Constituency and Dependency Relationship from a Tree Adjoining Grammar and Abstract Categorial Grammar Perspective
Aleksandre Maskharashvili, Sylvain Pogodalla

To cite this version:
Aleksandre Maskharashvili, Sylvain Pogodalla. Constituency and Dependency Relationship from a Tree Adjoining Grammar and Abstract Categorial Grammar Perspective. Proceedings of the Sixth International Joint Conference on Natural Language Processing (IJCNLP 2013), The Asian Federation of Natural Language Processing, Oct 2013, Nagoya, Japan. pp.1257-1263. hal-00868363

HAL Id: hal-00868363
https://inria.hal.science/hal-00868363v1
Submitted on 1 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Constituency and Dependency Relationship from a Tree Adjoining Grammar and Abstract Categorial Grammar Perspective

Aleksandre Maskharashvili and Sylvain Pogodalla
INRIA, 54600 Villers-lès-Nancy, France
LORIA, UMR 7503, 54506 Vandoeuvre-lès-Nancy, France
Sylvain.Pogodalla@inria.fr
Aleksandre.Maskharashvili@inria.fr

Abstract
This paper gives an Abstract Categorial Grammar (ACG) account of (Kallmeyer and Kuhlmann, 2012)’s process of transformation of the derivation trees of Tree Adjoining Grammar (TAG) into dependency trees. We make explicit how the requirement of keeping a direct interpretation of dependency trees into strings results into lexical ambiguity. Since the ACG framework has already been used to provide a logical semantics from TAG derivation trees, we have a unified picture where derivation trees and dependency trees are related but independent equivalent ways to account for the same surface–meaning relation.

1 Introduction
Tree Adjoining Grammars (TAG) (Joshi et al., 1975; Joshi and Schabes, 1997) is a tree grammar formalism relying on two operations between trees: substitution and adjunction. In addition to the tree generated by a sequence of such operations, there is a derivation tree which records this sequence. Derivation trees soon appeared as good candidates to encode semantic-like relations between the elementary trees they glue together. However, some mismatch between these trees and the relative scoping of logical connectives and relational symbols, or between these trees and the dependency relations, have been observed. Solving these problems often leads to modifications of derivation tree structures (Schabes and Shieber, 1994; Kallmeyer, 2002; Joshi et al., 2003; Rambow et al., 2001; Chen-Main and Joshi, To appear).

While alternative proposals have succeeded in linking derivation trees to semantic representations using unification (Kallmeyer and Romero, 2004; Kallmeyer and Romero, 2007) or using an encoding (Pogodalla, 2004; Pogodalla, 2009) of TAG into the ACG framework (de Groote, 2001), only recently (Kallmeyer and Kuhlmann, 2012) has proposed a transformation from standard derivation trees to dependency trees.

This paper provides an ACG perspective on this transformation. The goal is twofold. First, it exhibits the underlying lexical blow up of the yield functions associated with the elementary trees in (Kallmeyer and Kuhlmann, 2012). Second, using the same framework as (Pogodalla, 2004; Pogodalla, 2009) allows us to have a shared perspective on a phrase-structure architecture and a dependency one and an equivalence on the surface-meaning relation they define.

2 Abstract Categorial Grammars
ACGs provide a framework in which several grammatical formalisms may be encoded (de Groote and Pogodalla, 2004). They generate languages of linear λ-terms, which generalize both string and tree languages. A key feature is to provide the user direct control over the parse structures of the grammar, the abstract language, which allows several grammatical formalisms to be defined in terms of ACG, in particular TAG (de Groote, 2002). We refer the reader to (de Groote, 2001; Pogodalla, 2009) for the details and introduce here only few relevant definitions and notations.

Definition. A higher-order linear signature is defined to be a triple $\Sigma = (A, C, \tau)$, where:
- $A$ is a finite set of atomic types (also noted $A_\Sigma$),
- $C$ is a finite set of constants (also noted $C_\Sigma$),
- and $\tau$ is a mapping from $C$ to $\mathcal{T}_A$ the set of types built on $A$: $\mathcal{T}_A := A | \mathcal{T}_A \rightarrow \mathcal{T}_A$ (also noted $\mathcal{T}_\Sigma$).
A higher-order linear signature will also be called a vocabulary. \( \Lambda(\Sigma) \) is the set of \( \lambda \)-terms built on \( \Sigma \), and for \( t \in \Lambda(\Sigma) \) and \( \alpha \in \mathcal{F}_\Sigma \) such that \( t \) has type \( \alpha \), we note \( t :_\Sigma \alpha \) (the \( \Sigma \) subscript is omitted when it is obvious from the context).

