Why Synchronous Tree Substitution Grammars?

Andreas Maletti
Universitat Rovira i Virgili, Departament de Filologies Romàniques
Avinguda de Catalunya 35, 43002 Tarragona, Spain
andreas.maletti@urv.cat

Abstract

Synchronous tree substitution grammars are a translation model that is used in syntax-based machine translation. They are investigated in a formal setting and compared to a competitor that is at least as expressive. The competitor is the extended multi bottom-up tree transducer, which is the bottom-up analogue with one essential additional feature. This model has been investigated in theoretical computer science, but seems widely unknown in natural language processing. The two models are compared with respect to standard algorithms (binarization, regular restriction, composition, application). Particular attention is paid to the complexity of the algorithms.

1 Introduction

Every machine translation system uses a translation model, which is a formal model that describes the translation process. Either this system is handcrafted (in rule-based translation systems) or it is trained with the help of statistical processes. Brown et al. (1990) discuss automatically trainable translation models in their seminal paper on the latter approach. The IBM models of Brown et al. (1993) are string-based in the sense that they base the translation decision on the words and the surrounding context. In the field of syntax-based machine translation, the translation models have access to the syntax (in the form of parse trees) of the sentences. Knight (2007) presents a good exposition to both fields.

In this paper, we focus on syntax-based translation models, and in particular, synchronous tree substitution grammars (STSGs), or the equally powerful (linear and nondeleting) extended (top-down) tree transducers of Graehl et al. (2008). Chiang and Knight (2006) gives a good introduction to STSGs, which originate from the syntax-directed translation schemes of Aho and Ullman (1972) [nowadays more commonly known as synchronous context-free grammars]. Roughly speaking, an STSG has rules in which a nonterminal is replaced by two trees containing terminal and nonterminal symbols. In addition, the nonterminals in the two trees are linked and a rule is only applied to linked nonterminals.

Several algorithms for STSGs have been discussed in the literature. For example, we can

- train them [see Graehl et al. (2008)],
- attempt to binarize them using the methods of (Zhang et al., 2006; Huang et al., 2009; DeNero et al., 2009b),
- parse them [see DeNero et al. (2009a)], or
- attempt to compose them.

However, some important algorithms are partial because it is known that the construction might not be possible in general. This is the case, for example, for binarization and composition.

In the theoretical computer science community, alternative models have been explored. Such a model is the multi bottom-up tree transducer (MBOT) of Arnold and Dauchet (1982) and Lilin (1981), which essentially is the bottom-up analogue of STSGs with the additional feature that nonterminals can have an arbitrary rank (the rank of a nonterminal of an STSG can be considered to be fixed to 1). This model is even more expressive than STSGs, but still offers good computational properties. In this contribution, we will compare STSGs and MBOTs with respect to some standard algorithms. Generally, MBOTs offer algorithmic benefits over STSG, which can be summarized as fol-

lows:

- Every STSG can be transformed into an equivalent MBOT in linear time.
- MBOTs can be fully binarized in linear time whereas only partial binarizations (or asynchronous binarizations) are possible for STSGs.
- The input language of an MBOT M can be regularly restricted in $\mathcal{O}(|M| \cdot |S|^3)$, whereas the corresponding construction for an STSG M is in $\mathcal{O}(|M| \cdot |S|^{2\,\mathrm{rk}(M)+5})$ where $\mathrm{rk}(M)$ is the maximal number of nonterminals in a rule of the STSG M.
- MBOTs can be composed, whereas this cannot be achieved for STSGs.

Overall, we thus conclude that, from an algorithmic perspective, it would be beneficial to work with MBOTs instead of STSGs. However, the full power of MBOTs should not be tapped because, in general, MBOTs have the finite-copying property [see Engelfriet et al. (1980)], which complicates the algorithms for forward and backward application (see Section 7).

2 Preliminary definitions

An *alphabet* is a finite set of symbols. Our weighted devices use real-number weights, but the results translate easily to the more general setting of commutative semirings [see Golan (1999)]. A *weighted string automaton* as in Schützenberger (1961) and Eilenberg (1974) is a system (S, Γ, I, τ, F) where

- S and Γ are alphabets of states and input symbols, respectively,
- $I, F\colon S \to \mathbb{R}$ assign initial and final weights, respectively, and
- $\tau\colon S \times \Gamma \times S \to \mathbb{R}$ assigns a weight to each transition.

Let $w = \gamma_1 \cdots \gamma_k \in \Gamma^*$ be an input string of length k. A *run* on w is $r\colon \{0, \ldots, k\} \to S$. The *weight* of the run r is $\mathrm{wt}(r) = \prod_{i=1}^{k} \tau(r_{i-1}, \gamma_i, r_i)$. The semantics of the automaton A then assigns to w the weight

$$A(w) = \sum_{r \text{ run on } w} I(r_0) \cdot \mathrm{wt}(r) \cdot F(r_k) \ .$$

A good introduction to weighted string automata can be found in Mohri (2009) and Sakarovitch (2009).

