Generic binarization for parsing and translation

Matthias Büchse
Technische Universität Dresden
matthias.buechse@tu-dresden.de

Alexander Koller
University of Potsdam
koller@ling.uni-potsdam.de

Heiko Vogler
Technische Universität Dresden
heiko.vogler@tu-dresden.de

Abstract

Binarization of grammars is crucial for improving the complexity and performance of parsing and translation. We present a versatile binarization algorithm that can be tailored to a number of grammar formalisms by simply varying a formal parameter. We apply our algorithm to binarizing tree-to-string transducers used in syntax-based machine translation.

1 Introduction

Binarization amounts to transforming a given grammar into an equivalent grammar of rank 2, i.e., with at most two nonterminals on any right-hand side. The ability to binarize grammars is crucial for efficient parsing, because for many grammar formalisms the parsing complexity depends exponentially on the rank of the grammar. It is also critically important for tractable statistical machine translation (SMT). Syntax-based SMT systems (Chiang, 2007; Graehl et al., 2008) typically use some type of *synchronous grammar* describing a binary translation relation between strings and/or trees, such as synchronous context-free grammars (SCFGs) (Lewis and Stearns, 1966; Chiang, 2007), synchronous tree-substitution grammars (Eisner, 2003), synchronous tree-adjoining grammars (Nesson et al., 2006; DeNeefe and Knight, 2009), and tree-to-string transducers (Yamada and Knight, 2001; Graehl et al., 2008). These grammars typically have a large number of rules, many of which have rank greater than two.

The classical approach to binarization, as known from the Chomsky normal form transformation for context-free grammars (CFGs), proceeds rule by rule. It replaces each rule of rank greater than 2 by an equivalent collection of rules of rank 2. All CFGs can be binarized in this way, which is why their recognition problem is cubic. In the case of linear context-free rewriting systems (LCFRSs, (Weir, 1988)) the rule-by-rule technique also applies to every grammar, as long as an increased fanout it permitted (Rambow and Satta, 1999).

There are also grammar formalisms for which the rule-by-rule technique is not complete. In the case of SCFGs, not every grammar has an equivalent representation of rank 2 in the first place (Aho and Ullman, 1969). Even when such a representation exists, it is not always possible to compute it rule by rule. Nevertheless, the rule-by-rule binarization algorithm of Huang et al. (2009) is very useful in practice.

In this paper, we offer a generic approach for transferring the rule-by-rule binarization technique to new grammar formalisms. At the core of our approach is a binarization algorithm that can be adapted to a new formalism by changing a parameter at runtime. Thus it only needs to be implemented once, and can then be reused for a variety of formalisms. More specifically, our algorithm requires the user to (i) encode the grammar formalism as a subclass of *interpreted regular tree grammars* (IRTGs, (Koller and Kuhlmann, 2011)) and (ii) supply a collection of *b-rules*, which represent equivalence of grammars syntactically. Our algorithm then replaces, in a given grammar, each rule of rank greater than 2 by an equivalent collection of rules of rank 2, if such a collection is licensed by the b-rules. We define completeness of b-rules in a way that ensures that if any equivalent collection of rules of rank 2 exists, the algorithm finds one. As a consequence, the algorithm binarizes every grammar that can be binarized rule by rule. Step (i) is possible for all the grammar formalisms mentioned above. We show Step (ii) for SCFGs and tree-to-string transducers.

We will use SCFGs as our running example throughout the paper. We will also apply the algorithm to tree-to-string transducers (Graehl et al., 2008; Galley et al., 2004), which describe relations between strings in one language and parse trees of another, which means that existing methods for binarizing SCFGs and LCFRSs cannot be directly applied to these systems. To our knowledge, our binarization algorithm is the first to binarize such transducers. We illustrate the effectiveness of our system by binarizing a large tree-to-string transducer for English-German SMT.

Plan of the paper. We start by defining IRTGs in Section 2. In Section 3, we define the general outline of our approach to rule-by-rule binarization for IRTGs, and then extend this to an efficient binarization algorithm based on b-rules in Section 4. In Section 5 we show how to use the algorithm to perform rule-by-rule binarization of SCFGs and tree-to-string transducers, and relate the results to existing work.

2 Interpreted regular tree grammars

Grammar formalisms employed in parsing and SMT, such as those mentioned in the introduction, differ in the the derived objects—e.g., strings, trees, and graphs—and the operations involved in the derivation—e.g., concatenation, substitution, and adjoining. Interpreted regular tree grammars (IRTGs) permit a uniform treatment of many of these formalisms. To this end, IRTGs combine two ideas, which we explain here.

Algebras IRTGs represent the objects and operations symbolically using terms; the object in question is obtained by interpreting each symbol in the term as a function. As an example, Table 1 shows terms for a string and a tree, together with the denoted object. In the string case, we describe complex strings as concatenation (con^2) of elementary symbols (e.g., a, b); in the tree case, we alternate the construction of a sequence of trees (con^2) with the construction of a single tree by placing a symbol (e.g., α, β, σ) on top of a (possibly empty) sequence of trees. Whenever a term contains variables, it does not denote an object, but rather a function. In the parlance of universal-algebra theory, we are employing *initial-algebra semantics* (Goguen et al., 1977).

An *alphabet* is a nonempty finite set. Throughout this paper, let $X = \{x_1, x_2, \dots\}$ be a set, whose elements we call *variables*. We let X_k denote the set $\{x_1, \dots, x_k\}$ for every $k \geq 0$. Let Σ be an alphabet and $V \subseteq X$. We write $T_\Sigma(V)$ for the set of all terms over Σ with variables V, i.e., the smallest set T such that (i) $V \subseteq T$ and (ii) for every $\sigma \in \Sigma$, $k \geq 0$, and $t_1, \dots, t_k \in T$, we have $\sigma(t_1, \dots, t_k) \in T$. Alternatively, we view $T_\Sigma(V)$ as the set of all (rooted, labeled, ordered, unranked) *trees over* Σ *and* V, and draw them as usual. By T_Σ we abbreviate $T_\Sigma(\emptyset)$. The *set* $C_\Sigma(V)$ *of contexts over* Σ *and* V is the set of all trees over Σ and V in which each variable in V occurs exactly once.

