A Generalized View on Parsing and Translation

Alexander Koller
Dept. of Linguistics
University of Potsdam, Germany
koller@ling.uni-potsdam.de

Marco Kuhlmann
Dept. of Linguistics and Philology
Uppsala University, Sweden
marco.kuhlmann@lingfil.uu.se

Abstract

We present a formal framework that generalizes a variety of monolingual and synchronous grammar formalisms for parsing and translation. Our framework is based on regular tree grammars that describe derivation trees, which are interpreted in arbitrary algebras. We obtain generic parsing algorithms by exploiting closure properties of regular tree languages.

1 Introduction

Over the past years, grammar formalisms that relate pairs of grammatical structures have received much attention. These formalisms include *synchronous grammars* (Lewis and Stearns, 1968; Shieber and Schabes, 1990; Shieber, 1994; Rambow and Satta, 1996; Eisner, 2003) and *tree transducers* (Comon et al., 2007; Graehl et al., 2008). Weighted variants of both of families of formalisms have been used for machine translation (Graehl et al., 2008; Chiang, 2007), where one tree represents a parse of a sentence in one language and the other a parse in the other language. Synchronous grammars and tree transducers are also useful as models of the syntax-semantics interface; here one tree represents the syntactic analysis of a sentence and the other the semantic analysis (Shieber and Schabes, 1990; Nesson and Shieber, 2006).

When such a variety of formalisms are available, it is useful to take a step back and look for a generalized model that explains the precise formal relationship between them. There is a long tradition of such research on monolingual grammar formalisms, where e.g. linear context-free rewriting systems (LCFRS, Vijay-Shanker et al. (1987)) generalize various mildly context-sensitive formalisms. However, few such results exist for synchronous formalisms. A notable exception is the work by Shieber (2004), who unified synchronous tree-adjoining grammars with tree transducers.

In this paper, we make two contributions. First, we provide a formal framework – *interpreted regular tree grammars* – which generalizes both synchronous grammars, tree transducers, and LCFRS-style monolingual grammars. A grammar of this formalism consists of a regular tree grammar (RTG, Comon et al. (2007)) defining a language of derivation trees and an arbitrary number of *interpretations* which map these trees into objects of arbitrary algebras. This allows us to capture a wide variety of (synchronous and monolingual) grammar formalisms. We can also model heterogeneous synchronous languages, which relate e.g. trees with strings; this is necessary for applications in machine translation (Graehl et al., 2008) and in parsing strings with synchronous tree grammars.

Second, we also provide parsing and decoding algorithms for our framework. The key concept that we introduce is that of a *regularly decomposable algebra*, where the set of all terms that evaluate to a given object form a regular tree language. Once an algorithm that computes a compact representation of this language is known, parsing algorithms follow from a generic construction. All important algebras in natural language processing that we are aware of – in particular the standard algebras of strings and trees – are regularly decomposable.

In summary, we obtain a formalism that pulls together much existing research under a common formal framework, and makes it possible to obtain parsers for existing and new formalisms in a modular, universal fashion.

Plan of the paper. The paper is structured as follows. We start by laying the formal foundations in Section 2. We then introduce the framework of interpreted RTGs and illustrate it with some simple examples in Section 3. The generic parsing and decoding algorithms are described in Section 4. Section 5 discusses the role of binarization in our framework. Section 6 shows how interpreted RTGs can be applied to existing grammar formalisms.

Proceedings of the 12th International Conference on Parsing Technologies, pages 2–13,
October 5-7, 2011, Dublin City University.

2 Formal Foundations

For $n \geq 0$, we define $[n] = \{\, i \mid 1 \leq i \leq n \,\}$.

A *signature* is a finite set Σ of function symbols f, each of which has been assigned a non-negative integer called its *rank*. Given a signature Σ, we can define a *(finite constructor) tree* over Σ as a finite tree whose nodes are labeled with symbols from Σ such that a node with a label of rank n has exactly n children. We write T_Σ for the set of all trees over Σ. Trees can be written as *terms*; $f(t_1, \ldots, t_n)$ stands for the tree with root label f and subtrees $t_1, \ldots, t_n$. The *nodes* of a tree can be identified by paths $\pi \in \mathbb{N}^*$ from the root: The root has address ϵ, and the i-th child of the node at path π has the address πi. We write $t(\pi)$ for the symbol at path π in the tree t.

A *Σ-algebra* $\mathcal{A}$ consists of a non-empty set A called the *domain* and, for each symbol $f \in \Sigma$ with rank n, a total function $f^{\mathcal{A}} : A^n \to A$, the *operation* associated with f. We can *evaluate* a term $t \in T_\Sigma$ to an object $[\![t]\!]_{\mathcal{A}} \in A$ by executing the operations:

$$[\![f(t_1, \ldots, t_n)]\!]_{\mathcal{A}} = f^{\mathcal{A}}([\![t_1]\!]_{\mathcal{A}}, \ldots, [\![t_n]\!]_{\mathcal{A}})\,.$$

Sets of trees can be specified by *regular tree grammars (RTGs)* (Gécseg and Steinby, 1997; Comon et al., 2007). Formally, such a grammar is a structure $\mathcal{G} = (N, \Sigma, P, S)$, where N is a signature of nonterminal symbols, all of which are taken to have rank 0, Σ is a signature of terminal symbols, $S \in N$ is a distinguished start symbol, and P is a finite set of productions of the form $B \to t$, where B is a nonterminal symbol, and $t \in T_{N \cup \Sigma}$. The productions of a regular tree grammar are used as rewriting rules on terms. More specifically, the *derivation relation* of $\mathcal{G}$ is defined as follows. Let $t_1, t_2 \in T_{N \cup \Sigma}$ be terms. Then $\mathcal{G}$ derives t_2 from t_1 in one step, denoted by $t_1 \Rightarrow_{\mathcal{G}} t_2$, if there exists a production of the form $B \to t$ and t_2 can be obtained by replacing an occurrence of B in t_1 by t. The *(regular) language* $L(\mathcal{G})$ generated by $\mathcal{G}$ is the set of all terms $t \in T_\Sigma$ that can be derived, in zero or more steps, from the term S.

A *(tree) homomorphism* is a total function $h\colon T_\Sigma \to T_\Delta$ which expands symbols of Σ into trees over Δ while following the structure of the input tree. Formally, h is specified by pairs $(f, h(f))$, where $f \in \Sigma$ is a symbol with some rank n, and $h(f) \in T_{\Delta \cup \{x_1, \ldots, x_n\}}$ is a term with variables. Given $t \in T_\Sigma$, the value of t under h is defined as

$$h(f(t_1, \ldots, t_n)) = h(f)\{\, h(t_i)/x_i \mid i \in [n] \,\}\,,$$

where $\{\, t_i/x_i \mid i \in [n] \,\}$ represents the substitution that replaces all occurrences of x_i with the respective t_i. A homomorphism is called *linear* if every term $h(f)$ contains each variable at most once; and a *delabeling* if every term $h(f)$ is of the form $g(x_{\pi(1)}, \ldots, x_{\pi(n)})$ where n is the rank of f and π a permutation of $\{1, \ldots, n\}$.

