# On the Form-Meaning Relations Definable by CoTAGs

**Gregory M. Kobele**
University of Chicago
Chicago, Illinois
USA

**Jens Michaelis**
Bielefeld University
Bielefeld
Germany

## Abstract

Adding *cosubstitution* to the classical TAG operations of *substitution* and *adjunction*, *coTAGs* have been proposed as an "alternative conceptualization" to resolve the tension between the TAG mantra of locality of syntactic dependencies and the seeming non-locality of quantifier scope. CoTAGs follow the tradition of *synchronous TAGs (STAGs)* in that they derive syntactic and semantic representations simultaneously as pairs. We demonstrate that the mappings definable by coTAGs go beyond those of "simple" STAGs. While with regard to the first component, coTAGs are weakly and strongly equivalent to classical TAGs, the second projection of the synchronously derived representations, can in particular be—up to a homomorphism—the non-tree adjoining language MIX($k$), for any $k \geq 3$.

## 1 Introduction

Given a classical TAG as, e.g., defined in the handbook article by Joshi and Schabes (1997), the set of derivation trees constitutes a regular set of unordered labeled trees with an additional labeling of the tree edges. The nodes of a derivation tree are labeled with elementary trees, and each edge is labeled with a Gorn address. Such an address indicates the node of the elementary tree (labeling the outgoing node of the corresponding edge in the derivation tree) at which another elementary tree (labeling the incoming node of the corresponding edge in the derivation tree) was adjoined or substituted during the derivation represented by the derivation tree. The representation of a TAG derivation in this way is independent of the derivational order. This is the reason why the set of derivation trees of a classical TAG constitutes a regular tree set.

Work on semantics in the TAG framework often considers the derivation tree as a compact, order independent representation of all derivations yielding the same syntactic tree, but at the same time as a compact representation of different semantics associated with the same syntactic representation. Work in this line, in particular includes the extensive work of Kallmeyer and colleagues as well as Nesson and Shieber.[1]

Suggesting an "alternative conceptualization" to resolve the tension between the TAG mantra of locality of syntactic dependencies and the seeming non-locality of quantifier scope, Barker (2010) proposes to add to the classical TAG operations of *substitution* and *adjunction* as a third operation a modified version of substitution. He calls the resulting operation *cosubstitution* and the version of TAGs incorporating this operation *coTAGs*.

CoTAGs are defined in the spirit of *synchronous TAGs (STAGs)* as introduced by Shieber and Schabes (1990), deriving syntactic and semantic representations simultaneously as pairs. Syntactically, cosubstitution can essentially be understood as substitution, but with reversed roles of functor and argument. Barker notes that for this reason, from a purely syntactic perspective, in the context of simple (i.e. not multicomponent) TAGs, adding cosubstitution affects neither weak nor strong generative capacity (in the sense of derived string and tree languages).

Clearly, however, something is different: after adding the operation of cosubstitution, derivational order matters in the sense that one derived syntactic representation can potentially be associated with more than one simultaneously derived semantic representation. As Barker points out,

[1]See, e.g., Kallmeyer and Romero (2004) and Nesson and Shieber (2007), and references cited therein.

the introduction of the cosubstitution operator allows for a straightforward adaption of the notion of derivation tree such that two derivation trees can be different depending on when a cosubstitution step takes place.

We demonstrate that the form-meaning mappings definable by coTAGs go beyond those of "simple" STAGs (Shieber, 1994; Shieber, 2006).[2] In particular, the set of meanings, the second projection of the synchronously derived syntactic and semantic representations, can be—up to a homomorphism abstracting away from instances of $\lambda$ and variables—the non-tree adjoining language MIX($k$), for any $k \geq 3$. The complexity of the corresponding set of meanings is a reflex of the complexity of the connected derivation tree set, whose path language—up to a homomorphism—also provides the non-tree adjoining language MIX($k$). The result can already be established when restricting the attention to coTSGs, understood as coTAGs without classical adjunction. From this perspective it could be argued that the additional expressive power is really due to the cosubstitution operation alone.

## 2 CoTAGs

From the perspective of lexical entries, each coTAG consists of a finite set of pairs of labeled trees. The first component of such a pair $\langle \alpha_{\text{syn}}, \alpha_{\text{sem}} \rangle$ provides a syntactic representation, the second component a semantic representation of the corresponding lexical entry. Nodes in $\alpha_{\text{syn}}$ are uniquely linked to nodes in $\alpha_{\text{sem}}$. The label of a node $\nu_{\text{sem}}$ from $\alpha_{\text{sem}}$ linked to a node $\nu_{\text{syn}}$ from $\alpha_{\text{syn}}$ displays the semantic type of the corresponding syntactic subconstituent dominated by $\nu_{\text{syn}}$. An operation applying to $\nu_{\text{syn}}$ must be accompanied by a parallel operation applying to $\nu_{\text{sem}}$.

