# Translation Algorithms by Means of Language Intersection

Giorgio Satta*
Department of Information Engineering
University of Padua

*Synchronous rewriting grammars have been successfully exploited as translation models in machine translation applications. In this article we consider the problem of the design of translation algorithms based on synchronous context-free grammars. We revisit a methodology for the design of parsing algorithms based on the idea of language intersection, that has fully been developed in the parsing community, and show how to apply it in the design of translation algorithms.*

*We argue that the intersection methodology above can also be viewed as a framework for the comparison of translation algorithms and for the formal analysis of their properties. On this line, we observe how superficially different translation algorithms that have been proposed in the literature can be viewed as special cases of application of the intersection methodology. We also use our framework to reformulate and improve some results already presented in the literature.*

## 1. Introduction

State of the art machine translation architectures are all based on mathematical models called translation models. Generally speaking, a translation model accounts for all of the elementary operations that rule the process of translation between the different word orderings of the source and target languages. Translation models are usually enriched with statistical parameters, to help driving the search in the space of all valid transformations of the source sentence into the target sentence. Specialized algorithms are also provided for the automatic estimation of these parameters from corpora of translation pairs. Besides the task of natural language translation, translation models are also exploited in other applications, such as word and phrase alignments, multilingual document retrieval, automatic dictionary construction and several others.

* via Gradenigo 6/A, I-35131 Padova, Italy, E-mail: satta@dei.unipd.it

© yyyy Association for Computational Linguistics

The most successful translation models that are found in the literature exploit finite-state machinery. The approach started with the so-called IBM models (Brown et al. 1993), implementing a set of elementary operations, such as movement, duplication and translation, that independently act on individual words in the source sentence. These word-to-word models have been later enriched with the introduction of larger units such as phrases; see for instance work by Och, Tillmann, and Ney (1999), Och and Ney (2002) and Koehn, Och, and Marcu (2003). Still, the generative capacity of these models lies within the realm of finite-state machinery (Kumar and Byrne 2003), so they are unable to handle nested structures and do not provide the expressivity required to process language pairs with very different word orderings.

Rather independently of the above developments in statistical machine translation, the computational linguistics community also investigated the translation problem under a syntactic perspective, adopting formal approaches that extended the generative capacity of the finite-state paradigm by introducing hierarchical language descriptions. Borrowing from the theory of formal language and compilers, many of these approaches made use of synchronous rewriting; see for instance work by Shieber and Schabes (1990), Shieber (1994) and Dras (1999). In synchronous rewriting two formal grammars are exploited, one describing the source language and the other describing the target language. The productions of the two grammars are paired and, in the rewriting process, such pairs are always applied synchronously.

Very recently, there has been some sort of convergence between the lines of investigation presented above, and more sophisticated statistical translation models have been adopted by the machine translation community, in the attempt to give more structured accounts of the human language translation process and to improve state of the art translation accuracy. Formalisms based on hierarchical language descriptions

and synchronous rewriting have been adopted and empowered with the use of statistical parameters, and specialized estimation and translation (decoding) algorithms were newly developed for these formalisms. Among the several formalisms that have been proposed in the literature, we mention here the inversion transduction grammar of Wu (1997), the recursive system based on head transducers proposed by Alshawi, Bangalore, and Douglas (2000), the tree-editing system developed by Yamada and Knight (2001), the multitext grammars of Melamed (2003), the synchronous tree substitution grammar of Eisner (2003), and the synchronous context-free grammars with a single nonterminal proposed by Chiang (2005). Besides synchronous rewriting, other paradigms have been exploited in the development of hierarchical translation models. See for instance work by Galley et al. (2004) and May and Knight (2006), using tree transducer models (Knight and Graehl 2005), and work by Poutsma (2000) and Way (2003), inspired by the data-oriented parsing paradigm (Bod, Scha, and Sima'an 2003).

In this article we investigate the synchronous rewriting paradigm introduced above, and focus on the development of algorithms for the associated translation problem and other related problems as well, such as the problem of synchronous parsing and the tree translation problem. In the translation problem one is given as input a sentence in some source language, and must construct a suitable representation of all possible translations of that sentence in some target language. The theory and the algorithms presented in this article are intended as an abstract framework that serves two main goals:

- provide a general methodology that can be used to develop new algorithms for the translation problem and to investigate their formal properties;

- propose a unified treatment of superficially different algorithms that, in an application oriented setting, have been recently presented in the literature for the translation problem.

The framework we propose is naturally derived from a construction that has been originally developed in the theory of formal languages, the so-called Bar-Hillel construction (Bar-Hillel, Perles, and Shamir 1964). Later, such a construction has been largely adopted in natural language parsing applications; see for instance work by Lang (1991), Nederhof and Satta (2003) and references therein. In a practical perspective, what this means is that we will develop translation algorithms by entirely using well known, off-the-shelf parsing tools.

The present proposal deals with synchronous rewriting formalisms based on context-free grammars. This is motivated by the fact that, among all the approaches that adopt the synchronous rewriting paradigm, the choice of synchronous context-free rewriting is the most common in statistical machine translation applications. Although it is beyond the scope of this article to provide a detailed overview of all the synchronous formalisms cited above, we discuss some examples showing how the results presented in this article can be extended to those formalisms. Furthermore, the methodology we develop here can easily be extended to generatively more powerful synchronous systems, as for instance the synchronous tree adjoining grammars of Shieber (1994) and the generalized multitext grammars proposed by Melamed, Satta, and Wellington (2004b).

We are aware of few other attempts to provide a framework for the unified specification of translation algorithms based on models beyond the finite-state paradigm. Bertsch and Nederhof (2001) have adapted the range concatenation grammars of Boullier (2004) to generate languages of string pairs, thus implementing language transla-

tion, and have outlined a translation algorithm based on the same idea of language intersection considered in the present article. Since range concatenation grammars are generatively more powerful than the kind of synchronous grammars that are used in practical applications, their proposal could be used to provide translation algorithms for those formalisms. However, this idea is not pursued by the authors in their work. A second framework for the specification of translation algorithms has been presented in (Melamed 2004; Melamed and Wang 2005), based on the already mentioned multitext grammars, which are a variant of the synchronous context-free grammars presented in this article. Rather than focusing on the specification of a modular architecture for the development of translation algorithms, as we do in this article, the cited authors focus on the problem of grammar specification, showing how to transform multitext grammars in order to exploit standard deductive parsing techniques to obtain translation algorithms. Finally, in work by May and Knight (2006) several algorithms are specified for the combination of tree transducers. This can be viewed as a modular architecture for the development of translation algorithms. Since tree transducers are considerably different from the synchronous context-free grammars of this article, the two approaches involve quite different techniques and should be considered independent one of the other. On a perspective different from the algorithmic one developed in this article, a very interesting generative characterization of synchronous rewriting using tree transducers has been proposed in (Shieber 2004, 2006).

We conclude with a summary of the content of the sections to follow. In section 2 we introduce synchronous context-free grammars and some related definitions. Following a modular approach in the specification of computer algorithms, in sections 3 and 4 we present some basic constructions and representations, which we borrow from the literature on parsing and adapt to our synchronous context-free grammars. These tools

are then used in section 5 to specify several algorithms for the translation and related problems. We close with some discussion in section 6.

## 2. Synchronous context-free grammars

In this section we introduce synchronous context-free grammars, which is the central formalism in this article. Several translation models based on synchronous context-free grammars have been proposed in the literature, but most often these models have been introduced rather informally, and no precise definition of the underlying formalism and of the associated derive relation has been provided. What we propose here is based on the definition of syntax-directed transduction grammars (Lewis and Stearns 1968), successively called syntax-directed translation schemata (SDTS); see Aho and Ullman (1972) and references therein.

In a SDTS, the productions of two context-free grammars are paired in such a way that the occurrences of the nonterminal symbols in the right-hand side of a production in a pair are a permutation of the occurrences of the nonterminal symbols in the right-hand side of the other production. Thus, paired nonterminals are always equal. There is no restriction on the terminal symbols of the productions in a pair. SDTS are also associated with a rewriting relation imposing that paired nonterminals are rewritten synchronously, that is, at the same time, by means of some production pair. Overall, in a SDTS two paired context-free productions specify, through a permutation, how a particular sequence of phrases in the source string should be reordered in the target string. These permutations are then combined hierarchically, through the usual process of context-free rewriting. As a result, a SDTS assigns to a pair of strings two context-free parse trees that have the same skeleton but differ by a reordering of the children of each internal node and by the translation, insertion and deletion of the leaf nodes.

We generalize SDTS by allowing pairing of nonterminals that are not equal.[1] Although this generalization does not add to the weak generative power of the model, that is, the string mappings defined by the two models are the same, it does increase its strong generative capacity, that is, the parse tree mappings defined by SDTS are a proper subset of the parse tree mappings defined by synchronous context-free grammars. As a consequence of this fact, when the definitions of the two models are enriched with probabilities, synchronous context-free grammars can define certain parse tree distributions that cannot be captured by SDTS, as shown by Satta and Peserico (2005). The above generalization has been adopted in several translation models for natural language. This is for instance the case of the already mentioned multitext grammars and synchronous tree-substitution grammars. Authors that do not adopt such a generalization and follow the SDTS model, have also argued that imposing the above restriction may sacrifice the freedom of encoding meaningful syntactic information on the symbols of the grammar; see for instance (Wu 1997, pg. 381). More discussion motivating the choice of synchronous context-free grammars in place of SDTS is reported in section 6.

Before presenting the definition of synchronous context-free grammars we briefly introduce the notation we adopt in this article for context-free grammars, along with some ancillary notation. A **context-free grammar** (CFG) is a tuple $G = (V_N, V_T, P, S)$, where $V_N$ is a finite set of nonterminals, $V_T$ is a finite set of terminals with $V_T \cap V_N = \emptyset$, $S \in V_N$ is a special symbol called the start symbol, and $P$ is a finite set of productions having the form $A \rightarrow \gamma$, with $A \in V_N$ and $\gamma \in (V_T \cup V_N)^*$. In this article we adopt the following conventions: symbols $A, B, \ldots$ belong to $V_N$, symbols $a, b, \ldots$ belong to $V_T$,

1 The definition we propose here is taken from (Satta and Peserico 2005). In (Chiang 2005) the term synchronous context-free grammar is instead used to denote SDTS with the restriction that there can be only one nonterminal symbol besides the start symbol.

and symbols $u, v, w$ belong to $V_T^*$. The size of a CFG is defined as

$$|G| = \sum_{(A \rightarrow \gamma) \in P} |A\gamma| \,. \tag{1}$$

We assume the reader is already familiar with the notion of derivation associated with a CFG, based on the definition of the derive relation, written $\Rightarrow_G$; see for instance (Harrison 1978). The language (set of strings) derived in $G$ from the start symbol $S$ is denoted $L(G)$. Grammar $G$ is reduced if it does not contain any symbol or production that cannot be used in a derivation of a sting in $L(G)$. If we ignore the order of application of productions, derivations can be represented by means of parse trees, that is, ordered and node-labeled standard tree structures. We say that two parse trees $t_1$ and $t_2$ are **isomorphic**, written $t_1 \cong t_2$, if $t_1$ and $t_2$ have identical structures up to some relabeling of the internal nodes. Note that under this definition, $t_1 \cong t_2$ implies that $t_1$ and $t_2$ generate the same string.

In this article we call **parse forest** an arbitrary set of one or more parse trees that can be generated by some CFG. Parse forests are useful in representing alternative derivations for a set of one or more sentences, and are central in this article since they are the output of our translation algorithms. Despite the fact that in a CFG a single sentence of length $n$ can have a number of derivations that is an exponential function of $n$, several compact representations for parse forests have been proposed in the literature that take space polynomial in $n$. This is for instance the case of the and-or graphs and the hypergraphs. In this article parse forests are conveniently represented by means of CFGs. All of the mentioned representations are syntactic variations one of the other, as briefly discussed in section 6. We find it more convenient to use CFGs here for a reason of uniformity: since we already deal with translation models based on CFGs, the

choice of CFGs for the representation of parse forests does not force us to introduce any additional machinery.

In what follows we need to represent bijections between all the occurrences of nonterminals in two strings over $V_N \cup V_T$. This can be done by using an infinite set of indices and by annotating corresponding nonterminals with the same index. We draw indices from the set of natural numbers $\mathbb{N}$. We define

$$\mathcal{I}(V_N) = \{A^{(t)} \mid A \in V_N,\ t \in \mathbb{N}\}, \tag{2}$$

and call **indexed symbol** each symbols in $\mathcal{I}(V_N)$. We let $V_I = \mathcal{I}(V_N) \cup V_T$. As a convention, symbols $\alpha, \beta, \ldots$ are used to denote strings in $V_I^*$. For $\gamma \in V_I^*$, we write $\mathsf{index}(\gamma)$ to denote the set of all indices that appear in symbols in $\gamma$; more formally,

$$\mathsf{index}(\gamma) = \{t \mid \gamma = \gamma' A^{(t)} \gamma'',\ \gamma', \gamma'' \in V_I^*,\ A^{(t)} \in \mathcal{I}(V_N)\}. \tag{3}$$

Sometimes we need to strip off indices from the symbols of a string $\gamma \in V_I^*$. To this end we define $\mathsf{deindex}(A^{(t)}) = A$ for $A^{(t)} \in \mathcal{I}(V_N)$ and $\mathsf{deindex}(a) = a$ for $a \in V_T$. We then extend this notation to strings in $V_I^*$ by letting $\mathsf{deindex}(\varepsilon) = \varepsilon$ and $\mathsf{deindex}(X \cdot \gamma) = \mathsf{deindex}(X) \cdot \mathsf{deindex}(\gamma)$, for $X \in V_I$ and $\gamma \in V_I^*$.

**Definition 1**

Two strings $\gamma_1, \gamma_2 \in V_I^*$ are **synchronous** if each index in $\mathsf{index}(\gamma_1)$ occurs only once in $\gamma_1$, each index in $\mathsf{index}(\gamma_2)$ occurs only once in $\gamma_2$, and $\mathsf{index}(\gamma_1) = \mathsf{index}(\gamma_2)$.