3 Interpreted Regular Tree Grammars

We will now present a generalized framework for synchronous and monolingual grammars in terms of regular tree grammars, tree homomorphisms, and algebras. We will illustrate the framework with two simple examples here, but many other grammar formalisms can be seen as special cases too, as we will show in Section 6.

3.1 An Introductory Example

The derivation process of context-free grammar is usually seen as a string-rewriting process in which nonterminals are successively replaced by the right-hand sides of production rules. The actual parse tree is explained as a post-hoc description of the rules that were applied in the derivation.

However, we can alternatively view this as a two-step process which first computes a derivation tree and then interprets it as a string. Say we have the CNF grammar G in Fig. 2, and we want to derive the string $w =$ "Sue watches the man with the telescope". In the first step, we use G to generate a *derivation tree* like the one in Fig. 2a. The nodes of this tree are labeled with names of the production rules in G; nodes with labels r_7 and r_3 are licensed by G to be the two children of r_1 because r_1 has the two nonterminals NP and VP in its right-hand side, and the left-hand sides of r_7 and r_3 are NP and VP, respectively. In a second step, we can then *interpret* the derivation tree into w by interpreting each leaf labeled with a terminal production (say, r_7) as the string on its right-hand side ("Sue"), and each internal node as a string concatenation operation which arranges the string yields of its subtrees in the order given by the right-hand side of the production rule.

This view differs from the traditional perspective on context-free grammars in that it makes the derivation tree the primary participant in the derivation process. The string is only one particular interpretation of the derivation tree, and instead of a string we could also have interpreted it as some other kind of object. For instance, if we had inter-

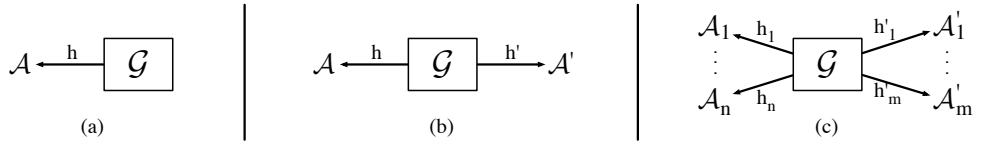

Figure 1: Interpreted tree grammars: (a) monolingual; (b) synchronous; (c) multiple "inputs" and "outputs".

preted r_7 as the *tree* N(Sue) and a rule like r_1 as a *tree operation* which takes the tree yields of its two subtrees and inserts them as the children of a root with label S, etc., the interpretation of a derivation tree would now be a tree; namely, an ordinary parse tree of G. We could even interpret the same derivation tree simultaneously as a string and as a tree using two different interpretation functions.

3.2 Interpreted Regular Tree Grammars

While this view is unnecessarily complex for explaining context-free grammars alone, the separation into two different generative processes (first derivation, then interpretation) is widely applicable. In particular, the derivation part can be independent of whether w is a string or some other algebra of objects. We formalize our view as follows. First, we need an interpretative component that maps derivation trees to terms of the relevant object algebra:

Definition 1 Let Σ be a signature. A *Σ-interpretation* is a pair $\mathcal{I} = (h, \mathcal{A})$, where $\mathcal{A}$ is a Δ-algebra, and h: $T_\Sigma \to T_\Delta$ is a homomorphism. □

We then capture the derivation process with a single regular tree grammar, and connect it with (potentially multiple) interpretations as follows:

Definition 2 An *interpreted regular tree grammar (IRTG)* is a structure $\mathbb{G} = (\mathcal{G}, \mathcal{I}_1, \ldots, \mathcal{I}_n)$, $n \geq 1$, where $\mathcal{G}$ is a regular tree grammar with terminal alphabet Σ, and the $\mathcal{I}_i$ are Σ-interpretations. □

Let $\mathcal{I}_i = (h_i, \mathcal{A}_i)$ be the ith interpretation. If we apply the homomorphism h_i to any tree $t \in L(\mathcal{G})$, we obtain a term $h_i(t)$, which we can evaluate to an object of $\mathcal{A}_i$. Based on this, we define the *language* generated by $\mathbb{G}$ as follows. We write $[\![t]\!]_{\mathcal{I}_i}$ as a shorthand for $[\![h_i(t)]\!]_{\mathcal{A}_i}$.

$$L(\mathbb{G}) = \{ \langle [\![t]\!]_{\mathcal{I}_1}, \ldots, [\![t]\!]_{\mathcal{I}_n} \rangle \mid t \in L(\mathcal{G}) \}$$

Given this notion, we can define an obvious *membership problem*: For a given tuple of objects $\vec{a} = \langle a_1, \ldots, a_n \rangle$, is $\vec{a} \in L(\mathbb{G})$? We can also define a *parsing task*: For every element $\vec{a} \in L(\mathbb{G})$, compute (some compact representation of) the set

$$\mathsf{parses}_{\mathbb{G}}(\vec{a}) = \{ t \in L(\mathcal{G}) \mid \forall i.\ [\![t]\!]_{\mathcal{I}_i} = a_i \}$$

We call the trees in this set the *derivation trees* of $\vec{a}$.

3.3 Monolingual Grammars

Let us use these definitions to make our example with context-free grammars as string-generating devices precise. This is a case with a single interpretation ($n = 1$), as illustrated in Fig. 1a.

We can adapt a standard construction (Goguen et al., 1977). Let G be a context-free grammar with nonterminals N, terminals T, and productions P. We start by defining a regular tree grammar $\mathcal{G}$. For a string $\alpha \in (N \cup T)^*$, let *nt*(α) denote the string of nonterminals in α, in the same order. We include into $\mathcal{G}$ all (and only) productions of the form $A \to p(A_1, \ldots, A_m)$, where $p = A \to \alpha$ is a production of G, and $A_1 \cdots A_m =$ *nt*(α). Note that by doing so, we view p as a symbol of rank $|$*nt*$(\alpha)|$. The nonterminals and the start symbol of $\mathcal{G}$ are as for G. We now interpret the trees generated by $\mathcal{G}$ over the *string algebra* over T, which we denote by T^*. The domain of this algebra is the set of all strings over T, and we have constants for the symbols in T and the empty string, as well as a single binary concatenation operation $\bullet$. As a last step, we use a homorphism *rb* to map each rule of G into a term over the signature of T^*: For each production p of the form above, *rb*(p) is the right-branching tree obtained from decomposing α into a series of concatenation operations, where the nonterminal A_i is replaced with the variable x_i. Thus we have constructed an IRTG grammar $\mathbb{G} = (\mathcal{G}, (rb, T^*))$. It can be shown that under this construction $L(\mathbb{G})$ is exactly $L(G)$, the string language of the original grammar.

