Synchronous Grammars as Tree Transducers

Stuart M. Shieber
Division of Engineering and Applied Sciences
Harvard University
shieber@deas.harvard.edu

Abstract

Tree transducer formalisms were developed in the formal language theory community as generalizations of finite-state transducers from strings to trees. Independently, synchronous tree-substitution and -adjoining grammars arose in the computational linguistics community as a means to augment strictly syntactic formalisms to provide for parallel semantics. We present the first synthesis of these two independently developed approaches to specifying tree relations, unifying their respective literatures for the first time, by using the framework of bimorphisms as the generalizing formalism in which all can be embedded. The central result is that synchronous tree-substitution grammars are equivalent to bimorphisms where the component homomorphisms are linear and complete.

1 Motivation

The typical natural-language pipeline can be thought of as proceeding by successive transformation of various data structures, especially strings and trees. For instance, low-level speech processing can be viewed as transduction of strings of speech samples into phoneme strings, then into triphone strings, finally into words strings. (Because of nondeterminism in the process, the nondeterministic string possibilities may be represented as a single lattice. Nonetheless, the underlying abstract operation is one of string transduction.) Morphological processes can similarly be modeled as character string transductions. For this reason, weighted finite-state transducers (WFST), a general formalism for string-to-string transduction, can serve as a kind of universal formalism for representing low-level natural-language processes (Mohri, 1997).

Higher-level natural-language processes can also be thought of as transductions, but on more highly structured representations, for instance trees. Semantic interpretation can be viewed as a transduction from a syntactic parse tree to a tree of semantic operations whose simplification to logical form can be viewed as a further transduction. This raises the question as to whether there is a universal formalism for NL tree transductions that can play the same role there that WFST plays for string transduction.

In this paper, we investigate the formal properties of synchronous tree-substitution and -adjoining grammars (STSG and STAG) from this perspective. In particular, we look at where the formalisms sit in the pantheon of tree transduction formalisms. As a particular result, we show that, contra previous conjecture, STSG is not equivalent to simple nondeterministic tree transducers, and place for the first time STSG and STAG into the tree transducer family. Essential to this unification of the two types of formalisms is the bimorphism characterization of tree transducers, little known outside the formal language theory community.

We begin by recalling the definitions of nondeterministic top-down tree transducers ($\downarrow TT$), and their description in terms of bimorphisms, and also provide a definition of STSG and STAG. We show that $\downarrow TT$ and STSG differ in their expressive properties; these differences argue in favor of the synchronous formalisms for NL use. Finally, we prove the equivalence between STSG and a new kind of bimorphism, which characterization makes some of the properties of STSG trivial. This view of STSG generalizes to provide a bimorphism characterization of STAG as well.

This work makes several contributions to our understanding of tree transducers and the synchronous formalisms. First, it provides the first unification of the two, placing both in a consistent framework, that of bimorphisms. Second, it provides intuition about appropriate properties of such formalisms for the purpose of natural-language processing applications, which may help inform the search for a universal NL tree transduction formalism.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 88-95.

2 Preliminaries

We start by defining the terminology and notations that we will use for strings, trees, and the like.

We will notate sequences with angle brackets, e.g., $\langle a, b, c\rangle$, with the empty string written ϵ. The number of elements in a set or sequence x will be notated $|x|$.

Trees will have nodes labeled with elements of a RANKED ALPHABET, a set of symbols $\mathcal{F}$, each with a non-negative integer RANK or ARITY assigned to it, say by a function $arity$, determining the number of children for nodes so labeled. Symbols with arity zero are called NULLARY symbols; with arity one, UNARY; with arity two, BINARY. We write $\mathcal{F}_n$ for the set of symbols in $\mathcal{F}$ with arity n. To express incomplete trees, trees with "holes" waiting to be filled, we will allow leaves to be labeled with variables, in addition to nullary symbols.

The set of TREES OVER A RANKED ALPHABET $\mathcal{F}$ AND VARIABLES $\mathcal{X}$, notated $\mathcal{T}(\mathcal{F}, \mathcal{X})$, is the smallest set such that

Nullary symbols at leaves $f \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}_0$;

Variables at leaves $x \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $x \in \mathcal{X}$;

Internal nodes $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}_n$, $n \geq 1$, and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

We abbreviate $\mathcal{T}(\mathcal{F}, \emptyset)$, where the set of variables is empty, as $\mathcal{T}(\mathcal{F})$, the set of GROUND TREES over $\mathcal{F}$. We will also make use of the set of n numerically ordered variables $\mathcal{X}_n = \{x_1, \ldots, x_n\}$, and write x, y, z as synonyms for x_1, x_2, x_3, respectively.

Trees can also be viewed as mappings from TREE ADDRESSES, sequences of integers, to the labels of nodes at those addresses. The address ϵ is the address of the root, $\langle 1\rangle$ the address of the first child, $\langle 1, 2\rangle$ the address of the second child of the first child, and so forth. We will use the notation $t@p$ to pick out the label of the node at address p in the tree t, that is, (using $\cdot$ for the insertion of an element on a list)

$$\begin{aligned} f(t_1, \ldots, t_n)@\epsilon &= f \\ f(t_1, \ldots, t_n)@(i \cdot p) &= t_i@p \\ &\text{for } 1 \leq i \leq n \quad . \end{aligned}$$

We can use trees with variables as CONTEXTS in which to place other trees. A tree in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ will be called a context, typically denoted with the symbol C. The notation $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ denotes the tree in $\mathcal{T}(\mathcal{F})$ obtained by substituting for each x_i the corresponding t_i.

