Weighted parsing of trees

Mark-Jan Nederhof
School of Computer Science, University of St Andrews
North Haugh, St Andrews, KY16 9SX, Scotland

Abstract

We show how parsing of trees can be formalized in terms of the intersection of two tree languages. The focus is on weighted regular tree grammars and weighted tree adjoining grammars. Potential applications are discussed, such as parameter estimation across formalisms.

1 Introduction

In parsing theory, strings and trees traditionally have had a very different status. Whereas strings in general receive the central focus, the trees involved in derivations of strings are often seen as auxiliary concepts at best. Theorems tend to be about the power of grammatical formalisms to produce strings (weak generative power) rather than trees (strong generative power).

This can be explained by looking at typical applications of parsing. In compiler construction for example, one distinguishes between parse trees and (abstract) syntax trees, the former being shaped according to a grammar that is massaged to make it satisfy relatively artificial constraints, e.g. that of LALR(1), which is required by many compiler generators (Aho et al., 2007). The form of syntax trees is often chosen to simplify phases of semantic processing that follow parsing. As the machinery used in such processing is generally powerful, this offers much flexibility in the choice of the exact shape and labelling of syntax trees, as intermediate form between parsing and semantic analysis.

In the study of natural languages, parse trees have played a more important role. Whereas linguistic utterances are directly observable and trees deriving them are not, there are nevertheless traditions within linguistics that would see one structural analysis of a sentence as strongly preferred over another. Furthermore, within computational linguistics there are empirical arguments to claim certain parses are correct and others are incorrect. For example, a question answering systems may verifiably give the wrong answer if the question is parsed incorrectly. See (Jurafsky and Martin, 2000) for general discussion on the role of parsing in NLP.

Despite the relative importance of strong generative power in computational linguistics, there is still much freedom in how exactly parse trees are shaped and how vertices are labelled, due to the power of semantic analysis that typically follows parsing. This has affected much of the theoretical investigations into the power of linguistic formalisms, and where strong equivalence is considered at all, it is often ”modulo relabelling” or allowing minor structural changes.

With the advent of syntax-based machine translation, trees have however gained much importance, and are even considered as the main objects of study. This is because many MT modules have trees both as input and output, which means the computational strength of such modules can be measured only in terms of the tree languages they accept and the transductions between tree languages that they implement. See for example (Knight, 2007).

In contrast, trees have always been the central issue in an important and well-established subfield of formal language theory that studies tree languages, tree automata and tree transducers (Gcseg and Steinby, 1997). The string languages generated by the relevant formalisms in this context are mostly taken to be of secondary importance, if they are considered at all.

This paper focuses on tree languages, but involves a technique that was devised for string languages, and shows how the technique carries over to tree languages. The original technique can be seen as the most fundamental idea in the field of context-free parsing, as it captures the essence of

finding hierarchical structure in a linear sequence. The generalization also finds structure in a linear sequence, but now the sequence corresponds to paths in trees each leading down from a vertex to a leaf. This means that the proposed type of parsing is orthogonal to the conventional parsing of strings.

The insights this offers have the potential to create new avenues of research into the relation between formalisms that were until now considered only in isolation. We seek credence to this claim by investigating how probability distributions can be carried over from tree adjoining grammars to regular tree grammars, and vice versa.

The implication that the class of tree languages of tree adjoining grammars (TAGs) is closed under intersection with regular tree languages is not surprising, as the linear context-free tree languages (LCFTLs) are closed under intersection with regular tree languages (Kepser and Mönnich, 2006). The tree languages of TAGs form a subclass of the LCFTLs, and the main construction in the proof of the closure result for the latter can be suitably restricted to the former.

The structure of this paper is as follows. The main grammatical formalisms considered in this paper are summarized in Section 2 and Section 3 discusses a number of analyses of these formalisms that will be used in later sections. Section 4 starts by explaining how parsing of a string can be seen as the construction of a grammar that generates the intersection of two languages, and then moves on to a type of parsing involving intersection of tree languages in place of string languages.

In order to illustrate the implications of the theory, we consider how it can be used to solve a practical problem, in Section 5. A number of possible extensions are outlined in Section 6.

2 Formalisms

In this section, we recall the formalisms of weighted regular tree grammars and weighted tree adjoining grammars. We use similar notation and terminology for both, in order to prepare for Section 4, where we investigate the combination of these formalisms through intersection. As a consequence of the required unified notation, we deviate to some degree from standard definitions, without affecting generative power however.

