# Two Equivalent Regularizations for Tree Adjoining Grammars

Anna Kasprzik

University of Trier

**Abstract.** In this paper two methods of how to make derivation in a Tree Adjoining Grammar a regular process without loss of expressive power are presented and compared. In a TAG, derivation is based upon the expansion of tree nodes into other trees. One regularization method is based on an algebraic operation called Lifting, while the other exploits an additional spatial dimension by transforming the components of a TAG into three-dimensional trees. The regularized grammars generate two kinds of "encoded" trees, from which the intended ones can be reconstructed by a simple decoding function. We can show the equivalence of these methods by giving a translation between lifted and three-dimensional trees and proving that via this translation it is possible to switch between the encodings without losing the information necessary for the reconstruction of the intended trees.

**Key words:** Tree Adjoining Grammar, Multi-Dimensional Trees, Lifting, Regularization

## 1 Introduction

A Tree Adjoining Grammar (TAG) is a special kind of tree grammar which has been developed by Joshi [6] in connection with studies on the formal treatment of natural languages. Joshi [6] claimed the least class of formal languages containing all natural languages to be situated between the context-free and the context-sensitive languages in the Chomsky Hierarchy, and named it the class of *mildly context-sensitive languages*. The string languages associated with TAGs fulfil all necessary conditions for this class.

As mild context-sensitivity represents a relatively high degree of complexity already, it would be of considerable use if there were a way to simplify derivation without giving up any of the expressive power. In general, regularizing a formalism has the obvious advantage that it makes the whole range of finite-state methods applicable, which is of interest for most areas based on formal language theory, e.g. natural language processing or grammatical inference, especially when objects more complex than strings are involved. As a matter of fact, for TAGs at least two such regularization methods exist, and it is the presentation and comparison of these two approaches that constitute the main part of this work.

## 2 Preliminaries

We presuppose some familiarity with classical formal language theory and trees. The necessary preliminaries can be found for example in [1], [5], and also [8]. We will now give the definition for Tree Adjoining Grammars.

**Definition 1.** *A TAG is a 5-tuple* $\langle \Sigma, N, I, A, S \rangle$*, where* $\Sigma$ *is the (non-ranked)* terminal labeling alphabet*,* $N$ *is the (non-ranked)* nonterminal labeling alphabet *with* $N \cap \Sigma = \emptyset$*,* $S$ *is the start symbol with* $S \in N$*,* $I$ *is a finite set of* initial trees *where the root is labeled with* $S$*, and* $A$ *is a finite set of* auxiliary trees*.*

Nonterminals label inner nodes, terminals label all leaf nodes but one, which is labeled by the nonterminal also labeling the root of the tree. This leaf is referred to as the *foot node*. Initial and auxiliary trees are referred to as *elementary trees*. New trees can be built by *adjunction*: A node in a tree is replaced by an auxiliary tree and the subtree formerly rooted at that node is attached to the foot node of the auxiliary tree.

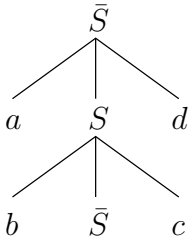
A TAG can be enriched by associating a pair of constraints with every node, stating if adjunction is required or not (*obligatory adjunction (OA) constraint*), and which auxiliary trees may be adjoined at that node (*selective adjunction (SA) constraint*). These constraints obliterate the roles of nonterminal and terminal symbols and the start symbol, and hence the distinction between initial and auxiliary trees as well. Rogers [10] defines *non-strict* TAGs:

**Definition 2.** *A non-strict TAG is a pair* $\langle E, I \rangle$ *where* $E$ *is a finite set of elementary trees in which each node is associated with a label from some alphabet, an SA constraint (a subset of* $E$*), and an OA constraint (Boolean valued).* $I \subseteq E$ *is a distinguished non-empty subset. Every elementary tree has a foot node.*

We will now give an example for a TAG generating the language $a^n b^n c^n d^n$.

*Example 1.* Let $G$ with $G = \langle \{\alpha, \beta\}, \{\alpha\} \rangle$ be a TAG (over the alphabet $\{a, b, c, d, S\}$). The only initial tree $\alpha$ and the only auxiliary tree $\beta$ are given below. Constraints at the inner nodes and the foot node are: $OA = 0$ and $SA = \{\beta\}$ for the ones without a bar, and $OA = 0$ and $SA = \emptyset$ for the ones labeled with '$\bar{S}$'. The bar stands for null adjunction, i.e., no adjunction is allowed at these nodes.

$\alpha =$ 


$\beta =$ 


A derivation for the word *aabbccdd* is shown in Figure 1.

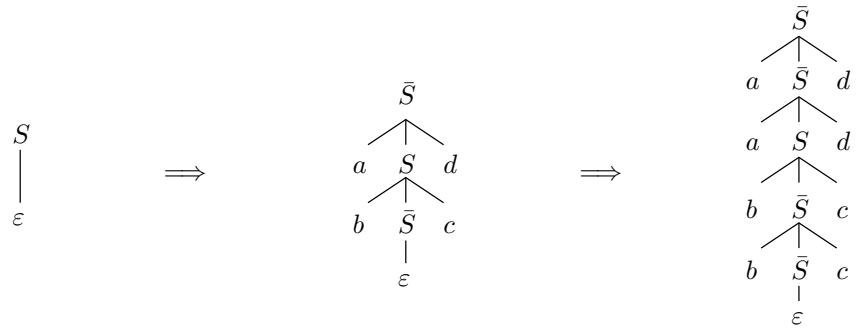



**Fig. 1.** A derivation for the word *aabbccdd*

## 3 Lifted Trees

The technique of Lifting belongs into the realm of treating formal language classes with algebraical means. For the necessary preliminaries concerning universal algebra (esp. many-sorted algebras) and trees as terms, see [2, 3]. For Lifting the notion of *derived alphabets* is essential:

