A Tree Transducer Model for Synchronous Tree-Adjoining Grammars

Andreas Maletti
Universitat Rovira i Virgili
Avinguda de Catalunya 25, 43002 Tarragona, Spain.
andreas.maletti@urv.cat

Abstract

A characterization of the expressive power of synchronous tree-adjoining grammars (STAGs) in terms of tree transducers (or equivalently, synchronous tree substitution grammars) is developed. Essentially, a STAG corresponds to an extended tree transducer that uses explicit substitution in both the input and output. This characterization allows the easy integration of STAG into toolkits for extended tree transducers. Moreover, the applicability of the characterization to several representational and algorithmic problems is demonstrated.

1 Introduction

Machine translation has seen a multitude of formal translation models. Here we focus on syntax-based (or tree-based) models. One of the oldest models is the *synchronous context-free grammar* (Aho and Ullman, 1972). It is clearly too weak as a syntax-based model, but found use in the string-based setting. *Top-down tree transducers* (Rounds, 1970; Thatcher, 1970) have been heavily investigated in the formal language community (Gécseg and Steinby, 1984; Gécseg and Steinby, 1997), but as argued by Shieber (2004) they are still too weak for syntax-based machine translation. Instead Shieber (2004) proposes *synchronous tree substitution grammars* (STSGs) and develops an equivalent bimorphism (Arnold and Dauchet, 1982) characterization. This characterization eventually led to the rediscovery of *extended tree transducers* (Graehl and Knight, 2004; Knight and Graehl, 2005; Graehl et al., 2008), which are essentially as powerful as STSG. They had been studied already by Arnold and Dauchet (1982) in the form of bimorphisms, but received little attention until rediscovered.

Shieber (2007) claims that even STSGs might be too simple to capture naturally occuring translation phenomena. Instead Shieber (2007) suggests a yet more powerful mechanism, *synchronous tree-adjoining grammars* (STAGs) as introduced by Shieber and Schabes (1990), that can capture certain (mildly) context-sensitive features of natural language. In the tradition of Shieber (2004), a characterization of the power of STAGs in terms of bimorphims was developed by Shieber (2006). The bimorphisms used are rather unconventional because they consist of a regular tree language and two embedded tree transducers (instead of two tree homomorphisms). Such embedded tree transducers (Shieber, 2006) are particular macro tree transducers (Courcelle and Franchi-Zannettacci, 1982; Engelfriet and Vogler, 1985).

In this contribution, we try to unify the picture even further. We will develop a tree transducer model that can simulate STAGs. It turns out that the adjunction operation of an STAG can be explained easily by explicit substitution. In this sense, the slogan that an STAG is an STSG with adjunction, which refers to the syntax, also translates to the semantics. We prove that any tree transformation computed by an STAG can also be computed by an STSG using explicit substitution. Thus, a simple evaluation procedure that performs the explicit substitution is all that is needed to simulate an STAG in a toolkit for STSGs or extended tree transducers like TIBURON by May and Knight (2006).

We show that some standard algorithms on STAG can actually be run on the constructed STSG, which often is simpler and better understood. Further, it might be easier to develop new algorithms with the alternative characterization, which we demonstrate with a product construction for input restriction in the spirit of Nederhof (2009). Finally, we also present a complete tree transducer model that is as powerful as STAG, which is an extension of the embedded tree transducers of Shieber (2006).

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1067–1076,
Uppsala, Sweden, 11–16 July 2010.

2 Notation

We quickly recall some central notions about trees, tree languages, and tree transformations. For a more in-depth discussion we refer to Gécseg and Steinby (1984) and Gécseg and Steinby (1997). A finite set Σ of labels is an alphabet. The set of all strings over that alphabet is Σ^* where ε denotes the empty string. To simplify the presentation, we assume an infinite set $X = \{x_1, x_2, \dots\}$ of variables. Those variables are syntactic and represent only themselves. In particular, they are all different. For each $k \geq 0$, we let $X_k = \{x_1, \dots, x_k\}$. We can also form trees over the alphabet Σ. To allow some more flexibility, we will also allow leaves from a special set V. Formally, a Σ-tree over V is either:

- a leaf labeled with an element of $v \in \Sigma \cup V$, or
- a node that is labeled with an element of Σ with $k \geq 1$ children such that each child is a Σ-tree over V itself.[1]

The set of all Σ-trees over V is denoted by $T_\Sigma(V)$. We just write T_Σ for $T_\Sigma(\emptyset)$. The trees in Figure 1 are, for example, elements of $T_\Delta(Y)$ where

$$\Delta = \{\text{S}, \text{NP}, \text{VP}, \text{V}, \text{DT}, \text{N}\}$$
$$Y = \{\text{saw}, \text{the}\} \ .$$

We often present trees as terms. A leaf labeled v is simply written as v. The tree with a root node labeled σ is written $\sigma(t_1, \dots, t_k)$ where $t_1, \dots, t_k$ are the term representations of its k children.

A tree language is any subset of $T_\Sigma(V)$ for some alphabet Σ and set V. Given another alphabet Δ and a set Y, a tree transformation is a relation $\tau \subseteq T_\Sigma(V) \times T_\Delta(Y)$. In many of our examples we have $V = \emptyset = Y$. Occasionally, we also speak about the translation of a tree transformation $\tau \subseteq T_\Sigma \times T_\Delta$. The *translation* of τ is the relation $\{(\mathrm{yd}(t), \mathrm{yd}(u)) \mid (t, u) \in \tau\}$ where $\mathrm{yd}(t)$, the *yield* of t, is the sequence of leaf labels in a left-to-right tree traversal of t. The yield of the third tree in Figure 1 is "the N saw the N". Note that the translation is a relation $\tau' \subseteq \Sigma^* \times \Delta^*$.