From the definition above, it follows that two synchronous strings must have the same number of occurrences of symbols from $\mathcal{I}(V_N)$ and must have the form

$$\gamma_1 = u_{10} A_{11}^{(t_1)} u_{11} A_{12}^{(t_2)} u_{12} \cdots u_{1n-1} A_{1n}^{(t_n)} u_{1n},$$

$$\gamma_2 = u_{20} A_{21}^{(t_{\pi(1)})} u_{21} A_{22}^{(t_{\pi(2)})} u_{22} \cdots u_{2n-1} A_{2n}^{(t_{\pi(n)})} u_{2n},$$

where $n \geq 0$, $u_{1i}, u_{2i} \in V_T^*$, $A_{1i}^{(t_i)}, A_{2i}^{(t_{\pi(i)})} \in \mathcal{I}(V_N)$, $t_i \neq t_j$ for $i \neq j$ and $\pi$ is a permutation defined on the set $\{1, \ldots, n\}$. Observe that the above notation implies that the $\pi(i)$-th nonterminal in $\gamma_1$ (from left to right) has the same index as the $i$-th nonterminal in $\gamma_2$. Equivalently, the $i$-th nonterminal in $\gamma_1$ has the same index as the $\pi^{-1}(i)$-th nonterminal in $\gamma_2$. This observation will be implicitly used in several places in this article. More in general, when representing synchronous strings, in this article we make use of functions $\pi$ that implement permutations on any finite set of integers of size $n \geq 1$ (not necessarily the set $\{1, \ldots, n\}$).

We can now define synchronous context-free grammars.

**Definition 2**

A **synchronous context-free grammar** (SCFG) is a tuple $G = (V_N, V_T, P, S_1, S_2)$, where $V_N$, $V_T$ are finite, disjoint sets of nonterminal and terminal symbols, respectively, $S_1, S_2 \in V_N$ are start symbols and $P$ is a finite set of synchronous productions. Each synchronous production has the form

$$[A_1 \rightarrow \alpha_1, \;\; A_2 \rightarrow \alpha_2],$$

where $A_1, A_2 \in V_N$ and where $\alpha_1, \alpha_2 \in V_I^*$ are synchronous strings.

**Example 1**

Let $V_T = \{a_1, b_1, a_2, b_2\}$ and $V_N = \{S, A_1, B_1, C_1, A_2, B_2, C_2\}$. A sample SCFG $G = (V_N, V_T, P, S, S)$ is defined by letting $P$ contain the following synchronous productions:

$$
\begin{array}{l}
s_1 : [S \rightarrow A_1^{(1)} C_1^{(2)},\ \ S \rightarrow A_2^{(1)} C_2^{(2)}], \\
s_2 : [C_1 \rightarrow B_1^{(1)} S^{(2)},\ \ C_2 \rightarrow B_2^{(1)} S^{(2)}], \\
s_3 : [C_1 \rightarrow B_1^{(1)} S^{(2)},\ \ C_2 \rightarrow S^{(2)} B_2^{(1)}], \\
s_4 : [C_1 \rightarrow B_1^{(1)},\ \ C_2 \rightarrow B_2^{(1)}], \\
s_5 : [A_1 \rightarrow a_1,\ \ A_2 \rightarrow a_2], \\
s_6 : [A_1 \rightarrow a_1,\ \ A_2 \rightarrow \varepsilon], \\
s_7 : [B_1 \rightarrow b_1,\ \ B_2 \rightarrow b_2].
\end{array}
$$

Note that in $G$ the nonterminal $B_2$, which is translated from $B_1$, can be optionally inverted with $S$ when it is rewritten from $C_2$. Also, the terminal symbol $a_2$, which is translated from $a_1$, can be optionally deleted.

For a synchronous production $s : [A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2]$, we call each of $A_1 \rightarrow \alpha_1$ and $A_2 \rightarrow \alpha_2$ a **production component** of $s$. The **rank** of $s$ is defined as $|\mathsf{index}(\alpha_1)|$ (which is the same as $|\mathsf{index}(\alpha_2)|$). The rank of $G$ is the maximum among all ranks of the synchronous productions in $P$. The size of a SCFG $G$ is defined as

$$
|G| = \sum_{[A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2] \in P} |A_1 \alpha_1 A_2 \alpha_2| \,. \tag{4}
$$

Similarly to the context-free case, in (4) we take the uniform-cost assumption that each symbol in $V_I$ can be represented by means of a constant amount of space. A more realistic assumption, in which each grammar symbol is represented by some integer, would add to (4) a logarithmic factor in the size of $V_I$. Since in this article we only distinguish between polynomial and exponential time algorithms, such a factor is irrelevant to our discussion.

In a SCFG, the derive relation is defined on synchronous strings in terms of simultaneous rewriting of two nonterminals with the same index by means of corresponding production components of a synchronous production. Some additional notation

will help us defining this relation precisely. A **reindexing** is a one-to-one function on $\mathbb{N}$. We extend a reindexing $f$ to $V_I$ by letting $f(A^{(t)}) = A^{(f(t))}$ for $A^{(t)} \in \mathcal{I}(V_N)$ and $f(a) = a$ for $a \in V_T$. We also extend $f$ to strings in $V_I^*$ by letting $f(\varepsilon) = \varepsilon$ and $f(X\gamma) = f(X)f(\gamma)$, for each $X \in V_I$ and $\gamma \in V_I^*$. We say that strings $\gamma_1, \gamma_2 \in V_I^*$ are **independent** if $\mathsf{index}(\gamma_1) \cap \mathsf{index}(\gamma_2) = \emptyset$.

**Definition 3**

Let $G = (V_N, V_T, P, S_1, S_2)$ be a SCFG and let $\gamma_1, \gamma_2$ be synchronous strings in $V_I^*$. The **derive** relation

$$[\gamma_1,\ \gamma_2] \Rightarrow_G [\delta_1,\ \delta_2] \tag{5}$$

holds whenever there exist an index $t$ in $\mathsf{index}(\gamma_1)$, a synchronous production $[A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2]$ in $P$ and some reindexing $f$ such that

1. $f(\alpha_1\alpha_2)$ and $\gamma_1\gamma_2$ are independent; and
2. for $i = 1, 2$

$$\gamma_i = \gamma_i' A_i^{(t)} \gamma_i'',$$

$$\delta_i = \gamma_i' f(\alpha_i) \gamma_i''.$$

We also write $[\gamma_1,\ \gamma_2] \Rightarrow_G^s [\delta_1,\ \delta_2]$ to explicitly indicate that the derive relation holds through the synchronous production $s \in P$.

It is not difficult to see that $\delta_1, \delta_2$ in (5) are synchronous strings. Thus, we can combine instances of the derive relation above in order to represent derivations in $G$, in the following way. Assume that, for integers $i$ and $n$ with $1 \leq i \leq n$ and $n \geq 1$, we

have relations

$$[\gamma_{1i-1},\ \gamma_{2i-1}] \Rightarrow_G^{s_i} [\gamma_{1i},\ \gamma_{2i}].$$

Then we say that $\sigma = s_1 s_2 \cdots s_n$ denotes a **synchronous derivation** and we write $[\gamma_{10},\ \gamma_{20}] \Rightarrow_G^{\sigma} [\gamma_{1n},\ \gamma_{2n}]$. String $\sigma = \varepsilon$ is also a synchronous derivation, rewriting a pair of synchronous strings into itself. As usual, we also denote generic synchronous derivations through the reflexive and transitive closure of $\Rightarrow_G$, written $\Rightarrow_G^*$.

A synchronous derivation can be associated with a pair of context-free derivations producing the strings that are generated on each of the two dimensions. In order to do this, we make use of certain functions that are technically called homomorphisms. An **homomorphism** is a total function mapping some source finite alphabet into a set of strings defined over some target finite alphabet. An homomorphism $h$ is usually extended to strings over the source alphabet by defining $h(\varepsilon) = \varepsilon$ and $h(a \cdot x) = h(a) \cdot h(x)$, where $a$ is a symbol in the source alphabet and $x$ is a string over the source alphabet. The homomorphisms we use in this article are defined over sets of productions of certain grammars, which we view as finite alphabets of atomic symbols. In addition, our homomorphisms always map a source symbol into strings of length one, that is, a single symbol in the target alphabet. Thus these homomorphisms simply act as symbol relabelings.

We associate with SCFG $G$ two homomorphisms $h_1$ and $h_2$, defined over the set $P$ of $G$'s synchronous productions. For a synchronous production $s \in P$ of the form $[A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2]$, we let $h_1(s) = A_1 \rightarrow \alpha_1$ and $h_2(s) = A_2 \rightarrow \alpha_2$.

Observe that, for each synchronous derivation in $G$ of the form $[S_1^{(1)},\ S_2^{(1)}] \Rightarrow_G^{\sigma} [w_1,\ w_2]$, we have that $h_1(\sigma)$ and $h_2(\sigma)$ are derivations in certain context-free grammars. (We will look more closely at these context-free grammars in the next section.) Therefore

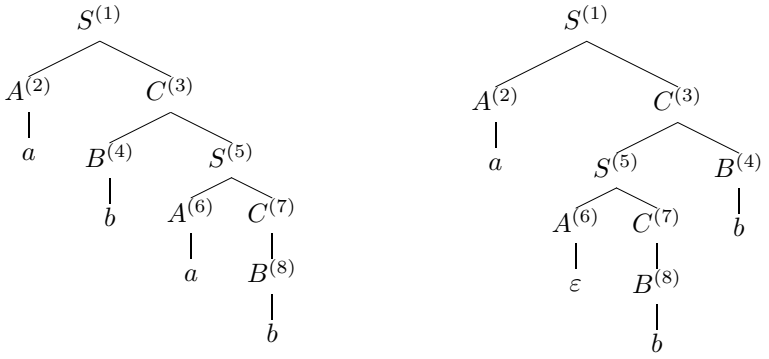


**Figure 1**
Parse tree components associated with derivation (6). Index annotation is added here to represent paired nonterminals, and is not part of the parse trees.

$\sigma$ can be associated in a natural way with a pair of context-free parse trees, encoding the derivations $h_1(\sigma)$ and $h_2(\sigma)$, respectively. These will be called the **parse tree components** of $\sigma$. Each parse tree component describes the way production components of $G$ have been used in deriving the relevant string, in each dimension.

**Example 2**

Consider again the SCFG $G$ from Example 1. The string pair $[a_1b_1a_1b_1,\ a_2b_2b_2]$ can be derived in $G$ through the following steps:

$$
\begin{aligned}
[S^{(1)},\ \ S^{(1)}] &\Rightarrow^{s_1}_G [A^{(2)}_1 C^{(3)}_1,\ \ A^{(2)}_2 C^{(3)}_2] \\
&\Rightarrow^{s_3}_G [A^{(2)}_1 B^{(4)}_1 S^{(5)},\ \ A^{(2)}_2 S^{(5)} B^{(4)}_2] \\
&\Rightarrow^{s_1}_G [A^{(2)}_1 B^{(4)}_1 A^{(6)}_1 C^{(7)}_1,\ \ A^{(2)}_2 A^{(6)}_2 C^{(7)}_2 B^{(4)}_2] \\
&\Rightarrow^{s_4}_G [A^{(2)}_1 B^{(4)}_1 A^{(6)}_1 B^{(8)}_1,\ \ A^{(2)}_2 A^{(6)}_2 B^{(8)}_2 B^{(4)}_2] \\
&\Rightarrow^{s_5}_G [a_1 B^{(4)}_1 A^{(6)}_1 B^{(8)}_1,\ \ a_2 A^{(6)}_2 B^{(8)}_2 B^{(4)}_2] \\
&\Rightarrow^{s_7}_G [a_1 b_1 A^{(6)}_1 B^{(8)}_1,\ \ a_2 A^{(6)}_2 B^{(8)}_2 b_2] \\
&\Rightarrow^{s_6}_G [a_1 b_1 a_1 B^{(8)}_1,\ \ a_2 B^{(8)}_2 b_2]
\end{aligned}
$$

$$\Rightarrow_G^{s_7} [a_1 b_1 a_1 b_1,\ \ a_2 b_2 b_2]. \tag{6}$$

In Figure 1 the parse tree components associated with the derivation in (6) are depicted. In $G$ we can also derive the string pair $[a_1 b_1 a_1 b_1,\ a_2 a_2 b_2 b_2]$ with a derivation very similar to (6), but avoiding the deletion of $a_2$ when rewriting the indexed nonterminal $A_2^{(6)}$. Furthermore, we can derive the pair $[a_1 b_1 a_1 b_1,\ a_2 b_2 a_2 b_2]$ using the latter derivation but avoiding the inversion of the indexed nonterminals $B_2^{(4)}$ and $S^{(5)}$ when rewriting $C_2^{(3)}$. Note that all these string pairs, and some more that can still be derived in $G$, share the first string component.

The **translation** generated by a SCFG $G$ is a binary relation over $V_T^*$ defined as

$$T(G) = \{[w_1,\ \ w_2] \mid [S_1^{(1)},\ \ S_2^{(1)}] \Rightarrow_G^* [w_1,\ \ w_2],\ w_1, w_2 \in V_T^*\}. \tag{7}$$

In most applications, SCFGs are used to translate from some source language to some target language. For this reason we introduce special notation to express the set of strings that are translations of a given input string $w_1$

$$T(G, w_1) = \{w_2 \mid [w_1,\ w_2] \in T(G)\}. \tag{8}$$

We say that a SCFG $G$ is **reduced** if every production of $G$ is used in some derivation $[S_1^{(1)},\ S_2^{(1)}] \Rightarrow_G^* [w_1,\ w_2]$, $[w_1,\ w_2] \in T(G)$. Any SCFG can always be transformed into a reduced SCFG generating the same translation. This process takes a linear amount of time in the size of the SCFG, if techniques similar to those used for the reduction of CFGs are exploited; see for instance (Harrison 1978).

