(\text{J})$

- Can be represented in terms of operator scope:

x seeks y

TRY(FIND(y))(x)

An Example - *De Re* vs *De Dicto* Readings

- John seeks a unicorn.

De Re

There is a particular unicorn that John is looking for.
$\exists y[\text{UNICORN}(y)\ \&\ \text{TRY}(\text{FIND}(y))(\text{J})]$

De Dicto

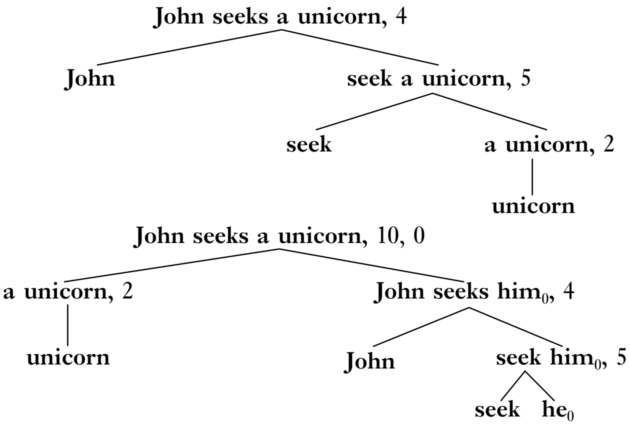
John will be satisfied with any unicorn.
$\text{TRY}(\exists y[\text{UNICORN}(y)\ \&\ \text{FIND}(y)])(\text{J})$

- Can be represented in terms of operator scope:

x seeks y

TRY(FIND(y))(x)

Derivations for *De Re* vs *De Dicto* Readings



Outline

Tree Adjoining Grammars

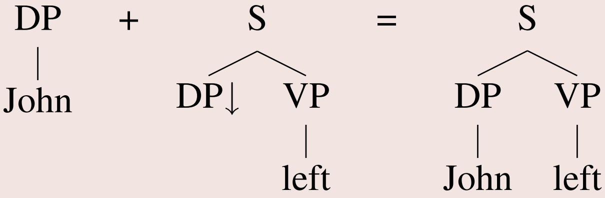
- A TAG consists of

  initial trees a finite set of trees with leaves labelled with either terminals or with $X\downarrow$, where X is a non-terminal symbol

  auxiliary trees as before, but with exactly one leaf node labelled with X^*, where X is the label of the root

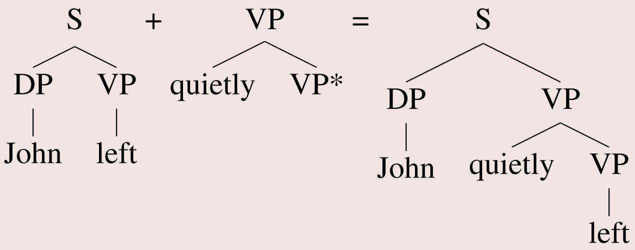
Tree Adjoining Grammars - Substitution

Substitute (1st Order Substitution)



Tree Adjoining Grammars - Adjunction

Adjoin (2nd Order Substitution)



Facts About Tree Adjoining Grammar

- Strongly equivalent to monadic linear CFTGs
  (Fujiyoshi & Kasai; Mönnich; Kepser & Rogers)
- Weakly equivalent to 2-MCFL$_{wn}$
  (Kanazawa)
- Derivation trees are terms over the set of lexical items
  - for each lexical item ℓ, enumerate its substitution and adoinable nodes (for each ℓ, there will be a finite number n_ℓ of such)
  - the rank of a lexical item ℓ is n_ℓ
  - (This is just an inside-out derivation.)

Barker's Innovation: Unrestricted Derivation and Scope

Fact

Linear CFTGs derive the same tree languages under all (IO, OI, unrestricted) derivation modes.