A *signature* is an alphabet Σ where each symbol is equipped with an *arity*. We write $\Sigma|_k$ for the subset of all k-ary symbols of Σ, and $\sigma|_k$ to denote $\sigma \in \Sigma|_k$. We denote the signature by Σ as well. A signature is *binary* if the arities do not exceed 2. Whenever we use $T_\Sigma(V)$ with a signature Σ, we assume that the trees are ranked, i.e., each node labeled by $\sigma \in \Sigma|_k$ has exactly k children.

Let Δ be a signature. A Δ-*algebra* $\mathcal{A}$ consists of a nonempty set A called the *domain* and, for each symbol $f \in \Delta$ with rank k, a total function $f^{\mathcal{A}} \colon A^k \to A$, the *operation* associated with f. We can *evaluate* any term t in $T_\Delta(X_k)$ in $\mathcal{A}$, to obtain a k-ary operation $t^{\mathcal{A}}$ over the domain. In particular, terms in T_Δ evaluate to elements of A. For instance, in the string algebra shown in Table 1, the term $\mathsf{con}^2(a, b)$ evaluates to ab, and the term $\mathsf{con}^2(\mathsf{con}^2(x_2, a), x_1)$ evaluates to a binary operation f such that, e.g., $f(b, c) = cab$.

Bimorphisms IRTGs separate the finite control (state behavior) of a derivation from its derived object (in its term representation; generational behavior); the former is captured by a regular tree language, while the latter is obtained by applying a tree homomorphism. This idea goes back to the *tree bimorphisms* of Arnold and Dauchet (1976).

Let Σ be a signature. A *regular tree grammar (RTG)* G *over* Σ is a triple (Q, q_0, R) where Q is a finite set (of *states*), $q_0 \in Q$, and R is a finite set of rules of the form $q \to \alpha(q_1, \dots, q_k)$, where $q \in Q$, $\alpha \in \Sigma|_k$ and $q, q_1, \dots, q_k \in Q$. We call α the *terminal symbol* and k the *rank* of the rule. Rules of rank greater than two are called *suprabinary*. For every $q \in Q$ we define the *language* $L^q(G)$ *derived from* q as the set $\{\alpha(t_1, \dots, t_k) \mid q \to \alpha(q_1, \dots, q_k) \in R, t_j \in L^{q_j}(G)\}$. If $q = q_0$, we drop the superscript and write $L(G)$ for the *tree language of* G. In the literature, there is a definition of RTG which also permits more than one terminal symbol per rule,

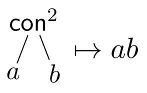
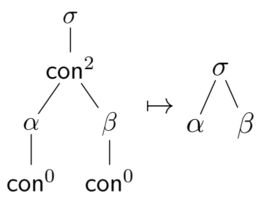
	strings over Γ	**trees over Γ**
example term and denoted object		
domain	Γ^*	T_Γ^* (set of sequences of trees)
signature Δ	$\{a\|_0 \mid a \in \Gamma\} \cup$ $\{\mathsf{con}^k\|_k \mid 0 \leq k \leq K, k \neq 1\}$	$\{\gamma\|_1 \mid \gamma \in \Gamma\} \cup$ $\{\mathsf{con}^k\|_k \mid 0 \leq k \leq K, k \neq 1\}$
operations	$a\colon () \mapsto a$ $\mathsf{con}^k\colon (x_1, \ldots, x_k) \mapsto x_1 \cdots x_k$	$\gamma\colon x_1 \mapsto \gamma(x_1)$ $\mathsf{con}^k\colon (x_1, \ldots, x_k) \mapsto x_1 \cdots x_k$

Table 1: Algebras for strings and trees, given an alphabet Γ and a maximum arity $K \in \mathbb{N}$.

or none. This does not increase the generative capacity (Brainerd, 1969).

A *(linear, nondeleting) tree homomorphism* is a mapping $h\colon T_\Sigma(X) \to T_\Delta(X)$ that satisfies the following condition: there is a mapping $g\colon \Sigma \to T_\Delta(X)$ such that (i) $g(\sigma) \in C_\Delta(X_k)$ for every $\sigma \in \Sigma|_k$, (ii) $h(\sigma(t_1, \ldots, t_k))$ is the tree obtained from $g(\sigma)$ by replacing the occurrence of x_j by $h(t_j)$, and (iii) $h(x_j) = x_j$. This extends the usual definition of linear and nondeleting homomorphisms (Gécseg and Steinby, 1997) to trees with variables. We abuse notation and write $h(\sigma)$ for $g(\sigma)$ for every $\sigma \in \Sigma$.

Let $n \geq 1$ and $\Delta_1, \ldots, \Delta_n$ be signatures. A *(generalized) bimorphism over* $(\Delta_1, \ldots, \Delta_n)$ is a tuple $\mathcal{B} = (G, h_1, \ldots, h_n)$ where G is an RTG over some signature Σ and h_i is a tree homomorphism from $T_\Sigma(X)$ into $T_{\Delta_i}(X)$. The *language* $L(\mathcal{B})$ *induced by* $\mathcal{B}$ is the tree relation $\{(h_1(t), \ldots, h_n(t)) \mid t \in L(G)\}$.

An IRTG is a bimorphism whose derived trees are viewed as terms over algebras; see Fig. 1. Formally, an *IRTG* $\mathbb{G}$ *over* $(\Delta_1, \ldots, \Delta_n)$ is a tuple $(\mathcal{B}, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ such that $\mathcal{B}$ is a bimorphism over $(\Delta_1, \ldots, \Delta_n)$ and $\mathcal{A}_i$ is a Δ_i-algebra. The *language* $L(\mathbb{G})$ *induced by* $\mathbb{G}$ is the relation $\{(t_1^{\mathcal{A}_1}, \ldots, t_n^{\mathcal{A}_n}) \mid (t_1, \ldots, t_n) \in L(\mathcal{B})\}$. We call the trees in $L(G)$ *derivation trees* and the terms in $L(\mathcal{B})$ *semantic terms*. We say that two IRTGs $\mathbb{G}$ and $\mathbb{G}'$ are *equivalent* if $L(\mathbb{G}) = L(\mathbb{G}')$. IRTGs were first defined in (Koller and Kuhlmann, 2011).

For example, Fig. 2 is an IRTG that encodes a synchronous context-free grammar (SCFG). It contains a bimorphism $\mathcal{B} = (G, h_1, h_2)$ consisting of an RTG G with four rules and homomorphisms h_1 and h_2 which map derivation trees to trees over the signature of the string algebra in Table 1. By evaluating these trees in the algebra, the symbols con^3 and con^4 are interpreted as concatenation, and we see that the first rule encodes the SCFG rule $A \to \langle BCD, DaBC \rangle$. Figure 3 shows a derivation tree with its two homomorphic images, which evaluate to the strings *bcd* and *dabc*.