3 Substitution

A standard operation on (labeled) trees is *substitution*, which replaces leaves with a specified label in one tree by another tree. We write $t[u]_A$ for (the result of) the substitution that replaces all leaves labeled A in the tree t by the tree u. If $t \in T_\Sigma(V)$ and $u \in T_\Delta(Y)$, then $t[u]_A \in T_{\Sigma\cup\Delta}(V \cup Y)$. We often use the variables of $X = \{x_1, x_2, \dots\}$ as substitution points and write $t[u_1, \dots, u_k]$ instead of $(\cdots(t[u_1]_{x_1})\dots)[u_k]_{x_k}$.

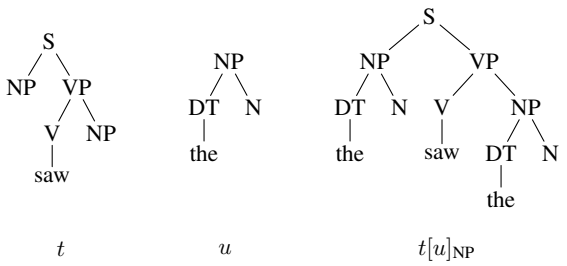

Figure 1: A substitution.

An example substitution is shown in Figure 1. The figure also illustrates a common problem with substitution. Occasionally, it is not desirable to replace all leaves with a certain label by the same tree. In the depicted example, we might want to replace one 'NP' by a different tree, which cannot be achieved with substitution. Clearly, this problem is avoided if the source tree t contains only one leaf labeled A. We call a tree *A-proper* if it contains exactly one leaf with label A.[2] The subset $C_\Sigma(X_k) \subseteq T_\Sigma(X_k)$ contains exactly those trees of $T_\Sigma(X_k)$ that are x_i-proper for every $1 \leq i \leq k$. For example, the tree t of Figure 1 is 'saw'-proper, and the tree u of Figure 1 is 'the'- and 'N'-proper.

In this contribution, we will also use substitution as an explicit operator. The tree $t[u]_{\text{NP}}$ in Figure 1 only shows the result of the substitution. It cannot be infered from the tree alone, how it was obtained (if we do not know t and u).[3] To make substitution explicit, we use the special binary symbols $\cdot[\cdot]_A$ where A is a label. Those symbols will always be used with exactly two children (i.e., as binary symbols). Since this property can easily be checked by all considered devices, we ignore trees that use those symbols in a non-binary manner. For every set Σ of labels, we let $\overline{\Sigma} = \Sigma \cup \{\cdot[\cdot]_A \mid A \in \Sigma\}$ be the extended set of labels containing also the substition symbols. The substitution of Figure 1 can then be ex-

[1] Note that we do not require the symbols to have a fixed rank; i.e., a symbol does not determine its number of children.

[2] A-proper trees are sometimes also called A-context in the literature.

[3] This remains true even if we know that the participating trees t and u are A-proper and the substitution $t[u]_A$ replacing leaves labeled A was used. This is due to the fact that, in general, the root label of u need not coincide with A.

pressed as the tree $\cdot[\cdot]_{\text{NP}}(t, u)$. To obtain $t[u]_{\text{NP}}$ (the right-most tree in Figure 1), we have to evaluate $\cdot[\cdot]_{\text{NP}}(t, u)$. However, we want to replace only one leaf at a time. Consequently, we restrict the evaluation of $\cdot[\cdot]_A(t, u)$ such that it applies only to trees t whose evaluation is A-proper. To enforce this restriction, we introduce an error signal $\perp$, which we assume not to occur in any set of labels. Let Σ be the set of labels. Then we define the function $\cdot^{\text{E}} \colon T_{\overline{\Sigma}} \to T_\Sigma \cup \{\perp\}$ by[4]

$$\sigma(t_1, \ldots, t_k)^{\text{E}} = \sigma(t_1^{\text{E}}, \ldots, t_k^{\text{E}})$$

$$\cdot[\cdot]_A(t, u)^{\text{E}} = \begin{cases} t^{\text{E}}[u^{\text{E}}]_A & \text{if } t^{\text{E}} \text{ is } A\text{-proper} \\ \perp & \text{otherwise} \end{cases}$$

for every $k \geq 0$, $\sigma \in \Sigma$, and $t, t_1, \ldots, t_k, u \in T_{\overline{\Sigma}}$.[5] We generally discard all trees that contain the error signal $\perp$. Since the devices that we will study later can also check the required A-properness using their state behavior, we generally do not discuss trees with error symbols explicitly.

4 Extended tree transducer

An extended tree transducer is a theoretical model that computes a tree transformation. Such transducers have been studied first by Arnold and Dauchet (1982) in a purely theoretic setting, but were later applied in, for example, machine translation (Knight and Graehl, 2005; Knight, 2007; Graehl et al., 2008; Graehl et al., 2009). Their popularity in machine translation is due to Shieber (2004), in which it is shown that extended tree transducers are essentially (up to a relabeling) as expressive as synchronous tree substitution grammars (STSG). We refer to Chiang (2006) for an introduction to synchronous devices.