To simplify some technical presentations in this article we sometimes use a specific normal form for SCFGs, defined in what follows. A translation generated by a SCFG can always be generated by some SCFG with all of its synchronous productions in the

form $[A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2]$, with either $\alpha_1, \alpha_2 \in (\mathcal{I}(V_N))^+$ or else $\alpha_1, \alpha_2 \in V_T^*$. To see this, assume a synchronous production $s$ of the form $[A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2]$, where

$$\alpha_1 = u_{10} A_{11}^{(t_1)} u_{11} \cdots u_{1r-1} A_{1r}^{(t_r)} u_{1r},$$

$$\alpha_2 = u_{20} A_{21}^{(t_{\pi_s(1)})} u_{21} \cdots u_{2r-1} A_{2r}^{(t_{\pi_s(r)})} u_{2r}.$$

We introduce fresh nonterminal symbols $B_{1i}$, $B_{2i}$ and fresh indices $t'_i$, $0 \leq i \leq r$. We then replace $s$ with a new synchronous production $[A_1 \rightarrow \alpha'_1,\ A_2 \rightarrow \alpha'_2]$, where

$$\alpha'_1 = B_{10}^{(t'_0)} A_{11}^{(t_1)} B_{11}^{(t'_1)} \cdots B_{1r-1}^{(t'_{r-1})} A_{1r}^{(t_r)} B_{1r}^{(t'_r)},$$

$$\alpha'_2 = B_{20}^{(t'_0)} A_{21}^{(t_{\pi_s(1)})} B_{21}^{(t'_1)} \cdots B_{2r-1}^{(t'_{r-1})} A_{2r}^{(t_{\pi_s(r)})} B_{2r}^{(t'_r)}.$$

Furthermore, for $0 \leq i \leq r$, we add to the grammar the new synchronous productions

$$[B_{1i} \rightarrow u_{1i},\ \ B_{2i} \rightarrow u_{2i}].$$

Note that the size of the SCFG obtained with the above construction is bounded by a linear function in the size of the source SCFG. Furthermore, for each derivation in the new SCFG, we can easily obtain the corresponding derivation in the source SCFG by means of only linear processing.

We close this section with the definition of link, a notion that plays an important role in the following sections. Let $s : [A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2]$ be a synchronous production of a SCFG $G$, and let

$$\alpha_1 = u_{10} A_{11}^{(t_1)} u_{11} \cdots u_{1r-1} A_{1r}^{(t_r)} u_{1r},$$

$$\alpha_2 = u_{20} A_{21}^{(t_{\pi_s(1)})} u_{21} \cdots u_{2r-1} A_{2r}^{(t_{\pi_s(r)})} u_{2r},$$

for some permutation $\pi_s$. If for some $i$ and $j$ with $1 \leq i, j \leq r$, nonterminals $A_{1i}$ and $A_{2j}$ above share a common index, we say that the pair $[A_{1i},\ A_{2j}]$ is a **link** of $G$. We also say that the nonterminals $A_{1i}$ and $A_{2j}$ are linked in some synchronous production. The set of all links of $G$ is denoted by $\mathcal{L}(G)$. In our notation, pair $[A_{1i},\ A_{2j}]$ is a link if and only if $t_i = t_{\pi_s(j)}$, that is, if and only if $i = \pi_s(j)$ or, equivalently, $j = \pi_s^{-1}(i)$. Thus, a link can always be expressed in the form $[A_{1i},\ A_{2\pi_s^{-1}(i)}] = [A_{1\pi_s(j)},\ A_{2j}]$. Furthermore, if a SCFG is reduced we have

$$\mathcal{L}(G) = \{[A_1,\ A_2] \mid [A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2] \in P\}. \tag{9}$$

In words, when $G$ is reduced, every link $[A_1,\ A_2]$ can be formed by taking the nonterminals in the left-hand side of the production components of some synchronous production.

The above definition of link implements for SCFGs a more general notion that, under the same name, has become central in the literature on synchronous parsing, starting from early work in (Shieber 1994). As discussed in (Melamed, Satta, and Wellington 2004a), a link can be thought of as a generalization of the notion of nonterminal in a traditional (non-synchronous) grammar. When one looks at SDTS, the notion of link is not apparent, since that formalism forces the nonterminal symbols in each link to be always the same.

### 3. Auto-projection of a SCFG

As seen in the last section, a SCFG generates a set that is composed by string pairs. In this section we deal with the problem of extracting from a SCFG a standard grammar, that is, a grammar that generates a string language. This construction is called auto-projection here, and is used later in the development of our translation algorithms.

We start by observing that a SCFG implements a sort of parallel rewriting system, where two CFGs simultaneously rewrite pairs of strings constrained by some relation that is implemented through the indices. From a slightly different perspective, we can also view a SCFG as a so-called controlled rewriting system, where the rewriting of one CFG is "controlled", that is, restricted, by the other. (See (Dassow and Păun 1989) for a general presentation of the paradigm of controlled rewriting.) However, in the case of a SCFG, rather than a one-way control of one grammar on the other, we could say that there is a mutual interaction between the rewriting processes of the two CFGs. The above observations can be made more precise with the help of some additional notation.

For $i \in \{1, 2\}$, the $i$-th **projection grammar** of $G$, written $\mathsf{proj}(G, i)$, is the CFG $(V_N, V_T, P_i, S)$, where

$$P_i = \{A_i \rightarrow \mathsf{deindex}(\alpha_i) \mid [A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2] \in P\}.$$

In words, $\mathsf{proj}(G, i)$ is the CFG obtained from $G$ by collecting all of the $i$-th production components of the synchronous productions in $P$, and by stripping off the indices from the right-hand sides. Furthermore, for $i \in \{1, 2\}$ we define the $i$-th **projection language** of $T(G)$ as

$$\mathsf{proj}(T(G), i) = \{w_i \mid [w_1,\ w_2] \in T(G)\}.$$

In words, $\mathsf{proj}(T(G), i)$ is the language obtained from $T(G)$ by collecting all of the $i$-th components of the string pairs in $T(G)$.

Recall from section 2 that each synchronous derivation is associated with two context-free derivations through the homomorphisms $h_1$ and $h_2$. We now see that, for each synchronous derivation $\sigma$ in $G$ and for $i \in \{1, 2\}$, $h_i(\sigma)$ is a context-free derivation in $\mathsf{proj}(G, i)$. This means that the relation $\mathsf{proj}(T(G), i) \subseteq L(\mathsf{proj}(G, i))$ always holds. In

the general case, we may have that $\mathsf{proj}(T(G), i) \neq L(\mathsf{proj}(G, i))$. This is because of the already mentioned fact that the projection grammars $\mathsf{proj}(G, i)$ interact with each other in the rewriting process of $G$.

**Example 3**

Consider the SCFG $G = (\{S, A\}, \{a, b\}, P, S, S)$, where $P$ consists of the following three synchronous productions:

$$\begin{array}{ll} [S \rightarrow \varepsilon, & S \rightarrow \varepsilon], \\ [S \rightarrow aA^{(1)}, & S \rightarrow bS^{(1)}], \\ [A \rightarrow S^{(1)}b, & S \rightarrow S^{(1)}a]. \end{array}$$

The generated translation is

$$T(G) = \{[a^n b^n,\ b^n a^n] \mid n \geq 0\},$$

and therefore $\mathsf{proj}(L(G), 2) = \{b^n a^n \mid n \geq 0\}$. On the other hand, $\mathsf{proj}(G, 2)$ is the CFG with productions

$$\begin{array}{l} S \rightarrow \varepsilon, \\ S \rightarrow bS, \\ S \rightarrow Sa, \end{array}$$

and therefore $L(\mathsf{proj}(G, 2)) = \{b^n a^m \mid n, m \geq 0\} \neq \mathsf{proj}(L(G), 2)$.

In view of machine translation applications, one might wonder whether the above property isn't something that should be avoided. In fact, from a translation model between two languages one would expect that projection on a single dimension delivers a grammar that is able to generate all and only the sentences of one of the two languages, rather than a grammar that overgenerates. In other words, it would be odd to have a model where the grammaticality of a sentence in one language is controlled by derivations of the other language, as we observe in Example 3. This observation is certainly reasonable, and such effects should never been observed in a translation

model for natural language. However, in later sections we will deal with SCFGs that are defined by subsets of the productions of a translation model. For instance, we will define new SCFGs by forcing a translation model to derive only one string on one of the two dimensions. For such kind of "artificial" SCFGs, we will observe the effects of controlled rewriting shown in Example 3.

The main problem we deal with in this section is the specification of a construction that, given a SCFG $G$ and an index $i \in \{1, 2\}$, provides a new grammar $G_i$ satisfying $L(G_i) = \mathsf{proj}(T(G), i)$. As we will see, $\mathsf{proj}(T(G), i)$ is always a context-free language, so $G_i$ does not need to be more expressive than a CFG. We will return to this last remark later.

In the following discussion we fix some SCFG $G = (V_N, V_T, P, S_1, S_2)$. We use two homomorphisms (see section 2) to map synchronous productions of $P$ to specific context-free productions defined over the set $\mathcal{L}(G)$ (the set of links of $G$), taken as the set of nonterminal symbols, and the set of terminal symbols $V_T$. As already done in section 2, we view productions as atomic symbols belonging to some finite alphabet. Let $s : [A_{10} \rightarrow \alpha_1,\ A_{20} \rightarrow \alpha_2]$ be a synchronous production in $P$, and let

$$\alpha_1 = u_{10} A_{11}^{(t_1)} u_{11} \cdots u_{1r-1} A_{1r}^{(t_r)} u_{1r},$$

$$\alpha_2 = u_{20} A_{21}^{(t_{\pi_s(1)})} u_{21} \cdots u_{2r-1} A_{2r}^{(t_{\pi_s(r)})} u_{2r},$$

for some permutation $\pi_s$. Recall from section 2 that, with the above notation, a link can always be expressed in the form $[A_{1i},\ A_{2\pi_s^{-1}(i)}]$ or $[A_{1\pi_s(j)},\ A_{2j}]$. We let

$$h_{a,1}(s) =$$

$$[A_{10},\ A_{20}] \rightarrow u_{10}[A_{11},\ A_{2\pi_s^{-1}(1)}]u_{11} \cdots u_{1r-1}[A_{1r},\ A_{2\pi_s^{-1}(r)}]u_{1r} \tag{10}$$

and

$$h_{a,2}(s) =$$

$$[A_{10},\ A_{20}] \rightarrow u_{20}[A_{1\pi_s(1)},\ A_{21}]u_{21}\cdots u_{2r-1}[A_{1\pi_s(r)},\ A_{2r}]u_{2r}. \tag{11}$$

We introduce next the notion of auto-projection grammar of a SCFG.

**Definition 4**

Let $G = (V_N, V_T, P, S_1, S_2)$ be a SCFG. For $i \in \{1, 2\}$ the $i$-th **auto-projection** of $G$ is the CFG

$$\textsf{auto-proj}(G, i) = (\mathcal{L}(G), V_T, P_i, [S_1,\ S_2])$$

where $P_i = \{h_{a,i}(s) \mid s \in P\}$.

Some simple examples of application of the above construction are reported in section 5. In what follows, we state and prove some properties of the auto-projection construction that will be used later in this article. The next lemma shows the rather intuitive property that a synchronous derivation can be "split" into two CFG derivations in the associated projection grammars. The proof is technically simple: we report it here since it helps understanding a more complex proof that needs to be developed later for 2.

**Lemma 1**

Let $G$ be a SCFG and let $G_i = \textsf{auto-proj}(G, i)$, $i \in \{1, 2\}$. For each synchronous derivation $\sigma$ in $G$ of the form

$$[S_1^{(1)},\ \ S_2^{(1)}] \Rightarrow_G^{\sigma}$$

$$[u_{10}A_{11}^{(t_1)}u_{11}\cdots u_{1n-1}A_{1n}^{(t_n)}u_{1n},\ \ u_{20}A_{21}^{(t_{\pi(1)})}u_{21}\cdots u_{2n-1}A_{2n}^{(t_{\pi(n)})}u_{2n}]$$

there exist context-free derivations in $G_1$ and $G_2$ of the form

$$[S_1,\ \ S_2] \Rightarrow_{G_1}^{h_{a,1}(\sigma)} u_{10}[A_{11},\ \ A_{2\pi^{-1}(1)}]u_{11}\cdots u_{1n-1}[A_{1n},\ \ A_{2\pi^{-1}(n)}]u_{1n},$$

$$[S_1,\ \ S_2] \Rightarrow_{G_2}^{h_{a,2}(\sigma)} u_{20}[A_{1\pi(1)},\ \ A_{21}]u_{21}\cdots u_{2n-1}[A_{1\pi(n)},\ \ A_{2n}]u_{2n}.$$

**Proof**

We prove the part of the lemma involving $G_1$ and $h_{a,1}$; the other part can be shown with a symmetrical argument.

Let $G = (V_N, V_T, P, S_1, S_2)$ and let $G_1 = (\mathcal{L}(G), V_T, P_1, [S_1,\ S_2])$. We proceed by induction on the length of $\sigma$. If $|\sigma| = 0$, we have $\sigma = \varepsilon$ and $[S_1^{(1)},\ S_2^{(1)}] \Rightarrow_G^{\varepsilon} [S_1^{(1)},\ S_2^{(1)}]$. Since $h_{a,1}(\varepsilon) = \varepsilon$, from the statement of the lemma we obtain the context-free derivation $[S,_1\ \ S_2] \Rightarrow_{G_1}^{\varepsilon} [S_1,\ S_2]$, which trivially holds.

