Unifying Synchronous Tree-Adjoining Grammars and Tree Transducers via Bimorphisms

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

Abstract

We place synchronous tree-adjoining grammars and tree transducers in the single overarching framework of bimorphisms, continuing the unification of synchronous grammars and tree transducers initiated by Shieber (2004). Along the way, we present a new definition of the tree-adjoining grammar derivation relation based on a novel direct inter-reduction of TAG and monadic macro tree transducers.

Tree transformation systems such as tree transducers and synchronous grammars have seen renewed interest, based on a perceived relevance to new applications, such as importing syntactic structure into statistical machine translation models or founding a formalism for speech command and control.

The exact relationship among a variety of formalisms has been unclear, with a large number of seemingly unrelated formalisms being independently proposed or characterized. An initial step toward unifying the formalisms was taken (Shieber, 2004) in making use of the formal-language-theoretic device of bimorphisms, previously used to characterize the tree relations definable by tree transducers. In particular, the tree relations definable by synchronous tree-substitution grammars (STSG) were shown to be just those definable by linear complete bimorphisms, thereby providing for the first time a clear relationship between synchronous grammars and tree transducers.

In this work, we show how the bimorphism framework can be used to capture a more powerful formalism, synchronous tree-adjoining grammars, providing a further uniting of the various and disparate formalisms.

After some preliminaries (Section 1), we begin by recalling the definition of tree-adjoining grammars and synchronous tree-adjoining grammars (Section 2). We turn then to a set of known results relating context-free languages, tree homomorphisms, tree automata, and tree transducers to extend them for the tree-adjoining languages (Section 3), presenting these in terms of restricted kinds of functional programs over trees, using a simple grammatical notation for describing the programs. This allows us to easily express generalizations of the notions: monadic macro tree homomorphisms, automata, and transducers, which bear (at least some of) the same interrelationships that their traditional simpler counterparts do (Section 4). Finally, we use this characterization to place the synchronous TAG formalism in the bimorphism framework (Section 5), further unifying tree transducers and other synchronous grammar formalisms. We also, in passing, provide a new characterization of the relation between TAG derivation and derived trees, and a new simpler and more direct proof of the equivalence of TALs and the output languages of monadic macro tree transducers.

1 Preliminaries

We will notate sequences with angle brackets, e.g., $\langle a, b, c \rangle$, or where no confusion results, simply as abc, with the empty string written ε.

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, determining the number of children for nodes so labeled. To emphasize the arity of a symbol, we will write it as a parenthesized superscript, for instance $f^{(n)}$ for a symbol f of arity n. Analogously, we write $\mathcal{F}^{(n)}$ for the set of symbols in $\mathcal{F}$ with arity n. Symbols with arity zero ($\mathcal{F}^{(0)}$) are called NULLARY symbols or CON-

STANTS. The set of nonconstants is written $\mathcal{F}^{(\geq 1)}$.

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 (i) $f \in \mathcal{T}(\mathcal{F},\mathcal{X})$ for all $f \in \mathcal{F}^{(0)}$; (ii) $x \in \mathcal{T}(\mathcal{F},\mathcal{X})$ for all $x \in \mathcal{X}$; and (iii) $f(t_1,\ldots,t_n) \in \mathcal{T}(\mathcal{F},\mathcal{X})$ for all $f \in \mathcal{F}^{(\geq 1)}$, 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, 1 the address of the first child, 12 the address of the second child of the first child, and so forth. We will use the notation t/p to pick out the subtree of the node at address p in the tree t. Replacing the subtree of t at address p by a tree t', written $t[p \mapsto t']$ is defined as (using $\cdot$ for the insertion of an element on a list)

$$\begin{aligned} t[\varepsilon \mapsto t'] &= t' \\ f(t_1,\ldots,t_n)[(i \cdot p) \mapsto t'] &= \\ f(t_1,\ldots,t_i[p \mapsto t'],\ldots,t_n) &\text{ for } 1 \leq i \leq n \end{aligned}$$

The HEIGHT of a tree t, notated $height(t)$, is defined as follows: $height(x) = 0$ for all $x \in \mathcal{X}$ and $height(f(t_1,\ldots,t_n)) = 1 + \max_{i=1}^{n} height(t_i)$ for all $f \in F$.

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. 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:

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

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.