Regarding the syntactic component of the coTAG-relation, coTAGs are identical to classical TAGs except for the following difference: whereas the roots of TAG-initial trees and substitution nodes of arbitrary TAG-elementary trees are labeled with elements from $\texttt{Cat}$ and $\texttt{Cat}{\downarrow}$, respectively,[3] the roots of syntactic coTAG-initial

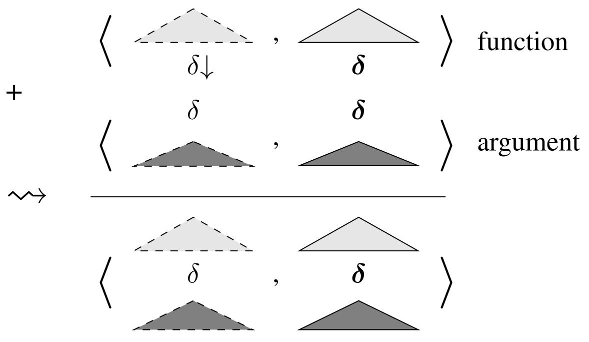


Figure 1: *Substitution* schematically. Syntactically, tree with root-label $\delta$ is substituted at leaf labeled $\delta{\downarrow}$; while semantically, the corresponding tree with root-label $\boldsymbol{\delta}$ is substituted at the linked node labeled $\boldsymbol{\delta}$.

trees and substitution nodes of arbitrary syntactic coTAG-elementary trees have labels from the sets $\texttt{Cat}(\uparrow\texttt{Cat})^*$ and $\texttt{Cat}(\uparrow\texttt{Cat})^*{\downarrow}$, respectively. Expressions of the syntactic component are constructed in very much the same manner as in TAGs, with one important addition: a derived structure $\beta_{\text{syn}}$ with syntactic root-label $\delta\uparrow B$ for some $B \in \texttt{Cat}$ and $\delta \in \texttt{Cat}(\uparrow\texttt{Cat})^*$ can be cosubstituted into a derived syntactic structure $\alpha_{\text{syn}}$ with syntactic root-label $B$ and a leaf labeled $\delta{\downarrow}$ derived structure. The result of applying cosubstitution in this situation is the same as substituting $\beta'_{\text{syn}}$ into $\alpha_{\text{syn}}$ at the corresponding leaf labeled $\delta{\downarrow}$, where $\beta'_{\text{syn}}$ results from $\beta_{\text{syn}}$ by replacing the root-label $\delta\uparrow B$ of $\beta_{\text{syn}}$ with $\delta$. From this perspective, cosubstitution can be understood as substitution reversing the roles of argument and functor, i.e., $\alpha_{\text{syn}}$ is rather cosubstituted onto the root of $\beta_{\text{syn}}$. As will become immediately clear, the "reversed perspective" of argument and functor chimes in with the operational semantic counterpart of applying cosubstitution.

Regarding the semantic component of the coTAG-relation, the trees derived represent well-typed lambda terms, which can be read off from the yield.[4] The matching semantic operation to applying substitution and adjunction syntactically is also substitution and adjunction, providing us with "functional application" in terms of lambda calculus, cf. Figure 1 for the case of substitution.[5]

[2]While we focus here on coTAGs, the results herein straightforwardly apply as well to *limited delay vector TAGs (LDV-TAGs)* (Nesson, 2009) which would allow for a different implementation of essentially the same mechanism.

[3]$\texttt{Cat}$ denotes the set of categories, i.e. the nonterminals, of the grammar.

[4]Arriving at a concrete lambda term is achieved by replacing each leaf-label which is a semantic type by a variable of corresponding type, when reading off the leaf-labels "from left to right."