Assume now that $\sigma = \sigma' s$, with $|\sigma'| \geq 0$ and $s$ a synchronous production in $P$. Assume also that $s$ has rank $r \geq 1$. (Recall that the rank of $s$ is the number of nonterminals in the right-hand side of one of the production components of $s$.) There must be integers $i_1$ and $j_1$, with $1 \leq i_1 \leq j_1 \leq n$ and $j_1 - i_1 + 1 = r$, such that the nonterminals $A_{1i_1}, \ldots, A_{1j_1}$ in the left string derived by $\sigma$ have been introduced by the indicated application of $s$. Correspondingly, there must be integers $i_2$ and $j_2$, with $1 \leq i_2 \leq j_2 \leq n$ and $j_2 - i_2 + 1 = r$, such that the nonterminals $A_{2i_2}, \ldots, A_{2j_2}$ in the right string derived by $\sigma$ have been introduced by the same application of $s$. Note also that, in the derived string, the nonterminals $A_{2i_2}, \ldots, A_{2j_2}$ must have the same set of indices as the nonterminals $A_{1i_1}, \ldots, A_{1j_1}$. From the above observations it follows that the synchronous production $s$ has the form

$$[B_1 \rightarrow v_{11} A_{1i_1}^{(t_{s,1})} u_{1i_1} \cdots u_{1j_1-1} A_{1j_1}^{(t_{s,r})} v_{12},$$

$$B_2 \rightarrow v_{21} A_{2i_2}^{(t_{s,\pi_s(1)})} u_{2i_2} \cdots u_{2j_2-1} A_{2j_2}^{(t_{s,\pi_s(r)})} v_{22}],$$

for some permutation $\pi_s$ defined on the set $\{1, \ldots, r\}$. There must then be some index $t$ such that the synchronous derivation $\sigma$ can be written in the form

$$
\begin{aligned}
&[S_1^{(1)},\ S_2^{(1)}] \\
&\Rightarrow_G^{\sigma'} [u_{10} A_{11}^{(t_1)} u_{11} \cdots u'_{1i_1-1} B_1^{(t)} u'_{1j_1} \cdots u_{1n-1} A_{1n}^{(t_n)} u_{1n}, \\
&\qquad u_{20} A_{21}^{(t_{\pi(1)})} u_{21} \cdots u'_{2i_2-1} B_2^{(t)} u'_{2j_2} \cdots u_{2n-1} A_{2n}^{(t_{\pi(n)})} u_{2n}] \\
&\Rightarrow_G^{s} [u_{10} A_{11}^{(t_1)} u_{11} \cdots u'_{1i_1-1} \cdot \\
&\qquad v_{11} A_{1i_1}^{(t_{i_1})} u_{1i_1} \cdots u_{1j_1-1} A_{1j_1}^{(t_{j_1})} v_{12} \cdot \\
&\qquad u'_{1j_1} \cdots u_{1n-1} A_{1n}^{(t_n)} u_{1n}, \\
&\qquad u_{20} A_{21}^{(t_{\pi(1)})} u_{21} \cdots u'_{2i_2-1} \cdot \\
&\qquad v_{21} A_{2i_2}^{(t_{\pi(i_2)})} u_{2i_2} \cdots u_{2j_2-1} A_{2j_2}^{(t_{\pi(j_2)})} v_{22} \cdot \\
&\qquad u'_{2j_2} \cdots u_{2n-1} A_{2n}^{(t_{\pi(n)})} u_{2n}].
\end{aligned}
$$

According to (10) we have

$$h_{a,1}(s) = [B_1, B_2] \rightarrow v_{11}[A_{1i_1},\ A_{2\pi_s^{-1}(1)}] u_{1i_1} \cdots u_{1j_1-1} [A_{1j_1},\ A_{2\pi_s^{-1}(r)}] v_{12}.$$

We apply the inductive hypothesis to derivation $\sigma'$ above and conclude with the desired context-free derivation

$$
\begin{aligned}
&[S_1,\ S_2] \\
&\Rightarrow_{G_1}^{h_{a,1}(\sigma')} u_{10}[A_{11},\ A_{2\pi^{-1}(1)}] u_{11} \cdots u'_{1i_1-1} [B_1,\ B_2] u'_{1j_1} \cdots \\
&\qquad \cdots u_{1n-1} [A_{1n},\ A_{2\pi^{-1}(n)}] u_{1n} \\
&\Rightarrow_{G_1}^{h_{a,1}(s)} u_{10}[A_{11},\ A_{2\pi^{-1}(1)}] u_{11} \cdots u'_{1i_1-1} \cdot
\end{aligned}
$$

$$v_{11}[A_{1i_1},\ A_{2\pi_s^{-1}(1)}]u_{1i_1}\cdots u_{1j_1-1}[A_{1j_1},\ A_{2\pi_s^{-1}(r)}]v_{12}\cdot$$

$$u'_{1j_1}\cdots u_{1n-1}[A_{1n},\ A_{2\pi^{-1}(n)}]u_{1n}.$$

The left out case in which $s$ has rank $0$ can be treated with a very similar argument. ■

We state the following theorem for the left dimension of a SCFG; a symmetrical statement holds for the right dimension.

**Theorem 1**

Let $[S_1^{(1)},\ S_2^{(1)}] \Rightarrow_G^{\sigma} [w_1,\ w_2]$ be a synchronous derivation for SCFG $G$, and let $t_1$ be its left parse tree component. Let also $t_{a,1}$ be the parse tree representing the CFG derivation $h_{a,1}(\sigma)$ in auto-proj$(G,1)$. Then $t_1 \cong t_{a,1}$.

**Proof**

Let $G_1 = \text{proj}(G,1)$. Then we have $S_1 \Rightarrow_{G_1}^{h_1(\sigma)} w_1$. Let also $G_{a,1} = \text{auto-proj}(G,1)$. From Lemma 1 we have $S_1 \Rightarrow_{G_{a,1}}^{h_{a,1}(\sigma)} w_1$. The statement then follows from the definition of homomorphisms $h_1$ and $h_{a,1}$. ■

For an homomorphism $h$ and a string $y$ over the target alphabet of $h$, we define $h^{-1}(y) = \{x \mid h(x) = y\}$. From the construction of $P_1$ in Definition 4, we have that $h_{a,1}^{-1}(p)$ is non-empty for every production $p \in P_1$. It follows that, for every context-free derivation $\rho \in P_1^*$, the set of synchronous derivation $h_{a,1}^{-1}(\rho)$ is always non-empty. The same observation also holds for $h_{a,2}$. The next result shows the inverse of Lemma 1, stating that auto-projection grammars do not overgenerate with respect to the source SCFG.

**Lemma 2**

Let $G$ be a SCFG and let $G_i = \textsf{auto-proj}(G, i)$, $i \in \{1, 2\}$. For each context-free derivation $\rho$ in $G_1$ of the form

$$[S_1,\ \ S_2] \Rightarrow^{\rho}_{G_1} u_{10}[A_{11},\ \ A'_{21}]u_{11} \cdots u_{1n-1}[A_{1n},\ \ A'_{2n}]u_{1n},$$

every $\sigma \in h^{-1}_{a,1}(\rho)$ is a synchronous derivation in $G$ of the form

$$[S^{(1)}_1,\ \ S^{(1)}_2] \Rightarrow^{\sigma}_{G}$$
$$[u_{10}A^{(t_1)}_{11}u_{11} \cdots u_{1n-1}A^{(t_n)}_{1n}u_{1n},\ \ u_{20}A^{(t_{\pi_\sigma(1)})}_{21}u_{21} \cdots u_{2n-1}A^{(t_{\pi_\sigma(n)})}_{2n}u_{2n}],$$

where $u_{2i}$, $0 \leq i \leq n$, are strings in $V^*_T$ and $A_{2\pi^{-1}_\sigma(i)} = A'_{2i}$, $1 \leq i \leq n$ (that is, nonterminals in each pair $[A_{1i},\ A'_{2i}]$ are coindexed in $\sigma$).

For each context-free derivation $\rho$ in $G_2$ of the form

$$[S_1,\ \ S_2] \Rightarrow^{\rho}_{G_2} u_{20}[A'_{11},\ \ A_{21}]u_{21} \cdots u_{2n-1}[A'_{1n},\ \ A_{2n}]u_{2n},$$

every $\sigma \in h^{-1}_{a,2}(\rho)$ is a synchronous derivation in $G$ of the form

$$[S^{(1)}_1,\ \ S^{(1)}_2] \Rightarrow^{\sigma}_{G}$$
$$[u_{10}A^{(t_1)}_{11}u_{11} \cdots u_{1n-1}A^{(t_n)}_{1n}u_{1n},\ \ u_{20}A^{(t_{\pi_\sigma(1)})}_{21}u_{21} \cdots u_{2n-1}A^{(t_{\pi_\sigma(n)})}_{2n}u_{2n}],$$

where $u_{1i}$, $0 \leq i \leq n$, are strings in $V^*_T$ and $A_{1\pi_\sigma(i)} = A'_{1i}$, $1 \leq i \leq n$ (that is, nonterminals in each pair $[A'_{1i},\ A_{2i}]$ are coindexed in $\sigma$).

**Proof**

We prove the part of the lemma involving $G_1$ and $h_{a,1}$; the other part can be shown by a symmetrical argument.

Let $G = (V_N, V_T, P, S_1, S_2)$ and let $G_1 = (\mathcal{L}(G), V_T, P_1, [S_1,\ S_2])$. We proceed by induction on the length of $\rho \in P_1^*$. If $|\rho| = 0$, we have $\rho = \varepsilon$ and $[S_1,\ S_2] \Rightarrow^{\varepsilon}_{G_1} [S_1,\ S_2]$. We then obtain $h^{-1}_{a,1}(\varepsilon) = \{\varepsilon\}$ and the synchronous derivation $[S_1^{(1)},\ S_2^{(1)}] \Rightarrow^{\varepsilon}_{G} [S_1^{(1)},\ S_2^{(1)}]$ trivially holds. It is easy to see that such a derivation satisfies the statement of the lemma.

Assume now that $\rho = \rho' p$, with $|\rho'| \geq 0$ and with $p$ a context-free production in $P_1$. We have $h^{-1}_{a,1}(\rho) = h^{-1}_{a,1}(\rho' p) = h^{-1}_{a,1}(\rho') \cdot h^{-1}_{a,1}(p)$. Hence each $\sigma \in h^{-1}_{a,1}(\rho)$ can be written in the form $\sigma' \cdot s$ with $\sigma' \in h^{-1}_{a,1}(\rho')$ and $s \in h^{-1}_{a,1}(p)$.

If production $p$ contains $r \geq 1$ nonterminals in its right-hand side, then there must be some integer $i_1$ with $1 \leq i_1 \leq n - r$ such that the nonterminals $[A_{1i_1+1},\ A'_{2i_1+1}], \ldots, [A_{1i_1+r},\ A'_{2i_1+r}]$ in the string derived by $\rho$ have been introduced by the indicated application of $p$. Then $p$ must have the form

$$[B_1, B_2] \rightarrow v_{1i_1}[A_{1i_1+1},\ A'_{2i_1+1}]u_{1i_1+1} \cdots u_{1i_1+r-1}[A_{1i_1+r},\ A'_{2i_1+r}]v_{1i_1+r},$$

for some strings $v_{1i_1}, v_{1i_1+r}$ with $u'_{1i_1} \cdot v_{1i_1} = u_{1i_1}$ and $v_{1i_1+r} \cdot u'_{1i_1+r} = u_{1i_1+r}$. The derivation $\rho$ can then be rewritten as

$$\begin{aligned}
&[S_1,\ \ S_1] \\
&\Rightarrow^{\rho'}_{G_1} u_{10}[A_{11},\ A'_{21}]u_{11} \cdots u'_{1i_1}[B_1, B_2]u'_{1i_1+r} \cdots u_{1n-1}[A_{1n},\ A'_{2n}]u_{1n} \\
&\Rightarrow^{p}_{G_1} u_{10}[A_{11},\ A'_{21}]u_{11} \cdots u'_{1i_1} \cdot \\
&\qquad v_{1i_1}[A_{1i_1+1},\ A'_{2i_1+1}]u_{1i_1+1} \cdots u_{1i_1+r-1}[A_{1i_1+r},\ A'_{2i_1+r}]v_{1i_1+r} \cdot \\
&\qquad u'_{1i_1+r} \cdots u_{1n-1}[A_{1n},\ A'_{2n}]u_{1n}.
\end{aligned}$$