We will use a notation akin to BNF to specify equations defining functional programs of various sorts. As an introduction to the notation we will use, here is a grammar defining trees over a ranked alphabet and variables (essentially identically to the definition given above):

$$\begin{aligned} f^{(n)} \in \mathcal{F}^{(n)} & \\ x \in \mathcal{X} \quad &::= \quad x_0 \mid x_1 \mid x_2 \mid \cdots \\ t \in \mathcal{T}(\mathcal{F},\mathcal{X}) \quad &::= \quad f^{(m)}(t_1,\ldots,t_m) \\ &\quad\;\; \mid \quad x \end{aligned}$$

The notation allows definition of classes of expressions (e.g., $\mathcal{F}^{(n)}$) and specifies metavariables over them ($f^{(n)}$). These classes can be primitive ($\mathcal{F}^{(n)}$) or defined ($\mathcal{X}$), even inductively in terms of other classes or themselves ($\mathcal{T}(\mathcal{F},\mathcal{X})$). We use the metavariables and subscripted variants on the right-hand side to represent an arbitrary element of the corresponding class. Thus, the elements $t_1,\ldots,t_m$ stand for arbitrary trees in $\mathcal{T}(\mathcal{F},\mathcal{X})$, and x an arbitrary variable in $\mathcal{X}$. Because numerically subscripted versions of x appear explicitly on the right hand side of the rule defining variables, numerically subscripted variables (e.g., x_1) on the right-hand side of all rules are taken to refer to the specific elements of x, whereas otherwise subscripted elements (e.g., x_i) are taken generically.

2 Tree-Adjoining Grammars

Tree adjoining grammar (TAG) is a tree grammar formalism distinguished by its use of a tree adjunction operation. Traditional presentations of TAG, which we will assume familiarity with, take the symbols in elementary and derived trees to be unranked; nodes labeled with a given nonterminal symbol may have differing numbers of children. (Joshi and Schabes (1997) present a good overview.) For example, foot nodes of auxiliary trees and substitution nodes have no children, whereas the similarly labeled root nodes must have at least one. Similarly, two nodes with the same label but differing numbers of children may match for the purpose of allowing an adjunction (as the root nodes of α_1 and β_1 in Figure 1). In order to integrate TAG with tree transducers, however, we move to a ranked alphabet, which presents some problems and opportunities. (In some ways, the ranked alphabet definition of TAGs is slightly more elegant than the traditional one.) Although the bulk of the later discussion integrating TAGs and transducers assumes (without loss of expressivity (Joshi and Schabes, 1997, fn. 6)) a limited form of TAG that includes adjunction but not substitution, we define the more complete form here.

We will thus take the nodes of TAG trees to be labeled with symbols from a ranked alphabet $\mathcal{F}$; a given symbol then has a fixed arity and a fixed

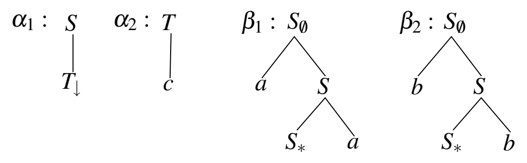
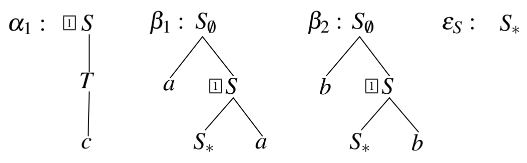


Figure 1: Sample TAG for the copy language $\{ wcw \mid w \in \{a,b\}^* \}$.



α_1 : $\boxed{1}S$
T
c
β_1 : $S_\emptyset$
a
$\boxed{1}S$
S_*
a
β_2 : $S_\emptyset$
b
$\boxed{1}S$
S_*
b
ε_S : S_*

Figure 2: Sample core-restricted TAG for the copy language $\{ wcw \mid w \in \{a,b\}^* \}$.

number of children. However, in order to maintain information about which symbols may match for the purpose of adjunction and substitution, we take the elements of $\mathcal{F}$ to be explicitly formed as pairs of an unranked label e and an arity n. (For notational consistency, we will use e for unranked and f for ranked symbols.) We will notate these elements, abusing notation, as $e^{(n)}$, and make use of a function $|\cdot|$ to unrank symbols in $\mathcal{F}$, so that $|e^{(n)}| = e$.

To handle foot nodes, for each non-nullary symbol $e^{(i)} \in \mathcal{F}^{(\geq 1)}$, we will associate a new nullary symbol e_*, which one can take to be the pair of e and $*$; the set of such symbols will be notated $\mathcal{F}_*$. Similarly, for substitution nodes, $\mathcal{F}_\downarrow$ will be the set of nullary symbols $e_\downarrow$ for all $e^{(i)} \in \mathcal{F}^{(\geq 1)}$. These additional symbols, since they are nullary, will necessarily appear only at the frontier of trees. Finally, to allow null adjoining constraints, for each $f \in \mathcal{F}^{(i)}$, we introduce a symbol $f_\emptyset$ also of arity i, and take $\mathcal{F}_\emptyset$ to be the set of all such symbols. We will extend the function $|\cdot|$ to provide the unranked symbol associated with these symbols as well, so $|e_\downarrow| = |e_*| = |e^{(i)}{}_\emptyset| = e$.