[5]For $\delta \in \texttt{Cat}(\uparrow\texttt{Cat})^*$ and $B \in \texttt{Cat}$, the boldface versions $\boldsymbol{\delta}$ and $\boldsymbol{\delta}\uparrow\boldsymbol{B}$ denote the corresponding semantic types of

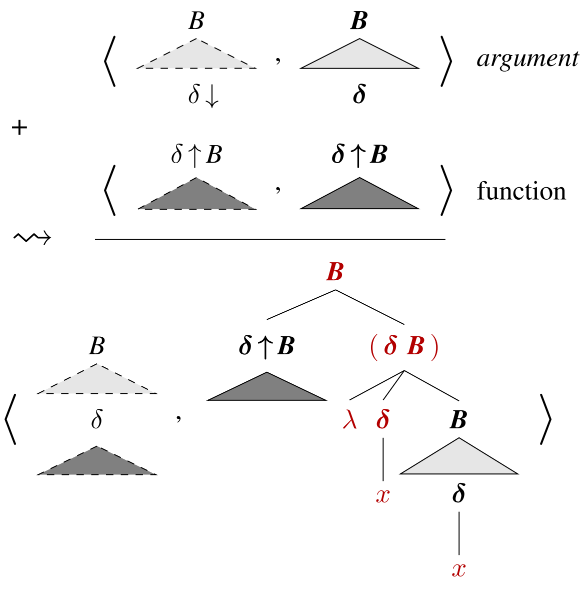


Figure 2: *Cosubstitution* schematically. Syntactically, tree with root-label with root-label $\delta{\uparrow}B$ is cosubstituted into tree with root-label $B$ at leaf labeled $\delta{\downarrow}$; while semantically, the corresponding tree with root-label $\boldsymbol{\delta}{\uparrow}\boldsymbol{B}$ is "quantified in" at the root of the corresponding tree with root-label $\boldsymbol{B}$.

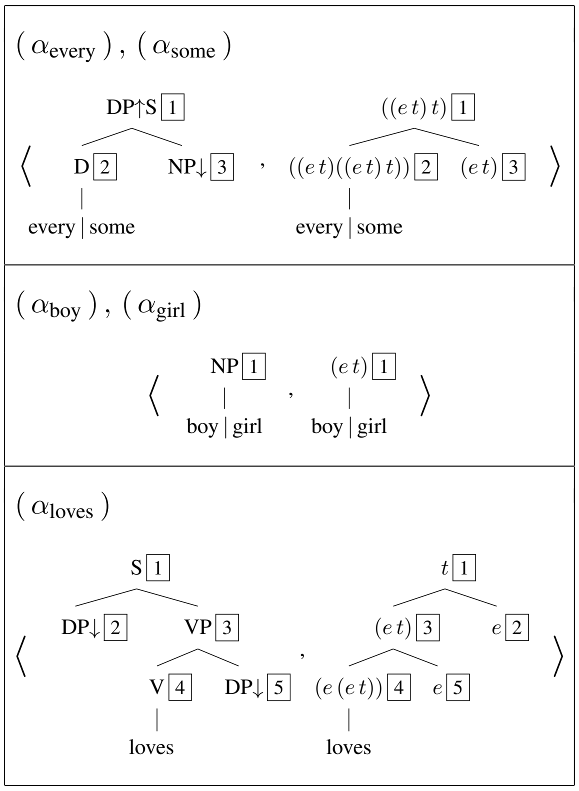


Figure 3: The example coTAG $G_{scope}$

The matching semantic operation to applying cosubstitution syntactically consists of a "quantifying in" step as outlined in Figure 2. Note that within the resulting semantic representation, new terminal leaves are introduced labeled by $\lambda$ and a variable $x$. The operation is set up such that $x$ is chosen to be "fresh."

As a concrete example, consider the grammar $G_{scope}$ presented in Figure 3.[6] One can start deriving the sentence "every boy loves some girl" either by substituting *boy* into *every*, and then cosubstituting *every boy* into the subject position of *loves* as shown in Figure 4, or by substituting *girl* into *some*, and then cosubstituting *some girl* into the object position of *loves* as shown in Figure 5. Both complete derivations of the sentence are given in Figure 6. The derived syntactic trees are identical, while the semantic trees are different, because of the different order of the derivation steps. Accordingly, we require a novel notion of derivation to represent this "timing" information, which we leave at an intuitive level in this paper for reasons of space.

Displaying a derivation tree, we use a solid line for drawing an edge in order to indicate an instance of substitution, and the edge label marks the address of the substitution site of the elementary tree into which substitution takes place. We use a dashed line in order to indicate an instance of cosubstitution, and the edge label marks the address of the substitution site of the elementary tree into which cosubstitution takes place. But in contrast to the case of substitution, this elementary tree labels the incoming node of the edge.