Applying the inductive hypothesis to $\rho'$ we obtain that every $\sigma' \in h_{a,1}^{-1}(\rho')$ is a synchronous derivation of the form

$$
\begin{aligned}
&[S_1^{(1)},\ S_2^{(1)}] \\
&\Rightarrow_G^{\sigma'} [u_{10} A_{11}^{(t_1)} u_{11} \cdots u'_{1i_1} B_1^{(t)} u'_{1i_1+r} \cdots u_{1n-1} A_{1n}^{(t_n)} u_{1n}, \\
&\qquad u_{20} A_{21}^{(t_{\pi_{\sigma'}(1)})} u_{21} \cdots u'_{2i_2} B_2^{(t)} u'_{2i_2+r} \cdots u_{2n-1} A_{2n}^{(t_{\pi_{\sigma'}(n)})} u_{2n}],
\end{aligned} \tag{12}
$$

for some integer $i_2$ with $1 \leq i_2 \leq n - r$, and for some choice of strings $u_{2,0}, \ldots, u_{2,i_2-1}$, strings $u'_{2i_2}, u'_{2i_2+r}$, and strings $u_{2,i_2+r+1}, \ldots, u_{2n}$ in $V_T^*$. From the inductive hypothesis we also obtain that, for each $i$ with $1 \leq i \leq n$ and $i \notin \{i_1 + 1, \ldots, i_1 + r\}$, the nonterminals in $[A_{1i},\ A'_{2i}]$ are coindexed in $\sigma'$, that is, we have

$$
A_{2\pi_{\sigma'}^{-1}(i)} = A'_{2i}. \tag{13}
$$

Let us consider the definition of homomorphism $h_{a,1}$ in (10). Observe that each $s \in h_{a,1}^{-1}(p)$ must have the form

$$
\begin{aligned}
s : [&B_1 \rightarrow v_{1i_1} A_{1i_1+1}^{(t_1)} u_{1i_1+1} \cdots u_{1i_1+r-1} A_{1i_1+r}^{(t_r)} v_{1i_1+r}, \\
&B_2 \rightarrow v_{2i_2} A_{2i_2+1}^{(t_{\pi_s(1)})} u_{2i_2+1} \cdots u_{2i_2+r-1} C_{2i_2+r}^{(t_{\pi_s(r)})} v_{2i_2+r}],
\end{aligned} \tag{14}
$$

for some choice of strings $v_{2i_2}, v_{2i_2+r}$ and strings $u_{2i_2+1}, \ldots, u_{2i_2+r-1}$ in $V_T^*$. Furthermore, we have that for each $k$ with $1 \leq k \leq r$ the nonterminal symbols $A_{1i_1+k}$ and $A'_{2i_1+k}$ are coindexed. This means that, for each $k$ with $1 \leq k \leq r$, we have

$$
A_{2i_2+\pi_s^{-1}(k)} = A'_{2i_1+k}. \tag{15}
$$

We can now combine (12) and (14) and conclude that every $\sigma \in h_{a,1}^{-1}(\rho)$ is a synchronous derivation of the form

$$
\begin{aligned}
&[S_1^{(1)},\ \ S_2^{(1)}] \\
&\Rightarrow_G^{\sigma} [u_{10} A_{11}^{(t_1)} u_{11} \cdots u'_{1i_1} \cdot \\
&\quad v_{1i_1} A_{1i_1+1}^{(t'_1)} u_{1i_1+1} \cdots u_{1i_1+r-1} A_{1i_1+r}^{(t'_r)} v_{1i_1+r} \cdot \\
&\quad u'_{1i_1+r} \cdots u_{1n-1} A_{1n}^{(t_n)} u_{1n}, \\
&\quad u_{20} A_{21}^{(t_{\pi_{\sigma'}(1)})} u_{21} \cdots u'_{2i_2} \cdot \\
&\quad v_{2i_2} A_{2i_2+1}^{(t'_{\pi_s(1)})} u_{2i_2+1} \cdots u_{2i_2+r-1} A_{2i_2+r}^{(t'_{\pi_s(r)})} v_{2i_2+r} \cdot \\
&\quad u'_{2i_2+r} \cdots u_{2n-1} A_{2n}^{(t_{\pi_{\sigma'}(n)})} u_{2n}].
\end{aligned}
$$

Furthermore, we can combine (13) and (15) above to show that, for each $i$ with $1 \leq i \leq n$, we have $A_{2\pi^{-1}(i)} = A'_{2i}$ for some permutation $\pi$ defined over $\{1, \ldots, n\}$.

The left out case in which $p$ has rank $0$ can be treated with a special case of the argument above. ■

Note that Lemma 2 establish a one-to-many correspondence from derivations in an auto-projection grammar and derivations in a source SCFG. This is so because in a SCFG there may be several parse tree components on one dimension that are associated with the same parse tree component on the other dimension. We can now establish the main result of this section.

**Theorem 2**

Let $G$ be a SCFG. For $i \in \{1, 2\}$ we have

$$L(\textsf{auto-proj}(G, i)) = \textsf{proj}(T(G), i).$$

**Proof**

Directly from Lemmas 1 and 2, by considering context-free derivations ending with strings in $V_T^*$, and by considering synchronous derivations ending with pairs of strings in $V_T^*$. ■

We discuss here an important consequence of Theorem 2. For $i \in \{1, 2\}$, the theorem shows that the $i$-th projection language of $G$ is always a context-free language. We have already remarked that a SCFG can be viewed as a controlled rewriting system, with some mutual interaction between the rewriting processes of two CFGs. Theorem 2 entails that, when we consider the languages obtained by projection of the translation generated by a SCFG, we do not observe any increase in the generative capacity with respect to the generative capacity of the class of the grammars involved in the synchronous system, taken individually. In the literature, this general property has been called the "weak language preservation property", and is one of the defining requirements of synchronous rewriting systems (Rambow and Satta 1996).

We conclude this section with some remarks on the computational complexity of the auto-projection construction. Observe that, given an input SCFG $G$ and an integer $i \in \{1, 2\}$, the output CFG $\mathsf{auto\text{-}proj}(G, i)$ contains the same number of productions as $G$. Furthermore, when translating a synchronous production $s$ of $G$ into the corresponding production $p$ of $\mathsf{auto\text{-}proj}(G, i)$, we simply "unfold" all of the indices in the right-hand side of the $i$-th production component of $s$, using the associated links. Thus, the length of $p$ is bounded by the length of $s$. We then conclude that $|\mathsf{auto\text{-}proj}(G, i)| = \mathcal{O}(|G|)$. Finally, it is not difficult to see that the whole construction of $\mathsf{auto\text{-}proj}(G, i)$ can be carried out in an amount of time proportional to the output, which is $\mathcal{O}(G)$. To do this, each synchronous production $s$ can be preprocessed in such a way that an array

is constructed, where each nonterminal occurrence in $s$ can be accessed starting from its own index.

**4. Intersection for SCFGs**

We introduce here some "intersection constructions" for SCFGs that will be exploited in the next sections, in the development of our translation algorithms. These constructions generalize a well-known technique, originally proposed by Bar-Hillel, Perles, and Shamir (1964), for the specification of a CFG generating the intersection of the languages generated by a source CFG and a source finite automaton. As already discussed in the introduction, such a technique is at the basis of several standard algorithms for CFG parsing, but has never been exploited before for SCFGs. Throughout this section, we let $G = (V_N, V_T, P, S_1, S_2)$ be a SCFG in the normal form discussed in section 2, requiring that for each $[A_1 \rightarrow \alpha_1,\ A_2 \rightarrow \alpha_2]$ in $P$ we have either $\alpha_1, \alpha_2 \in (\mathcal{I}(V_N))^+$ or else $\alpha_1, \alpha_2 \in V_T^*$. We denote by $r_G$ the rank of $G$.

To be used below, we briefly recall here the notion of finite automaton. A **finite automaton** (FA) is a tuple $M = (Q, V_T, \delta, q_{in}, F)$, where $Q$ is a finite set of states, $V_T$ is a finite set of terminal symbols with $V_T \cap Q = \emptyset$, $q_{in} \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta$ is a finite set of transitions, each of the form $(q_1, a, q_2)$, with $a \in V_T$ and $q_1, q_2 \in Q$. As a convention, symbols $q, q_0, q_1, \ldots$ denote elements of $Q$. Let $w = a_1 a_2 \cdots a_n$ with $n \geq 1$ and $a_i \in V_T$, $1 \leq i \leq n$. With some abuse of notation, if we have $(q_{i-1}, a_i, q_i) \in \delta$ for each $i$ with $1 \leq i \leq n$, then we write $(q_0, w, q_n) \in \delta^*$. Furthermore, for each $q \in Q$ we write $(q, \varepsilon, q) \in \delta^*$. This is used to denote a computation of $M$ on $w$, starting in state $q_0$ and ending in state $q_n$.

A FA $M$ is **deterministic** if, for every state $q_1$ and alphabet symbol $a$ we have at most one state $q_2$ such that $(q_1, a, q_2) \in \delta$. When $M$ is deterministic, there is at most one computation of $M$ on $w$ starting in state $q_1$ and ending in state $q_2$, for any choice

of $w$, $q_1$ and $q_2$. To simplify the presentation below, we assume that all of our FAs are deterministic and have a single final state. All of the translation applications we investigate in this article can be modeled using such assumption. Our results can be transfered to the case of general FAs with some additional refinement in the notation adopted here.

We start by considering the intersection on the left dimension between the translation generated by a SCFG and some regular language. More precisely, let $L_1 \times L_2$ denote the Cartesian product of languages $L_1$ and $L_2$, that is,

$$L_1 \times L_2 = \{[w_1,\ w_2] \mid w_1 \in L_1,\ w_2 \in L_2\}.$$

Given our SCFG $G$ and a FA $M$, we provide a new SCFG $G_\cap$ such that

$$T(G_\cap) = T(G) \cap (L(M_1) \times V_T^*). \tag{16}$$

Let $M = (Q, V_T, \delta, q_{in}, \{q_{fin}\})$. We define $G_\cap = (V_\cap, V_T, P_\cap, (q_{in}, S_1, q_{fin}), S_2)$, where

$$V_\cap = V_N \cup \{(q_1, A, q_2) \mid q_1, q_2 \in Q,\ A \in V_N\}.$$

The set of synchronous productions $P_\cap$ is constructed as follows.

- For each synchronous production in $P$ of the form

$$[A_1 \rightarrow A_{11}^{(t_1)} \cdots A_{1r}^{(t_r)},\ \ A_2 \rightarrow A_{21}^{(t_{\pi(1)})} \cdots A_{2r}^{(t_{\pi(r)})}],$$

  with $r \geq 1$ and $A_{1i}, A_{2i} \in V_N$, and for each sequence $q_0, \ldots, q_r$ of states in $Q$, we add to $P_\cap$ the synchronous production

$$[(q_0, A_1, q_r) \rightarrow (q_0, A_{11}, q_1)^{(t_1)} \cdots (q_{r-1}, A_{1r}, q_r)^{(t_r)},$$
$$A_2 \rightarrow A_{21}^{(t_{\pi(1)})} \cdots A_{2r}^{(t_{\pi(r)})}]. \tag{17}$$

- For each synchronous production in $P$ of the form $[A_1 \rightarrow x_1, \ \ A_2 \rightarrow x_2]$, with $x_1, x_2 \in V_T^*$, and for each pair $q_1, q_2 \in Q$ such that $(q_1, x_1, q_2) \in \delta^*$, we add to $P_\cap$ the synchronous production

$$[(q_1, A_1, q_2) \rightarrow x_1, \ \ A_2 \rightarrow x_2]. \tag{18}$$

Grammar $G_\cap$ may contain several useless synchronous productions. These can be removed in linear time, as already remarked in section 2. Some simple examples of application of the above construction are reported in section 5.

The intersection on the right dimension with a regular language can be symmetrically defined, and in the next section we will apply the left and right intersection constructions in cascades. Note also that in the above construction we take advantage of the definition of SCFG, allowing different nonterminals to be indexed. Had we dealt with the already discussed SDTS formalism, we would have to spread states from $M$ on the right dimension, making our construction less intuitive and, as will be apparent from the next section, also less modular. More discussion on this is provided in section 6.

Let us define homomorphism $h_\cap$ such that $h_\cap(s) = s'$ if and only if synchronous production $s \in P_\cap$ has been obtained from synchronous production $s' \in P$. We omit here the proof of the next theorem, since it does not add anything technically new to existing proofs of the corresponding CFG case; see for instance (Nederhof and Satta 2003).

**Theorem 3**

Let $G$, $M$ and $G_\cap$ be specified as above.

1. Homomorphism $h_\cap$ establishes a bijection between the set of all synchronous derivations in $G_\cap$ and the set of synchronous derivations in $G$ of string pairs in $L(M_1) \times V_T^*$.

2. Let $t_1$ and $t_2$ be parse tree components on the same dimension, associated with synchronous derivations $\sigma_\cap$ and $\sigma$, respectively, that are related through $h_\cap$. Then $t_1 \cong t_2$.

Theorem 3 implies (16), and states that the intersection construction "preserves" parse tree components on both dimensions, modulo some node relabeling.

Let us consider now the computational complexity of the construction presented above. All synchronous productions in (18) can be constructed in time (and space) $\mathcal{O}(|G| \cdot |Q|)$. This is so because $M$ is deterministic, and state $q_2$ can be uniquely identified from $q_1$ and $x_1$ in time proportional to $|x_1|$. Construction of the synchronous productions in (17) is more demanding. It is not difficult to see that, in the worst case, processing of such productions can take an amount of time $\Theta(|G| \cdot |Q|^{r_G+1})$, with $r_G$ the rank of the source SCFG $G$. We thus conclude that $\mathcal{O}(|G| \cdot |Q|^{r_G+1})$ is the time and space bound for our intersection construction.

When the SCFG $G$ is considered as part of the input, the above upper bound means that we observe an exponential space and time behavior. One might wonder whether some preliminary processing could be used to cast SCFG $G$ into an equivalent form, where $r$ is bounded by some constant independent of $G$. That would result in a polynomial time algorithm for the intersection construction for SCFGs. Unfortunately, this is not possible in general. Such a negative result follows from a similar property of SDTS proved in (Aho and Ullman 1969). As a consequence, some of the algorithms we are going to develop in the next section, based on the intersection construction, have an exponential worst case behavior. The natural question to ask is then: how much better than this can we do, both under a theoretical and a practical perspective? We will come back to this issue in the next section as well.

As a final remark, if in the above construction we ignore the right productions components of all the synchronous productions, we obtain the already mentioned intersection construction proposed by Bar-Hillel, Perles, and Shamir (1964) for CFGs. As it is well known (Harrison 1978), we can cast a CFG in binary form, that is, a form with productions having no more than two nonterminals in their right-hand side. Then from our upper bound above we obtain that the CFG intersection construction runs in time $\mathcal{O}(|G| \cdot |Q|^3)$. This construction will also be used in the next section.

We now present a second intersection construction for SCFGs. In this case, we restrict the parse tree components generated in one dimension to be members of a parse forest given as input. We need to introduce some additional notation. Let $G_f = (V_{N,f}, V_T, P_f, \eta_S)$ be a CFG whose generated trees represent the parse forest of interest. Each nonterminal $\eta \in V_{N,f}$ denotes a node that is shared by some of the trees in the parse forest. We write $\mathsf{label}(\eta) = A$, $A \in V_N$, if $A$ is the label of node $\eta$.

Let $G$ and $G_f$ be as above. We define the SCFG $G_{\cap,f} = (V_{\cap,f}, V_T, P_{\cap,f}, \eta_S, S_2)$, where $V_{\cap,f} = V_{N,f} \cup V_N$. The set of synchronous productions $P_{\cap,f}$ is constructed as follows.