**Definition.** An abstract categorial grammar is a quadruple \( \mathcal{G} = (\Sigma, \Xi, \mathcal{L}, s) \) where:

1. \( \Sigma \) and \( \Xi \) are two higher-order linear signatures, which are called the abstract vocabulary and the object vocabulary, respectively;
2. \( \mathcal{L} : \Sigma \rightarrow \Xi \) is a lexicon from the abstract vocabulary to the object vocabulary. It is a homomorphism\(^1\) that maps types and terms built on \( \Sigma \) to types and terms built on \( \Xi \). We note \( t :=_\mathcal{G} u \) if \( \mathcal{L}(t) = u \) and omit the \( \mathcal{G} \) subscript if obvious from the context.
3. \( s \in \mathcal{F}_\Sigma \) is a type of the abstract vocabulary, which is called the distinguished type of the grammar.

**Definition.** The abstract language of an ACG \( \mathcal{G} = (\Sigma, \Xi, \mathcal{L}, s) \) is \( \mathcal{A}(\mathcal{G}) = \{ t \in \Lambda(\Sigma) \mid t :_\Sigma s \} \)

The object language of the grammar \( \mathcal{O}(\mathcal{G}) = \{ t \in \Lambda(\Xi) \mid \exists u \in \mathcal{A}(\mathcal{G}). t = \mathcal{L}_G(u) \} \)

Since there is no structural difference between the abstract and the object vocabulary as they both are higher-order signatures, ACGs can be combined in different ways. Either by having a same abstract vocabulary shared by several ACGs in order to make two object terms (for instance a string and a logical formula) share the same underlying structure as \( \mathcal{G}_{\text{d-ed trees}} \) and \( \mathcal{G}_{\text{Log}} \) in Fig. 1. Or by making the abstract vocabulary of an ACG the object vocabulary of another ACG, allowing the latter to control the admissible structures of the former, as \( \mathcal{G}_{\text{yield}} \) and \( \mathcal{G}_{\text{d-ed trees}} \) in Fig. 1.

### 3 TAG as ACG

As Fig. 1 shows, the encoding of TAG into ACG uses two ACGs \( \mathcal{G}_{\text{d-ed trees}} = (\Sigma_{\text{d-ed}}, \Sigma_{\text{trees}}, \mathcal{L}_{\text{d-ed trees}}, s) \) and \( \mathcal{G}_{\text{yield}} = (\Sigma_{\text{trees}}, \Sigma_{\text{string}}, \mathcal{L}_{\text{yield}}, \tau) \). We exemplify the encoding\(^2\) of a TAG analyzing (1)\(^3\)

\[
(1) \text{ John Bill claims Mary seems to love }
\]

This sentence is usually analyzed in TAG with a derivation tree where the to love component scopes over all the other arguments, and where claims and seems are unrelated, as Fig. 2(a) shows.

The three higher-order signatures are:

- \( \Sigma_{\text{d-ed}} \): Its atomic types include \( s, \text{vp, np, sA, vA} \ldots \) where the \( X \) types stand for the categories \( X \) of the nodes where a substitution can occur while the \( \lambda A \) types stand for the categories \( X \) of the nodes where an adjunction can occur. For each elementary tree \( \gamma_{\text{aux, empty}} \) it contains a constant \( C_{\text{aux, empty}} \) whose type is based on the adjunction and substitution sites as Table 1 shows. It additionally contains constants \( I_X : X_A \) that are meant to provide a fake auxiliary tree on adjunction sites where no adjunction actually takes place in a TAG derivation.

- \( \Sigma_{\text{trees}} \): Its unique atomic type is \( \tau \) the type of trees. Then, for any \( X \) of arity \( n \) belonging to the ranked alphabet describing the elementary trees of the TAG, we have a constant \( X_n : \tau \rightarrow \tau \ldots \rightarrow \tau \rightarrow \tau \)

- \( \Sigma_{\text{string}} \): Its unique atomic type is \( \sigma \) the type of strings. The constants are the terminal symbols of the TAG (with type \( \sigma \)), the concatenation \( + : \sigma \rightarrow \sigma \rightarrow \sigma \) and the empty string \( \varepsilon : \sigma \).