## 3 Expressivity

For $k \geq 1$, we now provide a coTAG $G_k$ generating as its string language the regular language $\{(a_1 \cdots a_k)^m : m \geq 1\}$, while the path language of the set of derivation trees and the set of meanings, up to a homomorphism, provide the language MIX$(k)$, i.e. the set $\{w \in \{a_1, \ldots, a_k\}^* : |w|_{a_i} = |w|_{a_j}, 1 \leq i,j \leq k\}$.

$G_k$ consists of $k+4$ lexical entries, namely, the entries $\alpha_1, \ldots, \alpha_k, \beta_k, \gamma_k, \tau_k$ and $\sigma_k$, and for each entry, we use the (additional) subscripts $\_{\text{syn}}$ and $\_{\text{sem}}$ in order to refer to the entry's syntactic and semantic component, respectively, cf. Figure 7. $S$

$\delta{\downarrow}^{?}_{}$ and $\delta{\uparrow}B$, respectively. $\boldsymbol{\delta}{\uparrow}\boldsymbol{B}$ is the lifted type $((\,\boldsymbol{\delta}\,\boldsymbol{B})\,\boldsymbol{B}\,)$.

[6]Links will usually be marked with diacritics of the form $\boxed{n}$ for some $n \geq 1$. We may occasionally avoid explicitly mentioning the links between nodes of the syntactic and the semantic representation, when we think the canonical linking is obvious.

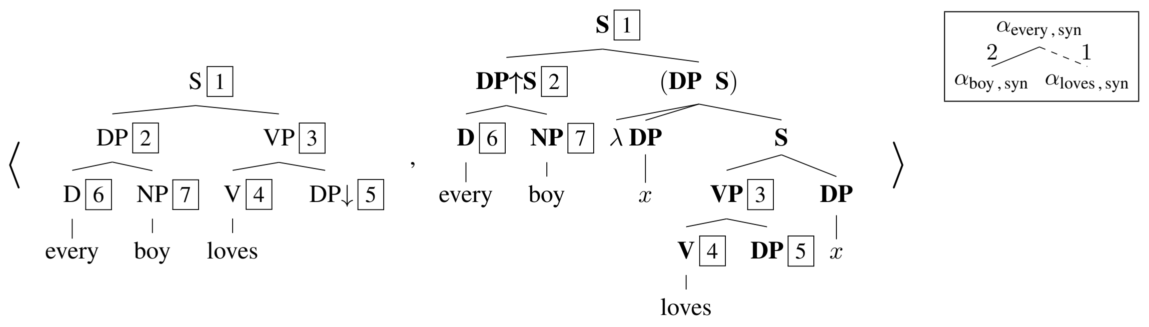


Figure 4: Cosubstituting *every boy* into the subject position before filling the object position of *loves*, derived syntactic tree and semantic tree, and derivation tree.

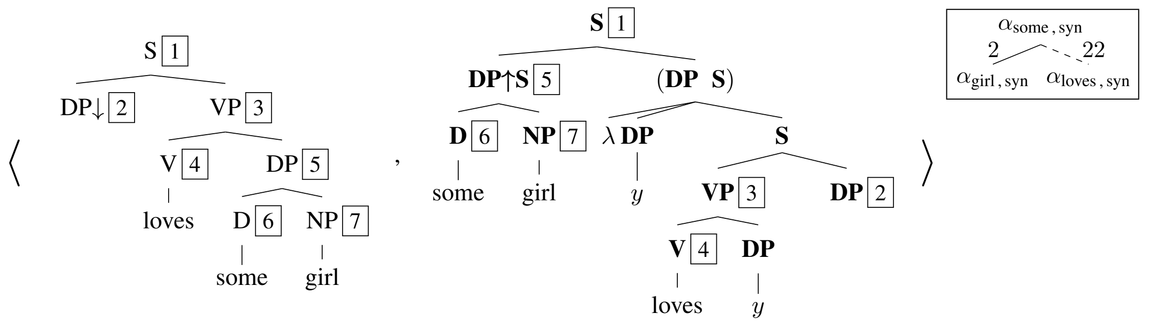


Figure 5: Cosubstituting *some girl* into the object position before filling the subject position of *loves*, derived syntactic and semantic tree, and derivation tree.