**Definition 3.** *Let $\Sigma$ be a ranked alphabet. The* derived $(\mathbb{N}^* \times \mathbb{N})$-indexed alphabet *of $\Sigma$, denoted by $D(\Sigma)$, is defined as follows. Let, for each $n \geq 0$, $\Sigma'_n = \{f' | f \in \Sigma_n\}$ be a new set of symbols; let for each $n$ and each $i$, $1 \leq i \leq n$, $\pi^n_i$ be a new symbol (the $i$th* projection symbol *of sort $n$); and let, for each $n, k \geq 0$, $c_{n,k}$ be a new symbol (the $(n,k)$th* composition symbol*). Then*

- $D(\Sigma)_{\varepsilon,0} = \Sigma'_0$ ;
- *for $n \geq 1$, $D(\Sigma)_{\varepsilon,n} = \Sigma'_n \cup \{\pi^n_i | 1 \leq i \leq n\}$ ;*
- *for $n, k \geq 0$, $D(\Sigma)_{nk^n,k} = \{c_{n,k}\}$ , and*
- $D(\Sigma)_{w,s} = \emptyset$ *otherwise.*

Note that all operators in $\Sigma$ are treated as constants in $D(\Sigma)$. Lifting a term over some $\Sigma$ just means translating it into a corresponding term over $D(\Sigma)$:

**Definition 4 ([8]).** *Let $\Sigma$ be a ranked alphabet. For $k \geq 0$, $\mathsf{LIFT}^\Sigma_k : T_\Sigma(X_k) \longrightarrow T^k_{D(\Sigma)}$ is defined as follows:*

$$\begin{aligned}
\mathsf{LIFT}^\Sigma_k(x_i) &= \pi^k_i \\
\mathsf{LIFT}^\Sigma_k(f) &= c_{0,k}(f') \text{ for } f \in \Sigma_0 \\
\mathsf{LIFT}^\Sigma_k(f(t_1, \ldots, t_n)) &= c_{n,k}(f', \mathsf{LIFT}^\Sigma_k(t_1), \ldots, \mathsf{LIFT}^\Sigma_k(t_n)) \\
&\qquad \text{for } n \geq 1, f \in \Sigma_n \text{ and } t_1, \ldots, t_n \in T(\Sigma, X_k) \, .
\end{aligned}$$

Lifting has a very useful side effect: If you note a term as a tree and lift it, all inner nodes become leaves. This obviously makes the operation of replacing those nodes by bigger structures a much easier process, to wit, speaking in terms of formal language theory, a regular process. Consequently any context-free

tree grammar (CFTG, based on the rewriting of inner nodes) over a signature $\Sigma$ can be translated into a regular tree grammar (RTG, rewriting of leaves only) over the signature $D(\Sigma)$ by lifting the trees on the right hand sides of the productions and, since all nonterminals have become constants (i.e., leaf labels), by deleting the variables representing daughters on the left hand sides. The intended trees over the original signature can then be reconstructed using the information contained in the "encoded" trees the RTG generates via the following function:

$$\begin{aligned} rec(f') &= f(x_1, \ldots, x_n) \text{ for } f \in \Sigma_n \\ rec(\pi_i^n) &= x_i \\ rec(c(t, t_1, \ldots, t_n)) &= rec(t)[rec(t_1), \ldots, rec(t_n)] \ . \end{aligned}$$

The proof of the following lemma (contained implicitly in [2]) is given in [7]:

**Lemma 1.** *Suppose $\Gamma = (\Sigma, \mathsf{F}, S, X, P)$ is a CFTG and $L(\Gamma)$ the tree language it generates. Then there is a derived regular tree grammar $\Gamma^L = (D(\Sigma), D(\mathsf{F}), S', P^L)$ such that $L(\Gamma)$ is the image of $L(\Gamma^L)$ under the mapping rec.*

Morawietz [8] has collected some properties of trees generated by a lifted CFTG over some signature $D(\Sigma)$: a) All inner nodes are and no leaf is labeled by some composition symbol $c_{n,k}$, b) any node labeled with a symbol in $\Sigma'_n$, $n \geq 1$, is on a leftmost branch, and c) for any node $p$ labeled with some projection symbol $\pi_i^n$ there is a unique node $\mu$ which properly dominates $p$ and whose $i$th sister will eventually evaluate to the value of $\pi_i^n$ under the mapping *rec*. Moreover, $\mu$ will be the first node properly dominating $p$ on a left branch. We will call lifted trees that fulfil condition c) *closed* lifted trees.

Lifting can be used to regularize TAGs as well. However, since TAGs function a little differently from CFTGs (which is linked to the fact that TAG trees are labeled by non-ranked symbols), this is not possible without some transformation. It has been proven by Fujiyoshi and Kasai [4] that every TAG can be translated into a spine grammar, which is a special kind of CFTG, where in the tree on the right-hand side of every production there exists a path from the root to a variable-labeled leaf and every other variable is the child of a node on that path. See [4] for the exact definition, construction, and proof of weak equivalence to TAGs. Their method can be easily adapted to non-strict TAGs as well. Example 2 shows the result of applying the translation of Fujiyoshi and Kasai [4] to the TAG of Example 1, and its lifted version.

*Example 2.* The new CFTG $G' = (\Sigma_0 \cup \Sigma_1 \cup \Sigma_3, \mathsf{F}_0 \cup \mathsf{F}_1 \cup \mathsf{F}_3, S', X, P)$ obtained from the TAG $G$ in Example 1 is defined as follows: $\Sigma_0 = \{\varepsilon, a, b, c, d\}$, $\Sigma_1 = \{s_1\}$, $\Sigma_3 = \{s_3\}$, $X = \{x_1, x_2, x_3\}$, $\mathsf{F}_0 = \{S'\}$, $\mathsf{F}_1 = \{S_1\}$, $\mathsf{F}_3 = \{S_3\}$, and

$$P = \left\{ \begin{array}{c} S' \longrightarrow s_1(\varepsilon) \\ S' \longrightarrow S_1(\varepsilon) \\ S_1(x_1) \longrightarrow s_3(a, s_3(b, s_1(x_1), c), d) \\ S_1(x_1) \longrightarrow s_3(a, S_3(b, s_1(x_1), c), d) \\ S_3(x_1, x_2, x_3) \longrightarrow s_3(a, s_3(b, s_3(x_1, x_2, x_3), c), d) \\ S_3(x_1, x_2, x_3) \longrightarrow s_3(a, S_3(b, s_3(x_1, x_2, x_3), c), d) \end{array} \right\}$$

