15

Strong connectivity hypothesis and generative power in TAG

Alessandro Mazzei, Vincenzo Lombardo and Patrick Sturt [†]

Abstract
Dynamic grammars are relevant for psycholinguistic modelling and speech processing. However, formal treatments of dynamic grammars are rare, and there are few studies that examine the relationship between dynamic and phrase structure grammars. This paper introduces a TAG related dynamic grammar (DVTAG) together with a study of its expressive power. We also shed a new perspective on the *wrapping* operation in TAG.

Keywords Dynamic grammars, TAG, wrapping, incrementality

15.1 Introduction

Incrementality is a feature of human language analysis that is very relevant for language modeling and speech processing. An incremental processor takes a string in left-to-right order and starts to build a syntactic and semantic representation before reaching the end of the sentence. The strong connectivity hypothesis is a parsimonious way to formalize the incrementality of the syntactic process: people incorporate each word into a single, totally connected syntactic structure before any further words follow (Stabler, 1994). Strong connectivity is supported by some psycholinguistic evidence (Kamide et al., 2003, Sturt and Lombardo, 2005).

Some traditional approaches to syntactic processing (both derivation

[†]We thank an anonymous reviewer and Jim Rogers for many useful comments.

FG-MoL 2005.
James Rogers (ed.).
Copyright © 2009, CSLI Publications.

and parsing) use a generative grammar and implement connectivity in the derivation/parsing algorithm (Abney and Johnson, 1991). An alternative approach is to change the perspective of investigation and model the strongly connected syntactic process in the grammar formalism. Dynamic systems model cognitive processes as sequences of states connected through transitions (Gelder, 1998). Like in automata theory, a state encodes the syntactic process until some point in the input string; then a string (one or several words) realizes the transition to the subsequent state. However, in dynamic approaches the automaton is not the result of a compilation of a generative grammar, but the formalism itself (cf. Woods (1986)), without any necessary involvement of generative rules.

The dynamic approach is not new in mathematical linguistics. Both left-associative grammars (Hausser, 1992) and dynamic dependency grammars (Milward, 1994) are examples of *dynamic grammars*. The common traits of dynamic grammars are the definition of a (non necessarily finite) recursive set of states, a finite subset of initial states, a finite set of axioms or rules. In addition to these, the definition of a dynamic grammar includes a way to assembly multiple applications of the axioms (or rules), usually an explicit or implicit specification of sequencing. However, an interesting abstraction in Milward's specification, called a deduction rule, allows several possibilities of assembling, and so provides a dynamic account of non–constituent coordination (Milward, 1994). In order to specify what are the legal strings of a language, both Hausser and Milward indicate a finite set of final states (in fact both approaches originate in a categorial grammar paradigm). We can say that left-associative grammars and dynamic dependency grammars incorporate the derivation process entirely. However, this is not strictly necessary if we design the axioms (or rules) on a generative basis. In this paper we take a hybrid dynamic–generative approach: states are partial structures that are licensed by a recursive process from a finite basic lexicon; transitions extend the basic structures through a finite number of generative operations. A state S_i represents a partial structure that spans the left fragment of a sentence $w_1...w_i...w_n$. In particular, we propose a Dynamic Version of TAG (DVTAG). DVTAG is a dynamic grammar that fulfills the strong connectivity hypothesis, has interesting consequences for constituency definition and linguistic descriptions, and shares basic tenets with LTAG (i.e. lexicalization, adjoining and extended domain of locality) (Mazzei, 2005). We provide a formal definition of DVTAG and a study of its expressive power. We show that DVTAG is a mildly context-sensitive formalism and is strongly equivalent to a restricted LTAG formalism, called *dynamic*

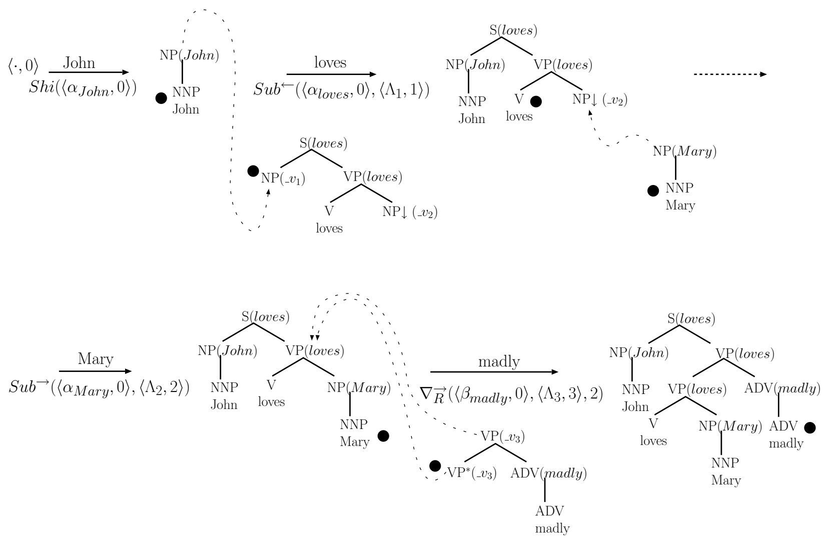

FIGURE 1: The DVTAG derivation of the sentence *John loves Mary madly.*

LTAG. Moreover we show that the introduction of wrapping operation, an alternative view of adjoining based on *flexible composition* of the derivation tree (Joshi, 2004), increases the generative power of DVTAG and dynamic LTAG.