A TAG is then a quadruple $\langle \mathcal{F}, S, I, A \rangle$, where $\mathcal{F}$ is a ranked alphabet; $S \in \mathcal{F}$ is a distinguished initial symbol; I is the set of initial trees, a finite subset of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_\emptyset \cup \mathcal{F}_\downarrow)$; and A is the set of auxiliary trees, a finite subset of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_\emptyset \cup \mathcal{F}_\downarrow \cup \mathcal{F}_*)$. An auxiliary tree β whose root is labeled f must have exactly one node labeled with $|f|_* \in \mathcal{F}_*$ and no other nodes labeled in $\mathcal{F}_*$; this node is its foot node, its address notated *foot*(β). In Figure 1, α_1 and α_2 are initial trees; β_1 and β_2 are auxiliary 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 or β for initial and auxiliary trees respectively. This further allows us to have multiple instances of a tree in I or A, distinguished by their index. (We will abuse notation by using the index and the tree that it names interchangably.)

The trees are combined by two operations, substitution and adjunction. Under substitution, a node labeled $e_\downarrow$ (at address p) in a tree α can be replaced by an initial tree α' with the corresponding label f at the root when $|f| = e$. The resulting tree, the substitution of α' at p in α, is $\alpha[p \mapsto \alpha']$. Under adjunction, an internal node of α at p labeled $f \in \mathcal{F}$ is *split apart*, replaced by an auxiliary tree β rooted in f' when $|f| = |f'|$. The resulting tree, the adjunction of β at p in α, is $\alpha[p \mapsto \beta[\mathit{foot}(\beta) \mapsto \alpha/p]]$. This definition (by requiring f to be in $\mathcal{F}$, not $\mathcal{F}_*$ or $\mathcal{F}_\downarrow$) maintains the standard convention, without loss of expressivity, that adjunction is disallowed at foot nodes and substitution nodes.

The TAG in Figure 1 generates a tree set whose yield is the non-context-free copy language $\{ wcw \mid w \in \{a,b\}^* \}$. The arities of the nodes are suppressed, as they are clear from context.

A derivation tree D records the operations over the elementary trees used to derive a given derived tree. Each node in the derivation tree specifies an elementary tree α, the node's child subtrees D_i recording the derivations for trees that are adjoined or substituted into that tree. A method is required to record at which node in α the tree specified by child subtree D_i operates. For trees recording derivations in context-free grammars, there are exactly as many substitution operations as nonterminals on the right-hand side of the rule used. Thus, child order in the derivation tree can be used to record the identity of the substitution node. But for TAG trees, operations occur throughout the tree, and some, namely adjunctions, can be optional, so a simple convention using child order is not possible. Traditionally, the branches in the derivation tree have been notated with the address of the node in the parent tree at which the child node operates. Figure 4 presents a derivation tree (a) using this notation, along with the corresponding derived tree (b) for the string *abcab*.

For simplicity below, we use a stripped down TAG formalism, one that loses no expressivity in weak generative capacity but is easier for analysis purposes.

First, we make all adjunction obligatory, in the

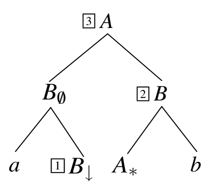


Figure 3: Sample TAG tree marked with diacritics to show the permutation of operable nodes.

sense that if a node in a tree allows adjunction, an adjunction must occur there. To get the effect of optional adjunction, for instance at a node labeled B, we add a vestigial tree of a single node $\varepsilon_B = B_*$, which has no adjunction sites and does not itself modify any tree that it adjoins into. It thus founds the recursive structure of derivations.

Second, now that it is determinate whether an operation must occur at a node, the number of children of a node in a derivation tree is determined by the elementary tree at that node; it is just the number of adjunction or substitution nodes in the tree, the OPERABLE NODES. All that is left to determine is the mapping between child order in the derivation tree and node in the elementary tree labeling the parent, that is, a permutation π on the operable nodes (or equivalently, their addresses), so that the i-th child of a node labeled α in a derivation tree is taken to specify the tree that operates at the node π_i in α. This permutation can be thought of as specified as part of the elementary tree itself. For example, the tree in Figure 3, which requires operations at the nodes at addresses ε, 12, and 2, may be associated with the permutation $\langle 12, 2, \varepsilon \rangle$. This permutation can be marked on the tree itself with numeric diacritics $\boxed{i}$, as shown in the figure.