Let us recall the formal definition. An *extended tree transducer* (for short: XTT)[6] is a system $M = (Q, \Sigma, \Delta, I, R)$ where

- Q is a finite set of *states*,
- Σ and Δ are alphabets of *input* and *output symbols*, respectively,
- $I \subseteq Q$ is a set of *initial states*, and
- R is a finite set of *rules* of the form

$$(q, l) \to (q_1 \cdots q_k, r)$$

[4]Formally, we should introduce an evaluation function for each alphabet Σ, but we assume that the alphabet can be infered.

[5]This evaluation is a special case of a *yield*-mapping (Engelfriet and Vogler, 1985).

[6]Using the notions of Graehl et al. (2009) our extended tree transducers are linear, nondeleting extended top-down tree transducers.

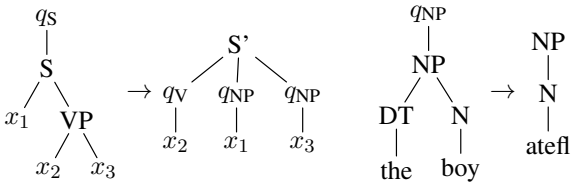

Figure 2: Example rules taken from Graehl et al. (2009). The term representation of the first rule is $(q_{\text{S}}, \text{S}(x_1, \text{VP}(x_2, x_3))) \to (w, \text{S}'(x_2, x_1, x_3))$ where $w = q_{\text{NP}} q_{\text{V}} q_{\text{NP}}$.

where $k \geq 0$, $l \in C_\Sigma(X_k)$, and $r \in C_\Delta(X_k)$.

Recall that any tree of $C_\Sigma(X_k)$ contains each variable of $X_k = \{x_1, \ldots, x_k\}$ exactly once. In graphical representations of a rule

$$(q, l) \to (q_1 \cdots q_k, r) \in R \ ,$$

we usually

- add the state q as root node of the left-hand side[7], and
- add the states $q_1, \ldots, q_k$ on top of the nodes labeled $x_1, \ldots, x_k$, respectively, in the right-hand side of the rule.

Some example rules are displayed in Figure 2.

The rules are applied in the expected way (as in a term-rewrite system). The only additional feature are the states of Q, which can be used to control the derivation. A sentential form is a tree that contains exclusively output symbols towards the root and remaining parts of the input headed by a state as leaves. A derivation step starting from ξ then consists in

- selecting a leaf of ξ with remaining input symbols,
- matching the state q and the left-hand side l of a rule $(q, l) \to (q_1 \cdots q_k, r) \in R$ to the state and input tree stored in the leaf, thus matching input subtrees $t_1, \ldots, t_k$ to the variables $x_1, \ldots, x_k$,
- replacing all the variables $x_1, \ldots, x_k$ in the right-hand side r by the matched input subtrees $q_1(t_1), \ldots, q_k(t_k)$ headed by the corresponding state, respectively, and
- replacing the selected leaf in ξ by the tree constructed in the previous item.

The process is illustrated in Figure 3.

Formally, a sentential form of the XTT M is a tree of $\text{SF} = T_\Delta(Q(T_\Sigma))$ where

$$Q(T_\Sigma) = \{q(t) \mid q \in Q, t \in T_\Sigma\} \ .$$

[7]States are thus also special symbols that are exclusively used as unary symbols.

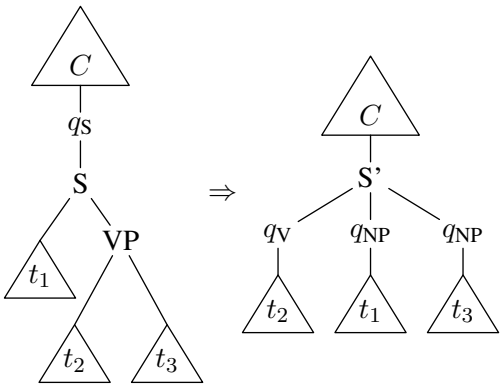

Figure 3: Illustration of a derivation step of an XTT using the left rule of Figure 2.

Given $\xi, \zeta \in \mathrm{SF}$, we write $\xi \Rightarrow \zeta$ if there exist $C \in C_\Delta(X_1)$, $t_1, \ldots, t_k \in T_\Sigma$, and a rule $(q, l) \to (q_1 \cdots q_k, r) \in R$ such that

- $\xi = C[q(l[t_1, \ldots, t_k])]$ and
- $\zeta = C[r[q_1(t_1), \ldots, q_k(t_k)]]$.

The tree transformation computed by M is the relation

$$\tau_M = \{(t, u) \in T_\Sigma \times T_\Delta \mid \exists q \in I \colon q(t) \Rightarrow^* u\}$$

where $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$. In other words, the tree t can be transformed into u if there exists an initial state q such that we can derive u from $q(t)$ in several derivation steps.

We refer to Arnold and Dauchet (1982), Graehl et al. (2008), and Graehl et al. (2009) for a more detailed exposition to XTT.

5 Synchronous tree-adjoining grammar

XTT are a simple, natural model for tree transformations, however they are not suitably expressive for all applications in machine translation (Shieber, 2007). In particular, all tree transformations of XTT have a certain locality condition, which yields that the input tree and its corresponding translation cannot be separated by an unbounded distance. To overcome this problem and certain dependency problems, Shieber and Schabes (1990) and Shieber (2007) suggest a stronger model called *synchronous tree-adjoining grammar* (STAG), which in addition to the substitution operation of STSG (Chiang, 2005) also has an adjoining operation.