Barker:

Γ scopes over Δ iff

1. Γ was substituted in after Δ was, or
2. Γ was substituted in after a tree containing Δ was

Outline

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\textbf{move}, \textbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\mathbf{move}, \mathbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\textbf{move}, \textbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\mathbf{move}, \mathbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - *Cat* is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\textbf{move}, \textbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F} \rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta \rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{$**move**, **merge**$\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\mathbf{move}, \mathbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\textbf{move}, \textbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\textbf{move}, \textbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Minimalist Grammars

- A minimalist grammar is given by a 4-tuple $\langle V, Cat, Lex, \mathcal{F}\rangle$ where
  - V is a finite set of vocabulary items
  - Cat is a finite set of *features* which have the following forms:
    - x is a *selectee* (or categorial) feature
    - $=\mathrm{x}$ is a *selector* feature
    - $-\mathrm{x}$ is a *licensee* (or movement) feature
    - $+\mathrm{x}$ is an *movement triggering* feature
  - $Lex \subset V^* \times Cat^*$ is a finite set of lexical items $\ell = \langle v, \delta\rangle$, which are pairs of sequences of vocabulary items v, and sequences of features δ
  - $\mathcal{F} = \{\mathbf{move}, \mathbf{merge}\}$ is the set of generating functions
    - **move** is a unary operation, which takes a single expression and rearranges its parts
    - **merge** is a binary operation, which takes two expressions and puts them together

Trees

- The domains of our generating functions will be trees where
  - internal nodes are binary branching and labelled with either $<$ or $>$, and whose
  - leaves are labelled with pairs $\langle \sigma, \delta \rangle$ of vocabulary item sequences σ and feature sequences δ

Notation and Definitions

- The head of an expression t is
  - t itself, if it is a leaf
  - the head of t_1, if $t = <(t_1, t_2)$
  - the head of t_2, if $t = >(t_1, t_2)$

Feature Checking

- The leaves of our trees contain sequences of features. These features determine which operations can apply.
- Once an operation applies, the features which allowed it are deleted.
- We can think of the grammatical operations as ‘trying’ to remove features from trees – well-formed expressions of a particular category x will be those without any remaining features except that the head has the single feature x

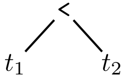
More Notation and Definitions

- Given t, we write t^f to denote the result of adding f as the first feature on the head of t:
  - if the head of t is $\langle \sigma, \delta \rangle$, then t^f is the tree just like t except that its head is $\langle \sigma, f\delta \rangle$
- t displays feature f, if the head of t is $\langle \sigma, f\delta \rangle$
  - t^f displays feature f
- Deleting the first feature of t^f gives us t

Merge

- $\langle t, t' \rangle \in \text{dom}(\textbf{merge})$ iff
  - $t = t_1^{=\mathrm{x}}$ and $t' = t_2^{\mathrm{x}}$

$$\textbf{merge}(t_1^{=\mathrm{x}}, t_2^{\mathrm{x}}) =$$



Some More Notation and Definitions

- A subtree t' of a tree t is a maximal projection iff t' does not project over its sister (if it has one)
- A particular occurrence of a subtree r in tree t is written $t = t'[r]$
  - The result of replacing a particular occurrence of subtree r with s in $t[r]$ is written $t[s]$
- I write ϵ for the empty leaf $\langle \epsilon, \epsilon \rangle$

Move

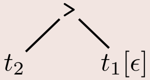
- $t^{+\mathrm{x}} \in \mathrm{dom}(\mathbf{move})$ iff
  - there is an occurrence of $t_2^{-\mathrm{x}}$ in $t^{+\mathrm{x}}$ ($t = t_1[t_2^{-\mathrm{x}}]$)
  - $t_2^{-\mathrm{x}}$ is a maximal projection in $t_1[t_2^{-\mathrm{x}}]$
- We require in addition that:

(SMC) there is exactly one occurrence of a maximal projection displaying $-\mathrm{x}$ in $t^{+\mathrm{x}}$

$$\mathbf{move}(t_1[t_2^{-\mathrm{x}}]^{+\mathrm{x}}) = [_{>}\ t_2 \quad t_1[\epsilon]\ ]$$

move

$$\textbf{move}(t_1[t_2^{-\mathrm{x}}]^{+\mathrm{x}}) =$$

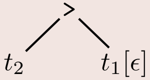


(*i*)	$\langle a, \mathrm{=x\ +y\ z} \rangle$
(*ii*)	$\langle b, \mathrm{=w\ x} \rangle$
(*iii*)	$\langle c, \mathrm{w\ -y} \rangle$

1. merge(ii,iii)
2. merge(i,1)
3. move(2)

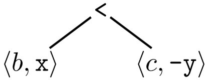
move

$$\textbf{move}(t_1[t_2^{-\mathrm{x}}]^{+\mathrm{x}}) =$$



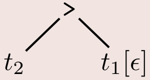
(*i*)	$\langle a, =\mathrm{x}\ +\mathrm{y}\ \mathrm{z}\rangle$
(*ii*)	$\langle b, =\mathrm{w}\ \mathrm{x}\rangle$
(*iii*)	$\langle c, \mathrm{w}\ -\mathrm{y}\rangle$