To simplify the theoretical discussion, we assume that each symbol that we use in trees has a fixed rank, which determines the number of children of each node with that label. A *ranked alphabet* $\Sigma = \bigcup_{k \geq 0} \Sigma_k$ is an alphabet whose symbols have assigned ranks. The set Σ_k contains all symbols of rank k. The set $T_\Sigma(V)$ of Σ-*trees* indexed by a set V is the smallest set such that $V \subseteq T_\Sigma(V)$ and $\sigma(t_1, \ldots, t_k) \in T_\Sigma(V)$ for every $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(V)$. The size $|t|$ of the tree $t \in T_\Sigma$ is the number of occurrences of symbols from $\Sigma \cup V$ that appear in t. A context c is a tree of $T_{\Sigma \cup \{\square\}}(V)$, in which the nullary symbol $\square$ occurs exactly once. The set of all such contexts is $C_\Sigma(V)$. The tree $c[t]$ is obtained from c by replacing the symbol $\square$ by t.

A *weighted synchronous tree substitution grammar* (STSG) is a system $(N, \Sigma, \Delta, I, P)$ where

- N is an alphabet of nonterminals,
- Σ and Δ are ranked alphabets of input and output symbols, respectively,
- $I\colon N \to \mathbb{R}$ assigns initial weights, and
- P is a finite set of productions $n\colon t \overset{a}{\leftrightarrow} u$ with $n \in N$, $t \in T_\Sigma(N)$, $a \in \mathbb{R}$, and $u \in T_\Delta(N)$ such that
  - every $n' \in N$ that occurs in t occurs exactly once in u and vice versa, and
  - $t \notin N$ or $u \notin N$.

Note that our distinction between nonterminals and terminals is rather uncommon for STSG [see Chiang (2005)], but improves the generative power. We chose the symbol "$\leftrightarrow$" because STSG productions are symmetric. The size $|n\colon t \overset{a}{\leftrightarrow} u|$ of a production is $|t| + |u|$, and the size $|M|$ of the STSG M is $\sum_{p \in P} |p|$. It is a *weighted tree substitution grammar* (TSG) if $t = u$ for all productions $n\colon t \overset{a}{\leftrightarrow} u \in P$. Further, it is in *normal form* if for every production $n\colon t \overset{a}{\leftrightarrow} u \in P$ there exist $\sigma \in \Sigma_k$, $\delta \in \Delta_k$, and nonterminals $n_1, \ldots, n_k, n'_1, \ldots, n'_k \in N$ such that $t = \sigma(n_1, \ldots, n_k)$ and $u = \delta(n'_1, \ldots, n'_k)$. A detailed exposition to STSGs and STSGs in normal form (also called synchronous context-free grammars) can be found in Chiang (2005). Further details on TSGs can be found in Berstel and Reutenauer (1982) and Fülöp and Vogler (2009).

Equal nonterminals in t and u of a production $n\colon t \overset{a}{\leftrightarrow} u \in P$ are *linked*. To keep the presentation simple, we assume that those links are re-

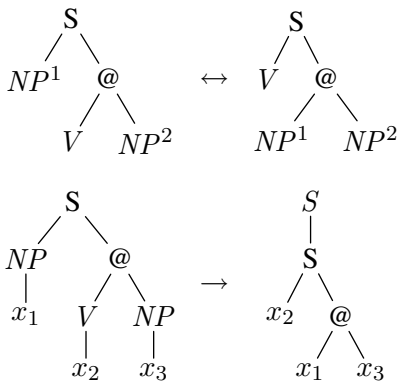


Figure 1: STSG production (top) and corresponding MBOT rule (bottom) where @ is an arbitrary symbol that is introduced during binarization.

membered also in sentential forms. In addition, we assume that $N \cap \Sigma = \emptyset$. For every $c, c' \in C_\Sigma(N)$ and $n \in N$, let $(c[n], c'[n]) \overset{a}{\Rightarrow} (c[t], c'[u])$ if

- there is a production $n\colon t \overset{a}{\leftrightarrow} u \in P$, and
- the explicit (the ones replacing $\square$) occurrences of n in $c[n]$ and $c'[n]$ are linked.

Left-most derivations are defined as usual, and the weight of a derivation $D\colon \xi_0 \overset{a_1}{\Rightarrow} \cdots \overset{a_k}{\Rightarrow} \xi_k$ is $\mathrm{wt}(D) = \prod_{i=1}^{k} a_i$. The weight assigned by the grammar M to a pair $(t, u) \in T_\Sigma \times T_\Delta$ is

$$M(t,u) = \sum_{n \in N} I(n) \cdot \sum_{\substack{D \text{ left-most derivation} \\ \text{from } (n,n) \text{ to } (t,u)}} \mathrm{wt}(D) \ .$$

The second restriction on productions ensures that derivations are of finite length, and thus that the sums in the definition of $M(t, u)$ are finite.