For common definitions of weighted regular tree grammars, the reader is referred to (Graehl and Knight, 2004). Weighted tree adjoining grammars are a straightforward generalization of probabilistic (or stochastic) tree adjoining grammars, as introduced by (Resnik, 1992) and (Schabes, 1992).

For both regular tree grammars (RTGs) and tree adjoining grammars (TAGs), we will write a labeled and ordered tree as $A(\alpha)$. where A is the label of the root node, and α is a sequence of expressions of the same form that each represent an immediate subtree. In our presentation, labels do not have explicit ranks, that is, the number of children of a node is not determined by its label. This allows an interesting generalization, to be discussed in Section 6.2.

Where we are interested in the string language generated by a tree-generating grammar, we may distinguish between two kinds of labels, the *terminal labels*, which may occur only at leaves, and the *nonterminal labels*, which may occur at any node. It is customary to write terminal leaves as a instead of $a()$. The *yield* of a tree is the string of occurrences of terminal labels in it, from left to right. Note that also nonterminal labels may occur at the leaves, but they will not be included in the yield; cf. epsilon rules in context-free grammars.

2.1 Weighted regular tree grammars

A *weighted regular tree grammar* (WRTG) is a 4-tuple $G = (S, L, R, s^{\vdash})$, where S and L are two finite sets of *states* and *labels*, respectively, $s^{\vdash} \in S$ is the *initial state*, and R is a finite set of *rules*. Each rule has the form:

$$s_0 \to A(s_1 \cdots s_m)\ \langle w \rangle,$$

where $s_0, s_1, \ldots, s_m$ are states ($0 \leq m$), A is a label and w is a weight.

Rewriting starts with a string containing only the initial state $s^{\vdash}$. This string is repeatedly rewritten by replacing the left-hand side state of a rule by the right-hand side of the same rule, until no state remains. It may be convenient to assume a canonical order of rewriting, for example in terms of left-most derivations (Hopcroft and Ullman, 1979).

Although alternative semirings can be considered, here we always assume that the weights of rules are non-negative real numbers, and the weight of a derivation of a tree is the product of the weights of the rule occurrences. If several (left-most) derivations result in the same tree, then

the weight of that tree is given by the sum of the weights of those derivations. Where we are interested in the string language, the weights of trees with the same yield are added to obtain the weight of that yield.

A (weighted) context-free grammar can be seen as a special case of a (weighted) regular tree grammar, where the set of states equals the set of labels, and rules have the form:

$$A \to A(B_1 \cdots B_m).$$

Also the class of (weighted) tree substitution grammars (Sima'an, 1997) can be seen as a special case of (weighted) regular tree grammars, by letting the set of labels overlap with the set of states, and imposing two constraints on the allowable rules. The first constraint is that for each label that is also a state, all defining rules are of the form:

$$A \to A(s_1 \cdots s_m).$$

The second constraint is that for each state that is not a label, there is exactly one rule with that state in the left-hand side. This means that exactly one subtree (or *elementary tree*) can be built top-down out of such states, down to a level where we again encounter states that are also labels. If desired, we can exclude infinite elementary trees by imposing an additional constraint on allowed sets of rules (no cycles composed of states that are not labels); alternatively, we can demand that the grammar does not contain any useless rules, which automatically excludes such infinite elementary trees.

2.2 Weighted linear indexed grammars

Although we are mainly interested in the tree languages of tree adjoining grammars, we will use an equivalent representation in terms of linear indexed grammars, in order to obtain a uniform notation with regard to regular tree grammars.

Thus, a *weighted linear indexed grammar* (WLIG) is a 5-tuple $G = (S, I, L, R, s^{\vdash})$, where S, I and L are three finite sets of *states*, *indices* and *labels*, respectively, $s^{\vdash} \in S$ is the *initial state*, and R is a finite set of *rules*. Each rule has one of the following four forms:

1. $s_0[\circ\circ] \to A(\ s_1[\,] \cdots s_{j-1}[\,]\ s_j[\circ\circ]\ s_{j+1}[\,] \cdots s_m[\,]\ )\ \langle w \rangle$,
   where $s_0, s_1, \ldots, s_m$ are states ($1 \leq j \leq m$), A is a label and w is a weight;
2. $s[\,] \to A()\ \langle w \rangle$;
3. $s[\circ\circ] \to s'[\iota\circ\circ]\ \langle w \rangle$, where ι is an index;
4. $s[\iota\circ\circ] \to s'[\circ\circ]\ \langle w \rangle$.