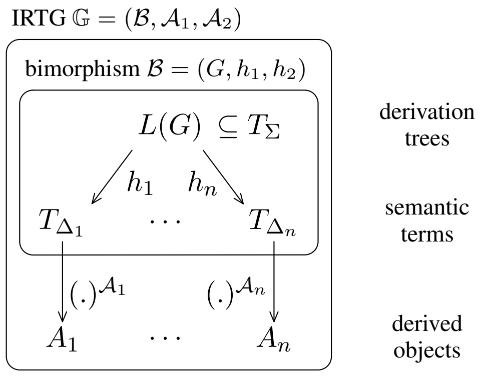

IRTG $\mathbb{G} = (\mathcal{B}, \mathcal{A}_1, \mathcal{A}_2)$
bimorphism $\mathcal{B} = (G, h_1, h_2)$
$L(G) \subseteq T_\Sigma$
h_1
h_n
T_{Δ_1} … T_{Δ_n}
$(.)^{\mathcal{A}_1}$
$(.)^{\mathcal{A}_n}$
$\mathcal{A}_1$ … $\mathcal{A}_n$
derivation trees
semantic terms
derived objects

Figure 1: IRTG, bimorphism overview.

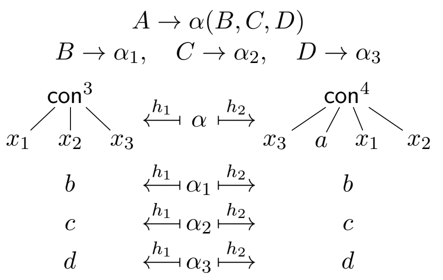

$A \to \alpha(B, C, D)$
$B \to \alpha_1, \quad C \to \alpha_2, \quad D \to \alpha_3$
$\mathsf{con}^3(x_1, x_2, x_3) \xleftarrow{h_1} \alpha \xrightarrow{h_2} \mathsf{con}^4(x_3, a, x_1, x_2)$
$b \xleftarrow{h_1} \alpha_1 \xrightarrow{h_2} b$
$c \xleftarrow{h_1} \alpha_2 \xrightarrow{h_2} c$
$d \xleftarrow{h_1} \alpha_3 \xrightarrow{h_2} d$

Figure 2: An IRTG encoding an SCFG.

IRTGs can be tailored to the expressive capacity of specific grammar formalisms by selecting suitable algebras. The string algebra in Table 1 yields context-free languages, more complex string al-

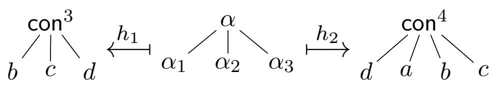


Figure 3: Derivation tree and semantic terms.

$$A \to \alpha'(A', D)$$
$$A' \to \alpha''(B, C)$$

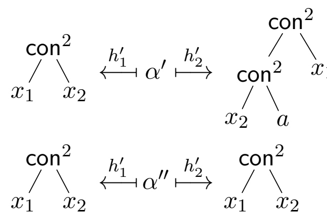


Figure 4: Binary rules corresponding to the α-rule in Fig. 2.

gebras yield tree-adjoining languages (Koller and Kuhlmann, 2012), and algebras over other domains can yield languages of trees, graphs, or other objects. Furthermore, IRTGs with $n = 1$ describe languages that are subsets of the algebra's domain, $n = 2$ yields synchronous languages or tree transductions, and so on.

3 IRTG binarization

We will now show how to apply the rule-by-rule binarization technique to IRTGs. We start in this section by defining the binarization of a rule in an IRTG, and characterizing it in terms of *binarization terms* and *variable trees*. We derive the actual binarization algorithm from this in Section 4.

For the remainder of this paper, let $\mathbb{G} = (\mathcal{B}, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be an IRTG over $(\Delta_1, \ldots, \Delta_n)$ with $\mathcal{B} = (G, h_1, \ldots, h_n)$.

3.1 An introductory example

We start with an example to give an intuition of our approach. Consider the first rule in Fig. 2, which has rank three. This rule derives (in one step) the fragment $\alpha(x_1, x_2, x_3)$ of the derivation tree in Fig. 3, which is mapped to the semantic terms $h_1(\alpha)$ and $h_2(\alpha)$ shown in Fig. 2. Now consider the rules in Fig. 4. These rules can be used to derive (in two steps) the derivation tree fragment ξ in Fig. 5e. Note that the terms $h'_1(\xi)$ and $h_1(\alpha)$ are *equivalent* in that they denote the same function over the string algebra, and so are the terms $h'_2(\xi)$ and $h_2(\alpha)$. Thus, replacing the α-rule by the rules in Fig. 4 does not change the language of the IRTG. However, since the new rules are binary,

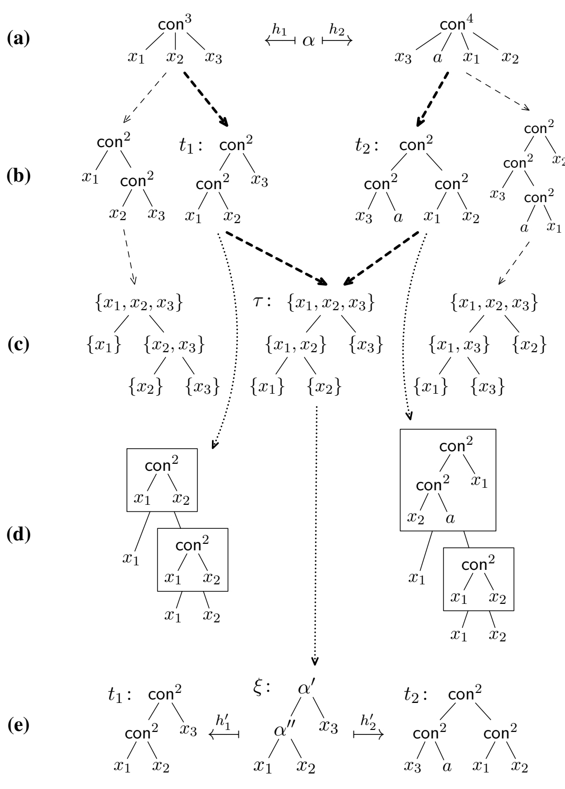


Figure 5: Outline of the binarization algorithm.

parsing and translation will be cheaper.