Consider the context-free grammar in Fig. 2. The RTG $\mathcal{G}$ contains production rules such as $\mathrm{S} \to r_1(\mathrm{NP}, \mathrm{VP})$; it generates an infinite language of trees, including the derivation trees shown in Fig. 2a and 2b. These trees can now be interpreted using *rb* with *rb*$(r_1) = x_1 \bullet x_2$,[1] *rb*$(r_7) =$ *Sue*, etc. This maps the tree in Fig. 2a to the term

Sue$\bullet$(*watches*$\bullet$(*the*$\bullet$(*man*$\bullet$(*with*$\bullet$(*the*$\bullet$*telescope*)))))

over the signature of T^*, which evaluates in the algebra T^* to the string w mentioned earlier. Similarly, *rb* maps the tree in Fig. 2b to the term

[1] Here and below, we write $\bullet$ in infix notation.

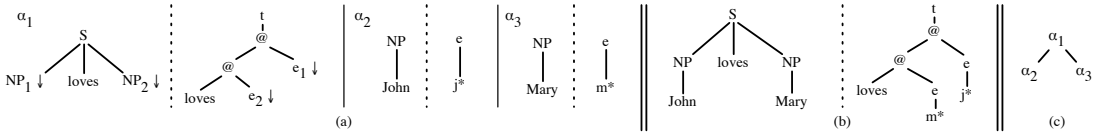

Figure 3: Synchronous TSG: (a) a lexicon consisting of three tree pairs; (b) a derived tree; (c) a derivation tree.

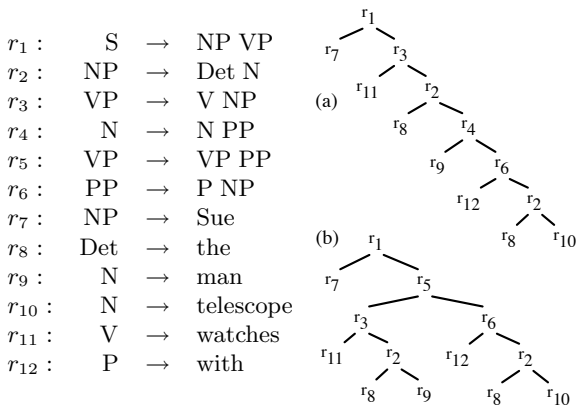

Figure 2: A CFG and two of its derivation trees.

Sue•((*watches*•(*the*•*man*))•(*with*•(*the*•*telescope*)))

This means that $L(\mathbb{G})$ is a set of strings which includes w. The tree language $L(\mathcal{G})$ contains further trees, which map to strings other than w. Therefore $L(\mathbb{G})$ includes other strings, but the trees in Fig. 2a and b are the only two derivation trees of w.

3.4 Synchronous Grammars

We can quite naturally represent grammars that describe binary relations between objects, i.e. synchronous grammars, as IRTGs with two interpretations ($n = 2$). We write these interpretations as $\mathcal{I}_L = (h_L, \mathcal{A}_L)$ ("left") and $\mathcal{I}_R = (h_R, \mathcal{A}_R)$ ("right"); see Fig. 1b.

Parsing with a synchronous grammar means to compute (a compact representation of) the set of all derivation trees for a given pair (a_L, a_R) from the set $\mathcal{A}_L \times \mathcal{A}_R$. This is precisely the parsing task that we defined in Section 3.2. A related task is *decoding*, in which we take the grammar $\mathbb{G} = (\mathcal{G}, \mathcal{I}_L, \mathcal{I}_R)$ as a translation device for mapping input objects $a_L \in \mathcal{A}_L$ to output objects $a_R \in \mathcal{A}_R$. We define the decoding task as the task of computing, for a given $a_L \in \mathcal{A}_L$, (a compact representation of) the following set, where $\mathbb{G}_L = (\mathcal{G}, \mathcal{I}_L)$:

$$\mathsf{decodes}_{\mathbb{G}}(a_L) = \{\, [\![t]\!]_{\mathcal{I}_R} \mid t \in \mathsf{parses}_{\mathbb{G}_L}(a_L) \,\}$$

To illustrate this with an example, consider the case of synchronous tree-substitution grammars (STSGs, Eisner (2003)). An STSG combines lexicon entries, as shown in Fig. 3a, into larger *derived trees* by replacing corresponding substitution nodes with trees from other lexicon entries. In the figure, we have marked the correspondence with numeric subscripts. The trees in Fig. 3a can be combined into the derived tree in Fig. 3b in two steps; this process is recorded in the derivation tree in Fig. 3c.

We capture an STSG G_S as an IRTG $\mathbb{G}$ by interpreting the (regular) language of derivation trees in appropriate *tree algebras*. The tree algebra T_Δ over some signature Δ consists of all trees over Δ; every symbol $f \in \Delta$ of rank m is interpreted as an m-place operation that returns the tree with root symbol f and its arguments as subtrees. To model STSG, we use the two tree algebras over all the symbols occurring in the left and right components of the lexicon entries, respectively. We can obtain an RTG $\mathcal{G}$ for the derivation trees using a standard construction (Schmitz and Le Roux, 2008; Shieber, 2004); its nonterminals are pairs $\langle A_L, A_R \rangle$ of nonterminals occurring in the left and right trees of G_S. To encode a lexicon entry α with root nonterminals A_L and A_R, left substitution nodes $A_L^1, \ldots, A_L^n$, and right substitution nodes $A_R^1, \ldots, A_R^n$, we add an RTG rule of the form

$$\langle A_L, A_R \rangle \to \alpha(\langle A_L^1, A_R^1 \rangle, \ldots, \langle A_L^n, A_R^n \rangle)\,.$$

We also let $h_L(\alpha)$ and $h_R(\alpha)$ be the left and right tree of α, with substitution nodes replaced by variables; h_L and h_R interpret derivation trees into derived trees in tree algebras T_Σ and T_Δ of appropriate (and possibly different) signatures. In the example, we obtain

$$\begin{aligned} \langle S, t \rangle &\to \alpha_1(\langle \mathrm{NP}, e \rangle, \langle \mathrm{NP}, e \rangle)\,, \\ h_L(\alpha_1) &= \mathrm{S}(x_1, \mathrm{loves}, x_2)\,, \qquad \text{and} \\ h_R(\alpha_1) &= t(@(@(\mathrm{loves}, x_2), x_1))\,. \end{aligned}$$

The variables reflect the corresponding substitution nodes. So if we let $\mathbb{G} = (\mathcal{G}, (h_L, T_\Sigma), (h_R, T_\Delta))$, $L(\mathbb{G})$ will be a language of pairs of derived trees, including the pair in Fig. 3b.