- For each CFG production in $P_f$ of the form $\eta_0 \rightarrow \eta_1 \cdots \eta_r$ with $r \geq 1$ and for each synchronous production in $P$ of the form

$$[A_1 \rightarrow A_{11}^{(t_1)} \cdots A_{1r}^{(t_r)}, \;\; A_2 \rightarrow A_{21}^{(t_{\pi(1)})} \cdots A_{2r}^{(t_{\pi(r)})}],$$

  with $A_{1i}, A_{2i} \in V_N$, such that $\mathsf{label}(\eta_0) = A_1$ and $\mathsf{label}(\eta_i) = A_{1i}$ for each $i$ with $1 \leq i \leq r$, we add to $P_{\cap,f}$ the production

$$[\eta_0 \rightarrow \eta_1^{(t_1)} \cdots \eta_r^{(t_r)}, \;\; A_2 \rightarrow A_{21}^{(t_{\pi(1)})} \cdots A_{2r}^{(t_{\pi(r)})}]. \tag{19}$$

- For each CFG production in $P_f$ of the form $\eta \rightarrow x_1$, and for each synchronous production in $P$ of the form $[A_1 \rightarrow x_1, \;\; A_2 \rightarrow x_2]$ such that

$\mathsf{label}(\eta) = A_1$, we add to $P_{\cap,f}$ the synchronous production

$$[\eta \rightarrow x_1, \;\; A_2 \rightarrow x_2]. \tag{20}$$

Again, the SCFG $G_{\cap,f}$ will have several useless symbols and needs to be reduced.

It is not difficult to show that a property similar to Theorem 3 holds for the construction above, so that each synchronous derivation $\sigma$ in $G_{\cap,f}$ derives a pair $[w_1, w_2] \in L(G_f) \times V_T^*$, with the restriction that the left parse tree component of $\sigma$ is a tree generated by $G_f$. With such a restriction, and modulo some relabeling of the internal nodes, the construction also preserves the parse tree component of the source SCFG $G$. Again, the proof of these two properties is rather straightforward and is omitted here.

As a final note, on the complexity of the construction of $G_{\cap,f}$, observe that for each synchronous production $s \in P$ we can add to $P_{\cap,f}$ a number of new productions bounded by $\mathcal{O}(|P_f|)$. Since each such a new production has size proportional to the size of $s$, the overall time bound for the complexity of the construction of $G_{\cap,f}$ is $\mathcal{O}(|G| \cdot |P_f|)$. Note that in this case we have a polynomial upper bound in the size of the input.

**5. Translation algorithm**

In this section we put together the machinery introduced in sections 3 and 4 and develop, in a modular way, algorithms for the translation problem and the related problems of synchronous recognition, synchronous parsing and tree translation, defined below. Throughout this section we consider a SCFG $G = (V_N, V_T, P, S_1, S_2)$ in the normal form of section 2, and with rank $r_G$.

The **translation problem** for SCFGs is defined as follows: given an input SCFG $G$ and an input string $w$, construct a CFG parse forest for all of the right parse tree

components of the synchronous derivations of $[w,\ u]$ under $G$, where $u \in T(G, w)$. (The term decoding is also used with a related meaning, in case of statistical translation models embedded within a noisy channel architecture.) In what follows we assume $w = a_1 a_2 \cdots a_n$, with $n \geq 0$ and $a_i \in V_T$ for each $i$, $1 \leq i \leq n$.

We define a FA $M$ such that $L(M) = \{w\}$. To do this, let $Q = \{q_i \mid 0 \leq i \leq n\}$, and let $M = (Q, V_T, \delta, q_0, \{q_n\})$, where $\delta = \{(q_{i-1}, a_i, q_i) \mid 1 \leq i \leq n\}$. Note that $M$ is deterministic. Our translation algorithm is defined by the following two steps.

**Step 1:** Apply to $G$ and $M$ the left intersection construction with a regular language from section 4, resulting in a new SCFG $G_{\cap,w}$ such that (Theorem 3)

$$T(G_{\cap,w}) = T(G) \cap (L(M) \times V_T^*) = \{[w,\ u] \mid [w,\ u] \in T(G)\}.$$

**Step 2:** Apply to $G_{\cap,w}$ the auto-projection construction from section 3, and produce a CFG $\mathsf{auto\text{-}proj}(G_{\cap,w}, 2)$ that generates the language (Theorem 2)

$$\begin{aligned} L(\mathsf{auto\text{-}proj}(G_{\cap,w}, 2)) &= \mathsf{proj}(T(G_{\cap,w}), 2) \\ &= \{u \mid [w,\ u] \in T(G)\} \ = \ T(G, w). \end{aligned}$$

This CFG is the output of the algorithm.

When we apply the appropriate node relabeling to the parse trees generated by CFG $\mathsf{auto\text{-}proj}(G_{\cap,w}, 2)$, we obtain all and only the right parse tree components that are assigned to strings $u \in T(G_{\cap,w})$ by the input SCFG $G$ when deriving $[w,\ u]$. This follows from Theorem 3 and (the symmetrical statement of) Theorem 1. We can then conclude that CFG $\mathsf{auto\text{-}proj}(G_{\cap,w}, 2)$ is the desired parse forest and the translation algorithm above is correct.

**Example 4**

Let us consider again the SCFG $G$ from Example 1, whose synchronous productions are reported below for convenience:

$$
\begin{array}{l}
s_1 : [S \rightarrow A_1^{(1)} C_1^{(2)},\ \ S \rightarrow A_2^{(1)} C_2^{(2)}], \\
s_2 : [C_1 \rightarrow B_1^{(1)} S^{(2)},\ \ C_2 \rightarrow B_2^{(1)} S^{(2)}], \\
s_3 : [C_1 \rightarrow B_1^{(1)} S^{(2)},\ \ C_2 \rightarrow S^{(2)} B_2^{(1)}], \\
s_4 : [C_1 \rightarrow B_1^{(1)},\ \ C_2 \rightarrow B_2^{(1)}], \\
s_5 : [A_1 \rightarrow a_1,\ \ A_2 \rightarrow a_2], \\
s_6 : [A_1 \rightarrow a_1,\ \ A_2 \rightarrow \varepsilon], \\
s_7 : [B_1 \rightarrow b_1,\ \ B_2 \rightarrow b_2].
\end{array}
$$

Assume the input string $w = a_1 b_1 a_1 b_1$. There are five different strings that can be obtained as a translation of $w$ through $G$, and we have

$$T(G, w) = \{a_2 b_2 a_2 b_2, a_2 b_2 b_2, b_2 a_2 b_2, b_2 b_2, a_2 a_2 b_2 b_2\}.$$

More precisely, when translating $w$, $G$ can optionally erase each occurrence of $a_2$ and can optionally invert the nonterminals $B_2$ and $S$ that are rewritten out of $C_2$. This would yield eight possible translations, but strings $a_2 b_2 b_2$ and $b_2 b_2$ are ambiguous, the former has three parse trees and the latter has two.

We now construct FA $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a_1, a_2, b_1, b_2\}, \delta, q_0, \{q_4\})$, with $\delta = \{(q_0, a_1, q_1), (q_1, b_1, q_2), (q_2, a_1, q_3), (q_3, b_1, q_4)\}$. In Step 1 of the algorithm above, and after the removal of useless symbols, we obtain SCFG $G_{\cap,w}$ with the synchronous

productions:

$$
\begin{array}{ll}
s'_1: & [(q_0,S,q_4) \rightarrow (q_0,A_1,q_1)^{(1)}(q_1,C_1,q_4)^{(2)},\ \ S \rightarrow A_2^{(1)}C_2^{(2)}],\\
s'_2: & [(q_1,C_1,q_4) \rightarrow (q_1,B_1,q_2)^{(1)}(q_2,S,q_4)^{(2)},\ \ C_2 \rightarrow B_2^{(1)}S^{(2)}],\\
s'_3: & [(q_1,C_1,q_4) \rightarrow (q_1,B_1,q_2)^{(1)}(q_2,S,q_4)^{(2)},\ \ C_2 \rightarrow S^{(2)}B_2^{(1)}],\\
s'_4: & [(q_2,S,q_4) \rightarrow (q_2,A_1,q_3)^{(1)}(q_3,C_1,q_4)^{(2)},\ \ S \rightarrow A_2^{(1)}C_2^{(2)}],\\
s'_5: & [(q_3,C_1,q_4) \rightarrow (q_3,B_1,q_4)^{(1)},\ \ C_2 \rightarrow B_2^{(1)}],\\
s'_6: & [(q_0,A_1,q_1) \rightarrow a_1,\ \ A_2 \rightarrow a_2],\\
s'_7: & [(q_0,A_1,q_1) \rightarrow a_1,\ \ A_2 \rightarrow \varepsilon],\\
s'_8: & [(q_1,B_1,q_2) \rightarrow b_1,\ \ B_2 \rightarrow b_2],\\
s'_9: & [(q_2,A_1,q_3) \rightarrow a_1,\ \ A_2 \rightarrow a_2],\\
s'_{10}: & [(q_2,A_1,q_3) \rightarrow a_1,\ \ A_2 \rightarrow \varepsilon],\\
s'_{11}: & [(q_3,B_1,q_4) \rightarrow b_1,\ \ B_2 \rightarrow b_2].
\end{array}
$$

We now move to Step 2 of the algorithm. We observe that the set $\mathcal{L}(G_{\cap,w})$ contains elements of the form $[(q_i,A,q_j),A']$, $A,A' \in V_N$. We construct the output CFG auto-proj$(G_{\cap,w},2)$ with nonterminals $\mathcal{L}(G_{\cap,w})$ and productions:

$$
\begin{array}{ll}
p_1: & [(q_0,S,q_4),S] \rightarrow [(q_0,A_1,q_1),A_2][(q_1,C_1,q_4),C_2],\\
p_2: & [(q_1,C_1,q_4),C_2] \rightarrow [(q_1,B_1,q_2),B_2][(q_2,S,q_4),S],\\
p_3: & [(q_1,C_1,q_4),C_2] \rightarrow [(q_2,S,q_4),S][(q_1,B_1,q_2),B_2],\\
p_4: & [(q_2,S,q_4),S] \rightarrow [(q_2,A_1,q_3),A_2][(q_3,C_1,q_4),C_2],\\
p_5: & [(q_3,C_1,q_4),C_2] \rightarrow [(q_3,B_1,q_4),B_2],\\
p_6: & [(q_0,A_1,q_1),A_2] \rightarrow a_2,\\
p_7: & [(q_0,A_1,q_1),A_2] \rightarrow \varepsilon,\\
p_8: & [(q_1,B_1,q_2),B_2] \rightarrow b_2,\\
p_9: & [(q_2,A_1,q_3),A_2] \rightarrow a_2,\\
p_{10}: & [(q_2,A_1,q_3),A_2] \rightarrow \varepsilon,\\
p_{11}: & [(q_3,B_1,q_4),B_2] \rightarrow b_2.
\end{array}
$$

It is not difficult to see that CFG auto-proj$(G_{\cap,w},2)$ generates all and only the strings in $T(G,w)$. Moreover, each such a string is generated through exactly the same component parse trees that are used by $G$ when translating from $w$, modulo a straightforward node relabeling. Let us go through the three leftmost derivations of string $a_2b_2b_2$ under auto-proj$(G_{\cap,w},2)$. We write $\Rightarrow^p$ to denote the derive relation in such a CFG by means of production $p$. Without using inversion of $B_2,S$ (that is, using production $p_2$ rather than production $p_3$) and erasing the second occurrence of $a_2$, we get the derivation

$$[(q_0,S,q_4),S] \Rightarrow^{p_1} [(q_0,A_1,q_1),A_2][(q_1,C_1,q_4),C_2]$$

$$\Rightarrow^{p_6} \; a_2[(q_1, C_1, q_4), C_2]$$

$$\Rightarrow^{p_2} \; a_2[(q_1, B_1, q_2), B_2][(q_2, S, q_4), S]$$

$$\Rightarrow^{p_8} \; a_2b_2[(q_2, A_1, q_3), A_2][(q_3, C_1, q_4), C_2]$$

$$\Rightarrow^{p_{10}} \; a_2b_2[(q_3, C_1, q_4), C_2]$$

$$\Rightarrow^{p_5} \; a_2b_2[(q_3, B_1, q_4), B_2]$$

$$\Rightarrow^{p_{11}} \; a_2b_2b_2.$$

If instead we use inversion of $B_2, S$, we get the two occurrences of $a_2$ at adjacent positions in the string, and we can either erase the first or the second occurrence in order to derive $a_2b_2b_2$, as reported in what follows:

$$[(q_0, S, q_4), S] \Rightarrow^{p_1} \; [(q_0, A_1, q_1), A_2][(q_1, C_1, q_4), C_2]$$

$$\Rightarrow^{p_7} \; [(q_1, C_1, q_4), C_2]$$

$$\Rightarrow^{p_3} \; [(q_2, S, q_4), S][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_4} \; [(q_2, A_1, q_3), A_2][(q_3, C_1, q_4), C_2][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_9} \; a_2[(q_3, C_1, q_4), C_2][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_5} \; a_2[(q_3, B_1, q_4), B_2][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_{11}} \; a_2b_2[(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_8} \; a_2b_2b_2,$$

$$[(q_0, S, q_4), S] \Rightarrow^{p_1} \; [(q_0, A_1, q_1), A_2][(q_1, C_1, q_4), C_2]$$

$$\Rightarrow^{p_6} \; a_2[(q_1, C_1, q_4), C_2]$$

$$\Rightarrow^{p_3} \; a_2[(q_2, S, q_4), S][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_4} \; a_2[(q_2, A_1, q_3), A_2][(q_3, C_1, q_4), C_2][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_{10}} \; a_2[(q_3, C_1, q_4), C_2][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_5} \; a_2[(q_3, B_1, q_4), B_2][(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_{11}} \; a_2 b_2[(q_1, B_1, q_2), B_2]$$

$$\Rightarrow^{p_8} \; a_2 b_2 b_2.$$

We now switch to the problem of **synchronous parsing** for SCFGs, defined as follows: given an input SCFG $G$ and an input string pair $[w_1,\ w_2]$, construct a SCFG parse forest for all the synchronous derivations of $[w_1,\ w_2]$ under $G$. Note that here we are viewing SCFGs as convenient representations of forests of aligned parse trees. The problem of synchronous parsing has applications in word and phrase alignments for multilingual corpora, automatic dictionary construction, and is also at the basis of training algorithms for translation models.