Finally, as mentioned before, we eliminate substitution (Joshi and Schabes, 1997, fn. 6). With these changes, the sample TAG grammar and derivation tree of Figures 1 and 4(a) might be expressed with the core TAG grammar and derivation tree of Figures 2 and 4(c).

3 Tree Transducers, Homomorphisms, and Automata

3.1 Tree Transducers

Informally, a TREE TRANSDUCER is a function from $\mathcal{T}(\mathcal{F})$ to $\mathcal{T}(\mathcal{G})$ defined such that the symbol at the root ofthe input tree and a current state determines an output context in which the recursive images of the subtrees are placed. Formally, we can define a transducer as a kind of functional program, that is, a set of equations characterized by the following grammar for equations Eqn. (The set of states is conventionally notated Q, with members notated q. One of the states is distinguished as the INITIAL STATE of the transducer.)[1]

$$\begin{array}{rcl}
q \in Q & & \\
f^{(n)} \in \mathcal{F}^{(n)} & & \\
g^{(n)} \in \mathcal{G}^{(n)} & & \\
x_i \in \mathcal{X} & ::= & x_0 \mid x_1 \mid x_2 \mid \cdots \\
Eqn & ::= & q(f^{(n)}(x_1, \ldots, x_n)) = \tau^{(n)} \\
\tau^{(n)} \in \mathcal{R}^{(n)} & ::= & g^{(m)}(\tau_1^{(n)}, \ldots, \tau_m^{(n)}) \\
& \mid & q_j(x_i) \quad \text{where } 1 \le i \le n
\end{array}$$

Intuitively speaking, the expressions in $\mathcal{R}^{(n)}$ are right-hand-side terms using variables limited to the first n.

For example, the grammar allows definition of the following set of equations defining a tree transducer:[2]

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

This transducer allows for the following derivation:

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

The relation defined by a tree transducer with initial state q is $\{ \langle t, u \rangle \mid q(t) = u \}$. By virtue of nondeterminism in the equations, multiple equations for a given state q and symbol f, tree transducers define true relations rather than merely functions.

TREE HOMOMORPHISMS are a subtype of tree transducers, those with only a single state, hence essentially stateless. Other subtypes of tree transducers can be defined by restricting the trees τ

[1]Strictly speaking, what we define here are nondeterministic top-down tree transducers.

[2]Full definitions of tree transducers typically describe a transducer in terms of a set of states, an input and output ranked alphabet, and an initial state, in addition to the set of transitions, that is, defining equations. We will leave off these details, in the expectation that the sets of states and symbols can be inferred from the equations, and the initial state determined under a convention that it is the state defined in the textually first equation.

Note also that we avail ourselves of consistent renaming of the variables x_1, x_2, and so forth, where convenient for readability.

that form the right-hand sides of equations, the elements of $\mathcal{R}^{(n)}$ used. A transducer is LINEAR if all such τ are linear; is COMPLETE if τ contains every variable in $\mathcal{X}_n$; is ε-FREE if $\tau \notin \mathcal{X}_n$; is SYMBOL-TO-SYMBOL if $height(\tau) = 1$; and is a DELABELING if τ is complete, linear, and symbol-to-symbol.

Another subcase is TREE AUTOMATA, tree transducers that compute a partial identity function; these are delabeling tree transducers that preserve the label and the order of arguments. Because they compute only the identity function, tree automata are of interest for their domains, not the mappings they compute. Their domains define tree languages, in particular, the so-called REGULAR TREE LANGUAGES.

3.2 The Bimorphism Characterization of Tree Transducers

Tree transducers can be characterized directly in terms of equations defining a simple kind of functional program, as above. There is an elegant alternative characterization of tree transducers in terms of a constellation of elements of the various subtypes of transducers — homomorphisms and automata — we have introduced, called a bimorphism.

A bimorphism is a triple $\langle L, h_i, h_o \rangle$, consisting of a regular tree language L (or, equivalently, a tree automaton) and two tree homomorphisms h_i and h_o. The tree relation defined by a bimorphism is the set of tree pairs that are generable from elements of the tree language by the homomorphisms, that is,

$$\mathcal{L}(\langle L, h_i, h_o \rangle) = \{\langle h_i(t), h_o(t) \rangle \mid t \in L\} \quad .$$

We can limit attention to bimorphisms in which the input or output homomorphisms are restricted to a certain type, linear (L), complete (C), epsilon-free (F), symbol-to-symbol (S), delabeling (D), or unrestricted (M). We will write $B(I,O)$ where I and O characterize a subclass of homomorphisms for the set of bimorphisms for which the input homomorphism is in the subclass indicated by I and the output homomorphism is in the subclass indicated by O. Thus, $B(D,M)$ is the set of bimorphisms for which the input homomorphism is a delabeling but the output homomorphism can be arbitrary.