Parsing as defined above amounts to computing a common derivation tree for a given pair of derived trees; given only a left derived tree, the decoding

problem is to compute the corresponding right derived trees. However, in an application of STSG to machine translation or semantic construction, we are typically given a *string* as the left input and want to decode it to a right output tree. We can support this by left-interpreting the derivation trees directly as strings: We use the appropriate string algebra T^* (consisting of the symbols of Σ with arity zero) for $\mathcal{A}_L$, and map every lexicon entry to a term that concatenates the string yields of the elementary trees. In the example grammar, we can let $h'_L(\alpha_1) = ((x_1 \bullet \text{loves}) \bullet x_2)$, $h'_L(\alpha_2) = \text{John}$, and $h'_L(\alpha_3) = \text{Mary}$. With this local change, we obtain a new IRTG $\mathbb{G}' = (\mathcal{G}, (h'_L, T^*), (h_R, T_\Delta))$, whose language contains pairs of a (left) string and a (right) tree. One such pair has the string "John loves Mary" as the left and the right-hand tree in Fig. 3b as the right component. Therefore $\mathsf{decodes}$("John loves Mary"), i.e. the set of right derived trees that are consistent with the input string, contains the right-hand tree in Fig. 3b.

We conclude this section by remarking that decoding can be easily generalized to n input objects and m output objects, all of which can be taken from different algebras (see Fig. 1c).

4 Algorithms

In the previous section, we have taken a view on parsing and translation in which languages and translations are obtained as the interpretation of regular tree grammars. One advantage of this way of looking at things is that it is possible to define completely generic parsing algorithms by exploiting closure properties of regular tree languages.

4.1 Parsing

The fundamental problem that we must solve is to compute, for a given IRTG $\mathbb{G} = (\mathcal{G}, (h, \mathcal{A}))$ and object $a \in \mathcal{A}$, a regular tree grammar $\mathcal{G}_a$ such that $L(\mathcal{G}_a) = \mathsf{parses}_{\mathbb{G}}(a)$. A parser for IRTGs with multiple interpretations follows from this immediately. Assume that $\mathbb{G} = (\mathcal{G}, \mathcal{I}_1, \ldots, \mathcal{I}_n)$; then

$$\mathsf{parses}_{\mathbb{G}}(a_1, \ldots, a_n) = \bigcap_{i=1}^{n} \mathsf{parses}_{(\mathcal{G}, \mathcal{I}_i)}(a_i)\,.$$

Because regular tree languages are closed under intersection, we can parse the different a_i separately and then intersect all the $\mathcal{G}_{a_i}$.

The general idea of our parsing algorithm is as follows. Suppose we were able to compute the set $\mathsf{terms}_{\mathcal{A}}(a)$ of all possible terms t over $\mathcal{A}$ that evaluate to a. Then $\mathsf{parses}_{\mathbb{G}}(a)$ can be written as $h^{-1}(\mathsf{terms}_{\mathcal{A}}(a)) \cap L(\mathcal{G})$. Of course, $\mathsf{terms}_{\mathcal{A}}(a)$ may be a large or infinite set, so computing it in general algebras is infeasible. But now assume an algebra $\mathcal{A}$ in which $\mathsf{terms}_{\mathcal{A}}(a)$ is a regular tree language for every $a \in \mathcal{A}$, and in which we can compute, for each a, a regular tree grammar $D(a)$ with $L(D(a)) = \mathsf{terms}_{\mathcal{A}}(a)$. Since regular tree languages are effectively closed under both inverse homomorphisms and intersections (Comon et al., 2007), we obtain a parsing algorithm which first computes $D(a)$, and then $\mathcal{G}_a$ as the grammar for $h^{-1}(L(D(a))) \cap L(\mathcal{G})$.

Formally, this can be done for the following class of algebras.

Definition 3 A Σ-algebra $\mathcal{A}$ is called *regularly decomposable* if there is a computable function $D(\cdot)$ which maps every object $a \in \mathcal{A}$ to a regular tree grammar $D(a)$ such that $L(D(a)) = \mathsf{terms}_{\mathcal{A}}(a)$.$\square$

Consider the example of context-free grammars. We have shown in Section 3.3 how these can be seen as an IRTG with an interpretation into T^*. The string algebra T^* is regularly decomposable because the possible term representations of a string simply correspond to its bracketings: For a string $w = w_1 \cdots w_n$, the grammar $D(w)$ consists of a rule $A_{i-1,i} \to w_i$ for each $1 \leq i \leq n$, and a rule $A_{i,k} \to A_{i,j} \bullet A_{j,k}$ for all $0 \leq i < j < k \leq n$. In our example "Sue watches the man with the telescope" from Section 3.2, these are rules such as $A_{2,3} \to \text{the}$, $A_{3,4} \to \text{man}$, $A_{2,4} \to A_{2,3} \bullet A_{3,4}$, and so on. The grammar generates a tree language consisting of the 132 binary bracketings of the sentence, including the two mentioned in Section 3.3.

Tree algebras are an even simpler example of a regularly decomposable algebra. For a given tree $t \in T_\Sigma$, the grammar $D(t)$ consists of the rules $A_\pi \to f(A_{\pi 1}, \ldots, A_{\pi n})$ for all nodes π in t with label f. $D(t)$ generates a language that contains a single tree, namely t itself. Thus we can use the parsing algorithm to parse tree inputs (say, in the context of an STSG) just as easily as string inputs.

4.2 Computing Inverse Homomorphisms

The performance bottleneck of the parsing algorithm is the computation of the inverse homomorphisms. The input of this problem is h and $D(a)$; the task is to compute an RTG $\mathcal{H}'$ that uses terminal symbols from the signature Σ of $\mathcal{G}$ and the same nonterminals as $D(a)$, such that $h(L(\mathcal{H}')) = L(D(a))$. This problem is nontrivial

$$\frac{h(f)(\pi) = x_i \quad A \in N_{D(a)}}{[f, \pi, A, \{A/x_i\}]}\ (var)$$

$$\frac{\begin{array}{c} A \to g(A_1, \ldots, A_n) \text{ in } \mathcal{H} \quad h(f)(\pi) = g \\ [f, \pi 1, A_1, \sigma_1] \quad \cdots \quad [f, \pi n, A_n, \sigma_n] \\ \sigma = \text{merge}(\sigma_1, \ldots, \sigma_n) \neq \textbf{fail} \end{array}}{[f, \pi, A, \sigma]}\ (up)$$

Figure 4: Algorithm for computing $h^{-1}(\mathcal{H})$.

because h may not be a delabeling, so a term $h(f)$ may have to be parsed by multiple rule applications in $D(a)$ (see e.g. $h'_L(\alpha_1)$ in Section 3.4), and we cannot simply take the homomorphic pre-images of the production rules of $D(a)$. One approach (Comon et al., 2007) is to generate all possible production rules $A \to f(B_1, \ldots, B_m)$ out of terminals $f \in \Sigma$ and $D(a)$-nonterminals and check whether $A \Rightarrow^*_{D(a)} h(f(B_1, \ldots, B_m))$. Unfortunately, this algorithm blindly combines arbitrary tuples of nonterminals. For parsing with context-free grammars in Chomsky normal form, this approach leads to an $O(n^4)$ parsing algorithm.