Table 1 illustrates \( \mathcal{L}_{\text{d-ed trees}} \): \( \mathcal{L}_{\text{yield}} \) is defined as follows:

- \( \mathcal{L}_{\text{yield}}(\tau) = \sigma \)
- for \( n > 0 \), \( \mathcal{L}_{\text{yield}}(X_n) = \lambda x_1 \cdots x_n.x_1 + \cdots + x_n \)

\(^1\) In addition to defining \( \mathcal{L} \) on the atomic types and on the constants of \( \Sigma \), we have:
- if \( \alpha \rightarrow_\Sigma \beta \) then \( \mathcal{L}(\alpha \rightarrow_\Sigma \beta) = \mathcal{L}(\alpha) \rightarrow \mathcal{L}(\beta) \)
- if \( x \in \Lambda(\Sigma) \) (resp. \( \lambda x.t \in \Lambda(\Sigma) \)) then \( \mathcal{L}(x) = x \) (resp. \( \mathcal{L}(\lambda x.t) = \lambda x.\mathcal{L}(t) \) and \( \mathcal{L}(t u) = \mathcal{L}(t) \mathcal{L}(u) \)) with the proviso that for any constant \( c :_\Sigma \alpha \) of \( \Sigma \) we have \( \mathcal{L}(c) : \mathcal{L}(\alpha) \).

\(^2\) We refer the reader to (Pogodalla, 2009) for the details.

\(^3\) The TAG literature typically uses this example, and (Kallimeyer and Kuhlmann, 2012) as well, to show the mismatch between the derivation trees and the expected semantics and the relative scopes of the predicates.

\(^4\) With \( \mathcal{L}_{\text{d-ed trees}}(X_A) = \tau \rightarrow \tau \) and for any other type \( X, \mathcal{L}_{\text{d-ed trees}}(X_A) = \tau \).
• for \( n = 0 \), \( X_0 : \tau \) represents a terminal symbol and \( \mathcal{L}^\text{yield}(X_0) = X \).

Then, the derivation tree, the derived tree, and the yield of Fig. 2 are represented by:

\[
\begin{align*}
t_0 &= C_{\text{to love}}(C_{\text{claims}} I_{\text{S}} C_{\text{Bill}})(C_{\text{seems}} I_{\text{VP}}) C_{\text{Mary}} C_{\text{John}} \\
\mathcal{L}_{\text{d-ed trees}}(t_0) &= s_2 (np_1 \text{John}) (s_2 (np_1 \text{Bill}) (vp_2 \text{claims} (s_2 (np_1 \text{Mary}) (vp_2 \text{seems} (vp_1 \text{to love})))) \\
\mathcal{L}^\text{yield}(\mathcal{L}_{\text{d-ed trees}}(t_0)) &= \text{John} + \text{Bill} + \text{claims} + \text{Mary} + \text{seems} + \text{to love}
\end{align*}
\]

Moreover, in case an initial tree accepts several adjunction of CTAs, (Kallmeyer and Kuhlmann, 2012) hypothesizes that the farther from the head a CTA is, the higher it is in the dependency tree. In the case of *to love*, the \( s \) node is farther from the head than the \( vp \) node. Therefore any adjunction on the \( s \) node (e.g. *claims*) should be higher than the one on the \( vp \) node (e.g. *seems*) in the dependency tree. We represent the dependency tree for (1) as \( t'_0 = d_{\text{claims}} d_{\text{Bill}} (d_{\text{seems}}(d_{\text{to love}} d_{\text{John}} d_{\text{Mary}})). \)

In order to do such reversing operations, (Kallmeyer and Kuhlmann, 2012) uses Macro Tree Transducers (MTTs) (Engelfriet and Vogler, 1985). Note that the MTTs they use are linear, i.e. non-copying. It means that any node of an input tree cannot be translated more than once. (Yoshinaka, 2006) has shown how to encode such MTTs as the composition \( \mathcal{G}' \circ \mathcal{G}^{-1} \) of two ACGs, and we will use a very similar construct.