The tree relations definable by tree transducers turn out to be exactly this class $B(D,M)$ (Comon et al., 1997). The bimorphism notion thus allows us to characterize the tree transductions purely in terms of tree automata and tree homomorphisms.

We have shown (Shieber, 2004) that the tree relations defined by synchronous tree-substitution grammars were exactly the relations $B(LC,LC)$. Intuitively speaking, the tree language in such a bimorphism represents the set of derivation trees for the synchronous grammar, and each homomorphism represents the relation between the derivation tree and the derived tree for one of the projected tree-substitution grammars. The homomorphisms are linear and complete because the tree relation between a tree-substitution grammar derivation tree and its associated derived tree is exactly a linear complete tree homomorphism. To characterize the tree relations defined by a synchronous tree-adjoining grammar, it similary suffices to *find a simple homomorphism-like characterization of the tree relation between TAG derivation trees and derived trees*. In Section 5 below, we show that linear complete embedded tree homomorphisms, which we introduce next, serve this purpose.

4 Embedded Tree Transducers

Embedded tree transducers are a generalization of tree transducers in which states are allowed to take a single additional argument in a restricted manner. They correspond to a restrictive subcase of macro tree transducers with one recursion variable. We use the term "embedded tree transducer" rather than the more cumbersome "monadic macro tree transducer" for brevity and by analogy with embedded pushdown automata (Schabes and Vijay-Shanker, 1990), another automata-theoretic characterization of the tree-adjoining languages.

We modify the grammar of transducer equations to add an extra argument to each occurrence of a state q. To highlight the special nature of the extra argument, it is written in angle brackets before the input tree argument. We uniformly use the otherwise unused variable x_0 for this argument in the left-hand side, and add x_0 as a possible right-hand side itself. Finally, right-hand-side occurrences of states may be passed an arbitrary further right-hand-side tree in this argument.

$$\begin{array}{rcl}
q \in Q & & \\
f^{(n)} \in \mathcal{F}^{(n)} & & \\
x_i \in \mathcal{X} & ::= & x_0 \mid x_1 \mid x_2 \mid \cdots \\
Eqn & ::= & q\langle[x_0]\rangle(f^{(n)}(x_1,\ldots,x_n)) = \tau^{(n)} \\
\tau^{(n)} \in \mathcal{R}^{(n)} & ::= & f^{(m)}(\tau_1^{(n)},\ldots,\tau_m^{(n)}) \\
& \mid & x_0 \\
& \mid & q_j\langle\tau_j^{(n)}\rangle(x_i) \quad \text{where } 1 \le i \le n
\end{array}$$

Embedded transducers are strictly more expressive than traditional transducers, because the extra argument allows unbounded communication between positions unboundedly distant in depth in the output tree. For example, a simple embedded transducer can compute the reversal of a string, e.g., $1(2(2(nil)))$ reverses to $2(2(1(nil)))$. (This is not computable by a traditional tree transducer.) It is given by the following equations:

$$\begin{array}{rcl} r\langle\rangle(x) & = & r'\langle nil\rangle(x) \\ r'\langle x_0\rangle(nil) & = & x_0 \\ r'\langle x_0\rangle(1(x)) & = & r'\langle 1(x_0)\rangle(x) \\ r'\langle x_0\rangle(2(x)) & = & r'\langle 2(x_0)\rangle(x) \end{array} \tag{1}$$

This is, of course, just the normal accumulating reverse functional program, expressed as an embedded transducer. The additional power of embedded transducers is, we will show in this section, exactly what is needed to characterize the additional power that TAGs represent over CFGs in describing tree languages. In particular, we show that the relation between a TAG derivation tree and derived tree is characterized by a deterministic linear complete embedded tree transducer (DLCETT).

The relation between tree-adjoining languages and embedded tree transducers may be implicit in a series of previous results in the formal-language theory literature.[3] For instance, Fujiyoshi and Kasai (2000) show that linear, complete monadic context-free tree grammars generate exactly the tree-adjoining languages via a normal form for spine grammars. Separately, the relation between context-free tree grammars and macro tree transducers has been described, where the relationship between the monadic variants of each is implicit. Thus, taken together, an equivalence between the tree-adjoining languages and the image languages of monadic macro tree transducers might be pieced together. In the present work, we define the relation between tree-adjoining languages and linear complete monadic tree transducers directly, simply, and transparently, by giving explicit constructions in both directions, carefully handling the distinction between the unranked trees of tree-adjoining grammars and the ranked trees of macro tree transducers and other important issues of detail in the constructions.