For a context $C \in \mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ and a sequence of n trees $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, the SUBSTITUTION OF $t_1, \ldots, t_n$ INTO C, notated $C[t_1, \ldots, t_n]$, is defined inductively as follows:

$$\begin{aligned} &(f(u_1, \ldots, u_m))[t_1, \ldots, t_n] \\ &\quad = f(u_1[t_1, \ldots, t_n], \ldots, u_m[t_1, \ldots, t_n]) \\ &x_i[t_1, \ldots, t_n] = t_i \quad . \end{aligned}$$

A tree $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is LINEAR if and only if no variable in $\mathcal{X}$ occurs more than once in t.

3 Tree Transducers and Bimorphisms

The variation in tree transducer formalisms is extraordinarily wide and the literature vast. For the purpose of this paper, we restrict attention to simple nondeterministic tree transducers operating top-down, which transform trees by replacing each node with a subtree as specified by the label of the node and the state of the transduction at that node.

A NONDETERMINISTIC TOP-DOWN TREE TRANSDUCER ($\downarrow TT$) is a tuple $\langle Q, \mathcal{F}_{in}, \mathcal{F}_{out}, \Delta, q_0\rangle$ where

- Q is a finite set of STATES;
- $\mathcal{F}_{in}$ is a ranked alphabet of INPUT SYMBOLS;
- $\mathcal{F}_{out}$ is a ranked alphabet of OUTPUT SYMBOLS;
- Δ is a set of TRANSITIONS each of the form

  $$q(f(x_1, \ldots, x_n)) \to C[q_1(x_1), \ldots, q_n(x_n)]$$

  for some $f \in \mathcal{F}_{in}$ of arity n, $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n \in \mathcal{X}_n$, and $C \in \mathcal{T}(\mathcal{F}_{out}, \mathcal{X}_n)$;
- $q_0 \in Q$ is a distinguished INITIAL STATE.

Given a tree transducer $\langle Q, \mathcal{F}_{in}, \mathcal{F}_{out}, \Delta, q_0\rangle$ and two trees $t \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out} \cup Q)$ and $t' \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out} \cup Q)$, tree t DERIVES t' IN ONE STEP, notated $t \vdash t'$ if and only if there is a transition $u \to u' \in \Delta$ with $u \in \mathcal{T}(\mathcal{F}_{in} \cup Q, \mathcal{X}_n)$ and $u' \in \mathcal{T}(\mathcal{F}_{out} \cup Q, \mathcal{X}_n)$ and trees $C \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out} \cup Q, \mathcal{X}_1)$ and $u_1, \ldots, u_n \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out})$, such that

$$t = C[u[u_1, \ldots, u_n]]$$

and

$$t' = C[u'[u_1, \ldots, u_n]] \quad .$$

The TREE RELATION defined by a $\downarrow TT$ $\langle Q, \mathcal{F}_{in}, \mathcal{F}_{out}, \Delta, q_0\rangle$ is the set of all tree pairs $\langle s, t\rangle \in \mathcal{T}(\mathcal{F}_{in}) \times \mathcal{T}(\mathcal{F}_{out})$ such that $q_0(s) \vdash^* t$.

For instance, the following rules specify a transducer that "rotates" subtrees of the form $f(t_1, f(t_2, t_3))$ to the tree $f(f(t_1, t_2), t_3)$. (By convention, we take the left-hand state of the first rule as the start state for the transducer.)

$$\begin{aligned} q(f(x, y)) &\to f(f(q(x), q_1(y)), q_2(y)) \\ q_1(f(x, y)) &\to q(x) \\ q_2(f(x, y)) &\to q(y) \\ q(a) &\to a \\ q(b) &\to b \end{aligned}$$

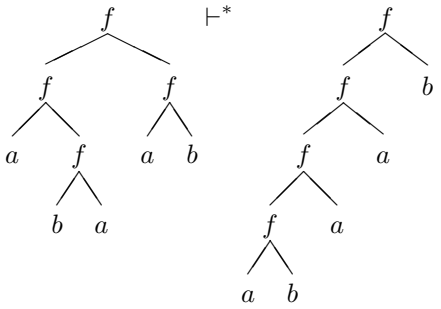

Figure 1: Local rotation computed by a nonlinear tree transducer

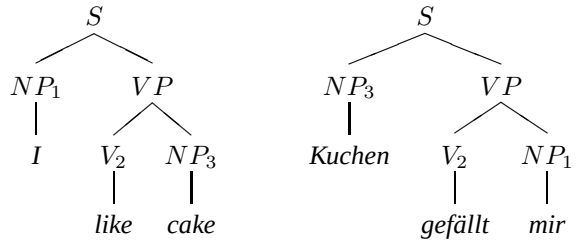

Figure 2: Example of local rotation in language translation divergence. Corresponding nodes are marked with matched subscripts.

The tree $f(f(a, f(b, a)), f(a, b))$ is transduced to $f(f(f(f(a, b), a), a), b)$ (as depicted graphically in figure 1) according to the following derivation:

$$
\begin{aligned}
& q(f(f(a, f(b, a)), f(a, b))) \\
& \quad \vdash \quad f(f(q(f(a, f(b, a))), q_1(f(a, b))), q_2(f(a, b))) \\
& \quad \vdash \quad f(f(f(\ f(q(a), q_1(f(b, a))), \\
& \qquad\qquad\quad q_2(f(b, a)) \ ), q(a)), q(b)) \\
& \quad \vdash \quad f(f(f(f(a, q(b)), q(a)), a), b) \\
& \quad \vdash \quad f(f(f(f(a, b), a), a), b)
\end{aligned}
$$

3.1 Nonlinearity Deprecated

Note that intrinsic use is made in this example of the ability to duplicate variables on the right-hand sides of rewrite rules. Transducers without such duplication are *linear*. Linear tree transducers are incapable of performing local rotations of this sort.