15.2 Informal introduction to DVTAG

In Fig. 1 we can see the DVTAG derivation of the sentence *John loves Mary madly.* Like LTAG (Joshi and Schabes, 1997), a Dynamic Version of Tree Adjoining Grammar (DVTAG) consists of a set of elementary trees, divided into initial trees and auxiliary trees, and attachment operations for combining them. Lexicalization is expressed through the association of a lexical *anchor* with each elementary tree. To encode lexical dependencies, each node in the elementary tree is augmented with a feature indicating the lexical head that projects the node. The head variable is a variable in logic terms: $\_v_3$ will be unified with the constant *loves* in the derivation of Fig. 1. The derivation process in DVTAG builds a constituency tree by combining the elementary trees via operations that are illustrated below. DVTAG implements the incremental process by constraining the *derivation process* to be a series of steps in which an elementary tree is combined with the partial tree

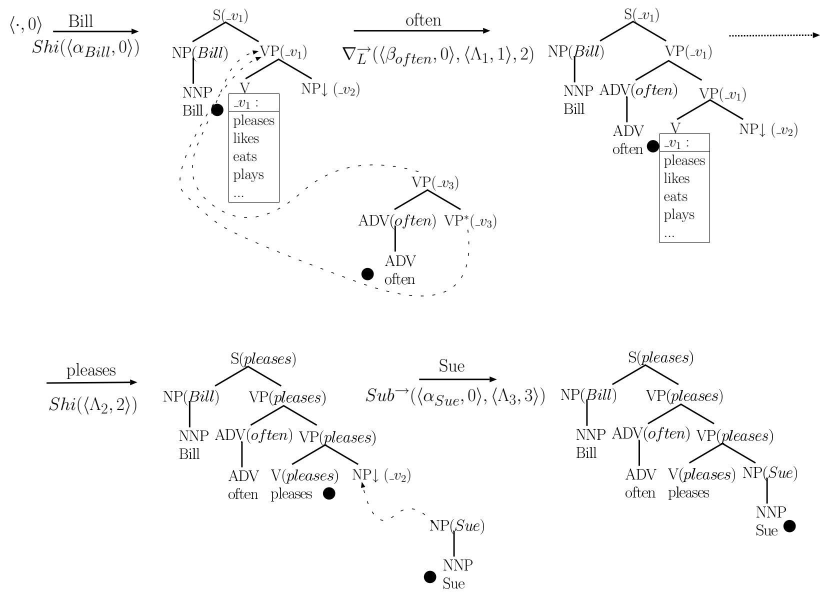

FIGURE 2: The DVTAG derivation of the sentence *Bill often pleases Sue.*

spanning the left fragment of the sentence. The result of a step is an updated partial structure. Specifically, at the processing step i, the elementary tree anchored by the i-th word in the sentence is combined with the partial structure spanning the words from 1 to $i-1$ positions; the result is a partial structure spanning the words from 1 to i. In DVTAG the derivation process starts from an elementary tree anchored by the first word in the sentence and that does not require any attachment that would introduce lexical material on the left of the anchor (such as in the case that a Substitution node is on the left of the anchor). This elementary tree becomes the first left-context that has to be combined with some elementary tree on the right. At the end of the derivation process the left-context spans the whole sentence, and is called the *derived tree*: the last tree of Fig.1 is the derived tree for the sentence *John loves Mary madly.*
In DVTAG we always combine a left context with an elementary tree, then there are seven attachment operations. Adjoining is split into two operations: *adjoining from the left* and *adjoining from the right.* The type of adjoining depends on the position of the lexical material in-

troduced by the auxiliary tree with respect to the material currently dominated by the adjoined node (which is in the left-context). In Fig. 1 we have an adjoining from the right in the case of the left auxiliary tree anchored by *madly*, and in Fig. 2 we have an adjoining from the left in the case of the left auxiliary tree anchored by *often*. Inverse operations account for the insertion of the left-context into the elementary tree. In the case of *inverse substitution* the left-context replaces a substitution node in the elementary tree; in the case of *inverse adjoining from the left* and *inverse adjoining from the right*, the left-context acts like an auxiliary tree, and the elementary tree is split because of the adjoining of the left context at some node. In Fig. 1 we have an inverse substitution in the case of the initial tree anchored by *John*. Finally, the *shift* operation either scans a lexical item which has been already introduced in the structure or derives a lexical item from some predicted preterminal node. The grounding of the variable $\_v_1$ in Fig. 2 is an example of shift.

It is important to notice that, during the derivation process, not all the nodes in the left-context and the elementary tree are accessible for performing operations: given the $i-1$-th word in the sentence we can compute a set of accessible nodes in the left-context (the *left-context fringe*); also, given the lexical anchor of the elementary tree, that in the derivation process matches the i-th word in the sentence, we can compute a set of accessible nodes in the elementary tree (the *elementary tree fringe*). To take into account this feature, the elementary tree in DVTAG are dotted tree, i.e. a couple $\langle \gamma, i \rangle$ formed by a tree γ and an integer i denoting the accessible fringe[1] of the tree (Mazzei, 2005).