1. **merge**(ii,iii)
2. **merge**(i,1)
3. **move**(2)



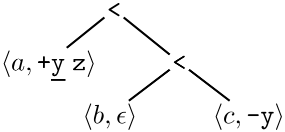
move

$$\textbf{move}(t_1[t_2^{-\mathrm{x}}]^{+\mathrm{x}}) =$$



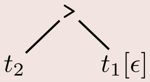
(*i*)	$\langle a, =\mathrm{x}\ +\mathrm{y}\ \mathrm{z}\rangle$
(*ii*)	$\langle b, =\mathrm{w}\ \mathrm{x}\rangle$
(*iii*)	$\langle c, \mathrm{w}\ -\mathrm{y}\rangle$

1. **merge**(ii,iii)
2. **merge**(i,1)
3. **move**(2)



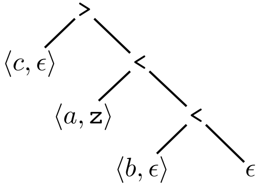
move

$$\textbf{move}(t_1[t_2^{-\mathrm{x}}]^{+\mathrm{x}}) =$$



(*i*)	$\langle a, =\mathrm{x}\ +\mathrm{y}\ \mathrm{z}\rangle$
(*ii*)	$\langle b, =\mathrm{w}\ \mathrm{x}\rangle$
(*iii*)	$\langle c, \mathrm{w}\ -\mathrm{y}\rangle$

1. **merge**(ii,iii)
2. **merge**(i,1)
3. **move**(2)



The SMC

- Having the **SMC** puts an upper bound on the number of moving pieces that an expression that could possibly converge can have:
  - if there are k licensee feature types, and you cannot have two expressions displaying the same feature at once, then at any given time you can have at most k moving pieces (one displaying each feature type)
- This allows us to represent a minimalist expression as a $(k{+}1)$-tuple of trees.
  - each of these trees only has features at its head
  - we can describe **move** and **merge** in these terms:

$$\frac{t^{+\mathrm{x}}, t_1, \ldots, t_n^{-\mathrm{x}}, \ldots}{>(t_n, t), t_1, \ldots} \quad \textbf{moveA}$$

$$\frac{t^{+\mathrm{x}}, t_1, \ldots, t_n^{-\mathrm{x}f}, \ldots}{>(\epsilon, t), t_1, \ldots, t_n^f, \ldots} \quad \textbf{moveB}$$

Factoring apart categories and strings

- Our current data structure is a sequence $\phi_0, \ldots, \phi_n$ of length up to $k+1$ of pairs $\langle w, \delta \rangle$ that has the following property:

  for $i \neq j$ the first feature of δ_i is different from the first feature of δ_j

- Let's factor out the syntactic information in such a sequence, and call this a *category*:

$$\langle \delta_0, \ldots, \delta_n \rangle$$

- There are only a finite number of relevant categories!
  (Michaelis)
- They are (a subset of) the k+1-tuples of suffixes of lexical feature sequences:

$$\text{suf}(Cat) := \{\delta : \langle \sigma, \gamma\delta \rangle \in Lex\}$$

MGs as Algebras

- We can recast MGs in an algebraic setting by:
  1. Taking as sorts
     $\mathcal{S} := \{\langle \delta_0, \delta_1, \ldots, \delta_k \rangle : \delta_i \in \text{suf}(\mathrm{Cat})\ \&\ \delta_i = \epsilon \vee -\mathrm{x}_i\delta_i'\}$
  2. Taking as operators
     - $move_s$, of type $s \rightarrow s^{\checkmark}$
       for all $s = \langle +\mathrm{x}\delta_0, \ldots, -\mathrm{x}\delta_i, \ldots \rangle$
     - $merge_{s,s'}$, of type $s \rightarrow s' \rightarrow s^{\checkmark} \oplus s'^{\checkmark}$
       for all $s = \langle =\mathrm{x}\delta_0, \ldots \rangle$, $s' = \mathrm{x}\delta_0', \ldots \rangle$
     - $\ell = \langle w, \delta \rangle$ of sort $\langle \delta, \epsilon, \ldots, \epsilon \rangle$ for each $\ell \in Lex$
- Here:
  - $\langle =\mathrm{x}\delta_0, \ldots \rangle^{\checkmark} = \langle \mathrm{x}\delta_0, \ldots \rangle^{\checkmark} = \langle \delta_0, \ldots \rangle$
  - $\langle +\mathrm{x}\delta_0, \ldots, -\mathrm{x}\delta_i, \ldots \rangle^{\checkmark} := \langle \delta_0, \ldots, u_k, \ldots, u_i, \ldots, \delta_k \rangle$, where
    - $u_k = \delta_i$ and $u_i = \epsilon$ if $\delta_i = -\mathrm{x}_k\delta$ and $\delta_k = \epsilon$
    - $u_k = \delta_k$ and $u_i = \epsilon$ otherwise
  - $s \oplus s' = \langle s_0, \delta_1, \ldots, \delta_k \rangle$, where
    - if $s_0' = -\mathrm{x}_i\delta$ then $s_i = s_i' = \epsilon$ and $\delta_i = s_0'$
    - otherwise $s_j = \epsilon$ iff $\delta_j = s_j'$