In the following, we will use syntactic simplifications such as

- several occurrences of the same nonterminal in a tree (disambiguated by decoration).
- symbols that are terminals (of Σ and Δ) and nonterminals. We will print nonterminals in italics and terminal symbols upright.
- omission of the nonterminal n (or the weight a) of a rule $n\colon t \overset{a}{\leftrightarrow} u$ if the terminal n occurs at the root of t and u (or $a = 1$).
- $n \overset{a}{\to} t$ instead of $n\colon t \overset{a}{\leftrightarrow} t$ if it is a TSG.

A sample STSG production (using those simplifications) is displayed in Figure 1. Our STSGs are essentially equivalent to the (nondeleting and linear) extended tree transducers of Graehl et al. (2008) and Maletti et al. (2009).

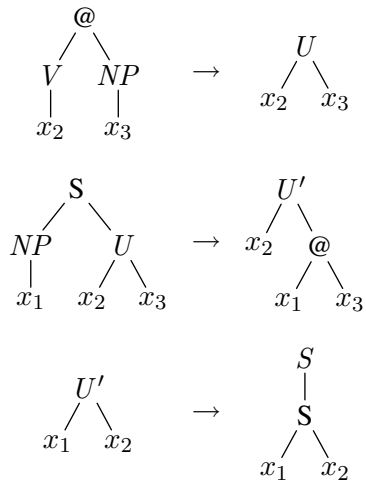


Figure 2: Sample MBOT rules in one-symbol normal form.

3 Multi bottom-up tree transducers

As indicated in the Introduction, we will compare STSGs to weighted multi bottom-up tree transducers, which have been introduced by Arnold and Dauchet (1982) and Lilin (1981). A more detailed (and English) presentation can be found in Engelfriet et al. (2009). Let us quickly recall the formal definition. We use a fixed set $X = \{x_1, x_2, \dots\}$ of (formal) variables. For a ranked alphabet S and $L \subseteq T_\Sigma(X)$ we let

$$S(L) = \{s(t_1, \dots, t_k) \mid s \in S_k, t_1, \dots, t_k \in L\}$$

and we treat elements of $S(L)$ like elements of $T_{\Sigma \cup S}(X)$.

Definition 1 *A* weighted multi bottom-up tree transducer *(MBOT) is a system* $(S, \Sigma, \Delta, F, R)$ *where*

- S, Σ, *and* Δ *are ranked alphabets of states, input symbols, and output symbols, respectively,*
- $F\colon S_1 \to \mathbb{R}$ *assigns final weights, and*
- R *is a finite set of rules* $l \overset{a}{\to} r$ *where* $a \in \mathbb{R}$, $l \in T_\Sigma(S(X))$, *and* $r \in S(T_\Delta(X))$ *such that*
  - *every* $x \in X$ *that occurs in* l *occurs exactly once in* r *and vice versa, and*
  - $l \notin S(X)$ *or* $r \notin S(X)$.

Roughly speaking, an MBOT is the bottom-up version of an extended top-down tree transducer, in which the states can have a rank different from 1. We chose the symbol "$\to$" because rules have a distinguished left- and right-hand side. The size $|l \overset{a}{\to} r|$ of

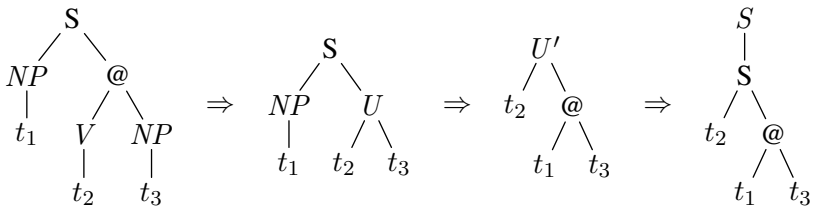


Figure 3: Derivation using the MBOT rules of Fig. 2.

a rule is $|l| + |r|$, and the size $|M|$ of an MBOT M is $\sum_{r \in R} |r|$. Again the second condition on the rules will ensure that derivations will be finite. Let us continue with the rewrite semantics for the MBOT $(S, \Sigma, \Delta, F, R)$. To simplify the presentation, we again assume that $S \cap (\Sigma \cup \Delta) = \emptyset$. We need the concept of substitution. Let $\theta \colon X \to T_\Delta$ and $t \in T_\Delta(X)$. Then $t\theta$ is the tree obtained by replacing every occurrence of $x \in X$ in t by $\theta(x)$.

Definition 2 *Let $c \in C_\Sigma(S(X))$ and $\theta \colon X \to T_\Delta$. Then $c[l\theta] \overset{a}{\Rightarrow} c[r\theta]$ if $l \overset{a}{\to} r \in R$. The weight of a derivation $D \colon \xi_0 \overset{a_1}{\Rightarrow} \cdots \overset{a_k}{\Rightarrow} \xi_k$ is $\mathrm{wt}(D) = \prod_{i=1}^{k} a_i$. The weight assigned by the MBOT M to a pair $(t, u) \in T_\Sigma \times T_\Delta$ is*

$$M(t,u) = \sum_{s \in S_1} F(s) \cdot \sum_{\substack{D \text{ left-most derivation} \\ \text{from } t \text{ to } s(u)}} \mathrm{wt}(D) \ .$$

We use the simplifications already mentioned in the previous section also for MBOTs. Figures 1 and 2 display example rules of an MBOT. The rules of Figure 2 are applied in a derivation in Figure 3. The first displayed derivation step uses the context $\mathrm{S}(NP(t_1), \square)$ and any substitution θ such that $\theta(x_2) = t_2$ and $\theta(x_3) = t_3$.