### 4.2 The Yield Functions

(Kallmeyer and Kuhlmann, 2012) adds to the transformation from derivation trees to dependency trees the additional constraint that the string associated with a dependency structure is computed directly from the latter, without any reference to the derivation tree. To achieve this, they use two distinct yield functions: \( \text{yield}_{\text{d-ed}} \) from derivation trees to strings, and \( \text{yield}_{\text{dep}} \) from dependency trees to strings.

Let us imagine an initial tree \( \gamma_i \) and an auxiliary tree \( \gamma_a \) with no substitution nodes. The yield of the derived tree resulting from the operations of the derivation tree \( \gamma \) of Fig. 3 defined in (Kallmeyer and Kuhlmann, 2012) is such that

\[
\begin{align*}
\text{yield}_{\text{d-ed}}(\gamma) &= a_1 + w_1 + a_2 + w_2 + a_3 \\
&= (\text{yield}_{\text{d-ed}}(\gamma_i)) (\text{yield}_{\text{d-ed}}(\gamma_a)) \\
&= (\lambda(x_1, x_2) a_1 + x_1 + a_2 + x_2 + a_3) (w_1, w_2)
\end{align*}
\]

where \( \langle x, y \rangle \) denotes a tuple of strings.

Because of the adjunction, the corresponding dependency structure has a reverse order \( \gamma' = \gamma_a(\gamma_i) \), the requirement on \( \text{yield}_{\text{dep}} \) imposes that

\[
\begin{align*}
\text{yield}_{\text{dep}}(\gamma') &= a_1 + w_1 + a_2 + w_2 + a_3 \\
&= (\text{yield}_{\text{dep}}(\gamma_{a})) (\text{yield}_{\text{dep}}(\gamma_i)) \\
&= (\lambda(x_1, x_2, x_3) a_1 + x_1 + x_2 + w_2 + x_3) (a_1, a_2, a_3)
\end{align*}
\]

In the interpretation of derivation trees as strings, initial trees (with no substitution nodes)

![Figure 2: John Bill claims Mary seems to love](image)
are interpreted as functions from tuples of strings into strings, and auxiliary trees as tuples of strings. The interpretation of dependency trees as strings leads us to interpret initial trees as tuples of strings and auxiliary trees as function from tuples of strings to strings.

![Figure 3: Yield from derivation trees](image)

Figure 3: Yield from derivation trees

Indeed, an initial tree can have several adjunction sites. In this case, to be ready for another adjunction after a first one, the first result itself should be a tuple of strings. So an initial tree (with no substitution nodes) with \( n \) adjunction sites is interpreted as a \((2n + 1)\)-tuple of strings. Accordingly, depending on the location where it can adjoin, an auxiliary tree is interpreted as a function from \((2k + 1)\)-tuples of strings to \((2k - 1)\)-tuples of strings.

Taking into account that to model trees having the substitution nodes is then just a matter of adding \( k \) string parameters where \( k \) is the number of substitution nodes in a tree. Then using the interpretation:

\[
\begin{align*}
\text{yield}_{\text{dep}}(d_{\text{to love}}) &= \lambda x_{11}. x_{21}. (x_{11}, x_{21}, \text{to love}, \varepsilon, \varepsilon) \\
\text{yield}_{\text{dep}}(d_{\text{seems}}) &= \lambda (x_{11}, x_{12}, x_{13}, x_{14}, x_{15}). \\
&\quad (x_{11}, x_{12} + \text{seems} + x_{13}x_{14} + x_{15}) \\
\text{yield}_{\text{dep}}(d_{\text{claims}}) &= \lambda x_{21}. (x_{11}, x_{13}, x_{14}) \\
&\quad (x_{11} + x_{21} + \text{claims} + x_{14} + x_{13})
\end{align*}
\]

we can check that

\[
\text{yield}_{\text{dep}}(d_{\text{to love}}) (d_{\text{seems}}(d_{\text{to love}} d_{\text{Mary}} d_{\text{John}})) = (\text{John } + \text{Bill } + \text{Mary seems } + \text{to love})
\]