The DVTAG derivation process requires the full connectivity of the left-context at all times. The extended domain of locality provided by LTAG elementary trees appears to be a desirable feature for implementing full connectivity. However, each new word in a string has to be connected with the preceding left-context, and there is no *a priori* limit on the amount of structure that may intervene between that word and the preceding context. For example in the sentence *Bill often pleases Sue* there is an intervening modifier between an argument and its predicative head (Fig.2). The elementary tree *Bill* is linguistically motivated up to the NP projection; the rest of the structure depends on connectivity. These extra nodes are called *predicted nodes*. A predicted preterminal node is referred by a set of lexical items, that represent a *predicted head*. So, the extended domain of locality available in LTAG has to be further extended. In particular, some structures have to be

[1] In the figures we represent the integer using a dot.

predicted as soon as there is some evidence from arguments or modifiers on the left.

15.3 Formalizing DVTAG

Now we provide a formal definition of DVTAG using strings of a formal language (a, b c, etc). In the definitions and proofs presented here, we use DVTAG elementary trees without defining the notions of underspecified heads and without the specification of the head-variable (see previous section). In fact these two features are important for linguistic description, but are irrelevant with respect to the generative power. First, we provide some general terminology. Let Σ be an alphabet of terminal symbols, and V an alphabet of non–terminal symbols. $a, b, c, d, ... \in \Sigma$ indicate terminal symbols, where ε is the null string. $A, B, C, D, ... \in V$ indicate non–terminal symbols, $x, y, w, z \in \Sigma^*$ are strings of terminals: $|x|$ is the length of the string x. We use w_i to denote the *i*-th word in the sentence w. We denote initial trees with α, auxiliary trees with β, derived and generic trees with γ, δ, ζ, Λ[2]; $\mathcal{N}$ denotes a node belonging to a tree, *label*($\mathcal{N}$) the label of this node. $foot(\beta)$ returns the foot node of an auxiliary tree β. $YIELD(\gamma)$ is a function that returns the frontier of γ with the exception of foot nodes (i.e. overt terminal nodes and substitution nodes). Finally, $YIELD_i(\gamma)$ is the function that returns the *i*–th element of the $YIELD(\gamma)$ whether $1 < i < |YIELD(\gamma)|$, otherwise it is undefined. As usual in TAG, to denote the position of a node $\mathcal{N}$ in a tree, we use its *Gorn Address.* Since DVTAG is a lexicalized formalism, each elementary tree is anchored by some lexical element. We refer to the leftmost lexical element in a tree with the expression *left-anchor.* We call a tree γ *direct* if $YIELD_1(\gamma)$ is its left-anchor, moreover we call γ *inverse* if $YIELD_1(\gamma)$ is a substitution node and $YIELD_2(\gamma)$ is its left-anchor. While initial trees are not different from the ones in LTAG, we distinguish three types of auxiliary trees: *left auxiliary trees* $\mathcal{A}_L$ as auxiliary trees where the foot node is the rightmost node of the frontier, *right auxiliary trees* $\mathcal{A}_R$ as auxiliary trees where the foot node is the leftmost node of the frontier, *wrapping auxiliary trees* $\mathcal{A}_W$ as auxiliary trees where the frontier extends on both the left and the right of the foot node. We introduce two useful notions that are borrowed from parsing algorithm tradition, namely *dotted tree* and *fringe.*

Definition 35 A **dotted tree** is a pair $\langle \gamma, i \rangle$ where γ is a tree and i is an integer such that $i \in 0...|YIELD(\gamma)|$.

[2]Conventionally we always use Λ to indicate the left-context.

Given a set of (LTAG) elementary tree $\mathcal{E}$ with $\langle \mathcal{E}, 0\rangle$, we will indicate the set of dotted trees such that if $\gamma \in \mathcal{E}$, then $\langle \gamma, 0\rangle \in \mathcal{E}$; we represent a generic element of $\langle \mathcal{E}, 0\rangle$ with the *underspecified* dotted tree $\langle \cdot, 0\rangle$. The fringe of $\langle \gamma, i\rangle$ is a set of nodes (split in a left fringe and a right fringe) that includes all the nodes that are accessible for operating upon, that is the fringe defines the domain of the attachment operations.

Definition 36 The **left-fringe** (Lfringe) of $\langle \gamma, i\rangle$ is the set difference between the set of nodes on the path from $YIELD_i(\gamma)$ to root and the set of nodes on the path from $YIELD_{i+1}(\gamma)$ to root. The **right-fringe** (Rfringe) of $\langle \gamma, i\rangle$ is the set difference between the set of nodes on the path from $YIELD_{i+1}(\gamma)$ to root, and the set of nodes on the path from $YIELD_i(\gamma)$ to root. Moreover, if there is a null lexical item ε on the left (on the right) of $YIELD_i(\gamma)$ ($YIELD_{i+1}(\gamma)$), all the nodes from ε up to the lowest common ancestor of ε and $YIELD_i(\gamma)$ ($YIELD_{i+1}(\gamma)$), belong to the right and left fringes.