It is argued by Chiang (2005) and Graehl et al. (2008) that STSGs (and extended tree transducers) have sufficient power for syntax-based machine translation. Knight (2007) presents a detailed overview that also mentions short-comings. Since our newly proposed device, the MBOT, should be at least as powerful as STSGs, we quickly demonstrate how each STSG can be coded as an MBOT. An STSG production and the corresponding MBOT rule are displayed in Figure 1. Since the correspondence is rather trivial, we omit a formal definition.

Theorem 3 *For every STSG M, an equivalent MBOT can be constructed in time $\mathcal{O}(|M|)$.*

4 Binarization

Whenever nondeterminism enters the playfield, binarization becomes an important tool for efficiency reasons. This is based on the simple, yet powerful observation that instead of making 5 choices from a space of n in one instant (represented by n^5 rules), it is more efficient (Wang et al., 2007) to make them one-by-one (represented by $5n$ rules). Clearly, this cannot always be done but positive examples exist in abundance; e.g., binarization of context-free grammars [see CHOMSKY normal form in Hopcroft and Ullman (1979)].

Binarization of tree language devices typically consists of two steps: (i) binarization of the involved trees (using the auxiliary symbol @) and (ii) adjustment (binarization) of the processing device to work on (and fully utilize) the binarized trees. If successful, then this leads to binarized derivation trees for the processing device. In Figure 4 we show the binarization of the trees in an STSG production. Another binarization of the rule of Figure 4 is displayed in Figure 1. The binarization is evident enough, so we can assume that all trees considered in the following are binarized.

The binarization in Figure 1 is unfortunate because the obtained production cannot be factorized such that only two nonterminals occur in each rule. However, the binarization of Figure 4 allows the factorization into $\mathrm{S}(U, NP) \leftrightarrow \mathrm{S}(U, NP)$ and $U \colon @(NP, V) \leftrightarrow @(V, NP)$, which are fully binarized productions. However, in general, STSGs (or SCFGs or extended tree transducers) cannot be fully binarized as shown in Aho and Ullman (1972).

Zhang et al. (2006) and Wang et al. (2007) show the benefits of fully binarized STSGs and present a linear-time algorithm for the binarization of *binarizable* STSGs. We show that those benefits can be reaped for *all* STSGs by a simple change of model.

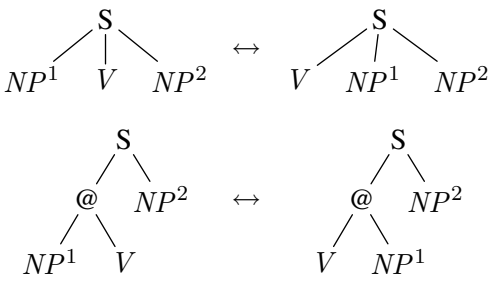


Figure 4: Binarization of trees in an STSG production. Top: Original — Bottom: Binarized trees.

We have already demonstrated that every STSG can be transformed into an equivalent MBOT in linear time. Next, we discuss binarization of MBOTs.

An MBOT is in *one-symbol normal form* if there is at most one input and at most one output symbol, but at least one symbol in each rule (see Figure 2). Raoult (1993) and Engelfriet et al. (2009) prove that every MBOT can be transformed into one-symbol normal form. The procedure presented there runs in linear time in the size of the input MBOT. Consequently, we can transform each STSG to an equivalent MBOT in one-symbol normal form in linear time. Finally, we note that a MBOT in one-symbol normal form has binarized derivation trees, which proves that we fully binarized the STSG.

Theorem 4 *For every STSG M an equivalent, fully binarized MBOT can be constructed in $\mathcal{O}(|M|)$.*

The construction of Engelfriet et al. (2009) is illustrated in Figure 2, which shows the rules of an MBOT in one-symbol normal form. Those rules are constructed from the unlucky binarization of Figure 1. In the next section, we show the benefit of the full binarization on the example of the BAR-HILLEL construction.

5 Input and output restriction

A standard construction for transformation devices (and recognition devices alike) is the regular restriction of the input or output language. This construction is used in parsing, integration of a language model, and the computation of certain metrics [see Nederhof and Satta (2003), Nederhof and Satta (2008), and Satta (2010) for a detailed account]. The construction is generally known as BAR-HILLEL construction [see Bar-Hillel et al. (1964) for the original construction on context-free grammars].