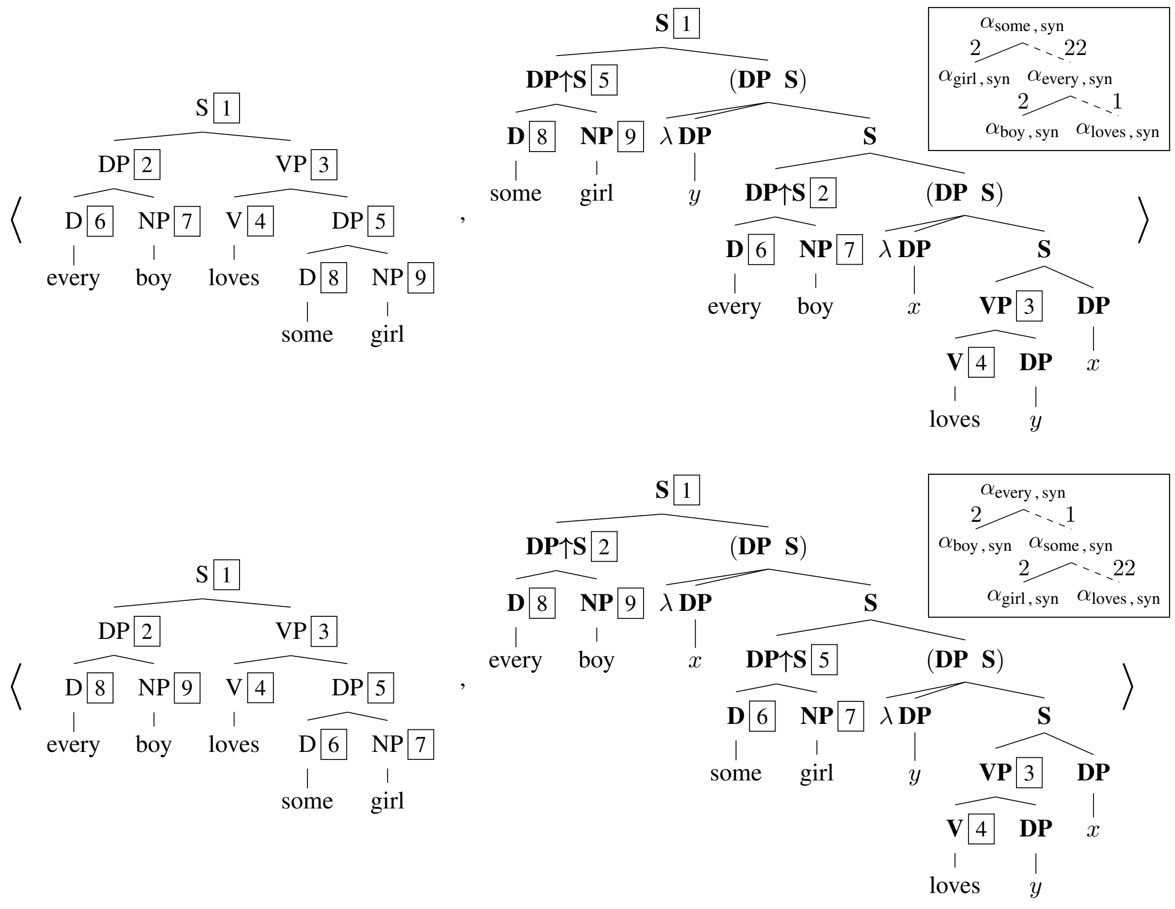
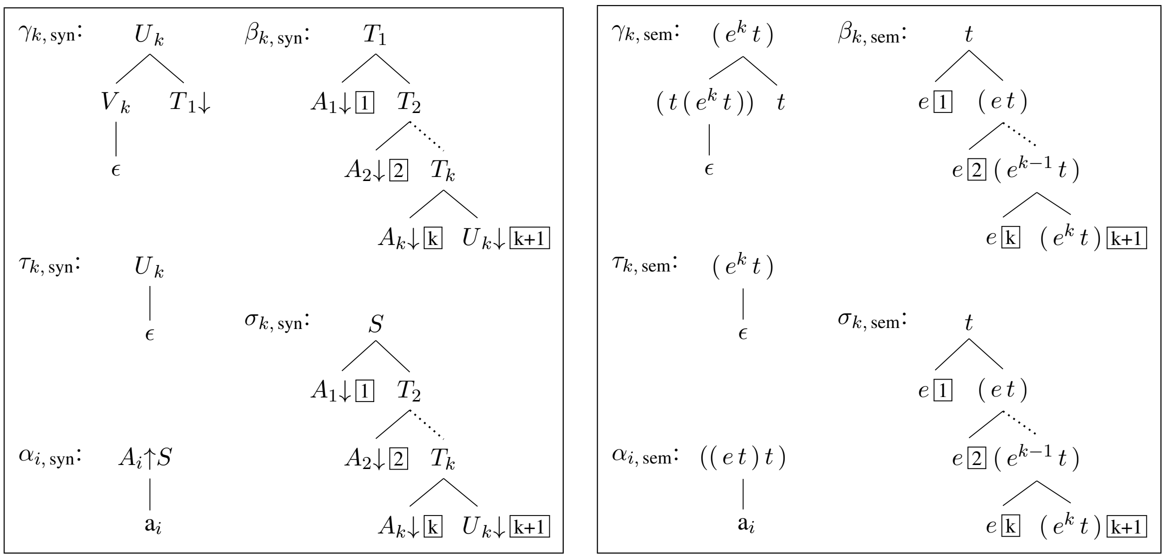