The problem can be solved more efficiently by the algorithm in Fig. 4. This algorithm computes an RTG $\mathcal{H}'$ for $h^{-1}(L(\mathcal{H}))$, where $\mathcal{H}$ is an RTG in a normal form in which every rule contains a single terminal symbol; bringing a grammar into this form only leads to a linear size increase (Gécseg and Steinby, 1997). The algorithm derives items of the form $[f, \pi, A, \sigma]$, stating that $\mathcal{H}$ can generate the subtree of $h(f)\sigma$ at node π if it uses A as the start symbol; the substitution σ is responsible for replacing the variables in $h(f)$ by nonterminal symbols. It starts by guessing all possible instantiations of each variable in $h(f)$ (rule *var*). It then computes items bottom-up, deriving an item $[f, \pi, A, \sigma]$ if there is a rule in $\mathcal{H}$ that can combine the nonterminals derived for the children $\pi 1, \ldots, \pi n$ of π into A (rule *up*). The substitution σ is obtained by merging all mappings in the $\sigma_1, \ldots, \sigma_n$; if some σ_i, σ_j assign different nonterminals to the same variable, the rule fails.

Whenever the algorithm derives an item of the form $[f, \epsilon, A, \sigma]$, it has processed a complete tree $h(f)$, and we add a production $A \to f(\sigma(x_1), \ldots, \sigma(x_n))$ to $\mathcal{H}'$; for variables x_i on which σ is undefined, we let $\sigma(x_i) = \$$ for the special nonterminal $\$$. We also add rules to $\mathcal{H}'$ which generate any tree from T_Σ out of $\$$.

The complexity of this algorithm is bounded by the number of instances of the *up* rule (McAllester, 2002). For parsing with context-free grammars, *up* is applied to rules of the form $A_{l,r} \to A_{l,k} \bullet A_{k,r}$ of $D(w)$; the premises are $[f, \pi 1, A_{l,k}, \sigma_1]$ and $[f, \pi 2, A_{k,r}, \sigma_2]$ and the conclusion $[f, \pi, A_{l,r}, \sigma]$. The substitution σ defines a segmentation of the substring between positions l and r into $m - 1$ smaller substrings, where m is the number of variables in the domain of σ. So the instances of *up* are uniquely determined by at most $m + 1$ string positions, where m is the total number of variables in the tree $h(f)$; the parsing complexity is $O(n^{m+1})$. By our encoding of context-free grammars into IRTGs, m corresponds to the maximal number of nonterminals in the right-hand side of a production of the original grammar. In particular, the generic algorithm parses Chomsky normal form grammars (where $m = 2$) in cubic time, as expected.

4.3 Parse Charts

We will now illustrate the operation of the parsing algorithm with our example context-free grammar from Fig. 2 and our example sentence $w =$ "Sue watches the man with the telescope". We first compute $D(w)$, which generates all bracketings of the sentence. Next, we use the algorithm in Fig. 4 to compute a grammar $\mathcal{H}'$ for $h^{-1}(L(D(w)))$ for the language of all derivation trees that are mapped by h to a term evaluating to w. $\mathcal{H}'$ contains rules such as $A_{2,4} \to r_2(A_{2,3}, A_{3,4})$, $A_{3,5} \to r_2(A_{3,4}, A_{4,5})$, and $A_{3,4} \to \text{man}$. That is, $\mathcal{H}'$ uses terminal symbols from Σ, but the nonterminals from $D(w)$. Finally, we intersect $\mathcal{H}'$ with $\mathcal{G}$ to retain only derivation trees that are grammatical according to $\mathcal{G}$. We obtain a grammar $\mathcal{G}_w$ for $\mathsf{parses}(w)$, which is shown in Fig. 5 (we have left out unreachable and unproductive rules). The nonterminals of $\mathcal{G}_w$ are pairs of the form $(N, A_{i,k})$, i.e. nonterminals of $\mathcal{G}$ and $\mathcal{H}'$; we abbreviate these pairs as $N_{i,k}$. Note that $L(\mathcal{G}_w)$ consists of exactly two trees, the derivation trees shown in Fig. 2.

There is a clear parallel between the RTG in Fig. 5 and a parse chart of the CKY parser for the same input. The RTG describes how to build larger parse items from smaller ones, and provides exactly the same kind of structure sharing for ambiguous sentences that the CKY chart would. For all intents and purposes, the RTG $\mathcal{G}_w$ *is* a parse chart. When we parse grammars of other formalisms, such as STSG, the nonterminals of $\mathcal{G}_a$ generally record non-

terminals of $\mathcal{G}$ and positions in the input objects, as encoded in the nonterminals of $D(a_1), \ldots, D(a_n)$; the spans $[i, k]$ occurring in CKY parse items simply happen to be the nonterminals of the $D(a)$ for the string algebra.

In fact, we maintain that the fundamental purpose of a chart is to act as a device for generating the set of derivation trees for an input. This tree-generating nature of parse charts is made explicit by modeling them directly as RTGs; the well-known view of parse charts as context-free grammars (Billot and Lang, 1989) captures the same intuition, but abuses context-free grammars (which are primarily string-generating devices) as tree description formalisms. One difference between the two views is that regular tree languages are closed under intersection, which means that parse charts that are modeled as RTGs can be easily restricted by external constraints (see Koller and Thater (2010) for a related approach), whereas this is hard in the context-free view.

4.4 Decoding

We conclude this section by explaining how to solve the decoding problem. Suppose that in the scenario of Fig. 1c, we have obtained a parse chart $\mathcal{G}_{\vec{a}}$ for a tuple $\vec{a} = \langle a_1, \ldots, a_n \rangle$ of inputs, if necessary by intersecting the individual parse charts $\mathcal{G}_{a_i}$. Decoding means that we want to compute RTGs for the languages $h'_j(L(\mathcal{G}_{\vec{a}}))$ where $j \in [m]$. The actual output objects can be obtained from these languages of terms by evaluating the terms.

In the case where the homomorphisms h'_j are linear, we can once again exploit closure properties: Regular tree languages are closed under the application of linear homomorphisms (Comon et al., 2007), and therefore we can apply a standard algorithm to compute the output RTGs from the parse chart. In the case of non-linear homomorphisms, the output languages are not necessarily regular, so decoding exceeds the expressive capacity of our framework. However, linear output homomorphisms are frequent in practice; see e.g. the analysis of synchronous grammar formalisms in (Shieber, 2004; Shieber, 2006). Some of the workload of a non-linear homomorphism may also be carried by the output algebra, whose operations may copy or delete material freely (as long as the algebra remains regularly decomposable). Notice that non-linear *input* homomorphisms are covered by the algorithm in Fig. 4.