Local rotations are typical of natural-language applications. For instance, many of the kinds of translation divergences between languages, such as that exemplified in Figure 2, manifest such rotations. Similarly, semantic bracketing paradoxes can be viewed as necessitating rotations. Thus, linear tree transducers are insufficient for NL modeling purposes.

Nonlinearity per se, the ability to make copies during transduction, is not the kind of operation that is characteristic of natural-language phenomena. Furthermore, nonlinear transducers are computationally problematic. The following nonlinear transducer generates a perfect binary tree whose height is identical to that of its single-strand input.

$$
\begin{array}{lcl}
q(f(x)) & \rightarrow & g(q(x), q(x)) \\
q(a) & \rightarrow & a
\end{array}
$$

For instance, the tree of height and size four, $f(f(f(a)))$, transduces to $g(g(g(a, a), g(a, a)), g(g(a, a), g(a, a)))$, of height four but with fifteen symbols. The size of this transducer's output is exponential in the size of its input. (The existence of such a transducer constitutes a simple proof of the lack of composition closure of tree transducers, as the exponential of an exponential grows faster than exponential.)

In summary, nonlinearity seems inappropriate on computational and linguistic grounds, yet is required for tree transducers to express the kinds of simple local rotations that are typical of natural-language transductions. By contrast, STSG, as described below, is intrinsically a linear formalism but can express rotations straightforwardly.

3.2 Tree Automata and Homomorphisms

Two subcases of tree transducers are especially important. First, tree transducers that implement the identity relation over their domain are TREE AUTOMATA. A tree is in the language specified by a tree automaton if it is transduced to itself by the automaton. The tree languages so recognized are the regular tree languages (or recognizable tree languages), and are coextensive with those definable by context-free grammars. We take tree automata to be quadruples by dropping one of the redundant alphabets from the corresponding tree transducer quintuple.

Second, TREE HOMOMORPHISMS are essentially tree transducers with only a single state, so that the replacement of a node by a subtree proceeds independently of its context. A homomorphism $h : \mathcal{T}(\mathcal{F}_{in}) \rightarrow \mathcal{T}(\mathcal{F}_{out})$ is specified by its kernel, a function $\hat{h} : \mathcal{F}_{in} \rightarrow \mathcal{T}(\mathcal{F}_{out}, \mathcal{X}_{\infty})$ such that $\hat{h}(f)$ is a tree in $\mathcal{T}(\mathcal{F}_{out}, \mathcal{X}_{arity(f)})$ for each symbol $f \in \mathcal{F}_{in}$. The kernel $\hat{h}$ is extended to the homomorphism h by the following recurrence:

$$h(f(t_1, \ldots, t_n)) = \hat{h}(f)[h(t_1), \ldots, h(t_n)]$$

that is, $\hat{h}(f)$ acts as a context in which the homomorphic images of the subtrees are substituted. Further restrictions can be imposed: A tree homomorphism h is LINEAR if $\hat{h}(f)$ is linear for all $f \in \mathcal{F}_{in}$; is COMPLETE if $\hat{h}(f)$ contains every variable in $\mathcal{X}_{arity(f)}$ for all $f \in \mathcal{F}_{in}$; is ϵ-FREE if $\hat{h}(f) \notin \mathcal{X}_{arity(f)}$ for all $f \in \mathcal{F}_{in}$;

is SYMBOL-TO-SYMBOL if $\hat{h}(f)$ has exactly one symbol, for all $f \in \mathcal{F}_{in}$; and is a DELABELING if h is complete, linear, and symbol-to-symbol.

The import of these two subcases of tree transducers lies in the fact that the tree relations definable by tree transducers have been shown also to be characterizable by composition from these simplified forms, via an alternate quite distinct formalization based on bimorphisms. A BIMORPHISM is a triple $\langle L, h_{in}, h_{out}\rangle$ consisting of a regular tree language and two tree homomorphisms. The tree relation defined by a bimorphism consists of all pairs of trees generable by applying the homomorphisms to elements of the tree language, that is, $\{\langle h_{in}(t), h_{out}(t)\rangle \mid t \in L\}$. Depending on the type of tree homomorphisms used in the bimorphism, different classes of tree relations are defined. In particular, if we restrict h_{in} to be a delabeling, the tree relations defined are exactly those definable by $\uparrow TT$. As a convenient notation for bimorphisms, we write $B(X, Y)$ for the class of bimorphisms where h_{in} is restricted to have property X and h_{out} to have property Y. We use the following abbrevations for the properties: L[inear], C[omplete], [ϵ-]F[ree], S[ymbol-to-symbol], D[elabeling], M[orphism without restriction]. Thus the tree relations $B(D, M)$ are exactly those definable by $\uparrow TT$. (See the survey by Comon et al. (1997) and works cited therein.) Though many classes of bimorphisms have been studied, to our knowledge, the class $B(LC, LC)$ investigated below has not.