Now we want to construct the binary rules systematically. In the example, we proceed as follows (cf. Fig. 5). For each of the terms $h_1(\alpha)$ and $h_2(\alpha)$ (Fig. 5a), we consider all terms that satisfy two properties (Fig. 5b): (i) they are equivalent to $h_1(\alpha)$ and $h_2(\alpha)$, respectively, and (ii) at each node at most two subtrees contain variables. As Fig. 5 suggests, there may be many different terms of this kind. For each of these terms, we analyze the bracketing of variables, obtaining what we call a *variable tree* (Fig. 5c). Now we pick terms t_1 and t_2 corresponding to $h_1(\alpha)$ and $h_2(\alpha)$, respectively, such that (iii) they have the same variable tree, say τ. We construct a tree ξ from τ by a simple relabeling, and we read off the tree homomorphisms h'_1 and h'_2 from a decomposition we perform on t_1 and t_2, respectively; see Fig. 5, dotted arrows, and compare the boxes in Fig. 5d with the homomorphisms in Fig. 4. Now the rules in Fig. 4 are easily extracted from ξ.

These rules are equivalent to r because of (i); they are binary because ξ is binary, which in turn holds because of (ii); finally, the decompositions of t_1 and t_2 are compatible with ξ because of (iii). We call terms t_1 and t_2 *binarization terms* if they satisfy (i)–(iii). We will see below that we can construct binary rules equivalent to r from any given sequence of binarization terms t_1, t_2, and that binarization terms exist whenever equivalent binary rules exist. The majority of this paper revolves around the question of finding binarization terms.

Rule-by-rule binarization of IRTGs follows the intuition laid out in this example closely: it means processing each suprabinary rule, attempting to replace it with an equivalent collection of binary rules.

3.2 Binarization terms

We will now make this intuition precise. To this end, we assume that $r = q \to \alpha(q_1, \ldots, q_k)$ is a suprabinary rule of G. As we have seen, binarizing r boils down to constructing:

- a tree ξ over some binary signature Σ' and
- tree homomorphisms $h'_1, \ldots, h'_n$ of type $h'_i \colon T_{\Sigma'}(X) \to T_{\Delta_i}(X)$,

such that $h'_i(\xi)$ and $h_i(\alpha)$ are equivalent, i.e., they denote the same function over $\mathcal{A}_i$. We call such a tuple $(\xi, h'_1, \ldots, h'_n)$ a *binarization* of the rule r. Note that a binarization of r need not exist. The problem of *rule-by-rule binarization* consists in computing a binarization of each suprabinary rule of a grammar. If such a binarization does not exist, the problem does not have a solution.

In order to define variable trees, we assume a mapping seq that maps each finite set U of pairwise disjoint variable sets to a sequence over U which contains each element exactly once. Let $t \in C_\Delta(X_k)$. The *variable set* of t is the set of all variables that occur in t. The *set* $S(t)$ *of subtree variables* of t consists of the nonempty variable sets of all subtrees of t. We represent $S(t)$ as a tree $v(t)$, which we call *variable tree* as follows. Any two elements of $S(t)$ are either comparable (with respect to the subset relation) or disjoint. We extend this ordering to a tree structure by ordering disjoint elements via seq. We let $v(L) = \{v(t) \mid t \in L\}$ for every $L \subseteq C_\Delta(X_k)$.

In the example of Fig. 5, t_1 and t_2 have the same set of subtree variables; it is $\{\{x_1\}, \{x_2\}, \{x_3\}, \{x_1, x_2\}, \{x_1, x_2, x_3\}\}$. If we assume that seq orders sets of variables according to the least variable index, we arrive at the variable tree in the center of Fig. 5.

Now let $t_1 \in T_{\Delta_1}(X_k), \ldots, t_n \in T_{\Delta_n}(X_k)$. We call the tuple $t_1, \ldots, t_n$ *binarization terms* of r if the following properties hold: (i) $h_i(\alpha)$ and t_i are equivalent; (ii) at each node the tree t_i contains at most two subtrees with variables; and (iii) the terms $t_1, \ldots, t_n$ have the same variable tree.

Assume for now that we have found binarization terms $t_1, \ldots, t_n$. We show how to construct a binarization $(\xi, h'_1, \ldots, h'_n)$ of r with $t_i = h'_i(\xi)$.

First, we construct ξ. Since $t_1, \ldots, t_n$ are binarization terms, they have the same variable tree, say, τ. We obtain ξ from τ by replacing every label of the form $\{x_j\}$ with x_j, and every other label with a fresh symbol. Because of condition (ii) in in the definition of binarization terms, ξ is binary.

In order to construct $h'_i(\sigma)$ for each symbol σ in ξ, we transform t_i into a tree t'_i with labels from $C_{\Delta_i}(X)$ and the same structure as ξ. Then we read off $h'_i(\sigma)$ from the node of t'_i that corresponds to the σ-labeled node of ξ. The transformation proceeds as illustrated in Fig. 6: first, we apply the *maximal decomposition operation* $\rightsquigarrow_d$; it replaces every label $f \in \Delta_i|_k$ by the tree $f(x_1, \ldots, x_k)$, represented as a box. After that, we keep applying the *merge operation* $\rightsquigarrow_m$ as often as possible; it merges two boxes that are in a parent-child relation, given that one of them has at most one child. Thus the number of variables in any box can only decrease. Finally, the *reorder operation* $\rightsquigarrow_o$ orders the children of each box according to the seq of their variable sets. These operations do not change the variable tree; one can use this to show that t'_i has the same structure as ξ.

Thus, if we can find binarization terms, we can construct a binarization of r. Conversely, for any given binarization $(\xi, h'_1, \ldots, h'_n)$ the semantic terms $h'_1(\xi), \ldots, h'_n(\xi)$ are binarization terms. This proves the following lemma.

Lemma 1 *There is a binarization of* r *if and only if there are binarization terms of* r.

3.3 Finding binarization terms

It remains to show how we can find binarization terms of r, if there are any.