Let SCFG $G$ be defined as above, and let $w_1 = a_1 a_2 \cdots a_n$ and $w_2 = b_1 b_2 \cdots b_m$, with $n, m \geq 0$ and $a_i, b_j \in V_T$ for each $i$ and $j$, $1 \leq i \leq n$, $1 \leq i \leq m$. Similarly to what we have done above, we define deterministic FAs $M_1 = (Q_1, V_T, \delta_1, q_{10}, \{q_{1n}\})$ such that $L(M_1) = \{w_1\}$, and $M_2 = (Q_2, V_T, \delta_2, q_{20}, \{q_{2m}\})$ such that $L(M_2) = \{w_2\}$. An algorithm for the problem of synchronous parsing can be defined by the following three steps.

**Step 1:** Apply to $G$ and $M_1$ the left intersection construction with a regular language from section 4, resulting in a new SCFG $G_{\cap,w_1}$ such that

$$T(G_{\cap,w_1}) = T(G) \cap (L(M_1) \times V_T^*).$$

**Step 2:** Apply to $G_{\cap,w_1}$ and $M_2$ the right intersection construction with a regular language, resulting in a new SCFG $G_{\cap,w_1,w_2}$ such that

$$\begin{aligned} T(G_{\cap,w_1,w_2}) &= T(G_{\cap,w_1}) \cap (V_T^* \times L(M_2)) \\ &= T(G) \cap (L(M_1) \times L(M_2)) \ = \ \{[w_1,\ w_2]\}. \end{aligned}$$

**Step 3:** Output the SCFG $G_{\cap,w_1,w_2}$.

From Theorem 3, and modulo some node relabeling, the SCFG $G_{\cap,w_1,w_2}$ is a representation of the desired parse forest of all aligned parse trees generating pair $[w_1,\ w_2]$ under $G$. From the above algorithm, it is also apparent that there is a strong similarity between the problem of synchronous parsing and the translation problem for SCFGs, as both can be reduced to an intersection problem with regular languages. Such a similarity has often been informally observed in the literature, but never formally proved in some framework.

The problem of synchronous parsing for SCFGs can be associated with a corresponding decision problem, called the problem of **synchronous recognition** for SCFGs, defined as follows: given an input SCFG $G$ and an input string pair $[w_1,\ w_2]$, decide whether $[w_1,\ w_2] \in T(G)$. The problem of synchronous recognition for SCFGs can be solved by running on $G$ and $[w_1,\ w_2]$ the algorithm for the problem of synchronous parsing presented above. We then answer yes if and only if the output SCFG $G_{\cap,w_1,w_2}$ generates a non-empty translation. Translation non-emptiness for SCFGs can in turn be tested by reducing the SCFG and checking whether the resulting set of synchronous productions is non-empty.

We now switch to the discussion of the computational complexity of the algorithms presented above and, more in general, of the considered computational problems. In

the intersection steps of both our algorithms we apply the intersection construction, which in the worst case requires an exponential time computation, as already observed in section 4. We thus conclude that our algorithms show exponential time behavior, when the SCFG is considered as part of the input. But on a theoretical perspective, such an exponential time behavior seems unavoidable, as discussed below.

It has been shown by Satta and Peserico (2005) that the problem of synchronous recognition for SCFGs is NP-hard. Such a problem seems to be the least complex one among the three problems considered so far, assuming polynomial time Turning reductions. In fact, we can reduce from the problem of synchronous recognition to the problem of synchronous parsing by testing the non-emptiness of the translation generated by a SCFG, as pointed out above. Such a test can be easily carried out in polynomial time by adapting standard techniques for the reduction of CFGs, as already observed in section 2. Similarly, we can reduce from the problem of synchronous recognition to the translation problem as follows. Given a SCFG $G$ and a string pair $[w_1,\ w_2]$, we first run the translation algorithm on $G$ and $w_1$. This results in a CFG $G'$ representing the desired parse forest. We can then test whether $w_2 \in L(G')$. Such a membership problem can be computed in polynomial time in the size of $G'$, as it is well known from the parsing literature; see for instance (Hopcroft and Ullman 1979). We thus conclude that both the translation problem and the problem of synchronous parsing for SCFGs are NP-hard, under a polynomial time Turing reduction.

Notice that the above discussion assumes definitions of the translation problem and of the problem of synchronous parsing for SCFGs in which the output parse forests are represented by means of SCFGs and CFGs, respectively. But the argument holds also for other representations of parse forests, as those discussed in section 6, which can be mapped to the CFG representation by means of linear time computations. In conclusion,

under the assumption that the classes P and NP are separable, for the solution of the problems discussed in this section one cannot hope to do a much better job than the algorithms we have presented.

Turning to an application oriented perspective, things look rather differently. The exponential time analysis for the intersection construction in section 4 holds under the assumption that the rank of the input SCFGs is unbounded. However, it might be the case that the synchronous productions we deal with do not implement the specific worst case permutations investigated in (Aho and Ullman 1969). This would in turn allow us to "factor" each synchronous production into a set of equivalent synchronous productions with rank bounded by some constant. In this case the intersection construction can be carried out in polynomial time (with degree of the polynomial depending on the above constant) and this is also the time upper bound for the algorithms described in this section, since all other steps use polynomial time procedures, as discussed in section 3. The above hypothesis about the type of permutations we deal with in natural language translation needs to be experimentally evaluated for individual language pairs. To date the issue still appears to be controversial; see for instance work by Fox (2002), Zhang et al. (2006) and Wellington, Waxmonsky, and Melamed (2006). We remark here that, in line with the above hypothesis, very efficient algorithms for the factorization of synchronous productions have been developed by Zhang et al. (2006) and Gildea, Satta, and Zhang (2006). These algorithms work in linear time and reduce as much as possible the rank of each synchronous production in a SCFG.

We now switch to the discussion of some applications of the framework we have developed so far. Observe that in the algorithm for the translation problem above we have not set any restriction on the right dimension of the source SCFG. However, one common application in statistical machine translation is to intersect the output of the

algorithm with a probabilistic regular language implementing a language model for the target language. Combining probabilities of the translation model with probabilities from an independent language model has proved very effective for the overall translation quality; see for instance work by Chiang (2005). Using our abstract framework, we provide in the next example a real case analysis of this technique.

**Example 5**

We show how to apply the developed framework to reformulate translation algorithms that have already been presented in the literature. We consider here the already mentioned Inversion Transduction Grammars (ITGs) originally proposed by Wu (1997) along with their probabilistic version, called Stochastic ITG (SITG). ITGs are a restricted form of SDTS, and thus of SCFGs, where synchronous productions display nonterminals in the right-hand side of the right production component in the same or in the reverse order as they appear in the right-hand side of left production component. Without sacrificing too much of the expressivity of the formalism, in the perspective of natural language translation, this simple restriction results in some very interesting properties, including the existence of canonical forms of rank two for ITGs which, as already discussed, does not hold for SDTS and SCFGs (Aho and Ullman 1969).

Wu and Wong (1998) report an algorithm for the translation problem based on SITGs. As far as we know, this is the first example of a dynamic programming method for the translation problem exploiting a hierarchical synchronous rewriting formalism. The algorithm takes as input a string $w$ in the source language, a SITG $G$ as a translation model and a 2-gram language model for the target language. The language model is used to filter the output of the translation model, and proves very effective in the perspective of the resulting accuracy of the system. The algorithm constructs from $w$ an indexed structure, which is implemented through a chart data structure (Jurafsky

and Martin 2000), representing all parse trees for strings in the target language that are translations of $w$ and that comply with the target language model. These parse trees are also assigned a probability that is a combination of probabilities computed through the translation model and the language model. A standard Viterbi method is then used to extract the parse tree(s) with highest probability and the associated generated string. Assuming that each word in the source language can be mapped into a set of words in the target language of size bounded by some constant, which is independent of $G$, the proposed algorithm works in time $\mathcal{O}(|G| \cdot |w|^7)$. An improved version of this algorithm has later been presented by Huang, Zhang, and Gildea (2005), where advanced dynamic programming techniques are exploited resulting in a time upper bound of $\mathcal{O}(|G| \cdot |w|^6)$.

We show here how the improved upper bound above can be obtained in a simple way using the framework we have presented. We focus on the algorithmic specification of the method and ignore probability assignments for the productions and transitions of the models. We can encode $w$ into a FA $M_1$ with set of states $Q_1$ of size $n+1$. We particularize the 2-gram language model for the target language to the set of strings that use the only words that are translations of words occurring in $w$. We can implement this model as a FA $M_2$, where each state encodes the last word read and conditions the next word to be read through its outgoing transitions. Using the above assumption that each word occurring in $w$ can be translated into at most a constant number of words in the target language, $M_2$ uses a set of states $Q_2$ of size $\mathcal{O}(|w|)$, and a number of transitions $\mathcal{O}(|w|^2)$. Finally, remember that for the SITG model our SCFG $G$ has rank $r_G = 2$. We can then apply twice the regular language intersection, on the two dimensions. Intersection on the left dimension of $G$ and $M_1$ produces a SCFG $G'$ of size $\mathcal{O}(|G| \cdot |Q_1|^{r_G+1})$. When we further intersect $G'$ on the right dimension with $M_2$, and use the above parameters,

we derive

$$\begin{aligned}
&\mathcal{O}(|G'| \cdot |Q_2|^{r_G+1}) = \\
&= \mathcal{O}(|G| \cdot |Q_1|^{r_G+1} \cdot |Q_2|^{r_G+1}) \\
&= \mathcal{O}(|G| \cdot |w|^3 \cdot |w|^3) \;=\; \mathcal{O}(|G| \cdot |w|^6),
\end{aligned}$$

which is the improved upper bound provided by Huang, Zhang, and Gildea (2005). This simple analysis shows the convenience of working with the abstract framework we have developed in this article.

We now discuss a variant of the translation problem, in which the input is a parse tree, or a parse forest, rather than a sentence. This problem has several applications in case a statistical parser for the source language is available, and can thus be exploited to restrict the translation process to a list of $N \geq 1$ most likely parse trees for the source sentence. The **tree translation problem** for SCFGs is defined as follows: given an input SCFG $G$ and an input CFG $G_f$ representing a parse forest, construct an output CFG representing the parse forest of all the right parse tree components that translate under $G$ the trees generated by $G_f$.

Assume $G$ as above and let $G_f = (V_{N,f}, V_T, P_f, \eta_{S_1})$ be a CFG as in section 4. Our translation algorithm is defined by the following two steps.

**Step 1:** Apply to $G$ and $G_f$ the left intersection construction with a parse forest from section 4, resulting in a new SCFG $G_{\cap,f}$ such that

$$T(G_{\cap,f}) = T(G) \cap (L(G_f) \times V_T^*).$$

**Step 2:** Apply the auto-projection construction from section 3, and output CFG auto-proj$(G_{\cap,f}, 2)$.

When we apply node relabeling to the parse trees generated by auto-proj$(G_{\cap,f}, 2)$, we obtain all and only the right parse tree components that are translations under $G$ of the parse trees generated by the input $G_f$. Step 1 of the algorithm runs in time $\mathcal{O}(|P_f| \cdot |G|)$, as discussed in section 4. This is also a space bound on the size of the produced grammar. Therefore, the linear time auto-projection construction in Step 2 also runs time $\mathcal{O}(|P_f| \cdot |G|)$. We conclude that the above algorithm solves the tree translation problem with a polynomial time upper bound of $\mathcal{O}(|G_f| \cdot |G|)$.

Once more, we propose an example with an analysis of an algorithm already presented in the literature, demonstrating the convenience of using our abstract framework.

**Example 6**

A statistical translation model has been presented by Yamada and Knight (2001), translating English parse trees into Japanese sentences. In this model a source probabilistic CFG is explicitly defined, while a target CFG is only implicitly provided through some elementary tree-editing operations associated with the productions of the source grammar, such as children reordering, and translation and insertion of leaf words. The authors provide an unsupervised maximum-likelihood estimation algorithm based on the expectation maximization method

The translation model can be viewed as a synchronous rewriting system, where synchronous productions implement the children reordering and the word translation operations. An insertion operation of say a word $b$ as rightmost child of a node $C_2$, in the context of a production $A_2 \rightarrow B_2 C_2$, can be implemented through the synchronous

productions

$$[A_1 \rightarrow C^{(1)}_{1A_1} B^{(2)}_{1A_1},\ \ A_2 \rightarrow B^{(2)}_{2A_2} C^{(1)}_{2A_2}],$$
$$[C_{1A_1} \rightarrow C^{(1)}_1 X^{(2)},\ \ C_{2A_2} \rightarrow C^{(1)}_2 X^{(2)}],$$
$$[X \rightarrow \varepsilon,\ \ X \rightarrow b].$$

The intermediate nonterminals above of the form $C_{2A_2}$ are used to record information about the parent node. In this way, the construction preserves the statistical parameters of the original model.

The estimation algorithm in (Yamada and Knight 2001) takes as input a corpus of pairs $[t, w]$, where $t$ is a parse tree and $w$ is a sentence translation of the yield of $t$. It then constructs an and-or graph representing all possible combinations of elementary operations of the model that translate $t$ into $w$. This structure is used to estimate the parameter models through an iterative procedure counting the occurrences of each elementary operation. Let $G$ be a SCFG implementing the translation model as above (again we ignore the attached probabilities here). We first intersect $G$ on the left with a parse forest representing $t$, and then intersect the resulting SCFG on the right with a FA representing $w$. The new SCFG that we obtain represents the parse forest of all synchronous derivations in $G$ with left parse tree component $t$ and right string $w$. This representation is a syntactic variant of the and-or graph above, and it has the advantage over the graph representation that it does not require additional definitions.