Let us recall the model in some detail. A tree-adjoining grammar essentially is a regular tree grammar (Gécseg and Steinby, 1984; Gécseg and Steinby, 1997) enhanced with an adjunction operation. Roughly speaking, an adjunction replaces a node (not necessarily a leaf) by an *auxiliary tree*, which has exactly one distinguished *foot node*. The original children of the replaced node will become the children of the foot node after adjunction. Traditionally, the root label and the label of the foot node coincide in an auxiliary tree aside from a star index that marks the foot node. For example, if the root node of an auxiliary tree is labeled A, then the foot node is traditionally labeled $A_\star$. The star index is not reproduced once adjoined. Formally, the adjunction of the auxiliary tree u with root label A (and foot node label $A_\star$) into a tree $t = C[A(t_1, \ldots, t_k)]$ with $C \in C_\Sigma(X_1)$ and $t_1, \ldots, t_k \in T_\Sigma$ is

$$C[u[A(t_1, \ldots, t_k)]_{A_\star}] \ .$$

Adjunction is illustrated in Figure 4.

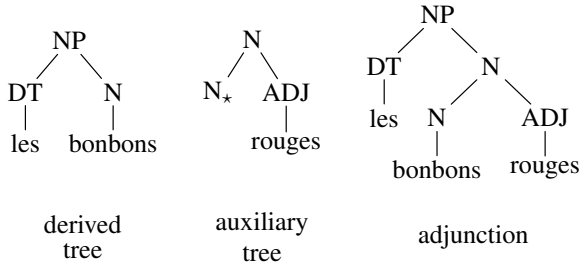

Figure 4: Illustration of an adjunction taken from Nesson et al. (2008).

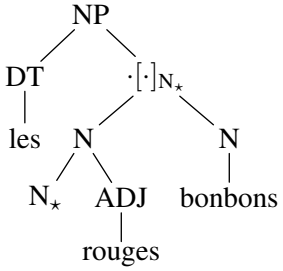

Figure 5: Illustration of the adjunction of Figure 4 using explicit substitution.

We note that adjunction can easily be expressed using explicit substitution. Essentially, only an additional node with the adjoined subtree is added. The result of the adjunction of Figure 4 using explicit substitution is displayed in Figure 5.

To simplify the development, we will make some assumptions on all tree-adjoining grammars (and synchronous tree-adjoining grammars). A *tree-adjoining grammar* (TAG) is a finite set of *initial trees* and a finite set of auxiliary trees. Our

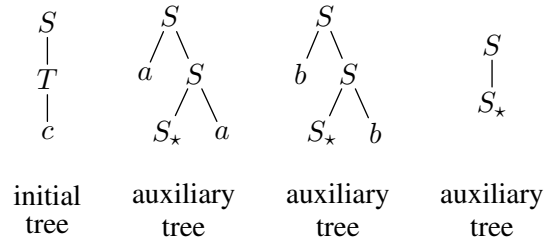

Figure 6: A TAG for the copy string language $\{wcw \mid w \in \{a, b\}^*\}$ taken from Shieber (2006).

TAG do not use substitution, but only adjunction. A derivation is a chain of trees that starts with an initial tree and each derived tree is obtained from the previous one in the chain by adjunction of an auxiliary tree. As in Shieber (2006) we assume that all adjunctions are mandatory; i.e., if an auxiliary tree can be adjoined, then we need to make an adjunction. Thus, a derivation starting from an initial tree to a derived tree is *complete* if no adjunction is possible in the derived tree. Moreover, we assume that to each node only one adjunction can be applied. This is easily achieved by labeling the root of each adjoined auxiliary tree by a special marker. Traditionally, the root label A of an auxiliary tree is replaced by $A_\emptyset$ once adjoined. Since we assume that there are no auxiliary trees with such a root label, no further adjunction is possible at such nodes. Another effect of this restriction is that the number of *operable nodes* (i.e., the nodes to which an adjunction must still be applied) is known at any given time.[8] A full TAG with our restrictions is shown in Figure 6.

Intuitively, a *synchronous tree-adjoining grammar* (STAG) is essentially a pair of TAGs. The synchronization is achieved by pairing the initial trees and the auxiliary trees. In addition, for each such pair (t, u) of trees, there exists a bijection between the operable nodes of t and u. Such nodes in bijection are *linked* and the links are preserved in derivations, in which we now use pairs of trees as sentential forms. In graphical representations we often indicate this bijection with integers; i.e., two nodes marked with the same integer are linked. A pair of auxiliary trees is then adjoined to linked nodes (one in each tree of the sentential form) in the expected manner. We will avoid a formal definition here, but rather present an example STAG and a derivation with it in Figures 7 and 8. For a

[8] Without the given restrictions, this number cannot be determined easily because no or several adjunctions can take place at a certain node.

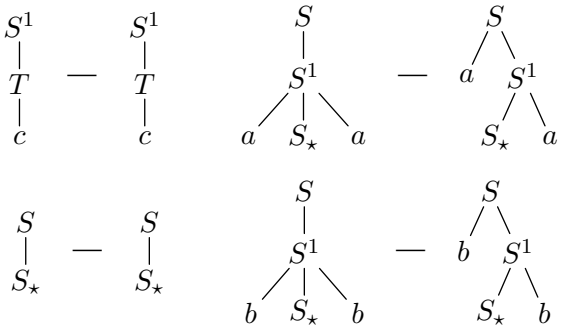

Figure 7: STAG that computes the translation $\{(wcw^{\mathrm{R}}, wcw) \mid w \in \{a, b\}^*\}$ where w^{R} is the reverse of w.

STAG G we write τ_G for the tree transformation computed by G.