**Remark.** The given interpretation of \(d_{\text{to love}}\) is only valid for structures reflecting adjunctions both on the \(S\) node and on the \(VP\) node of \(\gamma_{\text{to love}}\). So actually, an initial tree such as \(\gamma_{\text{to love}}\) yields four interpretations: one with the two adjunctions (5-tuple), two with one adjunction either on the \(VP\) node or on the \(S\) node (3-tuple), and one with no adjunction (1-tuple). The two first cases correspond to the sentences (2a) and (2b). Accordingly, we need multiple interpretations for the auxiliary trees, for instance for the two occurrences of \(\gamma_{\text{to love}}\) in (3) where the yield of the last one \(\text{yield}_{\text{dep}}(d_{\text{seems}})\) maps a 5-tuple to a 3-tuple, and the yield of the first one maps a 3-tuple to a 3-tuple. And \(\text{yield}_{\text{dep}}(d_{\text{claims}})\) maps a 3-tuple to a 1-tuple of strings. We will mimic this behavior by introducing as many different non-terminal symbols for the dependency structures in our ACG setting.

(2) a. John Bill claims Mary seems to love
    b. John Mary seems to love

(3) John Bill seems to claim Mary seems to love

**Remark.** Were we not interested in the yields but only in the dependency structures, we wouldn’t have to manage this ambiguity. This is true both for (Kallmeyer and Kuhlmann, 2012)’s approach and ours. But as we have here a unified framework for the two-step process they propose, this lexical blow up will result in a multiplicity of types as Section 5 shows.

\(^{3}\) As the two other ones are not correct English sentences, we can rule them out. However, from a general perspective, we should take such cases into account.
5 Disambiguated Derivation Trees

In order to encode the MTT acting on derivation trees, we introduce a new abstract vocabulary $\Sigma_{\text{der}}$ for disambiguated derivation trees as in (Yoshinaka, 2006). Instead of having only one constant for each initial tree as in $\Sigma_{\text{der}}$, we have as many of them as adjunction combinations. For instance, $\gamma_{\text{to love}}$ gives rise to the several constants in $\Sigma_{\text{der}}$: $C_{\text{to love}}^{11}$, $C_{\text{to love}}^{10}$, $C_{\text{to love}}^{01}$, $C_{\text{to love}}^{00}$. Here, $C_{\text{to love}}^{11}$ is used to model sentences where both adjunctions are performed into $\gamma_{\text{to love}}$. $C_{\text{to love}}^{10}$ and $C_{\text{to love}}^{01}$ are used for sentences where only one adjunction occurs. $C_{\text{to love}}^{00}$ is used when no adjunction occurs. This really mimics (Yoshinaka, 2006)’s encoding of (Kallmeyer and Kuhlmann, 2012) MTT rules: 

\[
\langle q_0, C_{\text{to love}}(x_1, x_2, x_3, x_4) \rangle \rightarrow \langle q_2, x_1 \rangle (q_4, x_4) \langle d_{\text{to love}}(q_1, x_1), q_3, x_3) \rangle \\
\langle q_0, C_{\text{to love}}(x_1, x_2, x_3) \rangle \rightarrow \langle q_2, x_1 \rangle \langle d_{\text{to love}}(q_1, x_1), q_3, x_3) \rangle \\
\langle q_0, C_{\text{to love}}(x_1, x_2) \rangle \rightarrow \langle q_4, x_1 \rangle \langle d_{\text{to love}}(q_1, x_1), q_3, x_3) \rangle \\
\langle q_0, C_{\text{to love}}(x_1) \rangle \rightarrow \langle d_{\text{to love}}(q_1, x_1), q_3, x_3) \rangle
\]

where the states $q_0$, $q_1$, $q_2$, $q_3$ and $q_4$ are given the names $s$, $np$, $s_3^{31}$, $np$, and $vp_3^{11}$ respectively.

Moreover, $s_3^{31}$, $vp_3^{11}$, ... $vp_3^{2(n+1)2(n−1)}$ are designed in order to indicate that a given adjunction has $n$ adjunctions above it (i.e. which scope over it). The superscripts $(2(n+1))(2(n−1))$ express that an adjunction that has $n$ adjunctions above it is translated as a function that takes a $2(n+1)$-tupple as argument and returns a $2(n−1)$-tupple.

To model auxiliary trees which are CTAs we need a different strategy. For each such adjunction tree $T$ we have two sets in $\Sigma_{\text{der}}$: $S_T$ the set of constants which can be adjoined into initial trees and $S_T^r$ the set of constants which can be adjoined into auxiliary trees.