Let $b_i \colon T_{\Delta_i}(X_k) \to \mathcal{P}(T_{\Delta_i}(X_k))$ the mapping with $b_i(t) = \{t' \in T_{\Delta_i}(X_k) \mid t$ and t' are equivalent, and at each node t' has at most two children with variables$\}$. Figure 5b shows some elements of $b_1(h_1(\alpha))$ and $b_2(h_2(\alpha))$ for our example. Terms $t_1, \ldots, t_n$ are binarization terms precisely when $t_i \in b_i(h_i(\alpha))$ and $t_1, \ldots, t_n$ have the same variable tree. Thus we can characterize binarization terms as follows.

Lemma 2 *There are binarization terms if and only if* $\bigcap_i v(b_i(h_i(\alpha))) \neq \emptyset$.

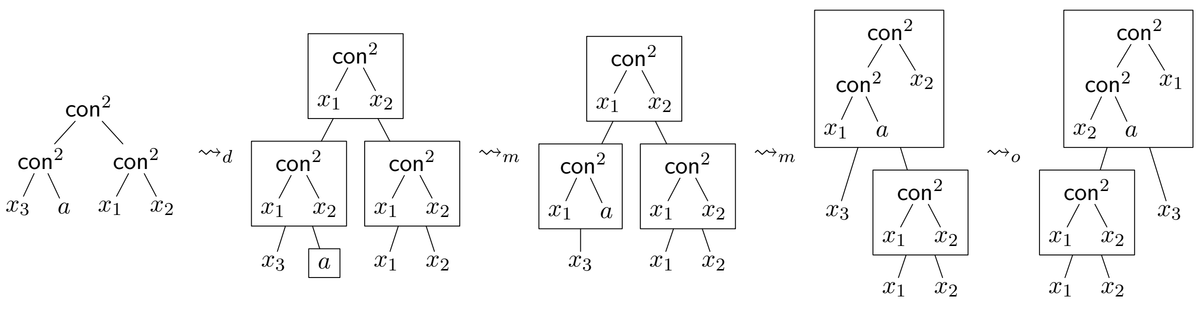


Figure 6: Transforming t_2 into t'_2.

This result suggests the following procedure for obtaining binarization terms. First, determine whether the intersection in Lemma 2 is empty. If it is, then there is no binarization of r. Otherwise, select a variable tree τ from this set. We know that there are trees $t_1, \dots, t_n$ such that $t_i \in b_i(h_i(\alpha))$ and $v(t_i) = \tau$. We can therefore select arbitrary concrete trees $t_i \in b_i(h_i(\alpha)) \cap v^{-1}(\tau)$. The terms $t_1, \dots, t_n$ are then binarization terms.

4 Effective IRTG binarization

In this section we develop our binarization algorithm. Its key task is finding binarization terms $t_1, \dots, t_n$. This task involves deciding term equivalence, as t_i must be equivalent to $h_i(\alpha)$. In general, equivalence is undecidable, so the task cannot be solved. We avoid deciding equivalence by requiring the user to specify an explicit approximation of b_i, which we call a *b-rule*. This parameter gives rise to a restricted version of the rule-by-rule binarization problem, which is efficiently computable while remaining practically relevant.

Let Δ be a signature. A *binarization rule (b-rule) over* Δ is a mapping $\mathfrak{b}\colon \Delta \to \mathcal{P}(T_\Delta(X))$ where for every $f \in \Delta|_k$ we have that $\mathfrak{b}(f) \subseteq C_\Delta(X_k)$, at each node of a tree in $\mathfrak{b}(f)$ only two children contain variables, and $\mathfrak{b}(f)$ is a regular tree language. We extend $\mathfrak{b}$ to $T_\Delta(X)$ by setting $\mathfrak{b}(x_j) = \{x_j\}$ and $\mathfrak{b}(f(t_1, \dots, t_k)) = \{t[x_j/t'_j \mid 1 \le j \le k] \mid t \in \mathfrak{b}(f), t'_j \in \mathfrak{b}(t_j)\}$, where $[x_j/t'_j]$ denotes substitution of x_j by t'_j. Given an algebra $\mathcal{A}$ over Δ, a b-rule $\mathfrak{b}$ over Δ is called *a b-rule over* $\mathcal{A}$ if, for every $t \in T_\Delta(X_k)$ and $t' \in \mathfrak{b}(t)$, t' and t are equivalent in $\mathcal{A}$. Such a b-rule encodes equivalence in $\mathcal{A}$, and it does so in an explicit and compact way: because $\mathfrak{b}(f)$ is a regular tree language, a b-rule can be specified by a finite collection of RTGs, one for each symbol $f \in \Delta$. We will look at examples (for the string and tree algebras shown earlier) in Section 5.

From now on, we assume that $\mathfrak{b}_1, \dots, \mathfrak{b}_n$ are b-rules over $\mathcal{A}_1, \dots, \mathcal{A}_n$, respectively. A binarization $(\xi, h'_1, \dots, h'_n)$ of r is a *binarization of* r *with respect to* $\mathfrak{b}_1, \dots, \mathfrak{b}_n$ if $h'_i(\xi) \in \mathfrak{b}_i(h_i(\alpha))$. Likewise, binarization terms $t_1, \dots, t_n$ are *binarization terms with respect to* $\mathfrak{b}_1, \dots, \mathfrak{b}_n$ if $t_i \in \mathfrak{b}_i(h_i(\alpha))$. Lemmas 1 and 2 carry over to the restricted notions. The problem of *rule-by-rule binarization with respect to* $\mathfrak{b}_1, \dots, \mathfrak{b}_n$ consists in computing a binarization with respect to $\mathfrak{b}_1, \dots, \mathfrak{b}_n$ for each suprabinary rule.

By definition, every solution to this restricted problem is also a solution to the general problem. The converse need not be true. However, we can guarantee that the restricted problem has at least one solution whenever the general problem has one, by requiring $v(\mathfrak{b}_i(h_i(\alpha)) = v(b(h_i(\alpha))$. Then the intersection in Lemma 2 is empty in the restricted case if and only if it is empty in the general case. We call the b-rules $\mathfrak{b}_1, \dots, \mathfrak{b}_1$ *complete* on $\mathbb{G}$ if the equation holds for every $\alpha \in \Sigma$.