6 Main result

In this section, we will present our main result. Essentially, it states that a STAG is as powerful as a STSG using explicit substitution. Thus, for every tree transformation computed by a STAG, there is an extended tree transducer that computes a representation of the tree transformation using explicit substitution. The converse is also true. For every extended tree transducer M that uses explicit substitution, we can construct a STAG that computes the tree transformation represented by τ_M up to a relabeling (a mapping that consistently replaces node labels throughout the tree). The additional relabeling is required because STAGs do not have states. If we replace the extended tree transducer by a STSG, then the result holds even without the relabeling.

Theorem 1 *For every STAG G, there exists an extended tree transducer M such that*

$$\tau_G = \{(t^{\mathrm{E}}, u^{\mathrm{E}}) \mid (t, u) \in \tau_M\} \ .$$

Conversely, for every extended tree transducer M, there exists a STAG G such that the above relation holds up to a relabeling.

6.1 Proof sketch

The following proof sketch is intended for readers that are familiar with the literature on embedded tree transducers, macro tree transducers, and bimorphisms. It can safely be skipped because we will illustrate the relevant construction on our example after the proof sketch, which contains the outline for the correctness.

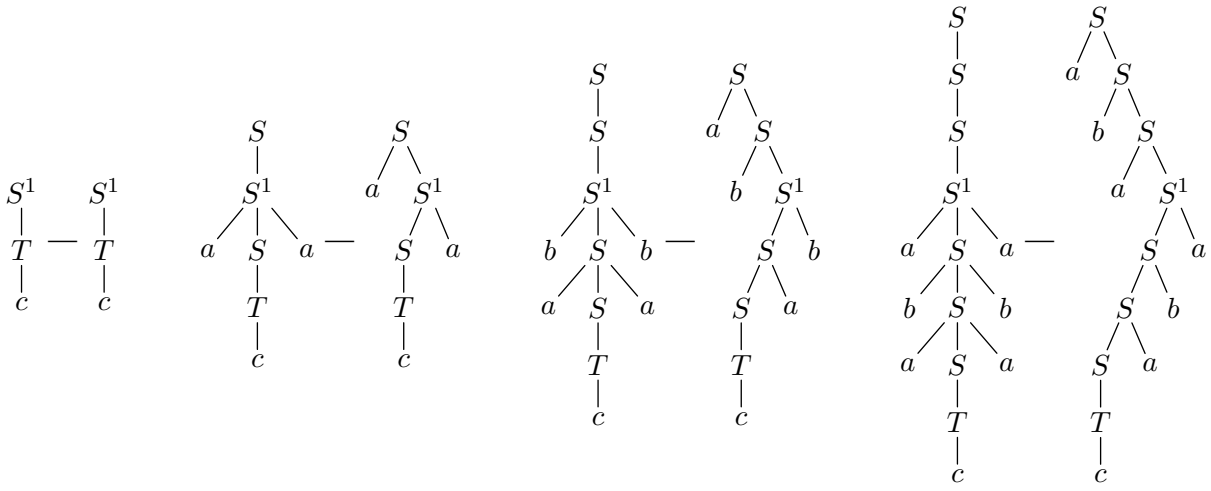

Figure 8: An incomplete derivation using the STAG of Figure 7.

Let $\tau \subseteq T_\Sigma \times T_\Delta$ be a tree transformation computed by a STAG. By Shieber (2006) there exists a regular tree language $L \subseteq T_\Gamma$ and two functions $e_1 \colon T_\Gamma \to T_\Sigma$ and $e_2 \colon T_\Gamma \to T_\Delta$ such that $\tau = \{(e_1(t), e_2(t)) \mid t \in L\}$. Moreover, e_1 and e_2 can be computed by *embedded tree transducers* (Shieber, 2006), which are particular 1-state, deterministic, total, 1-parameter, linear, and nondeleting macro tree transducers (Courcelle and Franchi-Zannettacci, 1982; Engelfriet and Vogler, 1985). In fact, the converse is also true up to a relabeling, which is also shown in Shieber (2006). The outer part of Figure 9 illustrates these relations. Finally, we remark that all involved constructions are effective.

Using a result of Engelfriet and Vogler (1985), each embedded tree transducer can be decomposed into a top-down tree transducer (Gécseg and Steinby, 1984; Gécseg and Steinby, 1997) and a yield-mapping. In our particular case, the top-down tree transducers are linear and nondeleting homomorphisms h_1 and h_2. Linearity and nondeletion are inherited from the corresponding properties of the macro tree transducer. The properties '1-state', 'deterministic', and 'total' of the macro tree transducer ensure that the obtained top-down tree transducer is also 1-state, deterministic, and total, which means that it is a homomorphism. Finally, the 1-parameter property yields that the used substitution symbols are binary (as our substitution symbols $\cdot[\cdot]_A$). Consequently, the yield-mapping actually coincides with our evaluation. Again, this decomposition actually is a characterization of embedded tree transducers. Now the set $\{(h_1(t), h_2(t)) \mid t \in L\}$ can be computed

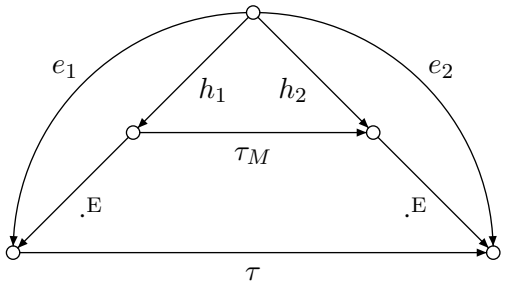

Figure 9: Illustration of the proof sketch.

by an extended tree transducer M due to results of Shieber (2004) and Maletti (2008). More precisely, every extended tree transducer computes such a set, so that also this step is a characterization. Thus we obtain that τ is an evaluation of a tree transformation computed by an extended tree transducer, and moreover, for each extended tree transducer, the evaluation can be computed (up to a relabeling) by a STAG. The overall proof structure is illustrated in Figure 9.