For instance, $\gamma_{\text{seem}}$ would generate $S_{\text{seem}}^4$ which includes $C_{\text{seem}}^{11}$, $C_{\text{seem}}^{10}$, $C_{\text{seem}}^{01}$, $C_{\text{seem}}^{00}$, $C_{\text{seem}}^{11}$, $C_{\text{seem}}^{10}$, $C_{\text{seem}}^{01}$, $C_{\text{seem}}^{00}$ etc. $C_{\text{seem}}^{00}$ is of type $vp_3^{11}$, which means that it can be adjoined into initial trees which contain $vp_3^{11}$ as its argument type (e.g. $C_{\text{to love}}^{01}$).

$C_{\text{seem}}^{11}$ is of type $s_3^{3−3} \rightarrow vp_3^{3−3} \rightarrow vp_3^{31}$. It means it expects two adjunctions at its $s$ and $vp$ nodes respectively and returns back a term of type $vp_3^{31}$ (as in John claims to appear to seem to love Mary). Here, $s_3^{3−3}$ and $vp_3^{3−3}$ are types used for modeling adjunction on adjuctions.

When an auxiliary tree is adjoined into another auxiliary tree as in (3), we do not allow the former to modify the tupleness of the latter.

Now it is easy to define $L_{\text{der}}$ from $\Sigma_{\text{der}}$ to $\Sigma_{\text{der}}$. It maps every type $X \in \Sigma_{\text{der}}$ to $X \in \Sigma_{\text{der}}$ and every $X^N_A$ to $X_A$; types without numbers are mapped to themselves, i.e. $s$ to $s$, $np$ to $np$, etc. Moreover, the different versions of some constant, that were introduced in order to extract the yield, are translated using only one constant and fake adjunctions. For instance:

\[
L_{\text{der}}(C_{\text{to love}}^{11}) = C_{\text{to love}} \\
L_{\text{der}}(C_{\text{to love}}^{10}) = \lambda x sos.C_{\text{to love}} \ s \ o \\
L_{\text{der}}(C_{\text{to love}}^{00}) = C_{\text{to love}} \ s \ vp
\]

6 Encoding a Dependency Grammar

The ACG of (Pogodalla, 2009) mapping TAG derivation trees to logical formulas already encoded some reversal of the predicate-argument structure. Here we map the disambiguated derivation trees to dependency structures. The vocabulary that define these dependency trees is $\Sigma_{\text{dep}}$. It is also designed to allow us to build two lexicons from it to $\Sigma_{\text{string}}$ (to provide a direct yield function) and to $\Sigma_{\text{Log}}$ (to provide a logical semantic representation).

In $\Sigma_{\text{dep}}$ constants are typed as follows: $d_{\text{to love}}^{i} : \tau_{1}^{i} \rightarrow \tau_{1}^{i} \rightarrow \tau_{5}^{i}$. Here, $\tau_{1}^{i}$ is the type into which the $np$ type is translated from disambiguated derivation tree. The superscript 1 indicates that $\tau_{1}^{i}$ will be translated into $1$-tuple into $\Sigma_{\text{string}}$. Now, it is easy to see that in order to translate $C_{\text{to love}}^{10} : s_{3}^{1} \rightarrow np \rightarrow np \rightarrow s$ and $C_{\text{to love}}^{01} : vp_{3}^{1} \rightarrow np \rightarrow np \rightarrow s$, we need to...
have constants like: $d_{\text{to} \text{love}}^3 S : \tau_{\text{np}} \to \tau_{\text{np}} \to \tau$ and
\[ d_{\text{to} \text{love}}^3 V : \tau_{\text{np}} \to \tau_{\text{np}} \to \tau. \]

Moreover, we have constants for adjunction trees, like $d_{\text{seems}}^3 : \tau_3 \to \tau_3$ that will be used in the translation of $C_{\text{seems53}}^1$, and $d_{\text{seems}}^3 : \tau_3 \to \tau_3$ for $C_{\text{seems53}}^{00}$. Furthermore, additional constants are needed to have things correctly typed. For this reason, the constants $d_1^3, d_2^3$ etc. are introduced. Each $d_{2n+1} \to \tau 2n+1 \to \tau 2n-1$.

Finally, non-CTAs like $n_A, n_A^1, V_A$ and $S_A$ are translated as $\tau_{n_A}^3, \tau_{n_A}^2, \tau_{n_A}^1$, and $\tau_{S_A}^3$ respectively. A superscript 2 indicates that they are modeled as 2-tuples in $\Sigma_{\text{adj}}$.