Now we show how to effectively compute binarization terms with respect to $\mathfrak{b}_1, \dots, \mathfrak{b}_n$, along the lines of Section 3.3. More specifically, we construct an RTG for each of the sets (i) $\mathfrak{b}_i(h_i(\alpha))$, (ii) $b'_i = v(\mathfrak{b}_i(h_i(\alpha)))$, (iii) $\bigcap_i b'_i$, and (iv) $b''_i = \mathfrak{b}_i(h_i(\alpha)) \cap v^{-1}(\tau)$ (given τ). Then we can select τ from (iii) and t_i from (iv) using a standard algorithm, such as the Viterbi algorithm or Knuth's algorithm (Knuth, 1977; Nederhof, 2003; Huang and Chiang, 2005). The effectiveness of our procedure stems from the fact that we only manipulate RTGs and never enumerate languages.

The construction for (i) is recursive, following the definition of $\mathfrak{b}_i$. The base case is a language $\{x_j\}$, for which the RTG is easy. For the recursive case, we use the fact that regular tree languages are closed under substitution (Gécseg and Steinby, 1997, Prop. 7.3). Thus we obtain an RTG G_i with $L(G_i) = \mathfrak{b}_i(h_i(\alpha))$.

For (ii) and (iv), we need the following auxiliary construction. Let $G_i = (P, p_0, R)$. We define the mapping $\mathsf{var}_i \colon P \to \mathcal{P}(X_k)$ such that for every $p \in P$, every $t \in L^p(G_i)$ contains exactly the variables in $\mathsf{var}_i(p)$. We construct it as follows. We initialize $\mathsf{var}_i(p)$ to "unknown" for every p. For every rule $p \to x_j$, we set $\mathsf{var}_i(p) = \{x_j\}$. For every rule $p \to \sigma(p_1, \ldots, p_k)$ such that $\mathsf{var}_i(p_j)$ is known, we set $\mathsf{var}_i(p) = \bigcup_j \mathsf{var}_i(p_j)$. This is iterated; it can be shown that $\mathsf{var}_i(p)$ is never assigned two different values for the same p. Finally, we set all remaining unknown entries to $\emptyset$.

For (ii), we construct an RTG G'_i with $L(G'_i) = b'_i$ as follows. We let $G'_i = (\{\langle \mathsf{var}_i(p)\rangle \mid p \in P\}, \mathsf{var}_i(p_0), R')$ where R' consists of the rules

$$
\begin{aligned}
&\langle\{x_j\}\rangle \to \{x_j\} \quad \text{if} \quad p \to x_i \in R\ ,\\
&\langle \mathsf{var}_i(p)\rangle \to \mathsf{var}_i(p)(\langle U_1\rangle, \ldots, \langle U_l\rangle))\\
&\quad \text{if} \quad p \to \sigma(p_1, \ldots, p_k) \in R,\\
&\qquad V = \{\mathsf{var}_i(p_j) \mid 1 \le j \le k\} \setminus \{\emptyset\},\\
&\qquad |V| \ge 2, \mathsf{seq}(V) = (U_1, \ldots, U_l)\ .
\end{aligned}
$$

For (iii), we use the standard product construction (Gécseg and Steinby, 1997, Prop. 7.1).

For (iv), we construct an RTG G''_i such that $L(G''_i) = b''_i$ as follows. We let $G''_i = (P, p_0, R'')$, where R'' consists of the rules

$$
\begin{aligned}
&p \to \sigma(p_1, \ldots, p_k)\\
&\quad \text{if} \quad p \to \sigma(p_1, \ldots, p_k) \in R,\\
&\qquad V = \{\mathsf{var}_i(p_j) \mid 1 \le j \le k\} \setminus \{\emptyset\},\\
&\qquad \text{if } |V| \ge 2, \text{ then}\\
&\qquad\ (\mathsf{var}_i(p), \mathsf{seq}(V)) \text{ is a fork in } \tau\ .
\end{aligned}
$$

By a fork $(u, u_1 \cdots u_k)$ in τ, we mean that there is a node labeled u with k children labeled u_1 up to u_k.

At this point we have all the ingredients for our binarization algorithm, shown in Algorithm 1. It operates directly on a bimorphism, because all the relevant information about the algebras is captured by the b-rules. The following theorem documents the behavior of the algorithm. In short, it solves the problem of rule-by-rule binarization with respect to b-rules $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$.

Theorem 3 *Let $\mathbb{G} = (\mathcal{B}, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ be an IRTG, and let $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$ be b-rules over $\mathcal{A}_1, \ldots, \mathcal{A}_n$, respectively.*

Algorithm 1 terminates. Let $\mathcal{B}'$ be the bimorphism computed by Algorithm 1 on $\mathcal{B}$ and $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$. Then $\mathbb{G}' = (\mathcal{B}', \mathcal{A}_1, \ldots, \mathcal{A}_n)$ is equivalent to $\mathbb{G}$, and $\mathbb{G}'$ is of rank 2 if and only if every suprabinary rule of $\mathbb{G}$ has a binarization with respect to $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$.

```
Input:   bimorphism $\mathcal{B} = (G, h_1, \ldots, h_n)$,
         b-rules $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$ over $\Delta_1, \ldots, \Delta_n$
Output:  bimorphism $\mathcal{B}'$
 1: $\mathcal{B}' \leftarrow (G|_{\le 2}, h_1, \ldots, h_n)$
 2: for rule $r \colon q \to \alpha(q_1, \ldots, q_k)$ of $G|_{>2}$ do
 3:    for $i = 1, \ldots, n$ do
 4:       compute RTG $G_i$ for $\mathfrak{b}_i(h_i(\alpha))$
 5:       compute RTG $G'_i$ for $v(\mathfrak{b}_i(h_i(\alpha)))$
 6:    compute RTG $G_v$ for $\bigcap_i L(G'_i)$
 7:    if $L(G_v) = \emptyset$ then
 8:       add $r$ to $\mathcal{B}'$
 9:    else
10:       select $t' \in L(G_v)$
11:       for $i = 1, \ldots, n$ do
12:          compute RTG $G''_i$ for
13:                $\mathfrak{b}''_i = \mathfrak{b}_i(h_i(\alpha)) \cap v^{-1}(t')$
14:          select $t_i \in L(G''_i)$
15:       construct binarization for $t_1, \ldots, t_n$
16:       add appropriate rules to $\mathcal{B}'$
```

Algorithm 1: Complete binarization algorithm, where $G|_{\le 2}$ and $G|_{>2}$ is G restricted to binary and suprabinary rules, respectively.

The runtime of Algorithm 1 is dominated by the intersection construction in line 6, which is $O(m_1 \cdot \ldots \cdot m_n)$ per rule, where m_i is the size of G'_i. The quantity m_i is linear in the size of the terms on the right-hand side of h_i, and in the number of rules in the b-rule $\mathfrak{b}_i$.