4 Synchronous Grammars and Bimorphisms

Tree-substitution grammars are composed of a set of elementary trees over a nonterminal and terminal vocabulary, allowing for nonterminal nodes at the leaves at which substitution of other elementary trees can occur (SUBSTITUTION NODES). They can be thought of as tree-adjoining grammars with substitution but no adjunction (hence no auxiliary trees). A synchronous tree-substitution grammar extends a tree-substitution grammar with the synchronization idea presented by Shieber (1992). In particular, grammars are composed of pairs of elementary trees, and pairs of substitution nodes, one from each tree in a pair, are linked to indicate that substitution of trees from a single elementary pair must occur at the linked nodes.

4.1 Tree-Substitution Grammars

A TREE-SUBSTITUTION GRAMMAR (TSG) comprises a set of ELEMENTARY TREES over a ranked alphabet $\mathcal{F}$, where certain frontier nonterminal (non-zero arity) nodes are marked as sites of substitution. The ability to have such nonterminal nodes with no children means that we must augment the definition of well-formed trees. We define the set of SUBSTITUTABLE TREES OVER A RANKED ALPHABET $\mathcal{F}$, notated $\mathcal{T}_{\downarrow}(\mathcal{F})$ as the smallest set such that

Nullary symbols at leaves $f \in \mathcal{T}_{\downarrow}(\mathcal{F})$ for all $f \in \mathcal{F}_0$;

Substitution nodes at leaves $f_{\downarrow} \in \mathcal{T}_{\downarrow}(\mathcal{F})$ for all $f \in F_n, n > 0$;

Internal nodes $f(t_1, \ldots, t_n) \in \mathcal{T}_{\downarrow}(\mathcal{F})$ for all $f \in \mathcal{F}_n$, $n \geq 1$, and $t_1, \ldots, t_n \in \mathcal{T}_{\downarrow}(\mathcal{F})$.

The marker $\downarrow$ marks the substitution nodes. In order to refer to the substitution nodes of a substitutable tree, we define the substitution paths of a tree t, $\downarrow\mathit{paths}(t)$ to comprise the paths to substitution nodes in t.

A tree-substitution grammar, then, is a triple, $\langle \mathcal{F}, P, S\rangle$ where $\mathcal{F}$ is a ranked alphabet comprising the vocabulary of the grammar, $S \in \mathcal{F}$ is the start symbol of the grammar, and $P \subseteq \mathcal{T}_{\downarrow}(\mathcal{F})$ is a set of elementary trees. In order to allow reference to a particular tree in the set P, we associate with each tree in P a unique index, conventionally notated with a subscripted α. This further allows us to have multiple instances of a tree in P, distinguished by their index. (We will abuse notation by using the index and the tree that it names interchangably.) Furthermore, we will assume that each grammar comes with an arbitrary ordering on the substitution node paths of a tree α_i, notating this permutation of $\downarrow\mathit{paths}(\alpha_i)$ by $\overline{\downarrow\mathit{paths}}(\alpha_i)$. We use this to mandate the child ordering of the children in derivation trees.

As a simple example, we consider the grammar with three elementary trees

$$\begin{array}{ll} \alpha_1 & S(NP_{\downarrow}, VP(V(like), NP_{\downarrow})) \\ \alpha_2 & NP(I) \\ \alpha_3 & NP(cake) \end{array}$$

and start symbol S. The arities of the symbols should be clear from their usage.

A DERIVATION for a grammar $G = \langle \mathcal{F}, P, S\rangle$ is a tree whose nodes are labeled with (indexes of) elementary trees, that is, a tree D in $\mathcal{T}(P)$, satisfying the following conditions:

1. For each node α in the tree D with substitution paths $\overline{\downarrow\mathit{paths}}(\alpha) = \langle p_1, \ldots, p_n\rangle$, the node must have n immediate children $\alpha_1, \ldots, \alpha_n$.

2. The root node of each child tree must match the corresponding substitution node in the parent, that is,

$$\alpha @ p_i = (\alpha_i @ \epsilon)_{\downarrow} \tag{1}$$

for all i, $1 \leq i \leq n$.

3. The tree α_r at the root of the derivation tree must be labeled at its root by the start symbol, that is, $\alpha_r @ \epsilon = S$.

For example, the derivation tree $\alpha_1(\alpha_3, \alpha_2)$ is a well-formed derivation tree for the sample grammar above, assuming that $\overline{\downarrow \mathit{paths}}(\alpha_1) = \langle\langle 2,2\rangle, \langle 1\rangle\rangle$. Note, for instance, that $\alpha_1 @ \langle 2,2\rangle = NP = \alpha_3 @ \epsilon$.

The derived tree for a derivation tree D is generated by performing all of the requisite substitutions. This can be defined directly, but to highlight the relationship with homomorphisms, we define it by mapping the substitutable trees into contexts, using a homomorphism kernel $\hat{h}_D$. For each tree $\alpha \in P$, with $\overline{\downarrow \mathit{paths}}(\alpha) = \langle p_1, \ldots, p_n\rangle$, $\hat{h}_D(\alpha)$ is the tree generated by replacing each node at address p_i by the variable x_i. For example, the context corresponding to the elementary tree $S(NP_\downarrow, VP(V(like), NP_\downarrow))$ with respect to the assumed substitution path ordering $\langle\langle 2,2\rangle, \langle 1\rangle\rangle$ is $S(x_2, VP(V(like), x_1))$. Because the substitution nodes of a tree all occur at its frontier, $\hat{h}_D(\alpha)$ is always a tree in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$, and by construction is linear and complete. Hence, the associated homomorphism h_D is also linear and complete.