STSGs (and extended tree transducers) are symmetric, so that input and output can freely be swapped. Let M be an STSG and A a weighted string automaton with states S. In the BAR-HILLEL construction for M and A, the maximal rank $\mathrm{rk}(M)$ of a symbol in the derivation forest of M enters as an exponent into the complexity $\mathcal{O}(|M| \cdot |S|^{2\,\mathrm{rk}(M)+5})$. Since full binarization is not possible in general, the maximal rank cannot be limited to 2. In contrast, full binarization is possible for MBOTs (with only linear overhead), so let us investigate whether we can exploit this in a BAR-HILLEL construction for MBOTs.

Let $M = (S, \Sigma, \Delta, F, R)$ be an MBOT in one-symbol normal form. The symbols in $\Sigma \cup \Delta$ have rank at most 2. Moreover, let $G = (N, \Sigma, \Sigma, I, P)$ be a TSG in normal form. We want to construct an MBOT M' such that $M'(t, u) = M(t, u) \cdot G(t)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$. In other words, each input tree should be rescored according to G; in the unweighted case this yields that the translation of M is filtered to the set of input trees accepted by G.

We occasionally write the pair (a, b) in angled parentheses ('$\langle$' and '$\rangle$'). In addition, we use the center line ellipsis '$\cdots$' (also with decoration) like a variable (especially for sequences).

Definition 5 *The input product* $\mathrm{Prod}(M, G)$ *is the MBOT* $\mathrm{Prod}(M, G) = (S \times N, \Sigma, \Delta, F', R')$ *where*

- $F'(\langle s, n\rangle) = F(s) \cdot I(n)$ *for every* $s \in S$ *and* $n \in N$,
- *for every rule* $s(\cdots) \xrightarrow{a} s'(\cdots') \in R$ *with* $s, s' \in S$ *and every* $n \in N$, *there exists a rule*

$$\langle s, n\rangle(\cdots) \xrightarrow{a} \langle s', n\rangle(\cdots') \in R' \ ,$$

- *for every rule* $\sigma(s_1(\cdots_1), \ldots, s_k(\cdots_k)) \xrightarrow{a} s(\cdots)$ *in* R *with* $\sigma \in \Sigma_k$ *and* $s, s_1, \ldots, s_k \in S$, *and every production* $n \xrightarrow{b} \sigma(n_1, \ldots, n_k) \in P$, *the following rule is in* R'*:*

$$\sigma(\langle s_1, n_1\rangle(\cdots_1), \ldots, \langle s_k, n_k\rangle(\cdots_k)) \xrightarrow{ab} \langle s, n\rangle(\cdots) \ .$$

The first type of rule (second item) does not involve an input symbol, and thus the nonterminal of G is just forwarded to the new state. Since no step with respect to G is made, only the weight of the rule of M is charged. The second type of rule (third item) uses a rule of R with the input symbol σ

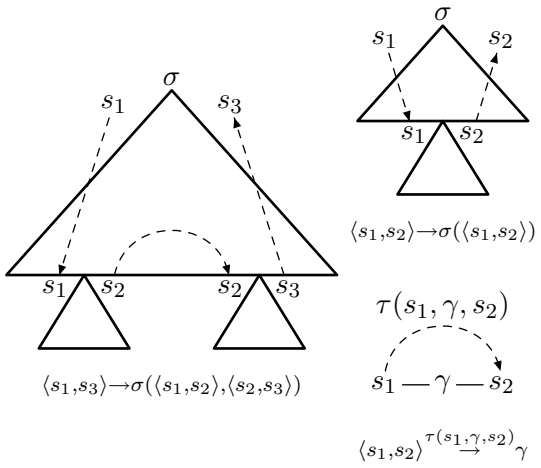


Figure 5: Constructing a TSG from a weighted string automaton.

and a production of P that also contains σ. The rule and the production are executed in parallel in the resulting rule and its weight is thus the product of the weights of the original rule and production. Overall, this is a classical product construction, which is similar to other product constructions such as Borchardt (2004). A straightforward proof shows that $M'(t,u) = M(t,u) \cdot G(t)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$, which proves the correctness.

Next, let us look at the complexity. The MBOT $\mathrm{Prod}(M, G)$ can be obtained in time $\mathcal{O}(|M| \cdot |G|)$. Furthermore, it is known [see, for example, Maletti and Satta (2009)] that for every weighted string automaton A with states S, we can construct a TSG G in normal form, which has size $\mathcal{O}(|\Sigma| \cdot |S|^3)$ and recognizes each tree of T_Σ with the weight that the automaton A assigns to its yield. The idea of this construction is illustrated in Figure 5. Consequently, our BAR-HILLEL construction has the well-known complexity $\mathcal{O}(|M| \cdot |S|^3)$. This should be compared to the complexity of the corresponding construction for an STSG M, which is in $\mathcal{O}(|M| \cdot |S|^{2\,\mathrm{rk}(M)+5})$ where $\mathrm{rk}(M)$ is the maximal number of (different) nonterminals in a production of M. Thus, the STSG should be transformed into an equivalent MBOT in one-symbol normal form, which can be achieved in linear time, and the BAR-HILLEL construction should be performed on this MBOT.