Outline

Adjuncts

- An adjunct is typically:
  - an optional element
  - iterable

Example

Modifying PPs:

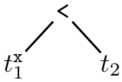
- John hit the wall.
- John hit the wall *with a hammer*.
- John hit the wall *in anger*.
- John hit the wall *with a hammer in anger*.
- John hit the wall *in anger with a hammer*.

⋮

Analyzing Adjuncts (Frey & Gärtner)

- introduce a new feature type: $\approx\mathrm{x}$

$$\textbf{adjoin}(t_1^{\mathrm{x}}, t_2^{\approx\mathrm{x}}) =$$



- algebraically, we add a new family of operators

$$\mathrm{adjoin}_{s,s'}$$

of type $s \to s' \to s \oplus s'^{\checkmark}$, where $s_0 = \mathrm{x}\delta_0$, and $s'_0 = \approx\mathrm{x}\delta'_0$

Motivating Late Adjunction

Condition C

A pronoun cannot c-command its antecedent.

- John$_i$ thinks that Mary likes him$_i$.
- *He$_i$ thinks that Mary likes John$_i$.

Condition C must hold at every level of representation

- *He$_i$ denied the rumor that John$_i$ had cheated.
- *Which rumor that John$_i$ had cheated did he$_i$ deny?

A puzzle

- *He$_i$ denied the rumor that John$_i$ started.
- Which rumor that John$_i$ started did he$_i$ deny?

A Solution: Late Adjunction

Example

- *Which rumor that John$_i$ had cheated did he$_i$ deny?
- Which rumor that John$_i$ started did he$_i$ deny?

A difference

- rumor [that John had cheated] $\rightsquigarrow$ *that John had cheated* is the content of the rumor (an essential property)
  (An **argument** of the noun)
- rumor [that John started] $\rightsquigarrow$ *that John started* is an accidental property of the rumor
  (An **adjunct** to the noun)

Lebeaux's proposal

Adjuncts can be inserted at any time.

Two derivations for *which rumor that John started*

Adjoin early; violate Condition C

1. [which rumor] + [that John started] (**adjoin**)
2. deny + [which rumor [that John started]]
3. [he] + [did deny which rumor [that John started]]

Adjoin late; no violation

1. deny + [which rumor]
2. [he] + [did deny which rumor]
3. [[which rumor] did he deny]
4. [[which rumor [that John started]] did he deny] (**adjoin**)

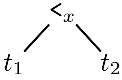
Late Adjunction, Formally

- We want to generalize the adjoin operation such that, if it could have applied to a term t, then it can (now) apply to any term $t' = C[t]$ containing t as a part.
- Note that, if it could have applied to t and to s, and $t' = D[t, s]$, then the result of adjoining to t' is nondeterministic.
- To indicate that adjunction *could* have applied to a phrase (and thus that it still can, retroactively), we extend the label set of internal nodes to include $<_{\mathrm{x}}$, and $>_{\mathrm{x}}$, for each category feature x.