In lifted form, this grammar then looks like this: $G'^L = (D(\Sigma), D(\mathsf{F}), S'', P^L)$ with $D(\Sigma)_{\varepsilon,0} = \{\varepsilon, a', b', c', d'\}$, $D(\Sigma)_{\varepsilon,1} = \{s'_1\}$, $D(\Sigma)_{\varepsilon,3} = \{s'_3, \pi^3_1, \pi^3_2, \pi^3_3\}$, $D(\Sigma)_{nk^n,k} = \{c\}$, $D(\mathsf{F})_{\varepsilon,0} = \{S''\}$, $D(\mathsf{F})_{\varepsilon,1} = \{S'_1\}$, $D(\mathsf{F})_{\varepsilon,3} = \{S'_3\}$, and

$$P^L = \left\{ \begin{array}{l} S'' \longrightarrow c(s'_1, \varepsilon) \\ S'' \longrightarrow c(S'_1, \varepsilon) \\ S'_1 \longrightarrow c(s'_3, a', (s'_3, b', c(s'_1, \pi_1), c'), d') \\ S'_1 \longrightarrow c(s'_3, a', (S'_3, b', c(s'_1, \pi_1), c'), d') \\ S'_3 \longrightarrow c(s'_3, a', (s'_3, b', c(s'_3, \pi_1, \pi_2, \pi_3), c'), d') \\ S'_3 \longrightarrow c(s'_3, a', (S'_3, b', c(s'_3, \pi_1, \pi_2, \pi_3), c'), d') \end{array} \right\}$$

Figure 2 shows two corresponding trees generated by $G'$ and $G'^L$ – note how the elementary trees of the original TAG are still distinguishable. In fact we can state the following: In a tree generated by a lifted TAG the elementary trees are represented by the tree parts between one composition symbol on a leftmost branch and the next, and the expanded nodes correspond to the mother nodes of these composition symbols. In order to see this, consider the following observations: a) An adjunction in a TAG corresponds to the rewriting of an inner node in the corresponding CFTG, and consequently to the rewriting of a leaf in the corresponding lifted CFTG.[1] b) A composition symbol on a leftmost branch is certain evidence that at this node a production of the lifted CFTG has been applied.[2] It follows from this that the parts separated by composition symbols on leftmost branches must be the elementary components of the original TAG. Foot nodes are represented by nodes whose children except the leftmost are all labeled by projection symbols.

## 4 Three-Dimensional Trees and their Yields

In this section we will consider a method based on a generalization of the concept of trees by Rogers [9, 10]. Starting from ordinary trees based on two-dimensional tree domains Rogers extends the concept both downwards (strings and points) and upwards and defines *labeled multi-dimensional trees*:

**Definition 5.** *Let* ${}^d1$ *be the class of all* $d$*th-order sequences of* 1*s:* ${}^0 1 := \{1\}$*, and* ${}^{n+1}1$ *is the smallest set satisfying (i)* $\langle\rangle \in {}^{n+1}1$*, and (ii) if* $\langle x_1, \ldots, x_l\rangle \in {}^{n+1}1$

[1] Recall Lemma 1. The individual derivation steps also correspond:
**Lemma 2.** *$t'$ is derived in $\Gamma^L$ from $t$ in $k$ steps, i.e., $t \Rightarrow t'$ via the productions $p^L_1, \ldots, p^L_k$ in $P^L$ if and only if there are corresponding productions $p_1, \ldots, p_k$ in $P$ such that $rec(t')$ is derived in $\Gamma$ from $rec(t)$ via those productions.*

[2] This can be explained as follows: Lifting a CFTG comprises lifting all the trees on the righthand sides of the rules. In such a lifted tree all leftmost daughters of any node are labeled with symbols in $\Sigma'_n$ (for some $n$), and never by composition symbols. Consequently, a composition symbol on a leftmost branch indicates that the tree cannot be contained as a whole in the rules of the grammar (i.e., as the right-hand side of a rule starting with "$S' \longrightarrow \ldots$") but that some production must have been applied that licenses the rewriting of a leaf by the subtree now rooted at the node in question (which, however, can contain other rewritings itself).

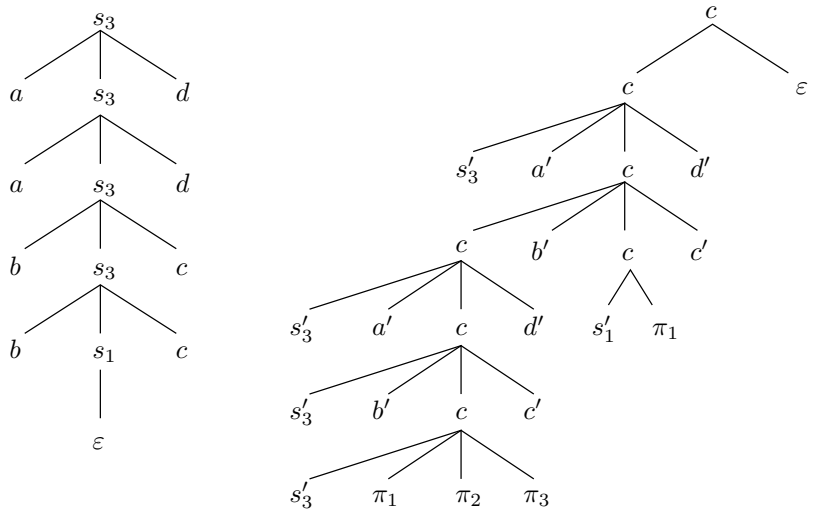



**Fig. 2.** Two trees, generated via corresponding rules of $G'$ and $G'^L$

*and* $y \in {}^n1$*, then* $\langle x_1, \ldots, x_l, y\rangle \in {}^{n+1}1$*. Let* $\mathbb{T}^0 := \{\emptyset, \{1\}\}$ *(point domains). A* $(d{+}1)$*-dimensional tree domain is a set of hereditarily prefix closed* $(d{+}1)$*st-order sequences of* $1$*s, i.e.,* $\mathsf{T} \in \mathbb{T}^{d+1}$ *iff*

- $\mathsf{T} \subseteq {}^{d+1}1$,
- $\forall s, t \in {}^{d+1}1 : s \cdot t \in \mathsf{T} \Rightarrow s \in \mathsf{T}$,
- $\forall s \in {}^{d+1}1 : \{w \in {}^d1 | s \cdot \langle w\rangle \in \mathsf{T}\} \in \mathbb{T}^d$ .