Figure 6: The two complete derived pairs of structures of *every boy loves some girl*, and derivation trees.

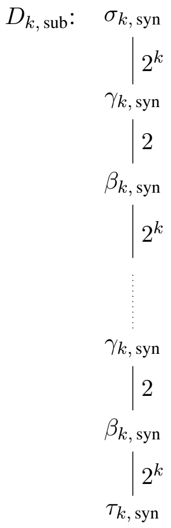


$\gamma_{k,\,\text{syn}}$: $U_k$ $V_k$ $T_1\downarrow$ $\epsilon$
$\beta_{k,\,\text{syn}}$: $T_1$ $A_1\downarrow\boxed{1}$ $T_2$ $A_2\downarrow\boxed{2}$ $T_k$ $A_k\downarrow\boxed{k}$ $U_k\downarrow\boxed{k+1}$
$\tau_{k,\,\text{syn}}$: $U_k$ $\epsilon$
$\sigma_{k,\,\text{syn}}$: $S$ $A_1\downarrow\boxed{1}$ $T_2$ $A_2\downarrow\boxed{2}$ $T_k$ $A_k\downarrow\boxed{k}$ $U_k\downarrow\boxed{k+1}$
$\alpha_{i,\,\text{syn}}$: $A_i\uparrow S$ $\text{a}_i$
$\gamma_{k,\,\text{sem}}$: $(e^k t)$ $(t(e^k t))$ $t$ $\epsilon$
$\beta_{k,\,\text{sem}}$: $t$ $e\boxed{1}$ $(e\,t)$ $e\boxed{2}$ $(e^{k-1}t)$ $e\boxed{k}$ $(e^k t)\boxed{k+1}$
$\tau_{k,\,\text{sem}}$: $(e^k t)$ $\epsilon$
$\sigma_{k,\,\text{sem}}$: $t$ $e\boxed{1}$ $(e\,t)$ $e\boxed{2}$ $(e^{k-1}t)$ $e\boxed{k}$ $(e^k t)\boxed{k+1}$
$\alpha_{i,\,\text{sem}}$: $((e\,t)\,t)$ $\text{a}_i$

Figure 7: The syntactic and the semantic components of $G_k$.

is the start symbol of $G_k$.

For $m \geq 1$, consider $w_{k,m} = (a_1 \ldots a_k)^m$. Each derivation tree for $w_{k,m}$ is a single path. All derivation trees for $w_{k,m}$ share a unique subtree, cf. Figure 8. From a bottom-up perspective, the derivation starts by substituting $\tau_k$ directly into $\sigma_k$ in case $m = 1$, or into $\beta_k$ otherwise. If $m > 1$, the resulting tree is substituted into $\gamma_k$, and the result in its turn is substituted into $\beta_k$ again. This procedure is repeated $m - 1$ times, before the resulting tree is substituted into $\sigma_k$. Note that the substitution site in each of these derivation steps is uniquely determined. Note also that we cannot cosubstitute any instance of $\alpha_i$ before we have substituted into $\sigma_k$, because cosubstitution of an $\alpha_i$ demands the presence of a root labeled $S$.



$D_{k,\,\text{sub}}$: $\sigma_{k,\,\text{syn}}$
$2^k$
$\gamma_{k,\,\text{syn}}$
$2$
$\beta_{k,\,\text{syn}}$
$2^k$
$\gamma_{k,\,\text{syn}}$
$2$
$\beta_{k,\,\text{syn}}$
$2^k$
$\tau_{k,\,\text{syn}}$

$$\sigma_{k,\,\text{syn}}\,(\underbrace{\gamma_{k,\,\text{syn}}\,(\beta_{k,\,\text{syn}}\,(\ldots(\gamma_{k,\,\text{syn}}\,(\beta_{k,\,\text{syn}}}_{m-1\ \text{times}}\,(\tau_{k,\,\text{syn}}\,)))\ldots)))$$

Figure 8: The unique subtree $D_{k,\,\text{sub}}$ of any derivation tree for $w_{k,m} = a_1^m \ldots a_k^m$.