Defining Late Adjunction

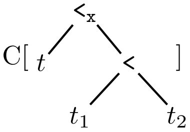
- We redefine merge so as to allow us to keep track of what category features were used.

$$\textbf{merge}(t_1^{=\mathrm{x}}, t_2^{\mathrm{x}}) =$$



- Then adjoin inserts the adjunct inside a structure at an appropriate maximal projection.

$$\textbf{adjoin}(C[<_{\mathrm{x}}(t, t_1)], t_2^{\approx \mathrm{x}}) \rightarrow$$



The Late Adjunction Algebra

- We take as sorts $\mathcal{S} := \{\langle\langle\delta_0, \delta_1, \ldots, \delta_k\rangle, C\rangle : \delta_i \in \text{suf}(\mathrm{Cat})$ & $\delta_i = \epsilon \vee -\mathrm{x}_i\delta_i'$ & $C \subseteq Cat\}$
- Taking as operators
  - move$_s$, of type $s \rightarrow s^{\checkmark}$
  - merge$_{s,s'}$, of type $s \rightarrow s' \rightarrow s^{\checkmark} \oplus_{\mathrm{x}} s'^{\checkmark}$
  - adjoin$_{s,s'}$ of type $s \rightarrow s' \rightarrow s \oplus s'^{\checkmark}$
  - $\ell = \langle w, \delta\rangle$ of sort $\langle\langle\delta, \epsilon, \ldots, \epsilon\rangle, \emptyset\rangle$ for each $\ell \in Lex$
- where $\langle s, C\rangle \oplus_{\mathrm{x}} \langle s', C'\rangle = \langle s \oplus s', C \cup C' \cup \{\mathrm{x}\}\rangle$

Note that this is semi-linear; these operators have the same interpretation in the Parikh domain as the previous ones.

Outline

1. On Late Operations
   - Montague Grammar
   - TAGs
2. Minimalist Grammars
   - Basic Definitions and Properties
   - Late Adjunction
3. Representing Late Adjunction in MGs
   - Via 3^{rd} Order ACGs
   - Via MSOTT With One Free Relation Variable
4. Conclusion

Abstract Categorial Grammars – Vocabulary

- A Vocabulary is a triple $\Sigma = \langle A, C, \tau \rangle$, where
  - A is a finite set of types
  - C is a finite set of constants
  - $\tau : C \rightarrow \mathcal{T}(A)$ maps each constant to its type (built up over A in the standard way)

Example

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$, where $A_1 = \{z\}$, $C_1 = \{a, b, f\}$, such that $\tau_1(f) = z \rightarrow z \rightarrow z$, and $\tau_1(a) = \tau_1(b) = z$. Then all linear lambda terms of type z are trees over $\{f^{(2)}, a^{(0)}, b^{(0)}\}$.

Example

Let $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, where $A_2 = \{\star\}$, and $C_2 = \{a, b\}$, where $\tau_2(c) = \star \rightarrow \star$ for all $c \in C$. Then the set of linear terms over Σ_2 are 'strings' of the form

$$\lambda x.w(x)$$

Abstract Categorial Grammars – Lexicon

- Given two vocabularies Σ_1, Σ_2, a lexicon $\mathcal{L} : \Sigma_1 \to \Sigma_2$ interprets Σ_1 inside of Σ_2 in the following manner:
  - each atomic type of Σ_1 is interpreted as a linear implicative type in Σ_2
  - each constant in Σ_1 is interpreted as a linear λ-term in Σ_2 whose type is gotten from the type of the constant in Σ_1 by replacing atomic types by their interpretations in Σ_2

Example

Interpreting the type z as the type $\star^2$, the constant f as the lambda term $\lambda u, v, x.u(v(x))$ of type $\star^2 \to \star^2 \to \star^2$, and the constants a and b as $\lambda x.a(x)$ and $\lambda x.b(x)$ of type $\star^2$, a lambda term in Σ_1 is interpreted as a term in Σ_2

$$f(a, f(b, b)) \rightsquigarrow \lambda x.a(b(b(x)))$$

MGs in ACGs

- We take as atomic types $A_G := \mathcal{S}$
- the constants are as before
- The linear lambda terms of atomic type are just the derivation trees of expressions of that type.

Note:

We want to take as our object vocabulary something with *tuples* (of trees or strings).

- $\langle \epsilon, \epsilon \rangle$:

$$\lambda w.w(\lambda x.x)(\lambda x.x)$$

- $\langle x, y \rangle \mapsto \langle ay, bx \rangle$:

$$\lambda f.\lambda w.f(\lambda xy.w(\lambda z.ayz)(\lambda z.bxz))$$

I will ignore this here.