*A* $\Sigma$*-labeled* $\mathsf{T}d$ *(*$d$-dimensional tree*) is a pair* $(T, \tau)$ *where* $T$ *is a* $d$*-dimensional tree domain and* $\tau : T \longrightarrow \Sigma$ *is an assignment of labels in the (non-ranked) alphabet* $\Sigma$ *to nodes in* $T$*. We will denote the class of all* $\Sigma$*-labeled* $\mathsf{T}d$ *as* $\mathbb{T}^d_\Sigma$*.*

Every $d$-dimensional tree can be conceived to be built up from $d$-dimensional *local trees*, that is, trees of depth at most one in their major dimension. Each of these smaller trees consists of a root and an arbitrarily large $(d-1)$-dimensional "child tree" consisting of the root's children. Composite trees can then be built from local ones by identifying the root of one local tree with a node in the child tree of another (see Figure 3 for an illustration). Rogers [10] also defines automata for multi-dimensional trees based on the notion of local trees.

Perhaps the most important concept Rogers adapts to multi-dimensionality is that of the *yield* of a tree. The yield of a two-dimensional tree is the string formed by its leaf labels. In Rogers' words, it is a projection of the tree onto the next lower level, i.e., its dimensions are reduced by one. $d$-dimensional trees with $d \geq 3$ have several yields, one for each dimension that is taken away, down to the one-dimensional string yield. Note that when taking the yield of a tree with $d \geq 3$, some thought has to go into the question of how to interweave the child trees of its local components to form a coherent $(d-1)$-dimensional tree, since

there are often several possibilities. Rogers solves this by a construction quite similar to foot nodes in TAGs. See [10] for the exact definition.

Rogers [10] has established a link between T3 trees and TAGs – he has proven the equivalence of T3 recognizing automata and non-strict TAGs:

**Theorem 1.** *A set of $\Sigma$-labeled two-dimensional trees is the yield of a recognizable set of $\Sigma$-labeled* T3 *iff it is generated by a non-strict TAG.*

From a certain perspective, trees accepted by a T3 automaton derived from a non-strict TAG are but a special sort of derivation trees[3] for that TAG in which one does not have to resort to tree names since both the elementary trees in question and the way they are combined can be displayed explicitly in the same object. Their direct yield is the set of the trees generated by that TAG, and their one-dimensional yield is the corresponding string language.

The representation of a TAG via three-dimensional trees obviously also constitutes a regularization: Trees are now constructed by adding local trees at the frontier of another tree (see Figure 3), which is a regular process, instead of inserting trees at the interior. As stated above, the trees generated by the original TAG can be extracted from the three-dimensional trees via the yield function. We have thus described the second regularization method for TAG.

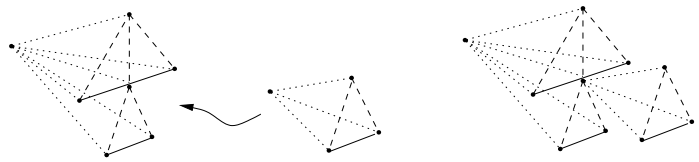


**Fig. 3.** Adjunction in TAG expressed via three-dimensional trees

We will now introduce an a bit more "term-like" representation for three-dimensional trees, which will include the concept of rank (in the second dimension), in order to facilitate the comparison to lifted trees. Let $\Sigma$ be an arbitrary ranked alphabet. We define the set $3\mathbb{D}_\Sigma$ of three-dimensional trees over $\Sigma$:

**Definition 6.** $(f,t) \in 3\mathbb{D}_\Sigma$ *if* $f \in \Sigma_0$ *($f$ is the root label) and* $t \in 3\mathbb{D}^+_\Sigma$ *($t$ is the structure formed by the nodes properly dominated by the root in the third dimension).* $3\mathbb{D}^+_\Sigma$ *is the set of* antitrunks *(three-dimensional trees without a root):*

- $(f,t,\langle t_1,\ldots,t_n\rangle) \in 3\mathbb{D}^+_\Sigma$ *if* $f \in \Sigma_n$, $t \in 3\mathbb{D}^+_\Sigma$ *and* $t_i \in 3\mathbb{D}^+_\Sigma$ *for* $0 \leq i \leq n$ *(the $t_i$ are the daughter antitrunks of $f$ in the second dimension).*
- $(f,\langle t_1,\ldots,t_n\rangle) \in 3\mathbb{D}^+_\Sigma$ *if* $f \in \Sigma_n$ *and* $t_i \in 3\mathbb{D}^+_\Sigma$ *for* $0 \leq i \leq n$.

[3] A derivation tree keeps track of the steps taken in the course of a derivation. For TAGs, its root is labeled with the name of an initial tree, and all other nodes are labeled with a pair containing the name of an auxiliary tree and an address in the tree named in the label of the mother of that node – the address of the node the auxiliary tree has been adjoined to.

In addition, we use a binary feature to indicate if a node is a foot node or not. For example, $(f, \langle\rangle, 1)$ for some label $f \in \Sigma$ and $t \in 3\mathbb{D}^+_\Sigma$ is a foot node, $(f, t, \langle t_1, t_2\rangle, 0)$ for $t_1, t_2 \in 3\mathbb{D}^+_\Sigma$ is not, and $(f, t, \langle t_1, t_2\rangle, 1)$ is not well-formed. We postulate the following conditions for foot nodes: a) Foot nodes are leaves in the second and third dimension, b) every contiguous two-dimensional tree in an antitrunk has to contain exactly one foot node, c) leaves in the second dimension are also leaves in the third.