Now we can define $L_{\text{dep}}$, the lexicon from $\Sigma_{\text{deriv}}$ to $\Sigma_{\text{dep}}$ translating disambiguated derivation trees into dependency trees:

\[
L_{\text{dep}}(S) = \tau_1 \\
L_{\text{dep}}(n_p) = \tau_{\text{np}} \\
L_{\text{dep}}(X_A^2) = \tau_{2n+1} \to \tau_{2n-1} \\
L_{\text{dep}}(d_{\text{to} \text{love}}^3) = \lambda S V s o.S(V(d_{\text{to} \text{love}}^3 s o)) \\
L_{\text{dep}}(d_{\text{to} \text{love}}^1) = \lambda S V s o.V(d_{\text{to} \text{love}}^3 s o) \\
L_{\text{dep}}(c_{\text{seems53}}^1) = \lambda x d_{\text{seems53}}^3 x \\
L_{\text{dep}}(c_{\text{seems53}}^0) = \lambda x d_{\text{seems53}}^3 x \\
L_{\text{dep}}(c_{\text{seems53}}^{00}) = \lambda x d_{\text{seems53}}^3 x \\
L_{\text{dep}}(\text{love}) = \lambda x d_{\text{seems53}}^3 x \\
L_{\text{dep}}(\text{bill}) = \lambda x d_{\text{seems53}}^3 x \\
L_{\text{dep}}(\text{john}) = \lambda x d_{\text{seems53}}^3 x.
\]

Furthermore, we describe $\Sigma_{\text{log}}$ 7 and define two lexicons: $L_{\text{dep yield}} : \Sigma_{\text{dep}} \to \Sigma_{\text{string}}$ and $L_{\text{dep log}} : \Sigma_{\text{dep}} \to \Sigma_{\text{log}}$. Table 2 provides examples of these two translations.

$L_{\text{dep yield}}$: It translates any atomic type $\tau^n$ or $\tau^n_\text{S}$ with $X \in \{n_A, n_A^1, \ldots\}$ as a n-tuple of string

\[ (\sigma \to \sigma \cdots \to \sigma) \to \sigma. \]

$\Sigma_{\text{log}}$: Its atomic types are $e$ and $t$ and we have the constants: $\text{john, mary, bill}$ of type $e$, the constant $\text{love}$ of type $e \to e \to t$, the constant $\text{claim}$ of type $e \to t \to t$ and the constant $\text{seem}$ of type $t \to t$.

$L_{\text{dep log}}$: Each $\tau_3^{2(n+1)}$ is mapped to $t$, $\tau_{\text{np}}^3$ is mapped to $(e \to t) \to \tau$, $\tau_{n_A}^2$ is mapped to $(e \to t) \to (e \to t) \to t$. The types

7We refer the reader to (Pogodalla, 2009) for the details.

8We encode a n-tuple $\langle M_1, \ldots, M_n \rangle$ as $\lambda f. f M_1 M_2 \ldots M_n$, where each $M_i$ has type $\sigma$.

of non-complement-taking verbal or sentential adjunctions $\tau_{\text{np}}^3$ and $\tau_{\text{S}}^3$ are translated as $t \to t$.

Let us show for the sentence (1) how the ACGs defined above work with the data provided in Table 2. Its representation in $\Sigma_{\text{deriv}}$ is: $T_0 = C_{\text{to} \text{love}}^{11} (C_{\text{claims}31} C_{\text{Bill}}) C_{\text{seems}53} C_{\text{Mary}} C_{\text{John}}$. Then $L_{\text{dep}}(T_0) = t_0$ and $L_{\text{dep}}(T_0) = d_1^{\text{claim}} d_{\text{bill}} (d_{\text{seems}53} (d_{\text{to} \text{love}}^3 \text{Mary} \text{John})) = t_0$ and finally $L_{\text{dep yield}}(t_0) = L_{\text{yield}}(L_{\text{d-ed tree}}(t_0)) = \lambda f. (\text{John} + (\text{Bill} + (\text{claims} + ((\text{seems} + \text{to} \text{love}) + \epsilon) + \epsilon)) + \epsilon)$. And $L_{\text{dep log}}(t_0) = \text{claim bill (seem love john mary)}$.