Adding Late Adjunction

- We could represent an adjunction site as a variable of type $a \rightarrow a$

$$\textbf{merge}(Y(\textbf{merge}(\text{kiss})(\text{Mary})))(\text{John}))$$

- A simple adjunct, whose syntactic category is $\langle \approx \mathrm{x}, \epsilon, \ldots, \epsilon \rangle$ (like 'yesterday'), can't be given directly to the function below (because this β-reduces to the 'normal' adjunction order)

$$\lambda Y.\textbf{merge}(Y(\textbf{merge}(\text{kiss})(\text{Mary})))(\text{John}))$$

- Instead we need to 'type raise' the adjunct: it takes functions of type $(a \rightarrow a) \rightarrow a$ into objects of type a

Adding Late Adjunction

- This doesn't work for adjuncts with moving pieces; the moving pieces must be added to the type of the resulting expression. A general solution assigns a fourth order type to adjuncts (here $s = \langle \approx\mathrm{x}\delta_0, \delta_1, \ldots, \delta_k \rangle$ is the syntactic category of the adjunct)

$$((a \rightarrow a) \rightarrow a) \rightarrow a \oplus s^{\checkmark}$$

- then we would have terms like

$$\text{yesterday}(\lambda Y.\textbf{merge}(Y(\textbf{merge}(\text{kiss})(\text{Mary})))(\text{John}))$$

- We could interpret an adjunct α as the following string operation of type $((\star^2 \rightarrow \star^2) \rightarrow \star^2) \rightarrow \star^2$:

$$\lambda P_{(\star^2\star^2)\star^2}.P(\lambda y_{\star^2}.y ^\frown /\alpha/)$$

From 4^{th} order to 3^{rd} order

- the type $((aa)a)a'$ is of order 4
- What we are doing is allowing the 'adjunct' to be a context
- But the adjunct only ever is pronounced on one side of its argument
- Thus we should be able to treat 'adjuncts' as terms
- we take a new set of types $(v_a)_{a \in A}$ (for 'variables' of type a)
- and the new family of constants **adj** : $v_a \to a \to a$
- Then we have terms like:

  $$\lambda x.\mathbf{merge}(\mathbf{adj}(x)((\mathbf{merge}(\text{kiss})(\text{Mary})))(\text{John})))$$

  of type $v_a \to b$
- an adjunct can then have the third-order type $(v_a \to b) \to b \oplus s^{\checkmark}$, and the interpretation:

  $$\lambda P_{\star^2 \star^2}.P(/\alpha/)$$

Problem

- Knowing that MGs with Late Adjunction can be represented in ACG(3) doesn't help us much:
  - ACG(3) contains NP-complete as well as non-semilinear string languages

(Yoshinaka & Kanazawa)

Outline

MSO Logic

- Quantification over nodes (x) and sets of nodes (X)
- atomic statements:
  - $x = y$
  - $x \in X$
  - $\text{lab}_\alpha(x)$ (the label of node x is α)
  - $\text{edge}_i(x, y)$ (y is the i^{th} daughter of x)

MSO-definable Transduction

The idea:

You make k copies of the original nodes, decide which ones you will keep (by giving them labels), and which edges to draw between the nodes you kept.

Definition:

A MSO graph transducer is a triple $\langle k, \Psi, X \rangle$, where

- $k \in \mathbb{N}$ is the number of copies of each node,
- $\Psi = \{\psi^i_\sigma(x) : \sigma \in \Sigma, i \leq k\}$ is the set of node formulae
  $\psi^i_\sigma(x)$ iff the i^{th} copy of x exists and has label σ
- $X = \{\chi^{i,j}_n(x, y) : n \leq max(\{rank(\sigma) : \sigma \in \Sigma\}), i, j \leq k\}$ is the set of edge formulae
  $\chi^{i,j}_n(x, y)$ iff the j^{th} copy of y is the n^{th} daughter of the i^{th} copy of x

MSO-definable Transduction

The idea:

You make k copies of the original nodes, decide which ones you will keep (by giving them labels), and which edges to draw between the nodes you kept.

Definition:

A MSO graph transducer is a triple $\langle k, \Psi, X \rangle$, where

- $k \in \mathbb{N}$ is the number of copies of each node,
- $\Psi = \{\psi^i_\sigma(x) : \sigma \in \Sigma, i \leq k\}$ is the set of node formulae $\psi^i_\sigma(x)$ iff the i^{th} copy of x exists and has label σ
- $X = \{\chi^{i,j}_n(x, y) : n \leq max(\{rank(\sigma) : \sigma \in \Sigma\}), i, j \leq k\}$ is the set of edge formulae $\chi^{i,j}_n(x, y)$ iff the j^{th} copy of y is the n^{th} daughter of the i^{th} copy of x

MSO-definable Transduction

The idea:

You make k copies of the original nodes, decide which ones you will keep (by giving them labels), and which edges to draw between the nodes you kept.