We will now define our own yield function $yd_\Sigma : 3\mathbb{D}^+_\Sigma \times \mathbb{N} \to T_{\Sigma^0}$ where $\Sigma^0$ is a ranked alphabet with $\Sigma^0_n = \Sigma_n \cup \Sigma_0$ for every $n \geq 0$ and $T_{\Sigma^0}$ is the set of all two-dimensional trees over $\Sigma^0$. Let $x_1, \ldots, x_l$ be elements of a countable set of variables. Let $t_v, t_1, \ldots, t_m \in 3\mathbb{D}^+_\Sigma$. $g_{rk} : \Sigma_0 \times \mathbb{N} \to \Sigma^0$ is a function that takes a label of rank 0 and yields another label consisting of the same symbol, but with the rank given in the second argument.

$$yd_\Sigma(t, l) = \begin{cases} f(yd_\Sigma(t_1, l), \ldots, yd_\Sigma(t_m, l)) & \text{if } f \in \Sigma_m,\ m \geq 0, \\ & t = (f, (t_1, \ldots, t_m), 0) \\ yd_\Sigma(t_v, m)[(yd_\Sigma(t_1, l), \ldots, yd_\Sigma(t_m, l)] & \text{if } f \in \Sigma_m,\ m \geq 1,\ t = \\ & (f, t_v, (t_1, \ldots, t_m), 0) \\ f_l(x_1, \ldots, x_l) & \text{if } t = (f, (), 1) \text{ and} \\ \quad \text{for } f_l = g_{rk}(f, l) & f \in \Sigma_0 \end{cases}$$

The function is subdivided into three different cases because we have to distinguish between foot nodes and non-foot nodes, and among the latter between nodes that have an extension in the third dimension and nodes that do not. The second argument keeps track of the number of daughters of the roots in the third dimension so that when a foot node is reached the correct number of variables can be attached (which are then substituted by the direct subtrees of the corresponding root). The function $yd$ works on antitrunks. In order to obtain the yield of a three-dimensional tree we have to apply a function $yd^{pre}_\Sigma : 3\mathbb{D}_\Sigma \to T_{\Sigma^0}$ first that detaches the root and initializes the second argument: $yd^{pre}_\Sigma(t_a) = yd_\Sigma(t, 0)$ for a tree $t_a = (f, t)$ with $f \in \Sigma$ and $t \in 3\mathbb{D}^+_\Sigma$.

## 5 Equivalence

We would like to establish the equivalence of the two regularization methods for TAGs presented in the previous sections, represented either by the transformation into a lifted spine grammar or into a T3 automaton. In order to do this, it is important to note several structural similarities between the tree-like objects defined by these devices: Both types of objects still include two kinds of information about the originally intended trees, namely which individual components, i.e., elementary trees they are composed of, and how these are put together, which is precisely the information needed in order to reconstruct the intended trees from the "encoded" ones.

In the case of the T3 method the elementary trees are the child structures of the local trees the T3 tree is composed of, and the points where the local trees

are joined together are the nodes that are expanded by these child structures. In a tree generated by a lifted TAG the elementary trees are represented by the tree parts between one composition symbol on a leftmost branch and the next, and the expanded nodes correspond to the mother nodes of these composition symbols, as already stated in Section 3.

We will give a direct formal translation between lifted and three-dimensional trees that exploits these structural similarities by finding corresponding points and making them match. Let us start by giving the function $h_{li}$ translating lifted into three-dimensional trees. For this, let $\mathbb{L}_{D(\Sigma)}$ be a set of trees over $D(\Sigma)$ characterized by the following:

- $\bigcup_{n\geq 0} D(\Sigma)_{\varepsilon,n} \in \mathbb{L}_{D(\Sigma)}$ .
- $c(f, t_1, \ldots, t_n) \in \mathbb{L}_{D(\Sigma)}$ if $f \in {\Sigma_n}'$ and $t_1, \ldots, t_n \in \mathbb{L}_{D(\Sigma)} \setminus \{\pi_i^n | 1 \leq i \leq n\}$ .
- $c(f, \pi_1, \ldots, \pi_n) \in \mathbb{L}_{D(\Sigma)}$ if $f \in \Sigma'_n$ and $n \geq 1$ .
- $c(t, t_1, \ldots, t_n) \in \mathbb{L}_{D(\Sigma)}$ if $n \geq 1$, $t, t_1, \ldots, t_n \in \mathbb{L}_{D(\Sigma)} \setminus \{\pi_i^n | 1 \leq i \leq n\}$ and $t$ contains projection symbols $\pi_1, \ldots, \pi_n$ that are not dominated by more than one composition symbol on a leftmost branch in $t$.

It is clear that all trees generated by a lifted CFTG derived from a TAG via the algorithm of Fujiyoshi and Kasai [4] and all their subtrees are contained in $\mathbb{L}_{D(\Sigma)}$. We will therefore take $\mathbb{L}_{D(\Sigma)}$ as the domain of our translation function. The range will be the set $3\mathbb{D}^+_\Sigma$: $h_{li} : \mathbb{L}_{D(\Sigma)} \to 3\mathbb{D}^+_\Sigma$.

Let in the following be $t_v, t_1 \ldots, t_n$, $q_1, \ldots, q_m \in \mathbb{L}_{D(\Sigma)}$ for all $n, m \geq 0$, and $t_v, t_1 \ldots, t_n \notin \{\pi_i | i \geq 1\}$. $g_{ze} : \bigcup_{n\geq 0} \Sigma'_n \to \Sigma_0$ is a function that takes a label and yields another label consisting of the same symbol, but with rank 0.

$$h_{li}(t) = \begin{cases} (f, (h_{li}(t_1), \ldots, h_{li}(t_n)), 0) & \text{if } t = c(f, t_1, \ldots, t_n) \\ & \text{with } f \in \Sigma'_n \text{ and } n \geq 0 \\ (\Diamond, h_{li}(t_v), (h_{li}(t_1), \ldots, h_{li}(t_n)), 0) & \text{if } t = c(t_v, t_1, \ldots, t_n), \\ & n \geq 1,\ t_v = c(q_1, \ldots, q_m) \\ & \text{and } m \geq 1 \\ (f, (), 1) \text{ with } f = g_{ze}(f_0) & \text{if } t = c(f_0, \pi_1, \ldots, \pi_n) \\ & \text{with } f_0 \in \Sigma'_n \text{ and } n \geq 1. \end{cases}$$