The proof requires reductions in both directions. First, we show that for any TAG we can construct a DLCETT that specifies the tree relation between the derivation trees for the TAG and the derived trees. Then, we show that for any DLCETT we can construct a TAG such that the tree relation between the derivation trees and derived trees is related through a simple homomorphism to the DLCETT tree relation.

[3]We are indebted to Uwe Mönnich for this observation.

4.1 From TAG to Transducer

Given an elementary tree α with the label A at its root, let the sequence $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ be a permutation on the nodes in α at which adjunction occurs. (We use this ordering by means of the diacritic representation below.) Then, if α is an auxiliary tree, construct the equation

$$q_A\langle x_0\rangle(\alpha(x_1,\ldots,x_n)) = \lfloor \alpha \rfloor$$

and if α is an initial tree, construct the equation

$$q_A\langle\rangle(\alpha(x_1,\ldots,x_n)) = \lfloor \alpha \rfloor$$

where the right-hand-side transformation $\lfloor \cdot \rfloor$ is defined by[4]

$$\begin{array}{rcl} \lfloor A_\emptyset(t_1,\ldots,t_n) \rfloor & = & A(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) \\ \lfloor \boxed{k} A(t_1,\ldots,t_n) \rfloor & = & q_A\langle \lfloor A_\emptyset(t_1,\ldots,t_n) \rfloor \rangle(x_k) \\ \lfloor A_* \rfloor & = & x_0 \\ \lfloor a \rfloor & = & a \end{array} \tag{2}$$

Note that the equations are linear and complete, because each variable x_i is generated once as the tree α is traversed, namely at position π_i in the traversal (marked with $\boxed{i}$), and the variable x_0 is generated at the foot node only. Thus, the generated embedded tree transducer is linear and complete. Because only one equation is generated per tree, the transducer is trivially deterministic.

By way of example, we consider the core TAG grammar given by the following trees:

$$\begin{array}{rl} \alpha : & \boxed{1}A(e) \\ \beta_A : & A_\emptyset(\boxed{1}B(a), \boxed{2}C(\boxed{3}D(A_*))) \\ \beta_B : & \boxed{1}B(b, B_*) \\ \varepsilon_B : & B_* \\ \varepsilon_C : & C_* \\ \varepsilon_D : & D_* \end{array}$$

[4]It may seem like trickery to use the diacritics in this way, as they are not really components of the tree being traversed, but merely reflexes of an extrinsic ordering. But their use is benign. The same transformation can be defined, a bit more cumbersomely, keeping the permutation π separate, by tracking the permutation and the current address p in a revised transformation $\lfloor \cdot \rfloor_{\pi,p}$ defined as follows:

$$\begin{array}{rcl} \lfloor A_\emptyset(t_1,\ldots,t_n) \rfloor_{\pi,p} & = & A(\lfloor t_1 \rfloor_{\pi,p\cdot 1}, \ldots, \lfloor t_n \rfloor_{\pi,p\cdot n}) \\ \lfloor A(t_1,\ldots,t_n) \rfloor_{\pi,p} & = & q_A\langle \lfloor A_\emptyset(t_1,\ldots,t_n) \rfloor_{\pi,p} \rangle(x_{\pi^{-1}(p)}) \\ \lfloor A_* \rfloor_{\pi,p} & = & x_0 \\ \lfloor a \rfloor_{\pi,p} & = & a \end{array}$$

We then use $\lfloor \alpha \rfloor_{\pi,\varepsilon}$ for the transformation of the tree α.

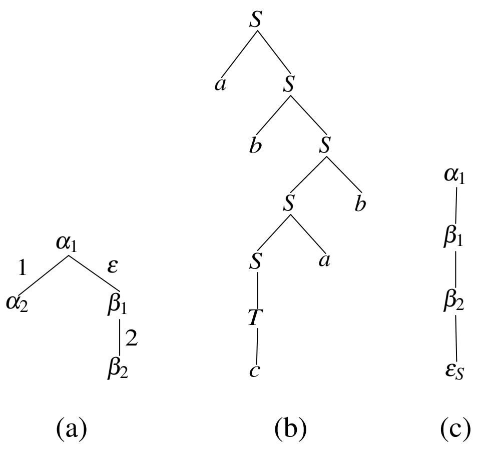


Figure 4: Derivation and derived trees for the sample grammars: (a) derivation tree for the grammar of Figure 1; (b) corresponding derived tree; (c) corresponding derivation tree for the core TAG version of the grammar in Figure 2.