Definition:

A MSO graph transducer is a triple $\langle k, \Psi, X\rangle$, where

- $k \in \mathbb{N}$ is the number of copies of each node,
- $\Psi = \{\psi^i_\sigma(x) : \sigma \in \Sigma, i \leq k\}$ is the set of node formulae
  $\psi^i_\sigma(x)$ iff the i^{th} copy of x exists and has label σ
- $X = \{\chi^{i,j}_n(x, y) : n \leq max(\{rank(\sigma) : \sigma \in \Sigma\}), i, j \leq k\}$ is the set of edge formulae
  $\chi^{i,j}_n(x, y)$ iff the j^{th} copy of y is the n^{th} daughter of the i^{th} copy of x

MSO-definable Transduction

The function induced by a MSO transducer $\langle k, \Psi, X \rangle$

Given input $g_1 = \langle V_1, E_1, \text{lab}_1 \rangle$, the output is the graph $\langle V, E, \text{lab} \rangle$ such that:

- $V := \{u_i : u \in V_1, \exists! \alpha.\ g_1, u \models \psi^i_\alpha(x)\}$
- $\text{lab}(u_i)$ is the unique α such that $g, u \models \psi^i_\alpha(x)$
- $E := \{\langle u_i, n, v_j \rangle : u_i, v_j \in V, \wedge\ g_1, u, v \models \chi^{i,j}_n(x, y)\}$
- $\langle u_i, n, v_j \rangle$ is the edge from u_i to v_j labelled n

A simple example – the yield of a tree

- We only need one copy of each node (i.e. we can interpret the yield of a tree inside the tree itself)
- We interpret the root as the leaf:

$$\psi_e^1(x) = \mathrm{root}(x)$$

- Leaves keep their labels:

$$\psi_\alpha^1(x) = \mathrm{leaf}(x) \wedge \mathsf{lab}_\alpha(x)$$

- Edges are drawn from one leaf to the next, and from the last leaf to the root:

$$\begin{aligned}\chi_1^{1,1}(x, y) = &(\mathrm{rightmost\text{-}leaf}(y, x) \wedge \mathrm{root}(y)) \\ &\vee \mathrm{adjacent\text{-}leaves}(x, y)\end{aligned}$$

MSOTT and Minimalist Grammars

- $MSOTT(REGT) = Tr(HR)$
- $MSOTT_{dir}(REGT) \subset MSOTT(REGT)$
- $T_{fc}(REGT) = MSOTT_{dir}(REGT)$
- $MG \subset T_{fc}(REGT)$
- Therefore, we have that for every MG G, there is a (direction preserving) MSO transducer T_G such that on every term t over G, $val_G(t) = T_G(t)$

Transducing Late Adjunction

- Two main problems:
  1. choosing one of the possible adjunction sites
  2. putting the late adjoined material there
- Clearly:

if no phonological material is ever adjoined, the mapping from MG derivation trees with late adjunction to the derived string is MSOTT$_{dir}$.
This can happen if, for every derivable expression of the form $t^{\approx \mathrm{x}}$,

- either $t = s^{-\mathrm{y}}$
- or $t = C[t_1^{-\mathrm{y}_1}, \ldots, t_k^{-\mathrm{y}_k}]$, and yield$(C[e, \ldots, e]) = e$

In this case we can simply avoid solving these problems!

- For the general case, let's look at each of these separately, beginning with the second one.

Moving subtrees 'down'

- Consider trees over the signature $\{\bullet^{(2)}, \circ^{(2)}, \oplus^{(2)}, a^{(0)}\}$ such that
  - each tree has exactly one node with label $\oplus$
  - each tree has exactly one node with label $\circ$
  - the node labelled $\oplus$ is contained within the first daughter of the node labelled $\circ$
- Given a tree $C[\circ(D[\oplus(t_1, t_2)], t')]$, we want to transduce it into the tree $C[D[\bullet(t_1, \bullet(t_2, t'))]]$
- This is like putting a late adjunct into its chosen position.
- Clearly, this is not realizable by a direction preserving MSO transduction.