Since STSGs are symmetric, our approach can also be applied to the output side of an STSG. However, it should be noted that we can apply it only to one side (the input side) of the MBOT. A construction for the output side of the MBOT can be defined, but it would suffer from a similarly high complexity as already presented for STSGs. More precisely, we expect a complexity of roughly $\mathcal{O}(|M| \cdot |S|^{2\,\mathrm{rk}(M)+2})$ for this construction. The small gain is due to the one-symbol normal form and binarization.

6 Composition

Another standard construction for transformations is (relational) composition. Composition constructs a translation from a language L to L'' given translations from L to L' and from L' to L''. Formally, given transformations $M' \colon T_\Sigma \times T_\Delta \to \mathbb{R}$ and $M'' \colon T_\Delta \times T_\Gamma \to \mathbb{R}$, the composition of M' and M'' is a tranformation $M' ; M'' \colon T_\Sigma \times T_\Gamma \to \mathbb{R}$ with

$$(M' ; M'')(t, v) = \sum_{u \in T_\Delta} M'(t, u) \cdot M''(u, v)$$

for every $t \in T_\Sigma$ and $v \in T_\Gamma$. Mind that the summation might be infinite, but we will only consider compositions, in which it is finite.

Unfortunately, Arnold and Dauchet (1982) show that the composition of two transformations computed by STSGs cannot necessarily be computed by an STSG. Consequently, there cannot be a general composition algorithm for STSGs.

Let us consider the problem of composition for MBOTs. Essentially, we will follow the unweighted approach of Engelfriet et al. (2009) to obtain a composition construction, which we present next. Let

$$M' = (S', \Sigma, \Delta, F', R') \quad \text{and}$$
$$M'' = (S'', \Delta, \Gamma, F'', R'')$$

be MBOTs in one-symbol normal form. We extend the rewrite semantics (see Definition 2) to trees that include symbols foreign to a MBOT. In other words, we (virtually) extend the input and output alphabets to contain all used symbols (in particular also the states of another MBOT). However, since we do not extend the set of rules, the MBOT cannot process foreign symbols. Nevertheless it can perform rewrite steps on known symbols (or apply rules that do not contain input symbols). We use $\Rightarrow_{R'}$ and $\Rightarrow_{R''}$ for derivation steps

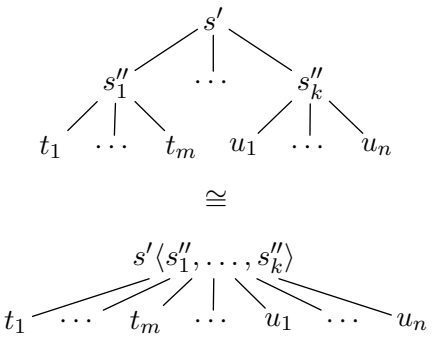


Figure 6: Identification in sentential forms.

that exclusively use rules of R' and R'', respectively. In addition, we identify $s'(s''_1(\cdot_1), \ldots, s''_k(\cdot_k))$ with $s'\langle s''_1, \ldots, s''_k\rangle(\cdot_1, \ldots, \cdot_k)$ for $s' \in S'$ and $s''_1, \ldots, s''_k \in S''$. This identification is illustrated in Figure 6.

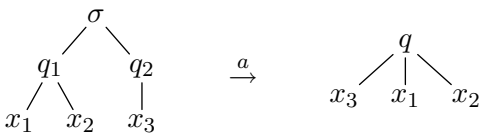
Definition 6 *The MBOT $M'; M'' = (S, \Sigma, \Gamma, F, R)$ is such that*

- *for every $s' \in S'_k$ and $s''_1 \in S''_{\ell_1}, \ldots, s''_k \in S''_{\ell_k}$ we have $s'\langle s''_1, \ldots, s''_k\rangle \in S_{\ell_1+\cdots+\ell_k}$,*
- *$F(s'\langle s'\rangle) = F'(s') \cdot F''(s'')$ for every $s' \in S'_1$ and $s'' \in S''_1$, and*
- *the rules $l \xrightarrow{a} r$ of R, all of which are such that the variables in l occur in order $(x_1, \ldots, x_k)$ from left-to-right, are constructed in 3 ways:*
  - *$l \overset{a}{\Rightarrow}_{R'} r$ by a single rule of R',*
  - *$l \overset{a}{\Rightarrow}_{R''} r$ by a single rule of R'', or*
  - *$l \overset{a_1}{\Rightarrow}_{R'} \xi \overset{a_2}{\Rightarrow}_{R''} r$ with $a = a_1 \cdot a_2$ and the applied rule of R' contains an output symbol.*

  If a rule $l \xrightarrow{a} r$ can be constructed in several ways (with exactly weight a), then the weights of all possibilities are added for the weight of the new rule.