In Fig. 3 left and right fringes are depicted as ovals in the dotted trees. Now we define seven attachment operations on dotted trees: two substitutions (similar to LTAG substitution), four adjoinings (similar to LTAG adjoining), and one shift.
The **Shift** operation $Shi(\langle \gamma, i\rangle)$ is defined on a single dotted tree $\langle \gamma, i\rangle$ and returns the dotted tree $\langle \gamma, i+1\rangle$. It can be applied only if a terminal symbol belongs to the right fringe of $\langle \gamma, i\rangle$: the dot is shifted on the right of the next overt lexical symbol of the yield $YIELD_{i+1}(\gamma)$ (Fig. 3-a).
The **Substitution** operation $Sub^{\rightarrow}(\langle \alpha, 0\rangle, \langle \gamma, i\rangle)$ is defined on two dotted trees: a dotted tree $\langle \gamma, i\rangle$ and an initial direct dotted tree $\langle \alpha, 0\rangle$. If there is in the right fringe of $\langle \gamma, i\rangle$ a substitution node $\mathcal{N}$and $label(\mathcal{N}) = label(root(\alpha))$, the operation returns a new dotted tree $\langle \delta, i+1\rangle$ such that δ is obtained by grafting α in $\mathcal{N}$(Fig. 3-b).
The **Inverse Substitution** operation $Sub^{\leftarrow}(\langle \zeta, 0\rangle, \langle \gamma, i\rangle)$ is defined on two dotted tree: a dotted tree $\langle \gamma, i\rangle$ and an inverse elementary dotted tree $\langle \zeta, 0\rangle$. If $root(\gamma)$ belongs to the left fringe of $\langle \gamma, i\rangle$, and there is a substitution node $\mathcal{N}$ belonging to the right fringe of $\langle \zeta, 0\rangle$ such that $label(\mathcal{N}) = label(root(\gamma))$, the operation returns a new dotted tree $\langle \delta, i+1\rangle$ such that δ is obtained by grafting γ in $\mathcal{N}$(Fig. 3-c).
The **Adjoining from the left** operation $\nabla_L^{\rightarrow}(\langle \beta, 0\rangle, \langle \gamma, i\rangle, add)$ is defined on two dotted trees: a dotted tree $\langle \gamma, i\rangle$ and a direct left or wrapping auxiliary dotted tree $\langle \beta, 0\rangle$. If there is in the right fringe of $\langle \gamma, i\rangle$ a non-terminal node $\mathcal{N}$ such that $label(\mathcal{N}) = label(root(\beta))$, the operation returns a new dotted tree $\langle \delta, i+1\rangle$ such that δ is obtained by grafting β in $\mathcal{N}$ (Fig. 3-d).

The **Adjoining from the right** operation $\nabla_{R}^{\rightarrow}(\langle\beta,0\rangle,\langle\gamma,i\rangle,add)$ is defined on two dotted trees: a dotted tree $\langle\gamma,i\rangle$ and a direct right auxiliary dotted tree $\langle\beta,0\rangle$. If there is in the left fringe of $\langle\gamma,i\rangle$ a non-terminal node $\mathcal{N}$ such that $label(\mathcal{N}) = label(root(\beta))$, the operation returns a new dotted tree $\langle\delta,i+1\rangle$ such that δ is obtained by grafting β in $\mathcal{N}$ (Fig. 3-e).
The **Inverse adjoining from the left** operation $\nabla_{L}^{\leftarrow}(\langle\zeta,0\rangle,\langle\gamma,i\rangle,add)$ is defined on two dotted trees: a dotted tree $\langle\gamma,i\rangle$ and a direct elementary dotted tree $\langle\zeta,0\rangle$. If $foot(\gamma)$ belongs to the fringe of $\langle\gamma,i\rangle$, and there is a node $\mathcal{N}$ belonging to right fringe of $\langle\zeta,0\rangle$ such that $label(\mathcal{N}) = label(foot(\gamma))$, the operation returns a new dotted tree $\langle\delta,i+1\rangle$ such that δ is obtained by grafting γ in $\mathcal{N}$ (Fig. 3-f).
Inverse adjoining from the right operation $\nabla_{R}^{\leftarrow}(\langle\zeta,0\rangle,\langle\gamma,i\rangle,add)$ is defined on two dotted tree: $\langle\gamma,i\rangle$ and the direct elementary dotted tree $\langle\zeta,0\rangle$. $\langle\zeta,0\rangle$ has a null lexical item (ε node) as first leaf. If $root(\gamma)$ belongs to the fringe of $\langle\gamma,i\rangle$, and there is a node $\mathcal{N}$ belonging to left fringe of $\langle\zeta,0\rangle$ such that $label(\mathcal{N}) = label(root(\gamma))$, the operation returns a new dotted tree $\langle\delta,i+1\rangle$ such that δ is obtained by grafting γ in $\mathcal{N}$ (Fig. 3-g).
Using the definitions given above, we can formally define DVTAG.