6.2 Example

Let us illustrate one direction (the construction of the extended tree transducer) on our example STAG of Figure 7. Essentially, we just prepare all operable nodes by inserting an explicit substitution just on top of them. The first subtree of that substitution will either be a variable (in the left-hand side of a rule) or a variable headed by a state (in the right-hand side of a rule). The numbers of the variables encode the links of the STAG. Two example rules obtained from the STAG of Figure 7 are presented in Figure 10. Using all XTT rules constructed for the STAG of Figure 7, we present

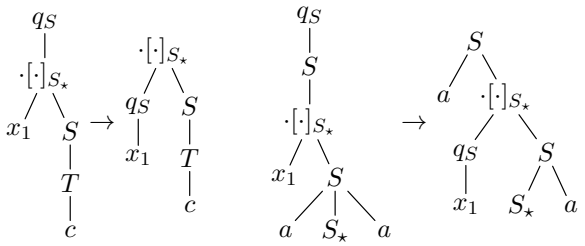

Figure 10: Two constructed XTT rules.

a complete derivation of the XTT in Figure 11 that (up to the final step) matches the derivation of the STAG in Figure 8. The matching is achieved by the evaluation $\cdot^{\mathrm{E}}$ introduced in Section 3 (i.e., applying the evaluation to the derived trees of Figure 11 yields the corresponding derived trees of Figure 8.

7 Applications

In this section, we will discuss a few applications of our main result. Those range from representational issues to algorithmic problems. Finally, we also present a tree transducer model that includes explicit substitution. Such a model might help to address algorithmic problems because derivation and evaluation are intertwined in the model and not separate as in our main result.

7.1 Toolkits

Obviously, our characterization can be applied in a toolkit for extended tree transducers (or STSG) such as TIBURON by May and Knight (2006) to simulate STAG. The existing infrastructure (input-output, derivation mechanism, etc) for extended tree transducers can be re-used to run XTTs encoding STAGs. The only additional overhead is the implementation of the evaluation, which is a straightforward recursive function (as defined in Section 3). After that any STAG can be simulated in the existing framework, which allows experiments with STAG and an evaluation of their expressive power without the need to develop a new toolkit. It should be remarked that some essential algorithms that are very sensitive to the input and output behavior (such as parsing) cannot be simulated by the corresponding algorithms for STSG. It remains an open problem whether the close relationship can also be exploited for such algorithms.

7.2 Algorithms

We already mentioned in the previous section that some algorithms do not easily translate from STAG to STSG (or vice versa) with the help of our characterization. However, many standard algorithms for STAG can easily be derived from the corresponding algorithms for STSG. The simplest example is the union of two STAG. Instead of taking the union of two STAG using the classical construction, we can take the union of the corresponding XTT (or STSG) that simulate the STAGs. Their union will simulate the union of the STAGs. Such properties are especially valuable when we simulate STAG in toolkits for XTT.

A second standard algorithm that easily translates is the algorithm computing the n-best derivations (Huang and Chiang, 2005). Clearly, the n-best derivation algorithm does not consider a particular input or output tree. Since the derivations of the XTT match the derivations of the STAG (in the former the input and output are encoded using explicit substitution), the n-best derivations will coincide. If we are additionally interested in the input and output trees for those n-best derivations, then we can simply evaluate the coded input and output trees returned by n-best derivation algorithm.

Finally, let us consider an algorithm that can be obtained for STAG by developing it for XTT using explicit substitution. We will develop a BAR-HILLEL (Bar-Hillel et al., 1964) construction for STAG. Thus, given a STAG G and a recognizable tree language L, we want to construct a STAG G' such that

$$\tau_{G'} = \{(t, u) \mid (t, u) \in \tau_G, t \in L\} \ .$$

In other words, we take the tree transformation τ_G but additionally require the input tree to be in L. Consequently, this operation is also called *input restriction*. Since STAG are symmetric, the corresponding output restriction can be obtained in the same manner. Note that a classical BAR-HILLEL construction restricting to a regular set of yields can be obtained easily as a particular input restriction. As in Nederhof (2009) a change of model is beneficial for the development of such an algorithm, so we will develop an input restriction for XTT using explicit substitution.

Let $M = (Q, \Sigma, \Delta, I, R)$ be an XTT (using explicit substitution) and $G = (N, \Sigma, I', P)$ be a tree substitution grammar (regular tree grammar) in normal form that recognizes L (i.e., $L(G) = L$). Let $S = \{A \in \Sigma \mid \cdot[\cdot]_A \in \Sigma\}$. A *context* is a mapping $c\colon S \to N$, which remembers a nonterminal of G for each substitution point. Given a rule