Intuitively, a single rule of R' without output symbols is used in the first type (because otherwise r would have the wrong shape). In the second type, a single rule of R'' without input symbols is used. Finally, in the third type, first a rule of R' that produces an output symbol of Δ is used and then this symbol is processed by a single rule of R''. Note that every rule of R' can produce at most one output symbol and the rules of R'' either process none or one input symbol due to the assumption that M' and M'' are in one-symbol normal form. We illustrate a rule of the first in Figure 7.

original rule:



constructed rule:

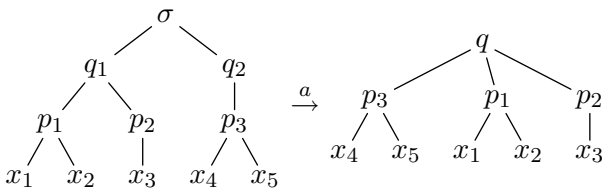


Figure 7: Example of a constructed rule of type 1.

The correctness proof of this construction can essentially (i.e., for the unweighted case) be found in Engelfriet et al. (2009). Before we can extend it to the weighted case, we need to make sure that the sum in the definition of composition is finite. We achieve this by requiring that

- for every $t \in T_\Sigma$ and $s \in S'_1$ there are finitely many $u \in T_\Delta$ such that $t \overset{a_1}{\Rightarrow} \cdots \overset{a_n}{\Rightarrow} s(u)$, or
- for every $v \in T_\Gamma$ and $s \in S''_1$ there are finitely many $u \in T_\Delta$ such that $u \overset{a_1}{\Rightarrow} \cdots \overset{a_n}{\Rightarrow} s(v)$.

In other words, M' may not have cyclic input ε-rules or M'' may not have cyclic output ε-rules. Now we can state the main theorem.

Theorem 7 *For all MBOTs M' and M'' with the above restriction the composition $M'; M''$ of their transformations can be computed by another MBOT.*

This again shows an advantage of MBOTs. The composition result relies essentially on the one-symbol normal form (or full binarization), which can always be achieved for MBOTs, but cannot for STSGs. Consequently, MBOTs can be composed, whereas STSGs cannot be composed in general. Indeed, STSGs in one-symbol normal form, which can be defined as for MBOTs, can be composed as well, which shows that the one-symbol normal form is the key for composition.

Finally, let us discuss the complexity of composition. Let $\mathrm{rk}(M')$ be the maximal rank of a state in S'. Then there are

- $\mathcal{O}(|M'| \cdot |S''|^{\mathrm{rk}(M')})$ rules of type 1,
- $\mathcal{O}(|M''| \cdot |S''|^{\mathrm{rk}(M')})$ rules of type 2, and

- $\mathcal{O}(|M'| \cdot |M''| \cdot |S''|^{\mathrm{rk}(M')})$ rules of type 3.

Each rule can be constructed in linear time in the size of the participating rules, so that we obtain a final complexity of $\mathcal{O}(|M'| \cdot |M''| \cdot |S''|^{\mathrm{rk}(M')})$. Note that if M' is obtained from an STSG M (via Theorem 4), then $\mathrm{rk}(M') \leq \mathrm{rk}(M)$. This shows that binarization does not avoid the exponent for composition, but at least enables composition in the general case. Moreover, the complexity could be slightly improved by the observation that our construction only relies on (i) M' having at most one output symbol per rule and (ii) M'' having at most one input symbol per rule.

7 Forward and backward application

We might want to apply a transformation not just to a single tree, but rather to a set of trees, which are, in some cases, already weighted. In general, the set of trees is given by a TSG G and we expect the result to be represented by a TSG as well. Forward and backward application amount to computing the image and pre-image of G under the transformation, respectively. Since STSG are symmetric, we need to solve only one of the problems if the transformation is given by an STSG. The other problem can then be solved by inverting the STSG (exchanging input and output) and using the method for the solved problem. We chose to address forward application here.

Forward application can be reduced to the problem of computing the co-domain (or range) with the help of a product construction for STSG, which is similar to the one presented in Definition 5. The co-domain cod_M of the tranformation computed by an STSG M assigns to each $t \in T_\Sigma$ the weight

$$\mathrm{cod}_M(t) = \sum_{u \in T_\Delta} M(t, u) \ .$$

This sum might not be well-defined. However, if $u \notin N$ for all productions $n\colon t \overset{a}{\leftrightarrow} u$ of the STSG, then the sum is well-defined and the output-side TSG (i.e., for every production $n\colon t \overset{a}{\leftrightarrow} u$ in the STSG there is a production $n \overset{a}{\rightarrow} u$ in the TSG) computes the co-domain. The restriction "$u \notin N$" guarantees that the output side is a TSG. Overall, domain, co-domain, and forward and backward applications (using the product construction) can be computed given such minor new requirements.