A MSO transduction for moving subtrees down

Keep the original nodes

- $\psi^0_\bullet(x) = \neg\text{leaf}(x)$

  `Internal nodes are labelled with` '$\bullet$'
- $\psi^0_a(x) = \text{leaf}(x)$

  `Leaves are labelled` 'a'

A MSO transduction for moving subtrees down

$$C[\circ_1(D[\oplus_2(q, r)], s)] \rightsquigarrow C[D[\bullet_2(q, \bullet_1(r, s))]]$$

Positioning $\circ$

- The parent of $\circ$ should instead be connected to its left daughter.
- The node $\circ$ should be the second daughter of the node labelled $\oplus$
- The node $\circ$ should have as its first daughter the second daughter of the node labelled $\oplus$
- All other edges are the same

A MSO transduction for moving subtrees down

$$C[\circ_1(D[\oplus_2(q,r)],s)] \rightsquigarrow C[D[\bullet_2(q,\bullet_1(r,s))]]$$

Positioning $\circ$

- The parent of $\circ$ should instead be connected to its left daughter.
- The node $\circ$ should be the second daughter of the node labelled $\oplus$
- The node $\circ$ should have as its first daughter the second daughter of the node labelled $\oplus$
- All other edges are the same
- $$\begin{aligned} \chi_0^{0,0}(x,y) = y, x \neq \circ \wedge \mathrm{edge}_0(x,y) \\ \vee\, \mathrm{edge}(x,\circ) \wedge \mathrm{edge}_0(\circ,y) \\ \vee\, x = \circ \wedge \mathrm{edge}_2(\oplus,y) \end{aligned}$$

A MSO transduction for moving subtrees down

$$C[\circ_1(D[\oplus_2(q,r)],s)] \rightsquigarrow C[D[\bullet_2(q,\bullet_1(r,s))]]$$

Positioning $\circ$

- The parent of $\circ$ should instead be connected to its left daughter.
- The node $\circ$ should be the second daughter of the node labelled $\oplus$
- The node $\circ$ should have as its first daughter the second daughter of the node labelled $\oplus$
- All other edges are the same
- $$\begin{aligned}\chi_1^{0,0}(x,y) = {} & x,y \neq \circ,\oplus \wedge \mathrm{edge}_1(x,y) \\ & \vee x = \oplus \wedge y = \circ \\ & \vee \mathrm{edge}(x,\circ) \wedge \mathrm{edge}_0(\circ,y)\end{aligned}$$

Generalizing to arbitrarily many subtrees

- Assume that we know which subtree should be adjoined where (R)
  - This relation should respect dominance (if xRy then x should dominate y)
  - This relation should be defined only on nodes labelled $\circ$
  - This relation should be an injective function!
    (multiple adjunction to the same XP is treated instead as adjunction to the unique adjunct (to the unique adjunct . . .) adjoining to XP)
- Then we can simply replace talk of $\circ$ and $\oplus$ with talk of x and y such that xRy
- Adding R to our trees essentially gives us a DAG where nodes have at most two parents.

Representing R in a tree

- We can code R into our trees, preserving regularity:
  - introduce a new symbol $\otimes^{(1)}$
  - $\otimes$ can be a child of $\otimes$ or of **adjoin**
  - at an adjoin node, the length of the $\otimes$ string it dominates indicates which of the possible adjunction sites it adjoins to (if there aren't enough adjunction sites, take the last one).
- However, MSO cannot 'walk' simultaneously along two branches (otherwise it could define **isomorphic**$(\cdot, \cdot)$)
- We *could* do this in FOL-DTC(2)...
- ...but we don't have a characterization of the FOL-DTC(2) definable tree transductions
  - On the automaton side, FOL-DTC(i) corresponds to pebble automata with i heads
  - single headed pebble tree to tree transducers with k pebbles correspond to $k + 1$ MTTs (Engelfriet & Maneth)

Taking Stock

- The substitution required by late adjunction is easy to do...
- *if* we know where to do it!
- Naively encoding this information into the tree is not MSO-usable,...
- and one logic that can deal with it seems like it would be beyond the IO-hierarchy!

Conclusion

- Late operations are formally natural
- . . . and linguists like them
- Adding late adjunction to MGs, we see that the relation between parse trees and derived tree is not an MSOTT.
- But if we include information about where each late adjoined element is supposed to appear, we have an MSO definable graph to tree transduction.
- Courcelle has shown that every MSO transduction on a graph of bounded treewidth results in a language of some hyperedge replacement grammar. Do the graphs we are using here have bounded treewidth? If yes, then
    - as our transduction gives trees,
    - and the sets of yields of CFHG definable sets of trees are always MCFLs
    - MGs with late adjunction would be MCFL

Thank you!