Definition 37 Let $\langle\mathcal{E},0\rangle$ be a finite set of elementary dotted trees, a DVTAG $G(\langle\mathcal{E},0\rangle)$ is a triple consisting of:
(1) Set of initial left-context: the finite set of direct dotted trees $\langle\Lambda_0,0\rangle \in \langle\mathcal{E},0\rangle$.
(2) Set of axiom schemata: there are three kinds of schemata to update the left-contexts:
1.

$$\langle\cdot,0\rangle \xrightarrow[Shi(\langle\Lambda_1,0\rangle)]{a} \langle\Lambda_1,1\rangle$$

where the terminal symbol **a** is the left-anchor of the initial left-context $\langle\Lambda_1,0\rangle$.
2.

$$\langle\Lambda_i,i\rangle \xrightarrow[Shi(\langle\Lambda_i,i\rangle)]{a} \langle\Lambda_{i+1},i+1\rangle$$

where the terminal symbol **a** belongs to the right fringe of $\langle\Lambda_i,i\rangle$ and i ranges over the natural numbers.
3.

$$\langle\Lambda_i,i\rangle \xrightarrow[op_a]{a} \langle\Lambda_{i+1},i+1\rangle$$

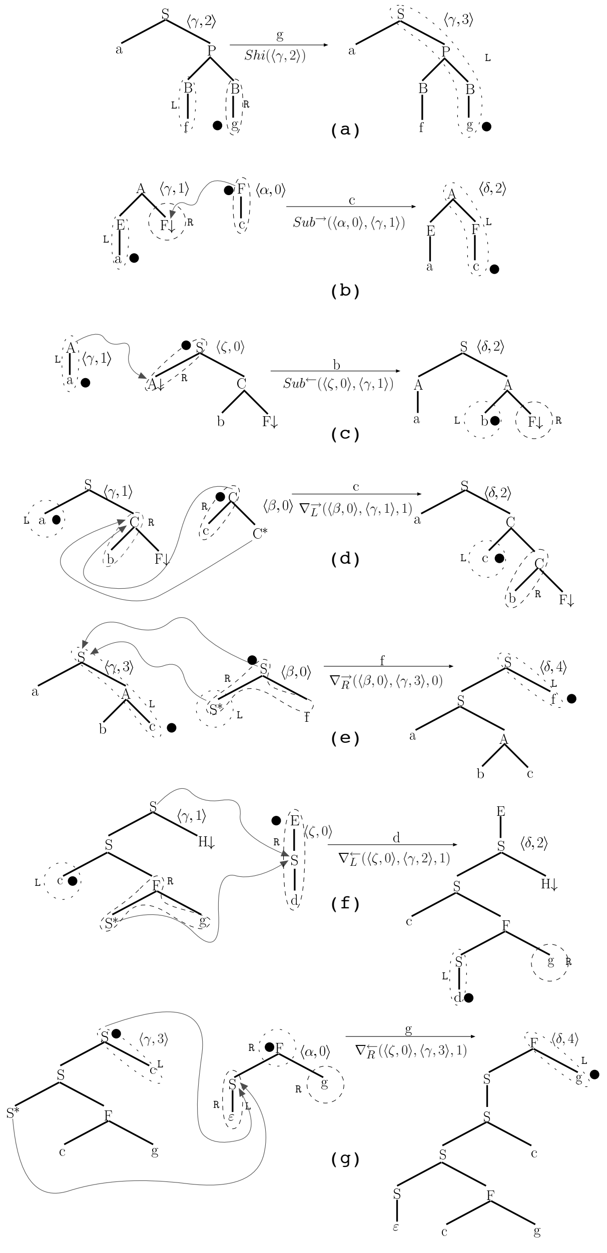

FIGURE 3: Operations in DVTAG.

where

$$op_a = \begin{cases} Sub^{\rightarrow}(\langle\alpha,0\rangle,\langle\Lambda_i,i\rangle) \\ Sub^{\leftarrow}(\langle\zeta,0\rangle,\langle\Lambda_i,i\rangle) \\ \nabla_L^{\rightarrow}(\langle\beta,0\rangle,\langle\Lambda_i,i\rangle,add) \\ \nabla_R^{\rightarrow}(\langle\beta,0\rangle,\langle\Lambda_i,i\rangle,add) \\ \nabla_L^{\leftarrow}(\langle\zeta,0\rangle,\langle\Lambda_i,i\rangle,add) \\ \nabla_R^{\leftarrow}(\langle\zeta,0\rangle,\langle\Lambda_i,i\rangle,add) \end{cases}$$

and **a** is the left-anchor of $\langle\alpha,0\rangle$, $\langle\zeta,0\rangle$, $\langle\beta,0\rangle$, $\langle\zeta,0\rangle$ respectively. For $\nabla_L^{\rightarrow}, \nabla_R^{\rightarrow}, \nabla_L^{\leftarrow}, \nabla_R^{\leftarrow}$, **add** is the Gorn address of the adjoining node, and where i ranges over the natural numbers.

(3) Deduction rule that specifies the sequentiality of the derivation process

$$\frac{\langle\Lambda_i,i\rangle \xrightarrow[op_{a_1}\ldots op_{a_n}]{a_1\ldots a_n} \langle\Lambda_j,j\rangle \qquad \langle\Lambda_j,j\rangle \xrightarrow[op_{b_1}\ldots op_{b_m}]{b_1\ldots b_m} \langle\Lambda_k,k\rangle}{\langle\Lambda_i,i\rangle \xrightarrow[op_{a_1}\ldots op_{a_n} op_{b_1}\ldots op_{b_m}]{a_1\ldots a_n b_1\ldots b_m} \langle\Lambda_k,k\rangle}$$

DVTAG defines implicitly the infinite space of derivable left-contexts, i.e. the (recursive) set of left-contexts that are reachable with a finite number of operations starting from an initial left-context. In order to define the language generated by a DVTAG we have to define the notion of final left-context. Both dynamic dependency grammars (Milward, 1994) and left-associative grammars (Hausser, 1992) explicitly define the finite set of the final state. In contrast, in our hybrid dynamic–generative approach we define the set of final left-contexts on the basis of their structure. We call a left-context $\langle\Lambda_n,n\rangle$ final if Λ_n has no substitution or foot nodes in the yield.