The derived tree described by the derivation tree $D_{k,\,\text{sub}}$ contains exactly $m$ substitution sites for every $A_i$. We can now cosubstitute into any of these sites in any order. Thus, for each permutation of $(a_1 \ldots a_k)^m$ there is a derivation of $(a_1 \ldots a_k)^m$ such that the permutation is reflected in the corresponding derivation tree in terms of the node labels $\alpha_{i,\text{syn}}$. More precisely, starting at the root following down the unique path, the node labels provide a permutation of $(\alpha_{1,\text{syn}} \ldots \alpha_{k,\text{syn}})^m$ before we hit the node label $\sigma_{k,\text{syn}}$.

## 4 Remarks on ACGs

The formalism of an *abstract categorial grammar (ACG)* (de Groote, 2001) has been introduced with the idea to provide a general framework in terms of linear logic allowing the encoding of existing grammatical models. An ACG distinguishes between an abstract language and an object language each of which is a set of linear lambda terms over some signature. Following the presentation by, e.g., Pogodalla (2004b), an ACG $\mathcal{G}$ defines 1) two sets of typed linear lambda terms, namely, a set $\Lambda_1$ based on the typed constant set $C_1$, and a set $\Lambda_2$ based on the typed constant set $C_2$; 2) a morphism $\mathcal{L} : \Lambda_1 \to \Lambda_2$; and 3) and a distinguished type $\boldsymbol{S}$. The abstract and the object language of $\mathcal{G}$ are defined as $A(\mathcal{G}) = \{t \in \Lambda_1 \,|\, t : \boldsymbol{S}\}$ and $O(\mathcal{G}) = \{\mathcal{L}(t) \in \Lambda_2 \,|\, t \in A(\mathcal{G})\}$, respectively.

In this way the ACG-framework, in particular, provides a logical setting in which an abstract language can be used as a specification of the derivation set of a grammar instantiation of some grammar formalism, and by applying two different morphisms to the abstract language, we can "simultaneously" obtain a syntactic object language and a semantic object language.

The *order* of an ACG is the maximal order of the types assigned to the abstract constants from $C_1$.[7] As demonstrated by de Groote (2002), each classical TAG can be encoded as second-order ACGs realizing the object language as the set of derived trees, and the set of derived strings can be extracted from that object language by composing the first second-order ACG with a second one. Salvati (2007) has shown more generally, that second-order ACGs, where the object language is realized over a string signature, derive exactly the string languages generated by, e.g., set-local multicomponent TAGs. Kanazawa (2010) has shown, that second-order ACGs, where the object language is realized over a tree signature, derive exactly the string languages generated by, e.g., *context-free graph grammars* (Bauderon and Courcelle, 1987).

Of course, the notion of derivation (trees) and derived trees we have informally presented in the previous sections is essentially one making use of binding and abstraction. Recasting the above notation into the ACG-framework provides one way of analyzing the properties of coTAGs, in particular, from the perspective of an comparison to other formalisms and approaches which fit into the ACG-shape.

Closer inspection reveals Barker's accompanying notion of derivation tree to be in line with the abstract language of the TAG-guided, ACG-based semantic analysis of Pogodalla (2004a; 2007), where the abstract language is a set of lambda terms containing third-order constants.

It is easy to see that, although as far as the syntactic component is concerned, coTAGs are strongly equivalent to TAGs, this is only because the syntactic interpretation of the higher-order derivations "goes through" the second-order derivation trees of TAGs. Once we turn to the domain of meanings, where the higher-order derivation terms are used essentially, we obtain a greater generative capacity. We have exemplified this above with regard to the language MIX$(k)$, for $k \geq 3$. Note that MIX$(3)$ is not a tree adjoining language (Kanazawa and Salvati, 2012), and that for $k \geq 4$, MIX$(k)$ is conjectured to be beyond the scope of set-local multicomponent TAGs.

## 5 Conclusion

As already mentioned in the introduction, Barker intends to provide an "alternative conceptualization" of TAG semantic computation. An earlier approach to TAG semantics in explicit terms of STAGs has been worked out by Nesson and Shieber (2007, and follow-up work). But in contrast to Barker's presentation, their initial version does not exceed the expressivity of simple STAGs (Shieber, 1994; Shieber, 2006), where also the second component does not exceed the (weak) generative capacity of TAGs.