7 Conclusion

In this paper, we have given an ACG perspective on the transformation of the derivation trees of TAG to the dependency trees proposed in (Kallmeyer and Kuhlmann, 2012). Figure 4 illustrates the architecture we propose. This transformation is a two-step process using first a macro-tree transduction then interpretation of dependency trees as (tuples of) strings. It was known from (Yoshinaka, 2006) how to encode a macro-tree transducer into a $\mathcal{D}_{\text{dep}} \circ \mathcal{D}_{\text{lib}}^{-1}$ ACG composition. Dealing with typed trees to represent derivation trees allows us to provide a meaningful (wrt. the TAG formalism) abstract vocabulary $\Sigma_{\text{deriv}}$ encoding this macro-tree transducer. The encoding of the second step then made explicit the lexical blow up for the interpretation of the functional symbols of the dependency trees in (Kallmeyer and Kuhlmann, 2012)'s construct. It also provides a push out (in the categorical sense) of the two morphisms from the disambiguated derivation trees to the derived trees and to the dependency trees. The diagram is completed with the yield function from the derived trees and from the dependency trees to the string vocabulary.

Finally, under the assumption of (Kallmeyer and Kuhlmann, 2012) of plausible dependency structures, we get two possible grammatical approaches to the surface-semantics relation that are related but independent: it can be equivalently modeled using either a phrase structure or a dependency model.
Table 2: Lexicons for yield and semantics from the dependency vocabulary

| \(\Sigma_{\text{der}}\) | \(\Sigma_{\text{tree}}\) | \(\Sigma_{\text{string}}\) |
|-----------------|-----------------|-----------------|
| \(\text{disambiguated derivation trees} \Lambda(\Sigma_{\text{der}}')\) | \(\text{dep. trees} \Lambda(\Sigma_{\text{dep}})\) | \(\text{strings} \Lambda(\Sigma_{\text{string}})\) |
| \(\text{dep. yield} \Lambda(\Sigma_{\text{yield}})\) | \(\text{dep. yield} \Lambda(\Sigma_{\text{yield}})\) | \(\text{logical formulas} \Lambda(\Sigma_{\text{Log}})\) |

Figure 4: General architecture

References

Joan Chen-Main and Aravind K. Joshi. To appear. A dependency perspective on the adequacy of tree local multi-component tree adjoining grammar. *Journal of Logic and Computation*.

Philippe de Groote and Sylvain Pogodalla. 2004. On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *Proceedings of ACL*, pages 148–155.

Philippe de Groote. 2002. Tree-adjointing grammars as abstract categorial grammars. In *Proceedings of TAG*+6, pages 145–150. Università di Venezia.

Joost Engelfriet and Heiko Vogler. 1985. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):71–146.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjointing grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 3, chapter 2. Springer.

Aravind K. Joshi, Leon S. Levy, and Masako Takanashi. 1975. Tree adjoining grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2003. Flexible composition in ltag: Quantifier scope and inverse linking. In Harry Bunt, Ielka van der Sluis, and Roser Morante, editors, *Proceedings of IWCS*–5.

Laura Kallmeyer and Marco Kuhlmann. 2012. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Proceedings of TAG*+11, pages 108–116.

Laura Kallmeyer and Maribel Romero. 2004. Ltag semantics with semantic unification. In *Proceedings of TAG*+7, pages 155–162.

Laura Kallmeyer. 2002. Using an enriched tag derivation structure as basis for semantics. In *Proceedings of TAG*+6.

Sylvain Pogodalla. 2004. Computing Semantic Representation: Towards ACG Abstract Terms as Derivation Trees. In *Proceedings of TAG*+7, pages 64–71. Vancouver, BC, Canada. http://hal.inria.fr/inria-00107768.

Sylvain Pogodalla. 2009. Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling. *ESSLLI 2009 Lecture Notes*, Part II. http://hal.inria.fr/hal-00749297.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-Substitution Grammars. *Computational Linguistics*.

Yves Schabes and Stuart M. Shieber. 1994. An alternative conception of tree-adjointing derivation. *Computational Linguistics*, 20(1):91–124.

Ryo Yoshinaka. 2006. Extensions and Restrictions of Abstract Categorial Grammars. Phd, University of Tokyo.