$$
\begin{array}{rcl@{\qquad}rcl}
\mathrm{S}_{0,7} & \to & r_1(\mathrm{NP}_{0,1}, \mathrm{VP}_{1,7}) & \mathrm{NP}_{5,7} & \to & r_2(\mathrm{Det}_{5,6}, \mathrm{N}_{6,7}) \\
\mathrm{VP}_{1,7} & \to & r_3(\mathrm{V}_{1,2}, \mathrm{NP}_{2,7}) & \mathrm{NP}_{0,1} & \to & r_7 \\
\mathrm{VP}_{1,7} & \to & r_5(\mathrm{VP}_{1,4}, \mathrm{PP}_{4,7}) & \mathrm{V}_{1,2} & \to & r_{11} \\
\mathrm{NP}_{2,7} & \to & r_2(\mathrm{Det}_{2,3}, \mathrm{N}_{3,7}) & \mathrm{Det}_{2,3} & \to & r_8 \\
\mathrm{N}_{3,7} & \to & r_4(\mathrm{N}_{3,4}, \mathrm{PP}_{4,7}) & \mathrm{N}_{3,4} & \to & r_9 \\
\mathrm{VP}_{1,4} & \to & r_3(\mathrm{V}_{1,2}, \mathrm{NP}_{2,4}) & \mathrm{P}_{4,5} & \to & r_{12} \\
\mathrm{NP}_{2,4} & \to & r_2(\mathrm{Det}_{2,3}, \mathrm{N}_{3,4}) & \mathrm{Det}_{5,6} & \to & r_8 \\
\mathrm{PP}_{4,7} & \to & r_6(\mathrm{P}_{4,5}, \mathrm{NP}_{5,7}) & \mathrm{N}_{6,7} & \to & r_{10}
\end{array}
$$

Figure 5: A "parse chart" RTG for the sentence "Sue watches the man with the telescope".

5 Membership and Binarization

A *binarization* transforms an m-ary grammar into an equivalent binary one. Binarization is essential for achieving efficient recognition algorithms, in particular the usual $O(n^3)$ time algorithms for context-free grammars, and $O(n^6)$ time recognition of synchronous context-free grammars. In this section, we discuss binarization in terms of IRTGs.

5.1 Context-Free Grammars

We start with a discussion of parsing context-free grammars. Let $\mathbb{G} = (\mathcal{G}, (rb, T^*))$ be a CFG as we defined it in Section 3.3. We have shown in Section 4.2 that our generic parsing algorithm processes a sentence $w = w_1 \ldots w_n$ in time $O(n^{m+1})$, where m is the maximal number of nonterminal symbols in the right-hand side of the grammar. To achieve the familiar cubic time complexity, an algorithm needs to convert the grammar into a binary form, either explicitly (by converting it to Chomsky normal form) or implicitly (as in the case of the Earley algorithm, which binarizes on the fly).

Strictly speaking, no algorithm that works on the binarized grammar is a parsing algorithm in the sense of 'parsing' as we defined it above. Under our view of things, such an algorithm does not compute the set $\mathsf{parses}_{\mathbb{G}}(w)$ of derivation trees of w according to the grammar $\mathbb{G}$, but according to a second, binarized grammar $\mathbb{G}' = (\mathcal{G}', (rb, T^*))$. The binarization then takes the form of a function *bin* that transforms terms over the signature of the RTG $\mathcal{G}$ into terms over the binary signature of the RTG $\mathcal{G}'$. For a binarized grammar, we have $m = 2$, and so the parsing complexity is $O(n^3)$ plus whatever time it takes to compute $\mathbb{G}'$ from $\mathbb{G}$ and w. Standard binarization techniques of context-free grammars are linear in the size of the grammar.

Although binarization does not simplify parsing in the sense of this paper, it *does* simplify the membership problem of $\mathbb{G}$: Given a string

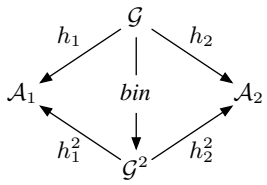

Figure 6: Binarization.

$w \in T^*$, is there some derivation tree $t \in \mathcal{G}$ such that $[\![h(t)]\!] = w$? Because $L(\mathbb{G}) = L(\mathbb{G}')$, this question can be decided by testing the emptiness of $\mathsf{parses}_{\mathbb{G}'}(w)$, without the need to compute $\mathsf{parses}_{\mathbb{G}}(w)$. Furthermore, the set $\mathsf{parses}_{\mathbb{G}'}(w)$ is useful not only for deciding membership in $L(\mathbb{G})$, but also for computing other quantities, such as inside probabilities of derivation trees of $\mathbb{G}$.

5.2 Synchronous Context-Free Grammars

Synchronous context-free grammars can be represented as IRTGs along the same lines as STSG grammars in Section 3.4. The resulting grammar $\mathbb{G} = (\mathcal{G}, (h_1, T_1^*), (h_2, T_2^*))$ consists of two 'context-free' interpretations of the RTG $\mathcal{G}$ into string algebras T_1^* and T_2^*; as above, the synchronization is ensured by requiring that related strings in T_1^* and T_2^* are interpretations of the same derivation tree $t \in L(\mathcal{G})$. As above, we can parse synchronously by parsing separately for the two interpretations and intersecting the results. This yields a parsing complexity for SCFG parsing of $O(n_1^{m+1} \cdot n_2^{m+1})$, where n_1 and n_2 are the lengths of the input strings and m is the rank of the RTG $\mathcal{G}$. Unlike in the monolingual case, this is now consistent with the result that the membership problem of SCFGs is NP-complete (Satta and Peserico, 2005).

The reason for the intractability of SCFG parsing is that SCFGs, in general, cannot be binarized. However, Huang et al. (2009) define the class of *binarizable* SCFGs, which can be brought into a weakly equivalent normal form in which all production rules are binary and the membership problem can be solved in time $O(n_1^3 \cdot n_2^3)$. The key property of binarizable SCFGs, in our terms, is that if r is any production rule pair of the SCFG, $h_1(r)$ and $h_2(r)$ can be chosen in such a way that they can be transformed into each other by locally swapping the subterms of a node. For instance, an SCFG rule pair $\langle A \to A_1\ A_2\ A_3\ A_4, B \to B_3\ B_4\ B_2\ B_1 \rangle$ can be represented by $h_1(r) = (x_1 \bullet x_2) \bullet (x_3 \bullet x_4)$ and $h_2(r) = (x_4 \bullet x_3) \bullet (x_1 \bullet x_2)$, and $h_2(r)$ can be obtained from $h_1(r)$ by swapping the children of the nodes ϵ and 2. In such a situation, we can binarize the rule $\langle A, B \rangle \to r(\langle A_1, B_1 \rangle, \ldots, \langle A_4, B_4 \rangle)$ in a way that follows the structure of $h_1(r)$, e.g.

$$\langle A, B \rangle \to r_1^\epsilon(\langle A_1^r, B_1^r \rangle, \langle A_2^r, B_2^r \rangle)$$
$$\langle A_1^r, B_1^r \rangle \to r_1^1(\langle A_1, B_1 \rangle, \langle A_2, B_2 \rangle)$$
$$\langle A_2^r, B_2^r \rangle \to r_1^2(\langle A_3, B_3 \rangle, \langle A_4, B_4 \rangle)$$

We can then encode the local rotations in two new left and right homomorphisms h_1^2 and h_2^2, i.e. $h_1^2(r_1^\epsilon) = h_1^2(r_1^1) = h_1^2(r_1^2) = h_2^2(r_1^2) = x_1 \bullet x_2$, $h_2^2(r_1^\epsilon) = h_2^2(r_1^1) = x_2 \bullet x_1$. To determine membership of some (a_1, a_2) in $L(\mathbb{G})$, we compute the pre-images of $D(a_1)$ and $D(a_2)$ under h_1^2 and h_1^2 and intersect them with the binarized version, $\mathcal{G}^2$, of $\mathcal{G}$. This can be done in time $O(n_1^3 \cdot n_2^3)$.