Also for transformations computed by MBOTs we can reduce the problem of forward application to the problem of computing the co-domain with the help of the product construction of Definition 5. However, the co-domain of an MBOT is not necessarily representable by a TSG, which is not due to well-definedness problems but rather the *finite-copying* property (Engelfriet et al., 1980) of MBOTs. This property yields that the co-domain might not be a regular tree language (or context-free string language). Consequently, we cannot compute forward or backward applications for arbitrary MBOT. However, if the MBOT is equivalent to an STSG (for example, because it was constructed by the method presented before Theorem 3), then forward and backward application can be computed essentially as for STSG. This can be understood as a warning. MBOT can efficiently be used (with computational benefits) as an alternative representation for transformations computed by STSG (or compositions of STSG). However, MBOT can also compute transformations, of which the domain or range cannot be represented by a TSG. Thus, if we train MBOT directly and utilize their full expressive power, then we might not be able to perform forward and backward application.

In the unweighted case, backward application can always be computed for MBOT. Moreover, it can be decided using (Ésik, 1984) whether all forward applications can be represented by TSGs. However, for a given specific TSG, it cannot be decided whether the forward application is representable by a TSG, which was proved by Fülöp (1994). A subclass of transformations computable by MBOT (that still contains all transformations computable by STSG), which allows all forward and backward applications, has been identified by Raoult (1993).

Conclusion and acknowledgement

We compared STSGs and MBOTs on several standard algorithms (binarization, regular restriction, composition, and application). We prove that MBOTs offer computational benefits on all mentioned algorithms as long as the original transformation is computable by an STSG.

The author was financially supported by the *Ministerio de Educación y Ciencia* (MEC) grants JDCI-2007-760 and MTM-2007-63422.

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.

Y. Bar-Hillel, M. Perles, and E. Shamir. 1964. On formal properties of simple phrase structure grammars. In *Language and Information: Selected Essays on their Theory and Application*, pages 116–150. Addison Wesley.

Jean Berstel and Christophe Reutenauer. 1982. Recognizable formal power series on trees. *Theoret. Comput. Sci.*, 18(2):115–148.

Björn Borchardt. 2004. A pumping lemma and decidability problems for recognizable tree series. *Acta Cybernet.*, 16(4):509–544.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. Mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

David Chiang and Kevin Knight. 2006. An introduction to synchronous grammars. In *Proc. ACL tutorial*.

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

John DeNero, Mohit Bansal, Adam Pauls, and Dan Klein. 2009a. Efficient parsing for transducer grammars. In *Proc. NAACL*, pages 227–235.

John DeNero, Adam Pauls, and Dan Klein. 2009b. Asynchronous binarization for synchronous grammars. In *Proc. ACL*, pages 141–144.

Samuel Eilenberg. 1974. *Automata, Languages, and Machines*. Academic Press.

Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. 1980. Tree transducers, L systems, and two-way machines. *J. Comput. System Sci.*, 20(2):150–202.

Joost Engelfriet, Eric Lilin, and Andreas Maletti. 2009. Extended multi bottom-up tree transducers: Composition and decomposition. *Acta Inform.*, 46(8):561–590.

Zoltán Ésik. 1984. Decidability results concerning tree transducers II. *Acta Cybernet.*, 6(3):303–314.

Zoltán Fülöp and Heiko Vogler. 2009. Weighted tree automata and tree transducers. In *Handbook of Weighted Automata*, chapter IX, pages 313–403. Springer.

Zoltán Fülöp. 1994. Undecidable properties of deterministic top-down tree transducers. *Theoret. Comput. Sci.*, 134(2):311–328.

Jonathan S. Golan. 1999. *Semirings and their Applications*. Kluwer Academic, Dordrecht.

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

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley.

Liang Huang, Hao Zhang, Daniel Gildea, and Kevin Knight. 2009. Binarization of synchronous context-free grammars. *Computational Linguistics*, 35(4):559–595.

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

Eric Lilin. 1981. Propriétés de clôture d'une extension de transducteurs d'arbres déterministes. In *CAAP*, volume 112 of LNCS, pages 280–289. Springer.

Andreas Maletti and Giorgio Satta. 2009. Parsing algorithms based on tree automata. In *Proc. IWPT*, pages 1–12.

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

Mehryar Mohri. 2009. Weighted automata algorithms. In *Handbook of Weighted Automata*, pages 213–254. Springer.

Mark-Jan Nederhof and Giorgio Satta. 2003. Probabilistic parsing as intersection. In *Proc. IWPT*, pages 137–148.

Mark-Jan Nederhof and Giorgio Satta. 2008. Computation of distances for regular and context-free probabilistic languages. *Theoret. Comput. Sci.*, 395(2–3):235–254.

Jean-Claude Raoult. 1993. Recursively defined tree transductions. In *Proc. RTA*, volume 690 of LNCS, pages 343–357. Springer.

Jacques Sakarovitch. 2009. Rational and recognisable power series. In *Handbook of Weighted Automata*, chapter IV, pages 105–174. Springer.

Giorgio Satta. 2010. Translation algorithms by means of language intersection. Manuscript.

Marcel Paul Schützenberger. 1961. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270.

Wei Wang, Kevin Knight, and Daniel Marcu. 2007. Binarizing syntax trees to improve syntax-based machine translation accuracy. In *Proc. EMNLP-CoNLL*, pages 746–754.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proc. NAACL-HLT*, pages 256–263.