Like our yield function from the previous section, this function is also subdivided into three cases. This time we have to distinguish between nodes that have projection symbols as daughters (the future foot nodes) and nodes that do not, and among those between nodes whose leftmost daughter is a symbol in $\Sigma'_n$ and nodes whose leftmost daughter is the root of another complex term (which is translated into an extension in the third dimension). As the function does not depend on the subscripts of the composition symbols ($c_{n,k}$ for some $n$ and $k$), they are left out. $h_{li}$ yields antitrunks. In order to translate the elements of $\mathbb{L}_{D(\Sigma)}$ into three-dimensional trees we have to apply a function $p_{li} : \mathbb{L}_{D(\Sigma)} \to 3\mathbb{D}_\Sigma$ first with $p_{li}(t) = (\Diamond, h_{li}(t))$ that attaches a three-dimensional root and then recurs

to the actual translation function. $\Diamond$ is a special placeholder for labelling three-dimensional roots, since the lifted trees do not contain the information about the labels they should have, i.e., the labels the nodes have in the original TAG before their expansion. $\Diamond$ can have any rank ($\Diamond \in \Sigma_n$ for all $n \geq 0$).

The function $h_{3d}$ that translates antitrunks into lifted trees has the set $3\mathbb{D}^+_\Sigma$ as its domain and the set $\mathbb{L}_{D(\Sigma)}$ as its range: $h_{3d} : 3\mathbb{D}^+_\Sigma \times \mathbb{N} \rightarrow \mathbb{L}_{D(\Sigma)}$. Let $t_v, t_1, \ldots, t_m \in 3\mathbb{D}^+_\Sigma$. $g_{rk(\varepsilon)} : \Sigma_0 \times \mathbb{N} \rightarrow \bigcup_{n \geq 0} \Sigma'_n$ is a function that takes a label of rank 0 and yields another label consisting of the same symbol, but of type $\langle \varepsilon, n \rangle$, where $n$ is fixed by the second argument.

$$h_{3d}(t,n) = \begin{cases} c_m(f, h_{3d}(t_1,n), \ldots, h_{3d}(t_m,n)) & \text{if } m \geq 0,\ f \in \Sigma_m, \\ & t = (f, (t_1, \ldots, t_m), 0) \\ c_m(h_{3d}(t_v, m), & \text{if } m \geq 1,\ f \in \Sigma_m \text{ and} \\ \quad h_{3d}(t_1,n), \ldots, h_{3d}(t_m,n)) & t = (f, t_v, (t_1, \ldots, t_m), 0) \\ c_n(f, \pi_1, \ldots, \pi_n) & \text{if } n \geq 1,\ f_0 \in \Sigma_0 \\ \quad \text{with } f = g_{rk(\varepsilon)}(f_0, n) & \text{and } t = (f_0, (), 1). \end{cases}$$

The cases for this function are identical to the ones for the yield function. Composition symbols on left branches are translated into extensions in the third dimension, and foot nodes are translated into nodes with the right number of sisters (i.e., the number of daughters of the node that was expanded by the corresponding elementary tree in the original TAG) labeled by projection symbols. Of the composition symbols $c_{n,k}$ only the index $n$ is given (the value of $k$ is not as immediate as the one of $n$ but can be easily inferred from the lifted tree afterwards). $h_{3d}$ takes antitrunks as its input. If we want to use $h_{3d}$ to translate a T3 tree $t_a = (f, t)$ with $f \in \Sigma$ and $t \in 3\mathbb{D}^+_\Sigma$, we first have to apply a function $p_{3d} : 3\mathbb{D}_\Sigma \rightarrow \mathbb{L}_{D(\Sigma)}$ with $p_{3d}(t_a) = h_{3d}(t, 0)$ that detaches the root of $t_a$ and initializes the second argument of the translation function.

Formally, the equivalence of Lifting and of T3 trees as regularization methods for TAG can be shown by proving that the direct decoding of an "encoded" tree and its translation into the other encoding and the decoding of the result yield exactly the same intended tree. More precisely, we can state the following theorem – see the attachment for the full proof.[4]

**Theorem 2.** *For all closed lifted trees $t_l \in \mathbb{L}_{D(\Sigma)}$ generated by the lifted version of some TAG (over $\Sigma$) and all trees $t_s \in 3\mathbb{D}_\Sigma$ generated by the three-dimensional version of the same TAG,*

- $rec(t_s) = yd^{pre}(p_{li}(t_s))$ , *and*
- $yd^{pre}(t_s) = rec(p_{3d}(t_s))$ .

[4] We also believe an even stronger equivalence in the sense of a bijection to hold: Let $A$ be the set of trees generated by a lifted spine grammar that is the regularized version of a TAG, and let $B$ be the set accepted by a T3 automaton that has been extracted from the same TAG. Then $p_{li}(A) = B$ and $p_{3d}(B) = A$, and even, for some tree $t_1 \in A$, $p_{3d}(p_{li}(t_1)) = t_1$ as well as $p_{li}(p_{3d}(t_2)) = t_2$ for some tree $t_2 \in B$.

Figure 4 shows two encoded trees and the corresponding intended tree, which is an element of the set generated by the TAG from Example 1 (with the derivation shown in Figure 1). The lifted tree in the lower right corner is an element of the set generated by the regularized version of that TAG as given in Example 2, and in the upper right corner is the matching three-dimensional tree.