5.3 A Generalized View on Binarization

The common theme of both examples we have just discussed is that binarization, when it is available, allows us to solve the membership problem in less time than the parsing problem. A lower bound for the membership problem of a tuple $\langle a_1, \ldots, a_n \rangle$ of inputs is $O(|D(a_1)| \cdots |D(a_n)|)$, because the pre-images of the $D(a_i)$ grammars are at least as big as the grammars themselves, and the intersection algorithm computes the product of these. This means that a membership algorithm is optimal if it achieves this runtime.

As we have illustrated above, the parsing algorithm from Section 4 is not optimal for monolingual context-free membership, because the RTG $\mathcal{G}$ has a higher rank than $D(a)$, and therefore permits too many combinations of input spans into rules. The binarization constructions above indicate one way towards a generic optimal membership algorithm. Assume that we have algebras $\mathcal{A}_1, \ldots, \mathcal{A}_n$, all of which over signatures with maximum rank k, and an IRTG $\mathbb{G} = (\mathcal{G}, (h_1, \mathcal{A}_1), \ldots, (h_n, \mathcal{A}_n))$, where $\mathcal{G}$ is an RTG over some signature Σ. Assume further that we have some other signature Δ, of maximum rank k, and a homomorphism $\mathit{bin} : T_\Sigma \to T_\Delta$. We can obtain a RTG $\mathcal{G}^2$ with $L(\mathcal{G}^2) = \mathit{bin}(L(\mathcal{G}))$ as in the SCFG example above. Now assume that there are delabelings $h_i^2 : T_\Delta \to T_{\mathcal{A}_i}$ such that $h_i^2(L(\mathcal{G}^2)) = h_i(L(\mathcal{G}))$ for all $i \in [n]$ (see Fig. 6). Then we can decide membership of a tuple $\langle a_1, \ldots, a_n \rangle$ by intersecting $\mathcal{G}^2$ with all the $(h_i^2)^{-1}(L(D(a_i)))$. Because the h_i^2 are delabelings, computing the pre-images can be done in linear time; therefore this membership algorithm is optimal. Notice that if the result of

the intersection is the RTG $\mathcal{H}$, then we can obtain $\mathsf{parses}(a_1, \ldots, a_n) = bin^{-1}(L(\mathcal{H}))$; this is where the exponential blowup can happen.

The constructions in Sections 5.1 and 5.2 are both special cases of this generalized approach, which however also maintains a clear connection to the strong generative capacity. It is not obvious to us that it is necessary that the homomorphisms h_i^2 must be delabelings for the membership algorithm to be optimal. Exploring this landscape, which ties in with the very active current research on binarization, is an interesting direction for future research.

6 Discussion and Related Work

We conclude this paper by discussing how a number of different grammar formalisms from the literature relate to IRTGs, and use this discussion to highlight a number of features of our framework.

6.1 Tree-Adjoining Grammars

We have sketched in Section 3.4 how we can capture tree-substitution grammars by assuming an RTG for the language of derivation trees and a homomorphism into the tree algebra which spells out the derived trees; or alternatively, a homomorphism into the string algebra which computes the string yield. This construction can be generalized to tree-adjoining grammars (Joshi and Schabes, 1997).

Assume first that we are only interested in the string language of the TAG grammar. Unlike in TSG, the string yield of a derivation tree in TAG may be discontinuous. We can model this with an algebra whose elements are strings and pairs of strings, along with a number of different concatenation operators that represent possible ways in which these elements can be combined. (These are a subset of the operations considered by Gómez-Rodríguez et al. (2010).) We can then specify a homomorphism, essentially the binarization procedure that Earley-like TAG parsers do on the fly, that maps derivation trees into terms over this algebra. The TAG string algebra is regularly decomposable, and $D(a)$ can be computed in time $O(n^6)$.

Now consider the case of mapping derivation trees into derived trees. This cannot easily be done by a homomorphic interpretation in an ordinary tree algebra. One way to deal with this, which is taken by Shieber (2006), is to replace homomorphisms by a more complex class of tree translation functions called *embedded pushdown tree transducers*. A second approach is to interpret homomorphically into a more powerful algebra. This approach is taken by Maletti (2010), who uses an ordinary tree homomorphism to map a derivation tree t into a tree t' of 'building instructions' for a derived tree, and then applies a function $\cdot^E$ to execute these building instructions and build the TAG derived tree. Maletti's approach fits nicely into our framework if we assume an algebra in which the building instruction symbols are interpreted according to $\cdot^E$.

Synchronous tree-adjoining grammars (Shieber and Schabes, 1990) can be modeled simply as an RTG with two separate TAG interpretations. We can separately choose to interpret each side as trees or strings, as described in Section 3.4.

6.2 Weighted Tree Transducers

One influential approach to statistical syntax-based machine translation is to use *weighted transducers* to map parse trees for an input language to parse trees or strings of the output language (Graehl et al., 2008). Bottom-up tree transducers can be modeled in terms of *bimorphisms*, i.e. triples $(h_L, \mathcal{G}, h_R)$ of an RTG $\mathcal{G}$ and two tree homomorphisms h_L and h_R that map a derivation $t \in L(\mathcal{G})$ into the input tree $h_L(t)$ and the output tree $h_R(t)$ of the transducer (Arnold and Dauchet, 1982). Thus bottom-up transducers fit into the view of Fig. 1b. Although Graehl et al. use extended top-down transducers and not bottom-up transducers, a first inspection of their transducers leads us to believe that nothing hinges on this specific choice for their application. The exact situation bears further investigation.

Graehl et al.'s transducers further differ from the setup we have presented above in that they are *weighted*, i.e. each derivation step is associated with a numeric weight (e.g., a probability), and we can ask for the optimum derivation for a given input. Our framework can be straightforwardly extended to cover this case by assuming that the RTG $\mathcal{G}$ is a weighted RTG (wRTG, Knight and Graehl (2005)). The parsing algorithm from Section 4.1 generalizes to an algorithm for computing a weighted chart RTG, from which the best derivation can be extracted efficiently (Knight and Graehl, 2005). Similarly, the decoding algorithm from Section 4.4 can be used to compute a weighted RTG for the output terms, and an algorithm for EM training can be defined directly on the weighted charts. In general, every grammar formalism that can be captured as an IRTG has a canonical weighted variant in this way. As probabilistic grammar formalisms,

these assume that all RTG rule applications are statistically independent. That is, the canonical probabilistic version of context-free grammars is PCFG, and the canonical probabilistic version of tree-adjoining grammar is PTAG (Resnik, 1992).