The expression $\circ\circ$ may be thought of as a variable denoting a string of indices on a stack, and this variable is to be consistently substituted in the left-hand and the right-hand sides of rules upon application during rewriting. In other words, stacks are copied from the left-hand side of a rule to at most one member in the right-hand side, which we will call the *head* of that rule. The expression $[\,]$ stands for the empty stack and $[\iota\circ\circ]$ denotes a stack with top element ι. Thereby, rules of the third type implement a stack push and rules of the fourth type implement a pop. Rewriting starts from $s^{\vdash}[\,]$. The four subsets of R containing rules of the respective four forms above will be referred to as R_1, R_2, R_3 and R_4.

In terms of tree adjoining grammars, which assume a finite number of elementary trees, the intuition behind the four types of rules is as follows. Rules of the first type correspond to continued construction of the same elementary tree. Rules of the third type correspond to the initiation of a newly adjoined auxiliary tree and rules of the fourth type correspond to its completion at a foot node, returning to an embedding elementary tree that is encoded in the index that is popped. Rules of the second type correspond to construction of leaves, as in the case of regular tree grammars. See further (Vijay-Shanker and Weir, 1994) for the equivalence of linear indexed grammars and tree adjoining grammars.

Note that regular tree grammars can be seen as special cases of linear indexed grammars, by excluding rules of the third and fourth types, which means that stacks of indices always remain empty (Joshi and Schabes, 1997).

2.3 Probabilistic grammars

A weighted regular tree grammar, or weighted linear indexed grammar, respectively, is called *probabilistic* if the weights are probabilities, that is, values between 0 and 1. A probabilistic regular tree grammar (PRTG) is *proper* if for each state s, the probabilities of all rules that have left-hand side s sum to one.

Properness for a probabilistic linear indexed grammar (PLIG) is more difficult to define, due to the possible overlap of applicability between

the four types of rules, listed in the section above. However, if we encode a given TAG as a LIG in a reasonable way, then a state s may occur both in left-hand sides of rules from R_1 and in left-hand sides of rules from R_3, but all other such overlap between the four types is precluded.

Intuitively, a state may represent an internal node of an elementary tree, in which case rules from both R_1 and R_3 may apply, or it may represent a non-foot leaf node, in which case a rule from R_2 may apply, or it may be a foot node, in which case a rule from R_4 may apply.

With this assumption that the only overlap in applicability is between R_1 and R_3, properness can be defined as follows.

- For each state s, either there are no rules in R_1 or R_3 with s in the left-hand side, or the sum of probabilities of all such rules equals one.
- For each state s, either there are no rules in R_2 with s in the left-hand side, or the sum of probabilities of all such rules equals one.
- For each state s and index ι, either there are no rules in R_4 with left-hand side $s[\iota\circ\circ]$, or the sum of probabilities of all such rules equals one.

We say a weighted regular tree grammar, or weighted linear indexed grammar, respectively, is *consistent* if the sum of weights of all (left-most) derivations is one. This is equivalent to saying that the sum of weights of all trees is one, and to saying that the sum of weights of all strings is one.

For each consistent WRTG (WLIG, respectively), there is an equivalent proper and consistent PRTG (PLIG, respectively). The proof lies in normalization. For WRTGs this is a trivial extension of normalization of weighted context-free grammars, as described for example by (Nederhof and Satta, 2003). For WLIGs (and weighted TAGs), the problem of normalization also becomes very similar once we consider that the set of *derivation trees* of tree adjoining grammars can be described with context-free grammars, and that this carries over to weighted derivation trees. See also (Sarkar, 1998).

WLIGs seemingly incur an extra complication, if a state may occur in combination with an index on top of the associated stack such that no rules are applicable. However, for LIGs that encode TAGs, the problem does not arise as, informally, one may always resume construction of the embedding elementary tree below the foot node of an adjoined auxiliary tree.

We say a LIG is in *TAG-normal form* if (a) at least one rule is applicable for each combination of state s and index ι such that $s[\iota\circ\circ]$ is derivable from $s^{\vdash}[\,]$, and (b) the only overlap in applicability of the four types of rules is between R_1 and R_3. Statements in what follows involving WLIGs (or PLIGs) in TAG-normal form also hold for weighted (or probabilistic) TAGs.

3 Analysis of grammars

We call a grammar rule *useless* if it cannot be part of any derivation of a tree (or of a string, in the case of grammars with an emphasis on string languages). We say a grammar is *reduced* if it does not contain useless rules.

Whereas most grammars written by hand or induced by a corpus or treebank are reduced, there are practical operations that turn reduced grammars into grammars with useless rules; we will see an example in the next section, where grammars are constructed that generate the intersection of two given languages. In order to determine whether the intersection is non-empty, it suffices to identify useless rules in the intersection grammar. If and only if *all* rules are useless, the generated language is empty.