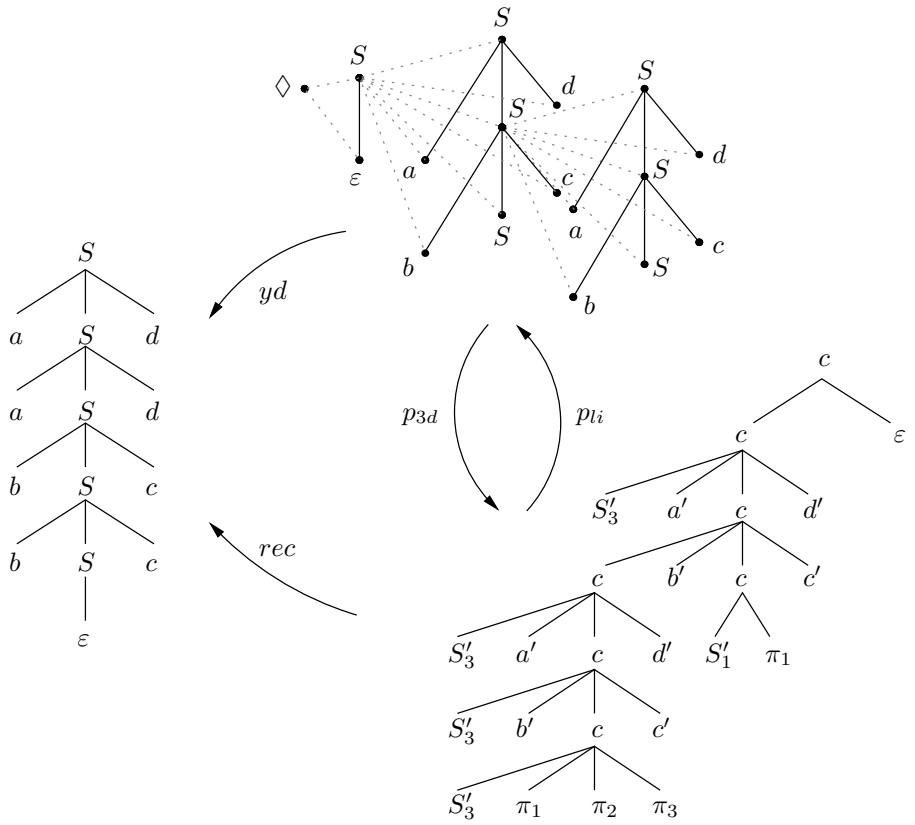



**Fig. 4.** Two encoded trees and the corresponding intended tree

## 6 Conclusion

In this paper, we have presented two regularization methods for TAGs. Regularized TAGs of both kinds define sets of "encoded" trees, which, however, still contain the necessary information to reconstruct the intended trees defined by the original TAG. Both methods operate by transforming the components of a TAG in a way that turns all inner nodes into leaves, thus making it possible to expand these nodes by means of a regular mechanism. Both methods exploit

a side effect of the theoretical concepts they are based on – algebraic Lifting and the notion of multi-dimensional trees – since neither was developed with the primary intention of regularizing the grammar formalism TAG.

However, the methods described here have even more in common: The two different kinds of objects generated by regularized TAGs exhibit a number of structural similarities, which we exploited in order to show their direct translatability. Objects of both kinds can be seen as a special sort of derivation tree for the originally intended tree, and it is only natural that this should be somehow related to regularization in that derivation trees, which are composed starting from a root and adding every further step of the derivation somewhere at the leaves, are a special sort of regular trees – recall that regularity in general represents a mode of constructing an object where one element after the other is added at the frontier of that object, and not somewhere in between. By letting us find different kinds of derivation trees for a formalism, in addition to gaining knowledge about how the properties of derivation as such can be modified, the study of regularization may thus perhaps give further insight about derivation in the formalism in particular as well.

A possible continuation of this work could be to search for more regularization methods for TAGs and determine if the structure of the objects created in the process resembles the structure of the objects treated here. One could even conjecture that every similar effort of reducing the complexity of derivation must result in similar properties, i.e., in objects that are directly translatable into lifted or three-dimensional trees as well.

## References

1. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley (1979)
2. Engelfriet, J., Schmidt, E.: IO and OI, part I. Journal of Computer and System Sciences 15, 328–353 (1977)
3. Engelfriet, J., Schmidt, E.: IO and OI, part II. Journal of Computer and System Sciences 16, 67–99 (1978)
4. Fujiyoshi, A., Kasai, T.: Spinal-formed context-free tree grammars. Theory of Computing Systems 33, 59–83 (2000)
5. F. Gécseg, M. Steinby: Tree automata. Akademiai Kiado (1984)
6. Joshi, A.K.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description. In: Dowty, D., Karttunen, L., Zwicky, A. (eds.) Natural Language Processing. Cambridge University Press (1985)
7. Mönnich, U.: On cloning contextfreeness. In: Kolb, H.P., Mönnich, U. (eds.) Studies in Generative Grammar, vol. 44, pp. 195–229. Mouton de Gruyter (1999)
8. Morawietz, F.: Two-Step Approaches to Natural Language Formalisms. Studies in Generative Grammar, vol. 64. Mouton de Gruyter (2003)
9. Rogers, J.: Syntactic Structures as Multi-dimensional Trees. Research on Language and Computation 1, 265–305 (2003)
10. Rogers, J.: wMSO Theories as Grammar Formalisms. Theoretical Computer Science 293, 291–320 (2003)

# Appendix

We shall here show the following: (1) For any closed lifted tree $t_s \in \mathbb{L}_{D(\Sigma)}$, as well as (2) for any three-dimensional tree $t_s \in 3\mathbb{D}_\Sigma$, it is true that the direct decoding of $t_s$ yields exactly the same tree as its translation into the other encoding and the decoding of the result. Both directions are proven by induction on the structure of the objects under concern. We will begin with (1).

(1) We show that $rec(t_s) = yd^{pre}(p_{li}(t_s))$ for all closed lifted trees $t_s \in \mathbb{L}_{D(\Sigma)}$. Since $yd^{pre}(p_{li}(t_s)) = yd^{pre}((\Diamond, h_{li}(t_s))) = yd(h_{li}(t_s), 0)$, this amounts to showing that $rec(t_s) = yd(h_{li}(t_s), 0)$. To that end, we will prove the more general equation $rec(t) = yd(h_{li}(t), m)$ $(*)$ for all subtrees $t$ of $t_s$ (including $t_s$) and any $m \geq 0$ in order to cover the recursive cases as well. For each of the cases, which correspond to the cases of $h_{li}$, the relevant results are underlined in order to make it easier to verify that $(*)$ holds. Assume that $(*)$ applies for $t_v, t_1, \ldots, t_n \in \mathbb{L}_{D(\Sigma)}$ and that $t_v, t_1, \ldots, t_n \notin \{\pi_i | i \geq 1\}$.

Case 1: $t = c(f, t_1, \ldots, t_n), f \in \Sigma'_n, n \geq 0$.