A final point is that Graehl et al. invest considerable effort into defining different versions of their transducer training algorithms for the tree-to-tree and tree-to-string translation cases. The core of their paper, in our terms, is to define synchronous parsing algorithms to compute an RTG of derivation trees for (tree, tree) and (tree, string) input pairs. In their setup, these two cases are formally completely different objects, and they define two separate algorithms for these problems. Our approach is more modular: The training and parsing algorithms can be fully generic, and all that needs to be changed to switch between tree-to-tree and tree-to-string is to replace the algebra and homomorphism on one side, as in Section 3.4. In fact, we are not limited to interpreting derivation trees into strings or trees; by interpreting into the appropriate algebras, we can also describe languages of graphs (Eaton et al., 2007), pictures (Drewes, 2006), 3D models (Bokeloh et al., 2010), and other objects with a suitable algebraic structure.

6.3 Generalized Context-Free Grammars

Finally, the view we advocate here embraces a tradition of grammar formalisms going back to generalized context-free grammar (GCFG, Pollard (1984)), which follows itself research in theoretical computer science (Mezei and Wright, 1967; Goguen et al., 1977). A GCFG grammar can be seen as an RTG over a signature Σ whose trees are evaluated as terms of some Σ-algebra $\mathcal{A}$. This is a special case of an IRTG, in which the homomorphism is simply the identical function on T_Σ, and the algebra is $\mathcal{A}$. In fact, we could have equivalently defined an IRTG as an RTG whose trees are interpreted in multiple Σ-algebras; the mediating homomorphisms do not add expressive power. We go beyond GCFG in three ways. First, the fact that we map the trees described by the RTG into terms of other algebras using different homomorphisms means that we can choose the signatures of these algebras and the RTG freely; in particular, we can reuse common algebras such as T^* for many different RTGs and homomorphisms. This is especially important in relation to the second advance, which is that we offer a generic parsing algorithm for arbitrary regularly decomposable algebras; because the algebras and RTGs are modular, we can reuse algorithms for computing $D(a)$ even when we change the homomorphism. Finally, we offer a more transparent view on synchronous grammars, which separates the different dimensions clearly.

An important special case of GCFG is that of *linear context-free rewrite systems* (LCFRS, Vijay-Shanker et al. (1987)). LCFRSs are essentially GCFGs with a "yield" homomorphism that maps objects of $\mathcal{A}$ to strings or tuples of strings. Therefore every grammar formalism that can be seen as an LCFRS, including certain dependency grammar formalisms (Kuhlmann, 2010), can be phrased as string-generating IRTGs. One particular advantage that our framework has over LCFRS is that we do not need to impose a bound on the length of the string tuples. This makes it possible to model formalisms such as combinatory categorial grammar (Steedman, 2001), which may be arbitrarily discontinuous (Koller and Kuhlmann, 2009).

7 Conclusion

In this paper, we have defined *interpreted RTGs*, a grammar formalism that generalizes over a wide range of existing formalisms, including various synchronous grammars, tree transducers, and LCFRS. We presented a generic parser for IRTGs; to apply it to a new type of IRTG, we merely need to define how to compute decomposition grammars $D(a)$ for input objects a. This makes it easy to define synchronous grammars that are heterogeneous both in the grammar formalism and in the objects that each of its dimensions describes.

The purpose of our paper was to pull together a variety of existing research and explain it in a new, unified light: We have not shown how to do something that was not possible before, only how to do it in a uniform way. Nonetheless, we expect that future work will benefit from the clarified formal setup we have proposed here. In particular, we believe that the view of parse charts as RTGs may lead to future algorithms which exploit their closure under intersection, e.g. to reduce syntactic ambiguity (Schuler, 2001).

Acknowledgments

We thank the reviewers for their helpful comments, as well as all the colleagues with whom we have discussed this work—especially Martin Kay for a discussion about the relationship to chart parsing.

References

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

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th ACL*.

M. Bokeloh, M. Wand, and H.-P. Seidel. 2010. A connection between partial symmetry and inverse procedural modeling. In *Proceedings of SIGGRAPH*.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`.

Frank Drewes. 2006. *Grammatical picture generation: a tree-based approach*. EATCS. Springer.

Nancy Eaton, Zoltán Füredi, Alexandr V. Kostochka, and Jozef Skokan. 2007. Tree representations of graphs. *European Journal of Combinatorics*, 28(4):1087–1098.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st ACL*.

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer.

Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. 1977. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95.

Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Proceedings of NAACL-HLT*.

Jonathan Graehl, Kevin Knight, and Jonathan May. 2008. Training tree transducers. *Computational Linguistics*, 34(4):391–427.

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

Aravind K. Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–123. Springer.

Kevin Knight and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Computational linguistics and intelligent text processing*, pages 1–24. Springer.

Alexander Koller and Marco Kuhlmann. 2009. Dependency trees and the strong generative capacity of CCG. In *Proceedings of the 12th EACL*.

Alexander Koller and Stefan Thater. 2010. Computing weakest readings. In *Proceedings of the 48th ACL*.

Marco Kuhlmann. 2010. *Dependency structures and lexicalized grammars: An algebraic approach*, volume 6270 of *Lecture Notes in Computer Science*. Springer.

P. M. Lewis and R. E. Stearns. 1968. Syntax-directed transduction. *Journal of the ACM*, 15(3):465–488.

Andreas Maletti. 2010. A tree transducer model for synchronous tree-adjoining grammars. In *Proceedings of the 48th ACL*.

David McAllester. 2002. On the complexity analysis of static analyses. *Journal of the Association for Computing Machinery*, 49(4):512–537.

Jorge E. Mezei and Jesse B. Wright. 1967. Algebraic automata and context-free sets. *Information and Control*, 11(1–2):3–29.

Rebecca Nesson and Stuart M. Shieber. 2006. Simpler TAG semantics through synchronization. In *Proceedings of the 11th Conference on Formal Grammar*.

Carl J. Pollard. 1984. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. Ph.D. thesis, Stanford University.

Owen Rambow and Giorgio Satta. 1996. Synchronous models of language. In *Proceedings of the 34th ACL*.

Philip Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of COLING*.

Giorgio Satta and Enoch Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*.

Sylvain Schmitz and Joseph Le Roux. 2008. Feature unification in tag derivation trees. In *Proceedings of the 9th TAG+ Workshop*.

William Schuler. 2001. Computational properties of environment-based disambiguation. In *Proceedings of the 39th ACL*.

Stuart Shieber and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the 13th COLING*.

Stuart Shieber. 1994. Restricting the weak generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–386.

Stuart M. Shieber. 2004. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 7)*.

Stuart M. Shieber. 2006. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *Proceedings of the 11th EACL*.

Mark Steedman. 2001. *The Syntactic Process*. MIT Press.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th ACL*.