In the case of context-free grammars (see for example (Sippu and Soisalon-Soininen, 1988)), the analysis to identify useless rules can be split into two phases:

1. a bottom-up phase to identify the grammar symbols that generate substrings, which may include the start symbol if the generated language is non-empty; and
2. a top-down phase to identify the grammar symbols that are reachable from the start symbol.

The intersection of the generating symbols and the reachable symbols gives the set of useful symbols. One can then identify useless rules as those that contain one or more symbols that are not useful.

The procedure for linear indexed grammars is similarly split into two phases, of which the first is given in Figure 1 in the form of a deduction system. The inference rules simultaneously derive

$$\frac{}{(s,s)}\ \Big\{ s \in S \qquad \text{(a)}$$

$$\frac{}{s}\ \Big\{ s[\,] \to A() \qquad \text{(b)}$$

$$\frac{s_1\ \cdots\ s_{j-1}\ (s_j, s)\ s_{j+1}\ \cdots\ s_m}{(s_0, s)}\ \Big\{ s_0[\circ\circ] \to A(s_1[\,]\ \cdots\ s_j[\circ\circ]\ \cdots\ s_m[\,]) \qquad \text{(c)}$$

$$\frac{s_1\ \cdots\ s_m}{s_0}\ \Big\{ s_0[\circ\circ] \to A(s_1[\,]\ \cdots\ s_j[\circ\circ]\ \cdots\ s_m[\,]) \qquad \text{(d)}$$

$$\frac{\begin{array}{c}(s_1, s_2)\\ (s_3, s_4)\end{array}}{(s_0, s_4)}\ \begin{cases} s_0[\circ\circ] \to s_1[\iota\circ\circ] \\ s_2[\iota\circ\circ] \to s_3[\circ\circ] \end{cases} \qquad \text{(e)}$$

$$\frac{\begin{array}{c}(s_1, s_2)\\ s_3\end{array}}{s_0}\ \begin{cases} s_0[\circ\circ] \to s_1[\iota\circ\circ] \\ s_2[\iota\circ\circ] \to s_3[\circ\circ] \end{cases} \qquad \text{(f)}$$

Figure 1: Simultaneous analysis of two kinds of subderivations in a LIG. Items (s, s') represent existence of one or more subderivations $s[\,] \to^* \alpha(s'[\,])$, where α is a tree with a gap in the form of an unresolved state s' associated with an empty stack. Furthermore, s and s' are connected through propagation of a stack of indices, or in other words, the occurrence of s' is the head of a rule, of which the left-hand side state is the head of another rule, etc., up to s. In the inference rules, items s represent existence of one or more subderivations $s[\,] \to^* \alpha$, where α is a complete tree (without any unresolved states).

two types of item. The generated language is non-empty if the item $s^{\vdash}$ can be derived.

We will explain inference rule (f), which is the most involved of the six rules. The two items in the antecedent indicate the existence of derivations $s_1[\,] \to^* \alpha(s_2[\,])$ and $s_3[\,] \to^* \beta$. Note that $s_1[\,] \to^* \alpha(s_2[\,])$ implies $s_1[\iota] \to^* \alpha(s_2[\iota])$, because an additional element in the bottom of a stack would not block an existing derivation. Hence $s_0[\,] \to s_1[\iota] \to^* \alpha(s_2[\iota]) \to \alpha(s_3[\,]) \to^* \alpha(\beta)$, which justifies the item s_0 in the consequent of the rule.

After determining which items can be derived through the deduction system, it is straightforward to identify those rules that are useful, by applying the inference rules in reverse, from consequent to antecedents, starting with $s^{\vdash}$.

The running time of the analysis is determined by how often each of the inference rules can be applied, which is bounded by the number of ways each can be instantiated with states and rules from the grammar. The six inference rules together give us $\mathcal{O}(|S| + |R_2| + |R_1| \cdot |S| + |R_1| + |R_3| \cdot |R_4| \cdot |S| + |R_3| \cdot |R_4|) = \mathcal{O}(|S| + |R_1| \cdot |S| + |R_2| + |R_3| \cdot |R_4| \cdot |S|) = |\mathcal{G}|^3$, where we assume a reasonable measure for the size $|\mathcal{G}|$ of a LIG $\mathcal{G}$, for example, the total number of occurrences of states, labels and indices in the rules.

It is not difficult to see that there is exactly one deduction of $s^{\vdash}$ in the deduction system for each complete derivation in the grammar. We leave the full proof to the interested reader, but provide the hint that items (s, s') can only play a role in a complete deduction provided s' is rewritten by a rule that pops an index from the stack. Because of this, derivations in the grammar of the form $s[\,] \to^* \alpha(s'[\,])$ or of the form $s[\,] \to^* \alpha$ can be divided in a unique way into subderivations representable by our items.