Our construction of the parse forest can be carried out in time $\mathcal{O}(|G| \cdot |t| \cdot |w|^d)$ with $d = \min\{r_t, r_G\}$. Here $r_t$ is the maximum number of children for a node of $t$, and $r_G$ is the rank of $G$. Thus in the worst case the algorithm runs in exponential time. This is the same time complexity reported by Yamada and Knight (2001). It is possible to improve upon this result, as discussed in what follows. As before, we intersect $G$ on the left with a parse forest representing $t$, resulting in a SCFG $G_t$. We then apply the auto-projection and construct a CFG $G'_t = \mathsf{auto\text{-}proj}(G_t, 2)$. We can now use standard linear

time algorithms to cast $G'_t$ in some form with at most two nonterminals in the right-hand side of its productions. Let $G''_t$ be such a grammar. Finally, we apply the standard CFG intersection construction discussed in section 4 to $G''_t$ and an FA encoding $w$, resulting in a new CFG $G_{t,w}$. The relevant nonterminals of $G_{t,w}$ have the form $(\eta, q_i, A_1, A_2, q_j)$ and represent all possible alignments of a node $\eta$ of $t$ labeled by nonterminal $A_1$, and a substring $a_{i+1}a_{i+2}\cdots a_j$ of $w$ which can be derived from nonterminal $A_2$. (Some other nonterminals are present in $G_{t,w}$ as a byproduct of the binarization algorithm, and can be ignored.) From such a parse forest we can retrieve all the information that is needed by the iterative procedure for the estimation of the model parameters. The construction of $G_{t,w}$ above requires an overall amount of time $\mathcal{O}(|G| \cdot |t| \cdot |w|^2)$, which is now polynomial in the size of the input.

We close with some practical remarks. The intersection construction with regular languages exploited by our algorithms has been presented in section 4 in its simplest form. This is in line with the goal of this article to establish a mathematical framework for the design of translation algorithms and the investigation of their theoretical properties. However, in the implementation of applications working with large size translation models, much more efficient methods can be devised for the computation of the intersection construction, avoiding the generation of huge numbers of useless nonterminals. Top-down and bottom-up strategies can be implemented, as is standard practice in the design of (string) parsing algorithms, resulting in a more selective search in the space of all candidate synchronous productions for the target grammar. In addition, when these models are enriched with probabilities, several heuristic strategies can also be implemented to filter the search space.

## 6. Discussion and future work

In this article we have borrowed from the parsing literature a well-established framework based on the idea of language intersections. We have adapted this methodology to the development of translation and related algorithms for SCFGs. The framework has also been used to establish formal properties of translation algorithms and to compare existing algorithms under a mathematical basis.

As already discussed, our choice of considering SCFGs rather than the SDTS of Aho and Ullman (1972) is motivated by the need of decoupling the two left-hand side symbols in a synchronous productions, which gives us more flexibility and more modularity in defining intersections on a single dimension of the grammar, leaving untouched the symbols in the other dimension. One might argue that the use of SDTS would spare the need for the auto-projection construction of section 3, using the already known result presented by Aho and Ullman (1972, Exercise 3.1.7, p. 235), stating that SDTS can be directly projected on both dimensions. But this advantage is only apparent, because in case one uses SDTS, the definition of the intersection on one dimension would have to embed the auto-projection construction, in order to satisfy the requirement that indexed nonterminals must be equal. This results in a lack of modularity. Our approach is logically cleaner, allowing us to keep the two constructions independent. Also, the added expressivity of SCFGs is particularly convenient when proving formal properties of the model.

In this article we have adopted the CFG representation for parse forests, which does not require any additional machinery since our translation models are based on CFG productions. CFG representations of parse forests are widely used in the parsing literature; see for instance work by Billot and Lang (1989), Lang (1991), Vijay-Shanker and Weir (1993), Lang (1994) and Nederhof (2005). Representations of parse forests

superficially different from CFGs are found in the literature, as for instance the and-or graphs used by Yamada and Knight (2001). An and-or graph is a graph whose vertices are of two special kinds, called and-vertices and or-vertices, and are usually alternated. These graphs are then visited by following all the outgoing edges at the and-vertices and by following only one outgoing edge at the or-vertices. As pointed out by Lang (1991), and-or graphs can also be viewed as CFGs, and the two representations can be related by a linear time computation. A second representation for parsing forests proposed by Klein and Manning (2004) is based on hypergraphs. In an hypergraph each edge, called hyperedge, may have more than one source or target vertex. The specific kind of hyperedges that are used to represent parse forests have a single target vertex, encoding an internal node $\nu$ in some parse tree, and several source vertices, encoding the children of $\nu$. Again, this notation is a syntactic variant of a CFG, as observed in (Nederhof 2005), and the two representations can be related by means of a linear time computation.

In order to keep the presentation at a simple level, in this article we have worked with a synchronous formalism that only allows the generation of parse tree pairs that have the same structure, modulo some relabeling of the nodes and some reordering of their children. More expressive formalisms than our SCFGs have been presented in the literature, that can generate parse trees with different structures. See for instance the already cited multitext grammars of Melamed (2003), which use the so-called independent rewriting, and the synchronous tree substitution grammar of Eisner (2003). Furthermore, synchronous systems beyond the generative power of context-free rewriting, involving so-called discontinuous constituents, are also used as translation models. See for instance the synchronous tree adjoining grammars of Shieber (1994) and the generalized multitext grammars proposed by Melamed, Satta, and Wellington

(2004b). It is not difficult to extend the intersection framework of this article to these formalisms.

A second important extension of the framework proposed here involves the use of probabilities, which are adopted in most real world machine translation applications. One can extend SCFGs by associating probabilities to each synchronous production, in a way similar to the stochastic extension of SDTS proposed by Maryanski and Thomason (1979). By slightly complicating the notation of section 4 we can extend our intersection algorithms to probabilistic formalisms. In the case of probabilistic FAs, this can be done by taking into account the transitions of the automaton and their probabilities into the synchronous productions of the intersection grammar, following the lines developed by Nederhof and Satta (2003) for the case of probabilistic CFGs. Things are however more complicated for the auto-projection construction of section 3. Our construction results in a many-to-one mapping from synchronous derivations to context-free parse tree components. Unfortunately, it is not possible to assign probabilities to the obtained context-free productions in such a way that the probability of a parse tree component is the sum of the probabilities of the source synchronous derivations. This fact follows from a result showing that probabilistic nondeterministic FAs cannot be converted into equivalent probabilistic deterministic FAs, in the general case (Vidal et al. 2005). One must therefore refine the auto-projection construction in such a way that a bijection is obtained from synchronous derivations to parse tree components. This requires a more complex notation than the one provided in this article, and is left for future work.

**Acknowledgments**

The author is indebted for discussion of ideas developed in this article to Liang Huang, Dan Melamed and Mark-Jan Nederhof. The intersection construction with a parse forest presented in section 4 is inspired by a similar construction for synchronous tree-substitution grammars, developed with Liang Huang in an unpublished work. The author has been partially supported by MIUR under project PRIN No. 2003091149_005.

**References**


Aho, A. V. and J. D. Ullman. 1969. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56.

Aho, A. V. and J. D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

Alshawi, Hiyan, Srinivas Bangalore, and Shona Douglas. 2000. Learning dependency translation models as collections of finite state head transducers. *Computational Linguistics*, 26(1):45–60, March.

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

Bertsch, E. and M.-J. Nederhof. 2001. On the complexity of some extensions of RCG parsing. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, pages 66–77, Beijing, China, October.

Billot, S. and B. Lang. 1989. The structure of shared forests in ambiguous parsing. In *27th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 143–151, Vancouver, British Columbia, Canada, June.

Bod, R., R. Scha, and K. Sima'an. 2003. *Data-Oriented Parsing*. CSLI Publications, University of Chicago Press.

Boullier, Pierre. 2004. Range concatenation grammars. In H. Bunt, J. Carroll, and G. Satta, editors, *New Developments in Parsing Technology*, volume 23 of *Text, Speech and Language Technology*. Kluwer Academic Publishers, pages 269–289.

Brown, Peter F., Stephen A. Della Pietra, Vincent Della J. Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–312, June.

Chiang, D. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proc. of the 43$^{rd}$ ACL*, pages 263–270.

Dassow, J. and G. Păun. 1989. *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, Germany.

Dras, M. 1999. A meta-level grammar: Redefining synchronous TAG for translation and paraphrase. In *Proc. of the 37$^{th}$ ACL*, pages 80–87, College Park, Maryland.

Eisner, Jason. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (Companion Volume)*, pages 205–208, Sapporo, July.

Fox, H. 2002. Phrasal cohesion and statistical machine translation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*.

Galley, M., M. Hopkins, K. Knight, and D. Marcu. 2004. Whats in a translation rule? In *Proc. of HLT/NAACL 2004 Conference*.

Gildea, D., G. Satta, and H. Zhang. 2006. Factoring synchronous grammars by sorting. In *Proceedings of the 44th Annual Conference of the Association for Computational Linguistics (COLING/ACL-06)*, Sydney, Australia.

Harrison, M.A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley.

Hopcroft, J. E. and J. D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA.

Huang, L., H. Zhang, and D. Gildea. 2005. Machine translation as lexicalized parsing with hooks. In *Proceedings of the 9$^{th}$ Int. Workshop on Parsing Technologies*, Vancouver, Canada.

Jurafsky, D. and J.H. Martin. 2000. *Speech and Language Processing*. Prentice-Hall.

Klein, D. and C. D. Manning. 2004. Parsing and hypergraphs. In H. Bunt, J. Carroll, and G. Satta, editors, *New Developments in Parsing Technology*, volume 23 of *Text, Speech and Language Technology*. Kluwer Academic Publishers.

Knight, K. and J. Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Proc. of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing), Lecture Notes in Computer Science*, Berlin, Germany. Springer-Verlag.

Koehn, P., F. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL03)*, pages 48–54, Morristown, NJ, USA. Association for Computational Linguistics.

Kumar, S. and W. Byrne. 2003. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of HLT-NAACL*.

Lang, B. 1991. Towards a uniform formal framework for parsing. In M. Tomita, editor, *Current Issues in Parsing Technology*. Kluwer Academic Publishers, chapter 11, pages 153–171.
Lang, B. 1994. Recognition can be harder than parsing. *Computational Intelligence*, 10(4).
Lang, Bernard. 1991. Towards a uniform formal framework for parsing. In M. Tomita, editor, *Current Issues in Parsing Technology*. Kluwer Academic Publishers, pages 153–171.
Lewis, P. M. and R. E. Stearns. 1968. Syntax-directed transduction. *Journal of the Association for Computing Machinery*, 15(3):465–488.
Maryanski, Fred J. and Michael G. Thomason. 1979. Properties of stochastic syntax-directed translation schemata. *International Journal of Computer and Information Sciences*, 8(2):89–110.
May, J. and K. Knight. 2006. Tiburon: A weighted tree automata toolkit. In *Proc. of the International Conference on Implementation and Application of Automata (CIAA), Lecture Notes in Computer Science*, volume 4094, Berlin, Germany. Springer-Verlag.
Melamed, D. and W. Wang. 2005. Proteus project working paper #2: Statistical machine translation by generalized parsingy. Proteus Project technical report #05-001. Available from http://nlp.cs.nyu.edu/pubs/.
Melamed, I. Dan. 2003. Multitext grammars and synchronous parsers. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, pages 158–165, Edmonton, Canada.
Melamed, I. Dan. 2004. Statistical machine translation by parsing. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, Barcelona, Spain.
Melamed, I. Dan, G. Satta, and B. Wellington. 2004a. Generalized multitext grammars. Technical Report 04-003, Proteus Project, New York University. `http://nlp.cs.nyu.edu/pubs/`.
Melamed, I. Dan, Giorgio Satta, and Benjamin Wellington. 2004b. Generalized multitext grammars. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, Barcelona, Spain.
Nederhof, M.-J. 2005. A general technique to train language models on language models. *Computational Linguistics*, 31(2):173–185.
Nederhof, M.-J. and G. Satta. 2003. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, LORIA, Nancy, France, April.
Och, Franz Josef and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, July.
Och, Franz Josef, Christoph Tillmann, and Hermann Ney. 1999. Improved alignment models for statistical machine translation. In *Proceedings of the 4nd Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 20–28, College Park, Maryland.
Poutsma, A. 2000. Data-oriented translation. In *Proc. of the 18$^{th}$ COLING*, pages 635–641, Saarbrücken, Germany.
Rambow, O. and G. Satta. 1996. Synchronous models of language. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, Santa Cruz, USA.
Satta, G. and E. Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proc. of the 2005 Conference on Empirical Methods in Natural Language Processing*, Vancouver, Canada.
Shieber, S. 2004. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 7)*, Vancouver, Canada.
Shieber, S. 2006. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, Trento, Italy.
Shieber, S. M. and Y. Schabes. 1990. Synchronous tree-adjoining grammars. In *Proc. of the 13$^{th}$ COLING*, pages 253–258, Helsinki, Finland.
Shieber, S.M. 1994. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385.
Vidal, E., F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. 2005. Probabilistic finite-state machines – Part I. *IEEE Trans. on Pattern analysis and Machine Intelligence*, 27(7):1013–1025.
Vijay-Shanker, K. and D. J. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.
Way, A. 2003. Machine translation using LFG-DOP. In R. Bod, R. Scha, and K. Sima'an, editors, *Data-Oriented Parsing*. CSLI Publications, Stanford, CA, pages 359–384.

Wellington, B., S. Waxmonsky, and D. Melamed. 2006. Empirical lower bounds on the complexity of translational equivalence. In *Proceedings of the 44th Annual Conference of the Association for Computational Linguistics (COLING/ACL-06)*, Sydney, Australia.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, September.

Wu, Dekai and Hongsing Wong. 1998. Machine translation with a stochastic grammatical channel. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, Montreal, Canada, July.

Yamada, Kenji and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 531–538, Toulouse, July.

Zhang, H., L. Huang, D. Gildea, and K. Knight. 2006. Synchronous binarization for machine translation. In *Proc. of HLT/NAACL 2006 Conference*, New York.