Starting with the auxiliary tree $\beta_A = A_\emptyset(\boxed{1}B(a), \boxed{2}C(\boxed{3}D(A_*)))$, the adjunction sites, corresponding to the nodes labeled B, C, and D at addresses 1, 2, and 21, have been arbitrarily given a preorder permutation. We therefore construct the equation as follows:

$$\begin{aligned}
&q_A\langle x_0\rangle(\beta_A(x_1,x_2,x_3))\\
&\quad= \lfloor A_\emptyset(\boxed{1}B(a),\boxed{2}C(\boxed{3}D(A_*)))\rfloor\\
&\quad= A(\lfloor\boxed{1}B(a)\rfloor, \lfloor\boxed{2}C(\boxed{3}D(A_*))\rfloor)\\
&\quad= A(q_B\langle\lfloor B_\emptyset(a)\rfloor\rangle(x_1), \lfloor\boxed{2}C(\boxed{3}D(A_*))\rfloor)\\
&\quad= A(q_B\langle B(\lfloor a\rfloor)\rangle(x_1), \lfloor\boxed{2}C(\boxed{3}D(A_*))\rfloor)\\
&\quad= \cdots\\
&\quad= A(q_B\langle B(a)\rangle(x_1), q_C\langle C(q_D\langle D(x_0)\rangle(x_3))\rangle(x_2))
\end{aligned}$$

Similar derivations for the remaining trees yield the (deterministic linear complete) embedded tree transducer defined by the following set of equations:

$$\begin{aligned}
q_A\langle\rangle(\alpha(x_1)) &= q_A\langle A(e)\rangle(x_1)\\
q_A\langle x_0\rangle(\beta_A(x_1,x_2,x_3)) &=\\
A(q_B\langle B(a)\rangle(x_1), q_C&\langle C(q_D\langle D(x_0)\rangle(x_3))\rangle(x_2))\\
q_B\langle x_0\rangle(\beta_B(x_1)) &= q_B\langle B(b,x_0)\rangle(x_1)\\
q_B\langle x_0\rangle(\varepsilon_B()) &= x_0\\
q_C\langle x_0\rangle(\varepsilon_C()) &= x_0\\
q_D\langle x_0\rangle(\varepsilon_D()) &= x_0
\end{aligned}$$

We can use this transducer to compute the derived tree for the derivation tree $\alpha(\beta_A(\beta_B(\varepsilon_B), \varepsilon_C, \varepsilon_D))$.

$$\begin{aligned}
&q_A\langle\rangle(\alpha(\beta_A(\beta_B(\varepsilon_B),\varepsilon_C,\varepsilon_D)))\\
&\quad= q_A\langle A(e)\rangle(\beta_A(\beta_B(\varepsilon_B),\varepsilon_C,\varepsilon_D))\\
&\quad= A(\ q_B\langle B(a)\rangle(\beta_B(\varepsilon_B)),\\
&\qquad\qquad q_C\langle C(q_D\langle D(A(e))\rangle(\varepsilon_D))\rangle(\varepsilon_C))\\
&\quad= A(q_B\langle B(b,B(a))\rangle(\varepsilon_B), C(q_D\langle D(A(e))\rangle(\varepsilon_D)))\\
&\quad= A(B(b,B(a)),C(D(A(e))))
\end{aligned}$$

As a final step, useful later for the bimorphism characterization of synchronous TAG, it is straightforward to show that the transducer so constructed is the composition of a regular tree language and a linear complete embedded tree homomorphism.

4.2 From Transducer to TAG

Given a linear complete embedded tree transducer, we construct a corresponding TAG as follows: For each rule of the form

$$q_i\langle[x_0]\rangle(f^{(m)}(x_1,\ldots,x_m)) = \tau$$

we build a tree named $\langle q_i, f, \tau\rangle$. Where this tree appears is determined solely by the state q_i, so we take the root node of the tree to be the state. Any foot node in the tree will also need to be marked with the same label, so we pass this information down as the tree is built inductively. The tree is therefore of the form $q_{i\emptyset}(\lceil\tau\rceil_i)$ where the right-hand-side transformation $\lceil\cdot\rceil_i$ constructs the remainder of the tree by the inductive walk of τ, with the subscript noting that the root is labeled q_i.

$$\begin{aligned}
\lceil f(t_1,\ldots,t_m)\rceil_i &= f_\emptyset(\lceil t_1\rceil_i,\ldots,\lceil t_m\rceil_i)\\
\lceil q_j\langle\tau\rangle(x_k)\rceil_i &= \boxed{k}q_j(\lceil\tau\rceil_i)\\
\lceil x_0\rceil_i &= q_{i*}\\
\lceil a\rceil_i &= a
\end{aligned}$$

Note that at x_0, a foot node is generated of the proper label. (Because the equation is linear, only one foot node is generated, and it is labeled appropriately by construction.) Where recursive processing of the input tree occurs $(q_j\langle\tau\rangle(x_l))$, we generate a tree that admits adjunctions at q_j. The role of the diacritic $\boxed{k}$ is merely to specify the permutation of operable nodes for interpreting derivation trees; it says that the k-th child in a derivation tree rooted in the current elementary tree is taken to specify adjunctions at this node.