We define the derived tree corresponding to a derivation tree D as the application of this homomorphism to D, that is $h_D(D)$. For the example above, the derived tree is that shown in Figure 2(a):

$$\begin{aligned} & h_D(\alpha_1(\alpha_3, \alpha_2)) \\ &= \hat{h}_D(\alpha_1)[h_D(\alpha_3), h_D(\alpha_2)] \\ &= S(x_2, VP(V(like), x_1))[\alpha_3, \alpha_2] \\ &= S(NP(I), VP(V(like), NP(cake))) \end{aligned}$$

4.2 Synchronous Tree-Substitution Grammars

We perform synchronization of tree-substitution grammars as per the approach taken for synchronizing tree-adjoining grammars in earlier work (Shieber, 1992). Synchronous grammars consist of pairs of elementary trees with a linking relation between nodes in one tree and nodes in the other. Simultaneous composition operations occur at linked nodes. In the case of synchronous tree-substitution grammars, the composition operation is substitution, so the linked nodes are substitution nodes.

We define a synchronous tree-substitution grammar, then, as a quintuple $G = \langle \mathcal{F}_{in}, \mathcal{F}_{out}, P, S_{in}, S_{out}\rangle$, where

- $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ are the input and output ranked alphabets, respectively,
- $S_{in} \in \mathcal{F}_{in}$ and $S_{out} \in \mathcal{F}_{out}$ are the input and output start symbols, and
- P is a set of elementary linked tree pairs, each of the form $\langle t, t', \frown\rangle$, where $t \in \mathcal{T}_\downarrow(\mathcal{F}_{in})$ and $t' \in \mathcal{T}_\downarrow(\mathcal{F}_{out})$ are input and output substitutable trees and $\frown \subseteq \downarrow\mathit{paths}(t) \times \downarrow\mathit{paths}(t')$ is a relation over substitution nodes from the two trees.

In order to guarantee that derivations for the synchronized grammars are isomorphic, we need to impose consistent orderings on the substitution nodes for paired trees. We therefore choose an arbitrary ordering $\langle p_{in,1} \frown p_{out,1}, \ldots, p_{in,n} \frown p_{out,n}\rangle$ over the linked pairs, and take $\overline{\downarrow \mathit{paths}}(t) = \langle p_{in,1}, \ldots, p_{in,n}\rangle$ and $\overline{\downarrow \mathit{paths}}(t') = \langle p_{out,1}, \ldots, p_{out,n}\rangle$.

We define $G_{in} = \langle \mathcal{F}_{in}, P_{in}, S_{in}\rangle$ where $P_{in} = \{t \mid \langle t, t', \frown\rangle \in P\}$; this is the left projection of the synchronous grammar onto a simple TSG. The right projection G_{out} can be defined similarly.

A synchronous derivation was originally defined as a pair $\langle D_{in}, D_{out}\rangle$ where (following Shieber (1992)):[1]

1. D_{in} is a well-formed derivation tree for G_{in}, and D_{out} is a well-formed derivation tree for G_{out}.
2. D_{in} and D_{out} are isomorphic.

The derived tree pair for a derivation $\langle D_{in}, D_{out}\rangle$ is then $\langle h_D(D_{in}), h_D(D_{out})\rangle$.

5 The Bimorphism Characterization of STSG

The central result we provide relating STSG to tree transducers is this: STSG is equivalent to $B(LC, LC)$. To show this, we must demonstrate that any STSG is reducible to a bimorphism, and vice versa.

5.1 Reducing STSG to $B(LC, LC)$

Given an STSG $G = \langle \mathcal{F}_{in}, \mathcal{F}_{out}, P, S_{in}, S_{out}\rangle$, we need to construct a bimorphism characterizing the same tree relation. All the parts are in place to do this. We start by recasting derivations as single derivation trees from which the left and right derivation trees can be projected via homomorphisms. Rather than taking a derivation to be a pair of isomorphic trees D_{in} and D_{out}, we take it to be the single tree D isomorphic to both, whose element at address p is $D @ p = \langle D_{in} @ p, D_{out} @ p\rangle$. Condition (2) on the well-formedness of a synchronous derivation thus being trivially satisfied, we simply need to require that the trees obtained by projecting this new derivation tree on its first and second elements are well-formed derivation trees in the projected TSGs. These projections D_{in} and D_{out} can be reconstructed by homomorphisms extending h_{in}

[1] In the earlier version, a third condition required that the isomorphic operations are sanctioned by links in tree pairs. This condition can be dropped here, as it follows from the previous definitions. In particular, since the substitution path orderings are chosen to be compatible, it follows that the isomorphic children of isomorphic nodes are substituted at linked paths.

that projects on the first component and h_{out} that projects on the second, respectively. These homomorphisms are trivially linear and complete (indeed, they are mere delabelings). Then the paired derived trees can be constructed as $h_D(h_{in}(D))$ and $h_D(h_{out}(D))$, respectively. Thus the mappings from the derivation tree to the derived trees are the compositions of two linear complete homomorphisms, hence linear complete homomorphisms themselves. We take the bimorphism characterizing the STSG tree relation to be $\langle L_D, h_D \circ h_{in}, h_D \circ h_{out} \rangle$ where L_D is the language of well-formed synchronous derivation trees.