The difference from coTAGs results from the fact that the semantic component may constitute a tree-local multicomponent TAG, which allows Nesson and Shieber to lexically represent, e.g., a scope-taking quantifier as a pair of an elementary auxiliary tree and an elementary substitution tree. Sticking to the realm of simple STAGs implies that in the context of nested quantifiers and *inverse linking*, when there are more than two quantifiers involved, not all logically possible readings are derivable. Nesson and Shieber discuss this point in the context of the example *two politicians spy on someone from every city*. Their approach delivers four possible readings. A particular fifth reading, namely, the case of scope ordering *every* > *two* > *some* is not available, which some authors, among them Barker, think is a possible reading, albeit hard to process.

Introducing *limited delay vector TAGs (LDV-TAGs)*, Nesson (2009) suggests a modification of the earlier STAG approach which also allows the derivation of the "fifth reading." If for any given LDV-TAG there is no hard upper bound on the degree of delay and on the number of multiple adjunctions that can take place at a single node, our argument for coTAG expressivity can be straightforwardly adapted to LDV-TAGs. That is to say,

[7]The order an atomic type is 1. For two types $\boldsymbol{\zeta}$ and $\boldsymbol{\eta}$, the order of $(\boldsymbol{\zeta}\boldsymbol{\eta})$ is defined as the maximum of the order of $\boldsymbol{\zeta}$ increased by 1 and the order of $\boldsymbol{\eta}$.

we can write an LDV-TAG whose set of derivation trees and set of derived meanings is, up to a homomorphism, MIX($k$) for arbitrary, but fixed $k \geq 1$.

Arbitrary delay allows for a qualitative increase in the relation-generating power of synchronous TAGs, demonstrated above in terms of coTAGs and the operation of cosubstitution. Recasting this in terms of ACGs allows for a further characterization, which makes clear the increase in derivational generative capacity by moving from second-order abstract constants to third-order ones.

## References

Chris Barker. 2010. Cosubstitution, derivational locality, and quantifier scope. In *Proceedings of the Tenth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+10),* New Haven, CT, pages 135–142.

Michel Bauderon and Bruno Courcelle. 1987. Graph expressions and graph rewriting. *Mathematical Systems Theory*, 20:83–127.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, Toulouse, pages 252–259. ACL.

Philippe de Groote. 2002. Tree-adjoining grammars as abstract categorial grammars. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+6),* Venezia, pages 145–150.

Aravind K. Joshi and Yves Schabes. 1997. Tree adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, Heidelberg.

Laura Kallmeyer and Maribel Romero. 2004. Ltag semantics with semantic unification. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+7),* Vancouver, BC, pages 155–162.

Makoto Kanazawa and Sylvain Salvati. 2012. MIX is not a tree-adjoining language. In *50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, Jeju Island. ACL.

Makoto Kanazawa. 2010. Second-order abstract categorial grammars as hyperedge replacement grammars. *Journal of Logic, Language and Information*, 19:137–161.

Rebecca Nesson and Stuart M. Shieber. 2007. Simpler TAG semantics through synchronization. In Wintner (2007), pages 129–142.

Rebecca Nesson. 2009. *Synchronous and Multicomponent Tree-Adjoining Grammars. Complexity, Algorithms and Linguistic Applications*. Ph.D. thesis, Havard University, Cambridge, MA.

Sylvain Pogodalla. 2004a. Computing semantic representation. Towards ACG abstract terms as derivation trees. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+7)*, Vancouver, BC, pages 64–71.

Sylvain Pogodalla. 2004b. Using and extending ACG technology. Endowing categorial grammars with an underspecified semantic representation. In *Proceedings of Categorial Grammars 2004*, Montpellier, pages 197–209.

Sylvain Pogodalla. 2007. Ambiguïté de portée et approche fonctionnelle des grammaires d'arbres adjoints. In *Traitement Automatique des Langues Naturelles (TALN 2007)*, Toulouse. 10 pages.

Sylvain Salvati. 2007. Encoding second order string ACG with deterministic tree walking transducers. In Wintner (2007), pages 143–156.

Stuart M. Shieber and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING '90)*, Helsinki, volume 3, pages 253–258.

Stuart M. Shieber. 1994. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10:371–385.

Stuart M. Shieber. 2006. Unifying synchronous tree adjoining grammars and tree transducers via bimorphisms. In *11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006),* Trento, pages 377–384. ACL.

Shuly Wintner, editor. 2007. *Proceedings of FG 2006: The 11th Conference on Formal Grammar*. CSLI Publications, Stanford, CA.