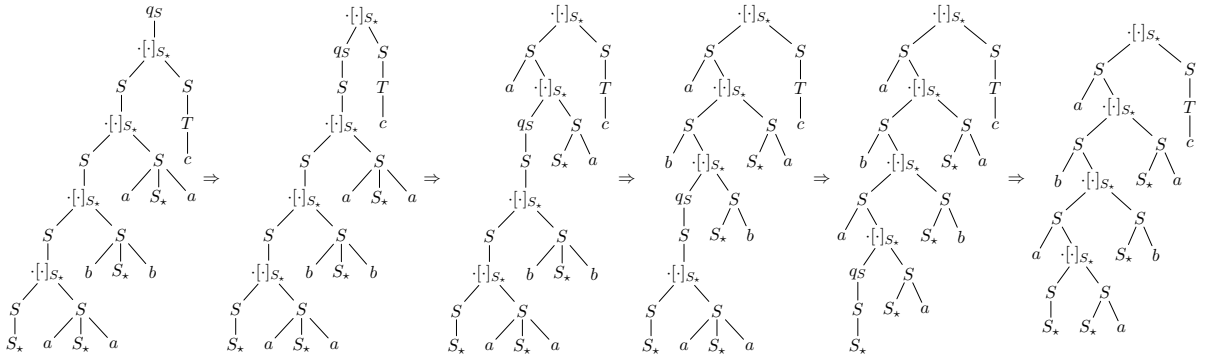

Figure 11: Complete derivation using the constructed XTT rules.

$(q, l) \to (q_1 \cdots q_k, r) \in R$, a nonterminal $p \in N$, and a context $c \in S$, we construct new rules corresponding to successful parses of l subject to the following restrictions:

- If $l = \cdot[\cdot]_A(l_1, l_2)$ for some $A \in \Sigma$, then select $p' \in N$, parse l_1 in p with context c' where $c' = c[A \mapsto p']$[9], and parse l_2 in p' with context c.
- If $l = A_\star$ with $A \in \Sigma$, then $p = c(A)$.
- Finally, if $l = \sigma(l_1, \ldots, l_k)$ for some $\sigma \in \Sigma$, then select $p \to \sigma(p_1, \ldots, p_k) \in P$ is a production of G and we parse l_i with nonterminal p_i and context c for each $1 \leq i \leq k$.

7.3 A complete tree transducer model

So far, we have specified a tree transducer model that requires some additional parsing before it can be applied. This parsing step has to annotate (and correspondingly restructure) the input tree by the adjunction points. This is best illustrated by the left tree in the last pair of trees in Figure 8. To run our constructed XTT on the trivially completed version of this input tree, it has to be transformed into the first tree of Figure 11, where the adjunctions are now visible. In fact, a second un-parsing step is required to evaluate the output.

To avoid the first additional parsing step, we will now modify our tree transducer model such that this parsing step is part of its semantics. This shows that it can also be done locally (instead of globally parsing the whole input tree). In addition, we arrive at a tree transducer model that exactly (up to a relabeling) matches the power of STAG, which can be useful for certain constructions. It is known that an embedded tree transducer (Shieber, 2006) can handle the mentioned un-parsing step.

An *extended embedded tree transducer with substitution* $M = (Q, \Sigma, \Delta, I, R)$ is simply an embedded tree transducer with extended left-hand sides (i.e., any number of input symbols is allowed in the left-hand side) that uses the special symbols $\cdot[\cdot]_A$ in the input. Formally, let

- $Q = Q_0 \cup Q_1$ be finite where Q_0 and Q_1 are the set of states that do not and do have a context parameter, respectively,
- Σ and Δ be ranked alphabets such that if $\cdot[\cdot]_A \in \Sigma$, then $A, A_\star \in \Sigma$,
- $Q\langle U \rangle$ be such that

$$Q\langle U \rangle = \{q\langle u \rangle \mid q \in Q_1, u \in U\} \cup \cup \{q\langle\rangle \mid q \in Q_0\} \ ,$$

- $I \subseteq Q\langle T_\Delta \rangle$, and
- R is a finite set of rules $l \to r$ such that there exists $k \geq 0$ with $l \in Q\langle\{y\}\rangle(C_\Sigma(X_k))$ and $r \in \mathrm{Rhs}_k$ where

$$\mathrm{Rhs}_k := \delta(\mathrm{Rhs}_k, \ldots, \mathrm{Rhs}_k) \mid \mid q_1\langle \mathrm{Rhs}_k \rangle(x) \mid q_0\langle\rangle(x)$$

with $\delta \in \Delta_k$, $q_1 \in Q_1$, $q_0 \in Q_0$, and $x \in X_k$. Moreover, each variable of l (including y) is supposed to occur exactly once in r.

We refer to Shieber (2006) for a full description of embedded tree transducers. As seen from the syntax, we write the context parameter y of a state $q \in Q_1$ as $q\langle y \rangle$. If $q \in Q_0$, then we also write $q\langle\rangle$ or $q\langle\varepsilon\rangle$. In each right-hand side, such a context parameter u can contain output symbols and further calls to input subtrees. The semantics of extended embedded tree transducers with substitution deviates slightly from the embedded tree transducer semantics. Roughly speaking, not its rules as such, but rather their evaluation are now applied in a term-rewrite fashion. Let

$$\mathrm{SF}' := \delta(\mathrm{SF}', \ldots, \mathrm{SF}') \mid \mid q_1\langle \mathrm{SF}' \rangle(t) \mid q_0\langle\rangle(t)$$

[9] c' is the same as c except that it maps A to p'.

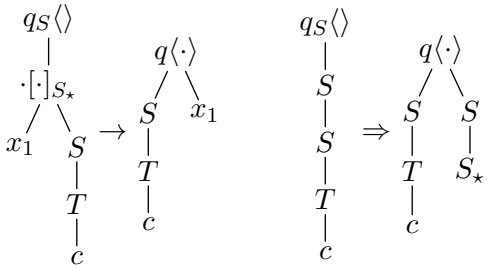

Figure 12: Rule and derivation step using the rule in an extended embedded tree transducer with substitution where the context parameter (if present) is displayed as first child.

where $\delta \in \Delta_k$, $q_1 \in Q_1$, $q_0 \in Q_0$, and $t \in T_\Sigma$.