Definition 38 A **string** $w_1 ... w_n$ (n natural number) is generated by a DVTAG $G(\langle\mathcal{E},0\rangle)$ if and only if $\langle\cdot,0\rangle \xrightarrow[op_{w_1}\ldots op_{w_n}]{w_1\ldots w_n} \langle\Lambda_n,n\rangle$ is derivable in $G(\langle\mathcal{E},0\rangle)$ and $\langle\Lambda_n,n\rangle$ is final. Moreover, a **tree** Λ_n (n natural number) is generated by a DVTAG $G(\langle\mathcal{E},0\rangle)$ if and only if $\langle\cdot,0\rangle \xrightarrow[op_{w_1}\ldots op_{w_n}]{YIELD(\Lambda_n)} \langle\Lambda_n,n\rangle$ is derivable in $G(\langle\mathcal{E},0\rangle)$ and $\langle\Lambda_n,n\rangle$ is final.

In contrast to the standard definition of generative grammars, the Definition 37 describes the derivation process as part of the definition of the grammar. Note that the notion of left-context corresponds to the notion of sentential form of the usual generative grammars. A dynamic grammar explicitly includes the derivability of a left-context, whereas this notion is outside of the grammar in the standard generative formalization. The dynamic approach allows us to define several deduction rules for derivation: Milward used this feature to take into account

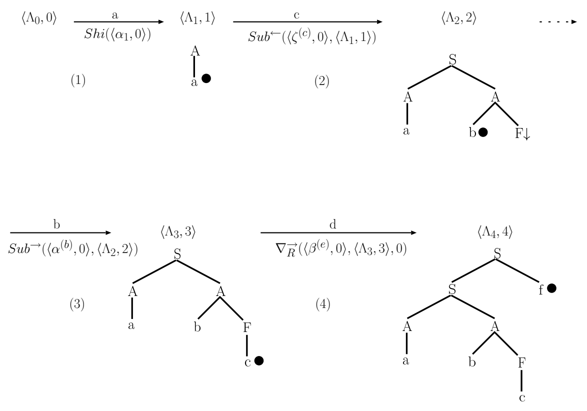

FIGURE 4: A DVTAG derivation for the sentence "abcf".

the non-constituent coordination in the dynamic dependency grammars (Milward, 1994). Finally, we formalize the notion of *derivation chain*:

Definition 39 Given a string $w_1...w_n$ derivable by $G(\langle\mathcal{E}, 0\rangle)$, a **derivation chain d** is the sequence of left-contexts $\langle\Lambda_0, 0\rangle\langle\Lambda_1, 1\rangle...\langle\Lambda_n, n\rangle$ such that $\langle\Lambda_0, 0\rangle$ is an initial left-context, $\langle\Lambda_n, n\rangle$ is a final left-context and $\langle\Lambda_{i-1}, i-1\rangle \xrightarrow[op_{w_i}]{w_i} \langle\Lambda_i, i\rangle$ $i \in 1...n$.

In Fig. 4 there is a DVTAG derivation chain for the sentence "abcf", using some elementary dotted tree of Fig. 3

15.4 Generative power of DVTAG

In this section we show that for each DVTAG there exists a LTAG that generates the same tree language, i.e. $\mathcal{L}(DVTAG) \subseteq \mathcal{L}(LTAG)$; then we show that $\mathcal{L}(CFG) \subset \mathcal{L}(DVTAG)$; the consequence of these results is that DVTAG is a mildly context-sensitive formalism (Vijay-Shanker et al., 1990). In the proof of the second result we introduce a restricted LTAG formalism called dynamic LTAG (dLTAG). For the proofs that follow it is useful recall the notion of derivation tree in LTAG. Given a $G_1(\mathcal{E})$ LTAG, a **derivation tree** D for G_1 is a tree in which each node of the tree is the identifier of an elementary tree γ belonging to $\mathcal{E}$, and each arc linking a parent γ to a child γ' and labeled with Gorn address t, represents that the tree γ' is an auxiliary tree adjoined at node t of the tree γ, or that γ' is an initial tree substituted at the node t in γ (Joshi and Schabes, 1997). The constituency tree that results from the derivation described by a derivation tree D is called the derived tree *derived*(D). In LTAG there are no constraints about the order in which the elementary trees are combined in the derivation tree. In order to compare DVTAG and LTAG derivations we introduce the notion of *partial derivation tree.* Given a derivation tree D, we define a **partial derivation tree** PD as a connected subgraph of D; a partial derived tree *derived*(PD) is yielded from a partial derivation tree PD.