To show that the language L_D is a regular tree language, we construct a top-down nondeterministic automaton $\langle Q_G, \mathcal{F}_G, \Delta_G, q_G \rangle$ recognizing it. The states of the automaton Q_G are elements of $\mathcal{F}_{in} \times \mathcal{F}_{out}$, expressing the allowable pair of symbols labeling the roots of the tree pair dominated by the state. The start state is $q_0 = \langle S_{in}, S_{out} \rangle$. The alphabet $\mathcal{F}_G$ of the trees is composed of pairs $\langle \alpha_{in}, \alpha_{out} \rangle$ of elementary trees, such that $\langle \alpha_{in}, \alpha_{out}, \frown \rangle \in P$, the arity of which is the number of substitution nodes in each tree, or equivalently, $|\frown|$. For each elementary tree pair $\langle \alpha_{in}, \alpha_{out}, \frown \rangle \in P$, where $\overline{\downarrow \mathit{paths}}(\alpha_{in}) = \langle p_1, \ldots, p_n \rangle$ and $\overline{\downarrow \mathit{paths}}(\alpha_{out}) = \langle r_1, \ldots, r_n \rangle$, there is a single transition in Δ_G of the form:

$$\begin{aligned}\langle \alpha_{in}@\epsilon, \alpha_{out}@\epsilon \rangle(\langle \alpha_{in}, \alpha_{out} \rangle(x_1, \ldots, x_n)) \\ \quad \rightarrow \langle \alpha_{in}, \alpha_{out} \rangle(\ \langle \alpha_{in}@p_1, \alpha_{out}@r_1 \rangle(x_1), \ldots, \\ \langle \alpha_{in}@p_n, \alpha_{out}@r_n \rangle(x_n)\ )\end{aligned}$$

We must verify that for any tree D recognized by this automaton $h_{in}(D)$ and $h_{out}(D)$ are well-formed derivation trees for their respective TSGs.

To show that $h_{in}(D)$ is a well-formed derivation tree (and symmetrically, for $h_{out}(D)$), we must demonstrate that the three definitional conditions hold. Consider a node in the tree of the form $\langle \alpha_{in}, \alpha_{out} \rangle$. This node must have been admitted by virtue of some transition of the form above.

1. By construction, there must be an elementary tree pair $\langle \alpha_{in}, \alpha_{out}, \frown \rangle \in P$, and the node must have n immediate children corresponding to $\overline{\downarrow \mathit{paths}}(\alpha_{in}) = \langle p_1, \ldots, p_n \rangle$.

2. Each child node, say the i-th, which we can notate $\langle \alpha_{in,i}, \alpha_{out,i} \rangle$, again by construction, must be admitted by a transition of the form $\langle \alpha_{in}@p_i, \alpha_{out}@r_i \rangle(\langle \alpha_{in,i}, \alpha_{out,i} \rangle(\cdots))$. Any matching transition enforces the requirement that $\langle \alpha_{in}@p_i, \alpha_{out}@r_i \rangle = \langle \alpha_{in,i}@\epsilon, \alpha_{out,i}@\epsilon \rangle$ hence that $\alpha_{in}@p_i = (\alpha_{in,i}@\epsilon)_{\downarrow}$ and $\alpha_{out}@r_i = (\alpha_{out,i}@\epsilon)_{\downarrow}$, as required.

3. Since the start state is $\langle S_{in}, S_{out} \rangle$, the root of the derivation tree must be a node $\langle \alpha_{in,r}, \alpha_{out,r} \rangle$ such that $\alpha_{in,r}@\epsilon = S_{in}$ and $\alpha_{out,r}@\epsilon = S_{out}$.

Thus, each of the two projection trees $h_{in}(D)$ and $h_{out}(D)$ are well-formed derivation trees for their respective grammars, and the tree relation defined by the STSG is in $B(LC, LC)$.

5.2 Reducing $B(LC, LC)$ to STSG

The other direction is somewhat trickier to prove, but can be done. Given a bimorphism $\langle L, h_{in}, h_{out} \rangle$ over input and output alphabets $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$, respectively, we construct a corresponding STSG $G = \langle \mathcal{F}'_{in}, \mathcal{F}'_{out}, P, S_{in}, S_{out} \rangle$. By "corresponding", we mean that the tree relation defined by the bimorphism is obtainable from the tree relation defined by the STSG via delabelings of the input and output that map $\mathcal{F}'_{in}$ to $\mathcal{F}_{in}$ and $\mathcal{F}'_{out}$ to $\mathcal{F}_{out}$. (Recall that delabelings are just many-to-one renamings of the symbols.)

As the language L is a regular tree language, it is generable by a nondeterministic top-down tree automaton $\langle Q, \mathcal{F}_d, \Delta, q_0 \rangle$. We use the states of this automaton in the input and output alphabets of the STSG. The input alphabet of the STSG is $\mathcal{F}'_{in} = \mathcal{F}_{in} \cup (Q \times \mathcal{F}_{in})$, composed of the input symbols of the bimorphism, along with some special symbols that pair states with the input symbols, and similarly for the output alphabet. The pair symbols mark the places in the tree where substitutions occur, allowing control for appropriate substitutions. In order to generate the trees actually related by the original bimorphism, the nodes labeled with such pairs can be projected on their second component by a simple delabeling.

The basic idea of the STSG construction is to construct an elementary tree pair for certain *sequences* of transitions from Δ. However, it is easiest understood by starting with the construction for the special case in which the homomorphisms are ϵ-free. In this case, as we will see, the pertinent sequences are just the single transitions. For the nonce, then, we assume h_{in} and h_{out} to be ϵ-free, relaxing this assumption later.