5 Applications

Algorithm 1 implements rule-by-rule binarization with respect to given b-rules. If a rule of the given IRTG does not have a binarization with respect to these b-rules, it is simply carried over to the new grammar, which then has a rank higher than 2. The number of remaining suprabinary rules depends on the b-rules (except for rules that have no binarization at all). The user can thus engineer the b-rules according to their current needs, trading off completeness, runtime, and engineering effort.

By contrast, earlier binarization algorithms for formalisms such as SCFG and LCFRS simply attempt to find an equivalent grammar of rank 2; there is no analogue of our b-rules. The problem these algorithms solve corresponds to the general rule-by-rule binarization problem from Section 3.

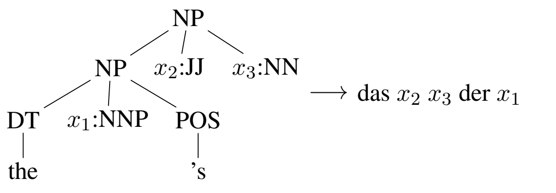


Figure 7: A rule of a tree-to-string transducer.

We show that under certain conditions, our algorithm can be used to solve this problem as well. In the following two subsections, we illustrate this for SCFGs and tree-to-string transducers, respectively. In the final subsection, we discuss how to extend this approach to other grammar formalisms as well.

5.1 Synchronous context-free grammars

We have used SCFGs as the running example in this paper. SCFGs are IRTGs with two interpretations into the string algebra of Table 1, as illustrated by the example in Fig. 2. In order to make our algorithm ready to use, it remains to specify a b-rule for the string algeba.

We use the following b-rule for both $\mathfrak{b}_1$ and $\mathfrak{b}_2$. Each symbol $a \in \Delta_i|_0$ is mapped to the language $\{a\}$. Each symbol con^k, $k \geq 2$, is mapped to the language induced by the following RTG with states of the form $[j, j']$ (where $0 \leq j < j' \leq k$) and final state $[0, k]$:

$$[j-1, j] \to x_j \qquad (1 \leq j \leq k)$$
$$[j, j'] \to \mathsf{con}^2([j, j''], [j'', j']) \qquad (0 \leq j < j'' < j' \leq k)$$

This language expresses all possible ways in which con^k can be written in terms of con^2.

Our definition of rule-by-rule binarization with respect to $\mathfrak{b}_1$ and $\mathfrak{b}_2$ coincides with that of Huang et al. (2009): any rule can be binarized by both algorithms or neither. For instance, for the SCFG rule $A \to \langle BCDE, CEBD \rangle$, the sets $v(\mathfrak{b}_1(h_1(\alpha)))$ and $v(\mathfrak{b}_2(h_2(\alpha)))$ are disjoint, thus no binarization exists. Two strings of length N can be parsed with a binary IRTG that represents an SCFG in time $O(N^6)$.

5.2 Tree-to-string transducers

Some approaches to SMT go beyond string-to-string translation models such as SCFG by exploiting known syntactic structures in the source or target language. This perspective on translation naturally leads to the use of tree-to-string transducers

NP $\to \alpha$(NNP, JJ, NN)

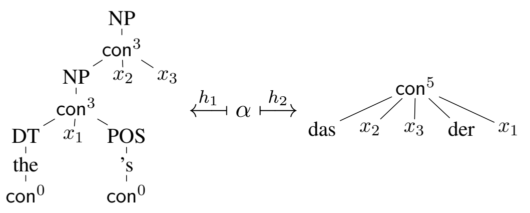


Figure 8: An IRTG rule encoding the rule in Fig. 7.

(Yamada and Knight, 2001; Galley et al., 2004; Huang et al., 2006; Graehl et al., 2008). Figure 7 shows an example of a tree-to-string rule. It might be used to translate "the Commission's strategic plan" into "das langfristige Programm der Kommission".

Our algorithm can binarize tree-to-string transducers; to our knowledge, it is the first algorithm to do so. We model the tree-to-string transducer as an IRTG $\mathbb{G} = ((G, h_1, h_2), \mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ is the string algebra, but this time $\mathcal{A}_1$ is the tree algebra shown in Table 1. This algebra has operations con^k to concatenate sequences of trees and unary γ that maps any sequence $(t_1, \ldots, t_l)$ of trees to the tree $\gamma(t_1, \ldots, t_l)$, viewed as a sequence of length 1. Note that we exclude the operation con^1 because it is the identity and thus unnecessary. Thus the rule in Fig. 7 translates to the IRTG rule shown in Fig. 8.

For the string algebra, we reuse the b-rule from Section 5.1; we call it $\mathfrak{b}_2$ here. For the tree algebra, we use the following b-rule $\mathfrak{b}_1$. It maps con^0 to $\{\mathsf{con}^0\}$ and each unary symbol γ to $\{\gamma(x_1)\}$. Each symbol con^k, $k \geq 2$, is treated as in the string case. Using these b-rules, we can binarize the rule in Fig. 8 and obtain the rules in Fig. 9. Parsing of a binary IRTG that represents a tree-to-string transducer is $O(N^3 \cdot M)$ for a string of length N and a tree with M nodes.

We have implemented our binarization algorithm and the b-rules for the string and the tree algebra. In order to test our implementation, we extracted a tree-to-string transducer from about a million parallel sentences of English-German Europarl data, using the GHKM rule extractor (Galley, 2010). Then we binarized the transducer. The results are shown in Fig. 10. Of the 2.15 million rules in the extracted transducer, 460,000 were suprabinary, and 67 % of these could be binarized. Binarization took 4.4 minutes on a single core of an Intel Core i5 2520M processor.

$$\mathrm{NP} \to \alpha'(\mathrm{NNP}, A')$$
$$A' \to \alpha''(\mathrm{JJ}, \mathrm{NN})$$

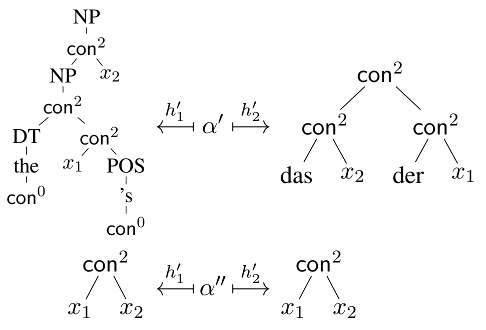


Figure 9: Binarization of the rule in Fig. 8.