Theorem 41 *Let*[3] $G_1(\mathcal{E})$ *be a LTAG, let* $G_2(\langle\mathcal{E},0\rangle)$ *be a DVTAG and let* D *be a derivation tree of* $G_1(\mathcal{E})$ *that derives the string* $w_1...w_n$ *from the elementary trees* $\gamma_1...\gamma_n$*. There exists a sequence of partial derivation trees* $PD_1, PD_2, ..., PD_n$ *of* D *such that each* PD_i *is composed by the trees* $\gamma_1...\gamma_i$ *if and only if there exists a derivation chain* d *of* $G_2(\langle\mathcal{E},0\rangle$ *such that* $d = \langle\Lambda_0,0\rangle\langle\Lambda_1,1\rangle...\langle\Lambda_n,n\rangle$ *with* $\Lambda_i = derived(PD_i)\ \forall i \in 1...n$*.*

Proof (*sketch*) From the definitions of partial derivation tree, PD_{i+1} is equal to PD_i but the node γ_{i+1}. Since LTAG operations preserve precedence and dominance relations among two nodes of the partial derived tree[4], we have that $w_1...w_i$ is a prefix of $YIELD(derived(PD_i))$. As a consequence, in *derived*(PD_{i+1}) there are no substitution nodes on the left of the left-anchor of γ_{i+1}. Since G_1 and G_2 use the same trees, we can show that there is a bijection between all the possible ways of inserting γ_{i+1} into PD_i in LTAG and the DVTAG operations that implement the transition between $\langle\Lambda_i, i\rangle$ and $\langle\Lambda_{i+1}, i+1\rangle$. This is shown for each i through an induction process. □

A corollary of this theorem shows that a LTAG G_1 generates only derivation trees D such that exists a sequence of partial derivation trees of $PD_1, PD_2, ..., PD_n$ such that each PD_i is composed by the trees $\gamma_1...\gamma_i$, if and only if there is a DVTAG G_2 that generates the same tree languages.

Given a partial derivation tree PD we write $w_i < w_j$ if on the yield of *derived*(PD) w_i precedes w_j and we write $\gamma_i < \gamma_j$ if $w_i < w_j$, where w_i and w_j are the left-anchors of γ_i and γ_j respectively.

Now we define dLTAG and then we show that dLTAG respects the hypotheses of the theorem 41.

Definition 40 A partial derivation tree PD is **dynamic** if: **(1)** Each node γ_i in PD has at most one child γ_j such that $\gamma_j < \gamma_i$, **(2)** If a node γ_i in PD has a child γ_j such that $\gamma_i < \gamma_j$, then γ_j does not have a child γ_k such that $\gamma_k < \gamma_j$.

[3]For space reasons in the proof sketch of this theorem we assume that each γ_i is anchored by only one lexical item w_i.

[4]I.e. the precedence and dominance relations among the nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ in PD_i will be the same in $PD_{i+j}\forall j \geq 0$.

We define dLTAG by constraining the derivation trees to be dynamic, thus satisfying the hypotheses of theorem 41 as shown in the the following theorem.

Theorem 42 *Let D be a derivation tree formed by the nodes $\gamma_1 ... \gamma_n$ of LTAG $G_1(\mathcal{E})$ that generates the string $w_1 ... w_n$. A sequence of partial derivation trees $PD_1, PD_2, ..., PD_n$ of D exists and PD_i is formed by $\gamma_1 ... \gamma_i$ ($\forall\ i \in 1...n$) if and only if D is dynamic.*

Proof (*sketch*) We can prove the theorem by reductio ad absurdum. Supposing that D is dynamic, we can show that if the strong connectivity of the sequence of partial derivation tree is not fulfilled by a node γ_i (hypotheses of theorem 42) we arrive at the absurdum that on the node γ_i, D does not fulfill the dynamic definition. Similarly, supposing that D fulfills the strong connectivity hypothesis, we can show that the two conditions of the Definition 40 cannot be violated. $\square$

A corollary of the theorems 41 and 42 states that DVTAG class of grammars generate the same tree languages of dynamic LTAG class of grammar. Given the constraints in the definition 40 we have that $\mathcal{L}(dLTAG) \subseteq \mathcal{L}(LTAG)$[5], since DVTAG is equivalent to dLTAG we have that $\mathcal{L}(DVTAG) \subseteq \mathcal{L}(LTAG)$. Now we show that $\mathcal{L}(CFG) \subset \mathcal{L}(DVTAG)$ by showing that $\mathcal{L}(CFG) \subset \mathcal{L}(dLTAG)$. This fact is proved by exploiting the notion of normal form for derivation tree in LTIG (Schabes and Waters, 1995). LTIG is a particular LTAG that is strongly equivalent to CFG, with some limitations on the elementary trees and some limitations on the operations. It is possible to transform a derivation tree for a LTIG in normal form into a dynamic derivation tree with a simple algorithm (Mazzei, 2005). As a consequence, for each CFG there is dynamic LTIG that generates the same tree language, then there is a DVTAG that generates the same tree language. Now we use the results described above to prove that DVTAG is a mildly context-sensitive formalism, i.e. DVTAG generates cross-serial dependencies (1), generates the context-free languages (2), is parsable in polynomial time (3), has the constant-growth property (4). DVTAG fulfills properties (3) and (4) as a direct consequence of the inclusion in LTAG. Moreover we have sketched the proof that $\mathcal{L}(CFG) \subset \mathcal{L}(DVTAG)$, then DVTAG can generate all the context-free grammars. About (1) we need to introduce the wrapping perspective in DVTAG (next section).