We define a simple nondeterministic transformation on trees in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ controlled by a sequence of $n+1$ states in Q:

$$\begin{aligned}\mathcal{C}(f(t_1, \ldots, t_k), q, q_1, \ldots, q_n) \\ = \{\langle q, f \rangle(t_1, \ldots, t_k)[\langle q_1, N_1 \rangle_{\downarrow}, \ldots, \langle q_n, N_n \rangle_{\downarrow}] \\ \mid N_1, \ldots, N_n \in \mathcal{F}\}\end{aligned}$$

In essence, the transformation replaces the root symbol by pairing it with the state q, and replaces the n variables with new pairs of a state q_i and an arbitrarily chosen symbol N_i. (The nondeterminism arises in the choice of the N_i.) These latter symbols are taken to be substitution

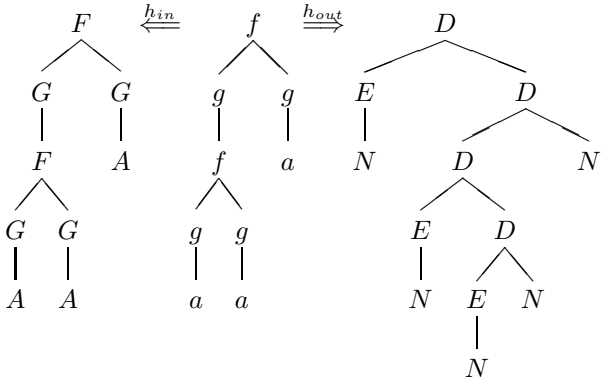

Figure 3: Example of bimorphism construction

nodes in the generated tree. Importantly, this transformation is partial; it applies to any tree in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$, with the exception of those trees that consist of a variable alone.

We use the transformation $\mathcal{C}$ to generate elementary tree pairs corresponding to transitions in Δ. For each transition $q(f(x_1, \ldots, x_n)) \rightarrow f(q_1(x_1), \ldots, q_n(x_n)) \in \Delta$, we construct the elementary tree pairs $\langle t_{in}, t_{out}, \frown \rangle$, where $t_{in} \in \mathcal{C}(\hat{h}_{in}(f), q, q_1, \ldots, q_n)$ and $t_{out} \in \mathcal{C}(\hat{h}_{out}(f), q, q_1, \ldots, q_n)$ and $\frown$ links the corresponding paths in the two trees, that is, the paths at which corresponding variables occur in the trees $\hat{h}_{in}(f)$ and $\hat{h}_{out}(f)$. Since h_{in} and h_{out} are linear and complete, this notion is well-defined. The applications of $\mathcal{C}$ are well-defined only when $\hat{h}_{in}(f)$ and $\hat{h}_{out}(f)$ are in the domain of $\mathcal{C}$, that is, it is not a lone variable, hence the requirement that h_{in} and h_{out} be ϵ-free.

An example may clarify the construction. Take the language of the bimorphism to be defined by the following two-state automaton:

$$\begin{array}{rcl} q(f(x,y)) & \rightarrow & f(q'(x), q'(y)) \\ q(a) & \rightarrow & a \\ q'(g(x)) & \rightarrow & g(q(x)) \end{array}$$

This automaton uses the states to alternate g's with f's and a's level by level. For instance, it admits the middle tree in Figure 3. With input and output homomorphisms defined by

$$\begin{array}{rclrcl} \hat{h}_{in}(f) & = & F(x,y) & \hat{h}_{out}(f) & = & D(y, D(x,N)) \\ \hat{h}_{in}(g) & = & G(x) & \hat{h}_{out}(g) & = & E(x) \\ \hat{h}_{in}(a) & = & A & \hat{h}_{out}(a) & = & N \end{array}$$

the bimorphism so defined generates the tree relation instance exemplified in the figure.

The construction given above generates the schematic elementary tree pairs in Figure 4 for this bimorphism. (The tree pairs are schematic in that we use a $*$ to stand for an arbitrary symbol in the appropriate alphabet.) The reader can verify that the grammar generates a tree pair whose delabeling is that shown in Figure 3 generated by the bimorphism.

Now, we turn to the considerably more subtle considerations of non-ϵ-free homomorphisms. In a linear complete homomorphism, the only possible case of non-ϵ-freeness that is possible is for unary function symbols, that is $\hat{h}(f) = x$, so that $h(f(x)) = h(x)$. Intuitively speaking, such cases in bimorphisms should (and will) correspond to STSG elementary trees that have just a single node, so that they contribute no structure to the derived trees.

If, for some symbol f, both h_{in} and h_{out} are non-ϵ-free, then any tree rooted in such a symbol, $f(t)$, is mapped, respectively, to $h_{in}(t)$ and $h_{out}(t)$. But in that case, we can eliminate the unary symbol f, eliminating transitions in the automaton of the form $q(f(x)) \rightarrow f(q'(x))$ by adding, for all transitions with q' on the left hand side, identical transitions with q on the left-hand side. We then construct the STSG for the simplified automaton.