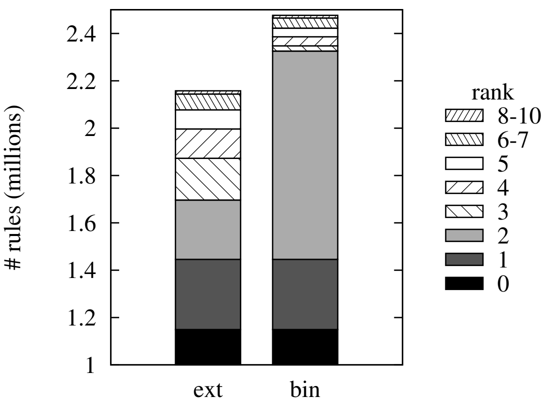


Figure 10: Rules of a transducer extracted from Europarl (ext) vs. its binarization (bin).

5.3 General approach

Our binarization algorithm can be used to solve the general rule-by-rule binarization problem for a specific grammar formalism, provided that one can find appropriate b-rules. More precisely, we need to devise a class $\mathcal{C}$ of IRTGs over the same sequence $\mathcal{A}_1, \dots, \mathcal{A}_n$ of algebras that encodes the grammar formalism, together with b-rules $\mathfrak{b}_1, \dots, \mathfrak{b}_n$ over $\mathcal{A}_1, \dots, \mathcal{A}_n$ that are complete on every grammar in $\mathcal{C}$, as defined in Section 4.

We have already seen the b-rules for SCFGs and tree-to-string transducers in the preceding subsections; now we have a closer look at the class $\mathcal{C}$ for SCFGs. We used the class of all IRTGs with two string algebras and in which $h_i(\alpha)$ contains at most one occurrence of a symbol con^k for every $\alpha \in \Sigma$. On such a grammar the b-rules are complete. Note that this would not be the case if we allowed several occurrences of con^k, as in $\mathsf{con}^2(\mathsf{con}^2(x_1, x_2), x_3)$. This term is equivalent to itself and to $\mathsf{con}^2(x_1, \mathsf{con}^2(x_2, x_3))$, but the b-rules only cover the former. Thus they miss one variable tree. For the term $\mathsf{con}^3(x_1, x_2, x_3)$, however, the b-rules cover both variable trees.

Generally speaking, given $\mathcal{C}$ and b-rules $\mathfrak{b}_1, \dots, \mathfrak{b}_n$ that are complete on every IRTG in $\mathcal{C}$, Algorithm 1 solves the general rule-by-rule binarization problem on $\mathcal{C}$. We can adapt Theorem 3 by requiring that $\mathbb{G}$ must be in $\mathcal{C}$, and replacing each of the two occurrences of "binarization with respect to $\mathfrak{b}_1, \dots, \mathfrak{b}_n$" by simply "binarization". If $\mathcal{C}$ is such that every grammar from a given grammar formalism can be encoded as an IRTG in $\mathcal{C}$, this solves the general rule-by-rule binarization problem of that grammar formalism.

6 Conclusion

We have presented an algorithm for binarizing IRTGs rule by rule, with respect to b-rules that the user specifies for each algebra. This improves the complexity of parsing and translation with any monolingual or synchronous grammar that can be represented as an IRTG. A novel algorithm for binarizing tree-to-string transducers falls out as a special case.

In this paper, we have taken the perspective that the binarized IRTG uses the same algebras as the original IRTG. Our algorithm extends to grammars of arbitrary fanout (such as synchronous tree-adjoining grammar (Koller and Kuhlmann, 2012)), but unlike LCFRS-based approaches to binarization, it will not *increase* the fanout to ensure binarizability. In the future, we will explore IRTG binarization with fanout increase. This could be done by binarizing into an IRTG with a more complicated algebra (e.g., of string tuples). We might compute binarizations that are optimal with respect to some measure (e.g., fanout (Gomez-Rodriguez et al., 2009) or parsing complexity (Gildea, 2010)) by keeping track of this measure in the b-rule and taking intersections of weighted tree automata.

Acknowledgments

We thank the anonymous referees for their insightful remarks, and Sarah Hemmen for implementing an early version of the algorithm. Matthias Büchse was financially supported by DFG VO 1011/6-1.

References

Alfred V. Aho and Jeffrey D. Ullman. 1969. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3:37–56.

André Arnold and Max Dauchet. 1976. Bi-transduction de forêts. In *Proc. 3rd Int. Coll. Automata, Languages and Programming*, pages 74–86. Edinburgh University Press.

Walter S. Brainerd. 1969. Tree generating regular systems. *Information and Control*, 14(2):217–231.

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

Steve DeNeefe and Kevin Knight. 2009. Synchronous tree-adjoining machine translation. In *Proceedings of EMNLP*, pages 727–736.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st ACL*, pages 205–208.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of HLT/NAACL*, pages 273–280.

Michael Galley. 2010. GHKM rule extractor. `http://www-nlp.stanford.edu/~mgalley/software/stanford-ghkm-latest.tar.gz`, retrieved on March 28, 2012.

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer-Verlag.

Daniel Gildea. 2010. Optimal parsing strategies for linear context-free rewriting systems. In *Proceedings of NAACL HLT*.

Joseph A. Goguen, Jim W. Thatcher, Eric G. Wagner, and Jesse B. Wright. 1977. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24:68–95.

Carlos Gomez-Rodriguez, Marco Kuhlmann, Giorgio Satta, and David Weir. 2009. Optimal reduction of rule length in linear context-free rewriting systems. In *Proceedings of NAACL HLT*.

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

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the 9th IWPT*, pages 53–64.

Liang Huang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proceedings of the 7th AMTA*, pages 66–73.

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

Donald E. Knuth. 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th IWPT*, pages 2–13.

Alexander Koller and Marco Kuhlmann. 2012. Decomposing TAG algorithms using simple algebraizations. In *Proceedings of the 11th TAG+ Workshop*, pages 135–143.

Philip M. Lewis and Richard E. Stearns. 1966. Syntax directed transduction. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:21–35.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

Rebecca Nesson, Stuart M. Shieber, and Alexander Rush. 2006. Induction of probabilistic synchronous tree-insertion grammars for machine translation. In *Proceedings of the 7th AMTA*.

Owen Rambow and Giorgio Satta. 1999. Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223(1–2):87–120.

David J. Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th ACL*, pages 523–530.