Given $\xi, \zeta \in \mathrm{SF}'$, we write $\xi \Rightarrow \zeta$ if there exist $C \in C_\Delta(X_1), t_1, \ldots, t_k \in T_\Sigma, u \in T_\Delta \cup \{\varepsilon\}$, and a rule $q\langle u\rangle(l) \to r \in R$[10] with $l \in C_\Sigma(X_k)$ such that

- $\xi = C[q\langle u\rangle(l[t_1, \ldots, t_k]^{\mathrm{E}})]$ and
- $\zeta = C[(r[t_1, \ldots, t_k])[u]_y]$.

Note that the essential difference to the "standard" semantics of embedded tree transducers is the evaluation in the first item. The tree transformation computed by M is defined as usual. We illustrate a derivation step in Figure 12, where the match $\cdot[\cdot]_{S_\star}(x_1, S(T(c)))^{\mathrm{E}} = S(S(T(c)))$ is successful for $x_1 = S(S_\star)$.

Theorem 2 *Every STAG can be simulated by an extended embedded tree transducer with substitution. Moreover, every extended embedded tree transducer computes a tree transformation that can be computed by a STAG up to a relabeling.*

8 Conclusions

We presented an alternative view on STAG using tree transducers (or equivalently, STSG). Our main result shows that the syntactic characterization of STAG as STSG plus adjunction rules also carries over to the semantic side. A STAG tree transformation can also be computed by an STSG using explicit substitution. In the light of this result, some standard problems for STAG can be reduced to the corresponding problems for STSG. This allows us to re-use existing algorithms for STSG also for STAG. Moreover, existing STAG algorithms can be related to the corresponding STSG algorithms, which provides further evidence of the close relationship between the two models. We used this relationship to develop a BAR-HILLEL construction for STAG. Finally, we hope that the alternative characterization is easier to handle and might provide further insight into general properties of STAG such as compositions and preservation of regularity.

[10] Note that u is ε if $q \in Q_0$.

Acknowledgements

ANDREAS MALETTI was financially supported by the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*. Prentice Hall.

André Arnold and Max Dauchet. 1982. Morphismes et bimorphismes d'arbres. *Theoret. Comput. Sci.*, 20(1):33–93.

Yehoshua Bar-Hillel, Micha Perles, and Eliyahu Shamir. 1964. On formal properties of simple phrase structure grammars. In Yehoshua Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison Wesley.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proc. ACL*, pages 263–270. Association for Computational Linguistics.

David Chiang. 2006. An introduction to synchronous grammars. In *Proc. ACL*. Association for Computational Linguistics. Part of a tutorial given with Kevin Knight.

Bruno Courcelle and Paul Franchi-Zannettacci. 1982. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191, 235–257.

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

Ferenc Gécseg and Magnus Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer.

Jonathan Graehl and Kevin Knight. 2004. Training tree transducers. In *HLT-NAACL*, pages 105–112. Association for Computational Linguistics. See also (Graehl et al., 2008).

Jonathan Graehl, Kevin Knight, and Jonathan May. 2008. Training tree transducers. *Computational Linguistics*, 34(3):391–427.

Jonathan Graehl, Mark Hopkins, Kevin Knight, and Andreas Maletti. 2009. The power of extended top-down tree transducers. *SIAM Journal on Computing*, 39(2):410–430.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proc. IWPT*, pages 53–64. Association for Computational Linguistics.

Kevin Knight and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Proc. CICLing*, volume 3406 of LNCS, pages 1–24. Springer.

Kevin Knight. 2007. Capturing practical natural language transformations. *Machine Translation*, 21(2):121–133.

Andreas Maletti. 2008. Compositions of extended top-down tree transducers. *Inform. and Comput.*, 206(9–10):1187–1196.

Jonathan May and Kevin Knight. 2006. TIBURON: A weighted tree automata toolkit. In *Proc. CIAA*, volume 4094 of LNCS, pages 102–113. Springer.

Mark-Jan Nederhof. 2009. Weighted parsing of trees. In *Proc. IWPT*, pages 13–24. Association for Computational Linguistics.

Rebecca Nesson, Giorgio Satta, and Stuart M. Shieber. 2008. Optimal k-arization of synchronous tree-adjoining grammar. In *Proc. ACL*, pages 604–612. Association for Computational Linguistics.

William C. Rounds. 1970. Mappings and grammars on trees. *Math. Systems Theory*, 4(3):257–287.

Stuart M. Shieber and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proc. Computational Linguistics*, volume 3, pages 253–258.

Stuart M. Shieber. 2004. Synchronous grammars as tree transducers. In *Proc. TAG+7*, pages 88–95.

Stuart M. Shieber. 2006. Unifying synchronous tree adjoining grammars and tree transducers via bimorphisms. In *Proc. EACL*, pages 377–384. Association for Computational Linguistics.

Stuart M. Shieber. 2007. Probabilistic synchronous tree-adjoining grammars for machine translation: The argument from bilingual dictionaries. In *Proc. Workshop on Syntax and Structure in Statistical Translation*, pages 88–95. Association for Computational Linguistics.

James W. Thatcher. 1970. Generalized^2 sequential machine maps. *J. Comput. System Sci.*, 4(4):339–367.