The situation is more complicated if only one of the two homomorphisms, say h_{in}, is non-ϵ-free. In this case, we have that $h_{in}(f(x)) = h_{in}(x)$ but $h_{out}(f(x)) = C[h_{out}(x)]$ for nontrivial context C, thus introducing structure on the output with no corresponding structure on the input. We will call such a unary symbol ASYMMETRIC. A sequence of asymmetric symbols can introduce unbounded amounts of material on the output with no corresponding material on the input (or vice versa). The key is thus to construct all possible such sequences of asymmetric symbols and chop them into a bounded set of minimal cycles, using these to generate single elementary tree pairs. We arrange that in such cycles, the state and symbol at the root will be identical to the state and symbol at the end of the sequence. For example, suppose we have asymmetric symbols f and g and an ϵ-free symbol k with the following automaton transitions:

$$\begin{array}{rcl} q(k(x)) & \rightarrow & k(q(x)) \\ q(f(x)) & \rightarrow & f(q(x)) \\ q(g(x)) & \rightarrow & g(q'(x)) \\ q'(f(x)) & \rightarrow & f(q'(x)) \\ q'(f(x)) & \rightarrow & f(q''(x)) \\ q'(g(x)) & \rightarrow & g(q'(x)) \\ q''(k(x)) & \rightarrow & k(\ldots) \end{array}$$

There is a minimal cycle such that $q'(f(g(f(x)))) = f(g(q'(f(x))))$. Note that the state q' and symbol f at the root are duplicated at the bottom. There is a similar cycle of the form $q'(f(f(x))) = f(q'(f(x)))$. For each such cycle, we construct a linked tree pair with a trivial input tree labeled with a pair of the state and an arbitrary symbol N from the input alphabet—$\langle q', N \rangle$ in

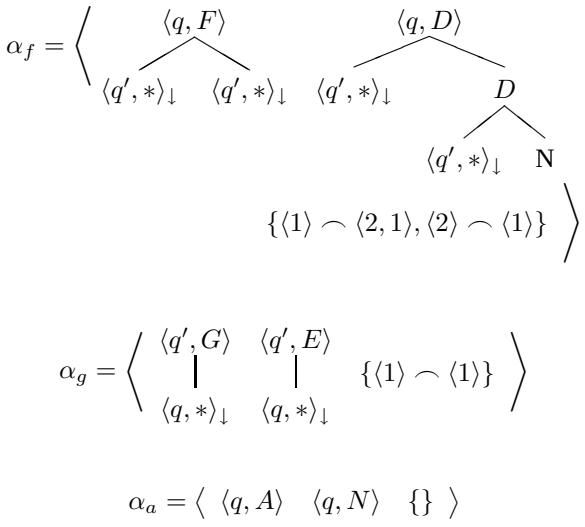

Figure 4: Generated STSG for example bimorphism

the example. The corresponding output tree is generated by composing the nontrivial output trees and applying $\mathcal{C}$ to this compound tree in the obvious way. Since the path language in the tree language of a tree automaton is regular, a decomposition of the paths into a bounded number of bounded-length cycles can always be done, leading to a finite number of elementary tree pairs. Note that since the label of the root for the appropriate input tree $\langle q', f\rangle$ is identical to the label to replace the (single) variable, the tree pair is constructed in a way consistent with $\mathcal{C}$, hence the workings of the rest of the STSG.

In addition, for each minimal sequence starting with a symbol that is non-ϵ-free on the input and leading to such a cyclic state/symbol pair, a tree pair is similarly generated. In the example, the sequence corresponding to the automaton subderivation $q(k(f(g(f(x))))) = k(f(g(q'(f(x)))))$ would lead us to generate a tree pair with $\langle \mathcal{C}(\hat{h}_{in}(k), q, q'), \mathcal{C}(\hat{h}_{in}(k)[\hat{h}_{out}(f)[\hat{h}_{out}(g)]], q, q'), \frown\rangle$ where $\frown$ links the two leaf nodes labeled with state/symbol pairs.

Similarly, we require elementary tree pairs corresponding to minimal tails of sequences of asymmetric symbols starting in a cyclic state/symbol pair and ending in a symbol non-ϵ-free on the input. These three types of sequences can be pieced together to form any possible sequence of unary symbols admitted by the automaton, and the corresponding tree pairs correspond to the compositions of the homomorphism trees.

6 Discussion

By placing STSG in the class of bimorphisms, which have already been used to characterize tree transducers, we provide the first synthesis of these two independently developed approaches to specifying tree relations, unifying their respective literatures for the first time. The relation between a TAG derivation tree and its derived tree is not a mere homomorphism. The appropriate morphism generalizing linear complete homomorphisms to allow adjunction can presumably be used to provide a bimorphism characterization of STAG as well, further unifying these strands of research.

The bimorphism characterization of STSG has immediate application. First, the symmetry of the tree relations defined by an STSG is a trivial corollary. Second, it has been claimed in passing that synchronous tree-substitution grammars are "equivalent to top-down tree transducers." (Eisner, 2003). This is clearly contravened by the distinction between $B(LC, LC)$ and $B(D, M)$. Third, the bimorphism characterization of tree transducers has led to a series of composition closure results. Similar techniques may now be applicable to synchronous formalisms, where no composition results are known. For instance, the argument for the lack of composition closure in $B(LCF, LCF)$ (Arnold and Dauchet, 1982) may be directly applicable to a similar proof for $B(LC, LC)$, hence for STSG; the conjecture remains for future work.

References

A. Arnold and M. Dauchet. 1982. Morphismes et bimorphismes d'arbres. *Theoretical Computer Science*, 20(1):33–93, March.

H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 1997. Tree automata techniques and applications. Available at: `http://www.grappa.univ-lille3.fr/tata`. release of October 1, 2002.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, July.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Stuart M. Shieber. 1992. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. In *Proceedings of the Second TAG Workshop*, University of Pennsylvania, Philadelphia, Pennsylvania, June 24–26.