$rec(t) = f(x_1, \ldots, x_n)[rec(t_1), \ldots, rec(t_n)] = \underline{f(rec(t_1), \ldots, rec(t_n))}$ .

$h_{li}(t) = (f, (h_{li}(t_1), \ldots, h(t_n)), 0)$ and

$yd(h_{li}(t), m) = f(x_1, \ldots, x_n)[yd(h_{li}(t_1), m), \ldots, yd(h_{li}(t_n), m)]$

$= \underline{f(yd(h_{li}(t_1), m), \ldots, yd(h_{li}(t_n), m))}$ for any $m$.

Case 2: $t = c(t_v, t_1, \ldots, t_n), n \geq 0,$ and $t_v = c(q_1, \ldots, q_l), l \geq 1$.

$rec(t) = \underline{rec(t_v)[rec(t_1), \ldots, rec(t_n)]}$ .

$h_{li}(t) = (\Diamond, h_{li}(t_v), (h_{li}(t_1), \ldots, h_{li}(t_n)))$ and

$yd(h_{li}(t), m) = \underline{yd(h_{li}(t_v), n)[yd(h_{li}(t_1), m), \ldots, yd(h_{li}(t_n), m)]}$ for any $m$.

Case 3: $t = c(f, \pi_1, \ldots, \pi_n), f \in \Sigma'_n, n \geq 1$.

$rec(t) = \underline{f(x_1, \ldots, x_n)}$ .

$h_{li}(t) = (f, (), 1)$ and $yd(h_{li}(t), m) = \underline{f(x_1, \ldots, x_m)}$ . *Problem!*

Obviously, Case 3 seems to be a problem, as $(*)$ only holds for $m = n$. This can be resolved as follows: Since $c(f, \pi_1, \ldots, \pi_n)$ is not a closed lifted tree, $t = c(f, \pi_1, \ldots, \pi_n)$ must be a genuine subtree of $t_s$, and, since we postulated that $t_s$ be closed, the projection symbols $\pi_1, \ldots, \pi_n$ in $t$ must of necessity point to some $n$ subtrees of $t_s$ whose roots are sisters to the right of some node $q$ labeled by a composition symbol on a left branch dominating the root $r$ of $t$ (the dominance

relation may have to be reflexive here, for $q = r$). The subtree rooted at the mother of $q$ is covered by Case 2, where the translation of the subtree rooted at $q$ itself (containing $t$) is fed into the first and the number $n$ is fed into the second argument of the $yd$ function. According to the definition of $\mathbb{L}_{D(\Sigma)}$, the path from $q$ to $r$ may not contain any other node labeled by a composition symbol on a left branch (which would cause the second argument of $yd$ to be changed to the number of sisters of that node), and thus $n$ is passed down the path until the point where the first argument of $yd$ is the translation of $t$. Consequently, when $yd(h_{li}(t), m)$ is compared to $rec(t)$ in Case 3, the value of $m$ is always $n$. □

(2) For the other direction, we want to show that $yd^{pre}(t_s) = rec(p_{3d}(t_s))$ for all $t_s \in 3\mathbb{D}_\Sigma$. Since $yd^{pre}(t_s) = yd(t_p, 0)$ and $rec(p_{3d}(t_s)) = rec(h_{3d}(t_p, 0))$ for $t_s = (f, t_p)$, $f \in \Sigma$, this amounts to showing that $yd(t_p, 0) = rec(h_{3d}(t_p, 0))$ for all $t_p \in 3\mathbb{D}^+_\Sigma$. We prove the more general equation $yd(t, m) = rec(h_{3d}(t, m))$ $(**)$ for all $t \in 3\mathbb{D}_\Sigma$ and any $m \geq 0$ in order to cover the recursive cases, as above. Again, the cases here correspond to the cases of $h_{3d}$, and the relevant results are underlined. Assume that $(**)$ applies for $t_v, t_1, \ldots, t_n \in 3\mathbb{D}^+_\Sigma$.

Case 1: $t = (f, (t_1, \ldots, t_n), 0), f \in \Sigma'_n, n \geq 0$.

$$yd(t, m) = f(x_1, \ldots, x_n)[yd(t_1, m), \ldots, yd(t_n, m)]$$
$$= \underline{f(yd(t_1, m), \ldots, yd(t_n, m))} \text{ for any } m.$$

$h_{3d}(t, m) = c(f, h_{3d}(t_1, m), \ldots, h_{3d}(t_n, m))$ and

$$rec(h_{3d}(t, m)) = f(x_1, \ldots, x_n)[rec(h_{3d}(t_1, m)), \ldots, rec(h_{3d}(t_n, m))]$$
$$= \underline{f(rec(h_{3d}(t_1, m)), \ldots, rec(h_{3d}(t_n, m)))} \text{ for any } m.$$

Case 2: $t = (f, t_v, (t_1, \ldots, t_n), 0), f \in \Sigma'_n, n \geq 1$.

$yd(t, m) = \underline{yd(t_v, n)[yd(t_1, m), \ldots, yd(t_n, m)]}$ for any $m$.

$h_{3d}(t, m) = c(h_{3d}(t_v, n), h_{3d}(t_1, m), \ldots, h_{3d}(t_n, m))$ and, for any $m$,

$rec(h_{3d}(t, m)) = \underline{rec(h_{3d}(t_v, n))[rec(h_{3d}(t_1, m)), \ldots, rec(h_{3d}(t_n, m))]}$ .

Case 3: $t = (f, (), 1), f \in \Sigma'_0$.

$yd(t, m) = \underline{f(x_1, \ldots, x_m)}$ for any $m$.

$h_{3d}(t, m)) = c(f, \pi_1, \ldots, \pi_m)$ and $rec(h_{3d}(t, m)) = \underline{f(x_1, \ldots, x_m)}$ for any $m$. □

We have proven that $rec(t_s) = yd^{pre}(p_{li}(t_s))$ for all closed lifted trees $t_s \in \mathbb{L}_{D(\Sigma)}$ and $yd^{pre}(t_s) = rec(p_{3d}(t_s))$ for all trees $t_s \in 3\mathbb{D}_\Sigma$.