The trees generated by this TAG are intended to correspond to the outputs of the corresponding tree transducer. Because of the more severe constraints on TAG, in particular that all combinatorial limitations on putting subtrees together must be manifest in the labels in the trees themselves, the outputs actually contain more structure than the corresponding transducer output. In particular, the state-labeled nodes are merely for bookkeeping. A homomorphism removing these nodes gives the desired transducer output. Most importantly, then, the weak generative capacity of TAGs and LCETTs are identical.

Some examples may clarify the construction. Recall the reversal embedded transducer in (1) above. The construction above generates a TAG containing the following trees. We have given them indicative names rather than the cumbersome ones of the form $\langle q_i, f, \tau\rangle$.

$$\begin{array}{ll} \alpha: & r_{\emptyset}(1 : r'(nil)) \\ \beta_{nil}: & r'_{\emptyset}(r'_{*}) \\ \beta_1: & r'_{\emptyset}(1 : r'(1_{\emptyset}(r'_{*}))) \\ \beta_2: & r'_{\emptyset}(1 : r'(2_{\emptyset}(r'_{*}))) \end{array}$$

It is simple to verify that the derivation tree

$$\alpha(\beta_1(\beta_2(\beta_2(\beta_{nil}))))$$

derives the tree

$$r(r'^{6}(2(r'(2(r'(1(r'(nil))))))))$$

Simple homomorphisms that extract the input function symbols on the input and drop the bookkeeping states on the output reduce these trees to $1(2(2(nil)))$ and $2(2(1(nil)))$ respectively, just as for the corresponding tree transducer.

5 Synchronous TAGs as Bimorphisms

The major advantage of characterizing TAG derivation in terms of tree transducers (via the compilation (2)) is the integration of synchronous TAGs into the bimorphism framework. A synchronous TAG (Shieber, 1994) is composed of a set of triples $\langle t_L, t_R, \frown\rangle$ where the two trees t_L and t_R are elementary trees and $\frown$ is a set of links specifying pairs of linked operable nodes from t_L and t_R. Without loss of generality, we can stipulate that each operable node in each tree is impinged upon by exactly one link in $\frown$. (If a node is unlinked, the triple can never be used; if overlinked, a set of replacement triples can be "multiplied out".) In this case, a projection of the triples on first or second component, with a permutation defined by the corresponding projections on the links, is exactly a TAG as defined above. Thus, derivations proceed just as in a single TAG except that nodes linked by some link in $\frown$ are simultaneously operated on by paired trees derived by the grammar.

In order to model a synchronous grammar formalism as a bimorphism, the well-formed derivations of the synchronous formalism must be characterizable as a regular tree language and the relation between such derivation trees and each of the paired derived trees as a homomorphism of some sort. For synchronous tree-substitution grammars, derivation trees are regular tree languages, and the map from derivation to each of the paired derived trees is a linear complete tree homomorphism. Thus, synchronous tree-substitution grammars fall in the class of bimorphisms $B(LC, LC)$. The other direction can be shown as well; all bimorphisms in $B(LC, LC)$ define tree relations expressible by an STSG.

A similar result follows immediately for STAG. Crucially relying on the result above that the derivation relation is a DLCETT, we can use the method of Shieber (2004) directly to characterize the synchronous TAG tree relations as just $B(ELC, ELC)$. We have thus integrated synchronous TAG with the other transducer and synchronous grammar formalisms falling under the bimorphism umbrella.

Acknowledgements

We wish to thank Mark Dras, Uwe Mönnich, Rebecca Nesson, James Rogers, and Ken Shan for helpful discussions on the topic of this paper. This work was supported in part by grant IIS-0329089 from the National Science Foundation.

References

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.

A. Fujiyoshi and T. Kasai. 2000. Spinal-formed context-free tree grammars. *Theory of Computing Systems*, 33:59–83.

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

Yves Schabes and K. Vijay-Shanker. 1990. Deterministic left to right parsing of tree adjoining languages. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Pittsburgh, Pennsylvania, 6–9 June.

Stuart M. Shieber. 1994. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385, November. Also available as cmp-lg/9404003.

Stuart M. Shieber. 2004. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, pages 88–95, Vancouver, Canada, May 20-22.