[5]We can define a procedure that takes as input a dLTAG and returns a LTAG that using *selective adjoining constraints* generates the same tree language.

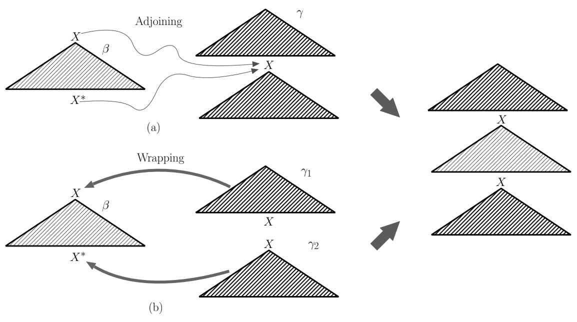

FIGURE 5: Adjoining operation (a) and wrapping operation (b).

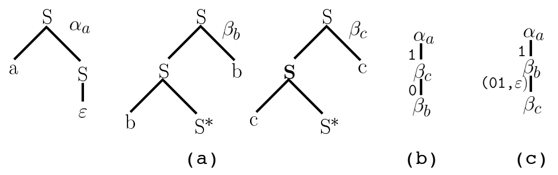

FIGURE 6: Derivation tree for the sentence "abcbc" in adjoining (b) and wrapping (c) perspective.

15.5 Wrapping and DVTAG generative power

Figure 5 shows the two perspectives for the adjoining operation, either inserting an auxiliary tree β into γ (Fig. 5-a) or splitting γ into γ_1 and γ_2 to include β (*wrapping perspective*, Fig. 5-b) (Joshi, 2004). If γ is an elementary tree, the wrapping operation does not change the weak and strong generative power of LTAG. However, since the wrapping operation can generate a wider set of derivation tree, on the subclass of dLTAG it does increase the strong generative power. A LTAG derivation that is not a dynamic derivation in standard LTAG can be a dynamic derivation in LTAG with wrapping operation. The LTAG elementary trees in Fig. 6-a generate the cross-serial dependencies (Joshi, 2004), but the derivation tree of Fig. 6-b for the string "abcbc" is not dynamic. As a consequence, we cannot derive the cross-serial dependencies in DVTAG. But if we rewrite this derivation tree by using wrapping, we obtain the tree of figure 6-c. In fact the adjoining of β_b in β_c becomes the wrapping of β_c in β_b. We can define a *dynamic wrapping* in DVTAG that maintains the strong equivalence between dynamic LTAG with wrapping and DVTAG with dynamic wrapping, and so we can produce a derivation chain that derives cross-serial dependencies. Then DVTAG is mildly context-sensitive.

15.6 Conclusions

In this paper we have defined a dynamic version of TAG. We have sketched the basic issues of the formalism, and using these notions we have proved that DVTAG is mildly context sensitive. We have shown that DVTAG can generate cross-serial dependencies using a dynamic wrapping.

References

Abney, S. P. and M. Johnson. 1991. Memory requirements and local ambiguities of parsing strategies. *J. of Psycholinguistic Research* 20(3):233–250.

Gelder, T. Van. 1998. The dynamical hypothesis in cognitive science. *Behavioral and Brain Sciences* 21:1–14.

Hausser, R. 1992. Complexity in left-associative grammar. *Theoretical Computer Science* 106:283–308.

Joshi, A. 2004. Starting with complex primitives pays off: Complicate locally, simplify globally. *Cognitive Science* 28(5):637–668.

Joshi, A. and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, pages 69–123. Springer.

Kamide, Y., G. T. M. Altmann, and S. L. Haywood. 2003. The time-course of prediction in incremental sentence processing: Evidence from anticipatory eye movements. *J. of Memory and Language Language* 49:133–156.

Mazzei, A. 2005. *Formal and empirical issues of applying dynamics to Tree Adjoining Grammars*. Ph.D. thesis, Dipartimento di Informatica, Università degli studi di Torino.

Milward, D. 1994. Dynamic dependency grammar. *Linguistics and Philosophy* 17(6):561–604.

Schabes, Y. and R. Waters. 1995. Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes Context-free grammar without changing the trees produced. *Computational Linguistics* 21(4):479–513.

Stabler, E. P. 1994. The finite connectivity of linguistic structure. In C. Clifton, L. Frazier, and K. Reyner, eds., *Perspectives on Sentence Processing*, pages 303–336. Lawrence Erlbaum Associates.

Sturt, P. and V. Lombardo. 2005. Processing coordinated structures: Incrementality and connectedness. *Cognitive Science* 29(2):291–305.

Vijay-Shanker, K., A. Joshi, and D. Weir. 1990. The convergence of mildly context-sensitive grammatical formalisms. In P. Sells, S. Shieber, and T. Wasow, eds., *Foundational Issues in Natual Language Processing*, pages 31–81. Cambridge MA: MIT Press.

Woods, W. A. 1986. Transition network grammars for natural language analysis. In B. J. Grosz, K. Sparck Jones, and B. L. Webber, eds., *Natural Language Processing*, pages 71–87. Los Altos, CA: Kaufmann.