The above deduction system is conceptually very close to a system of equations that expresses the sum of weights of all derivations in the grammar, or $in(s^{\vdash})$, in terms of similar values of the form $in(s)$, which is the sum of weights of all subderivations $s[\,] \to^* \alpha$, and $in(s, s')$, which is the sum of weights of all subderivations $s[\,] \to^* \alpha(s'[\,])$. The equations are given in Figure 2.

Although the expressions look unwieldy, they

$$
\begin{aligned}
in(s_0) \;=\; & \sum_{s_0[\,] \to A()\ \langle w \rangle} w \;+ \\
& \sum_{s_0[\circ\circ] \to A(s_1[\,] \cdots s_j[\circ\circ] \cdots s_m[\,])\ \langle w \rangle} w \cdot in(s_1) \cdot \ldots \cdot in(s_m) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to s_1[\iota\circ\circ]\ \langle w \rangle \\ s_2[\iota\circ\circ] \to s_3[\circ\circ]\ \langle v \rangle}} w \cdot v \cdot in(s_1, s_2) \cdot in(s_3) \\
in(s_0, s) \;=\; & \delta(s_0 = s) \;+ \\
& \sum_{s_0[\circ\circ] \to A(s_1[\,] \cdots s_j[\circ\circ] \cdots s_m[\,])\ \langle w \rangle} w \cdot in(s_1) \cdot \ldots \cdot in(s_j, s) \cdot \ldots \cdot in(s_m) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to s_1[\iota\circ\circ]\ \langle w \rangle \\ s_2[\iota\circ\circ] \to s_3[\circ\circ]\ \langle v \rangle}} w \cdot v \cdot in(s_1, s_2) \cdot in(s_3, s)
\end{aligned}
$$

Figure 2: The sum of weights of all derivations in a WLIG, or $in(s^{\vdash})$, is defined by the smallest non-negative solution to a system of equations. The function δ with a boolean argument evaluates to 1 if the condition is true and to 0 otherwise.

express exactly the 'inside' value of the weighted context-free grammar that we can extract out of the deduction system from Figure 1, by instantiating the inference rules in all possible ways, and then taking the consequent as the left-hand side of a rule, and the antecedent as the right-hand side. The weight is the product of weights of rules that appear in the side conditions. It is possible to effectively solve the system of equations, as shown by (Wojtczak and Etessami, 2007).

In the same vein we can compute 'outside' values for weighted linear indexed grammars, as straightforward analogues of the outside values of weighted and probabilistic context-free grammars. The outside value is the sum of weights of partial derivations that may lie 'outside' a subderivation $s[\,] \to^* \alpha$ in the case of $out(s)$, or a subderivation $s[\,] \to^* \alpha(s'[\,])$ in the case of $out(s, s')$. The equations in Figure 3 again follow trivially from the view of Figure 1 as weighted context-free grammar and the usual definition of outside values.

The functions in and out are particularly useful for PLIGs in TAG-normal form, as they allow the expected number of occurrences of state s to be expressed as:

$$E(s) \;=\; in(s) \cdot out(s)$$

Similarly, the expected number of subderivations of the form $s[\,] \to^* \alpha(s'[\,])$ is:

$$E(s, s') \;=\; in(s, s') \cdot out(s, s')$$

We will return to this issue in Section 5.

4 Weighted intersection

Before we discuss intersection on the level of trees, we first show how a well-established type of intersection on the level of strings, with weighted context-free grammars and weighted finite automata (WFAs), can be trivially extended to replace CFGs with RTGs or LIGs. The intersection paradigm is originally due to (Bar-Hillel et al., 1964). Extension to tree adjoining grammars and linear indexed grammars was proposed before by (Lang, 1994) and (Vijay-Shanker and Weir, 1993b).

4.1 Intersection of string languages

Let us assume a WLIG $\mathcal{G}$ with terminal and nonterminal labels. Furthermore, we assume a weighted finite automaton $\mathcal{A}$, with an input alphabet equal to the set of terminal labels of $\mathcal{G}$. The transitions of $\mathcal{A}$ are of the form:

$$q \overset{a}{\mapsto} q'\ \langle w \rangle,$$

where q and q' are states, a is a terminal symbol, and w is a weight. To simplify the presentation,

$$
\begin{aligned}
out(s') \;=\;& \delta(s' = s^{\vdash}) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to A(s_1[\,] \,\cdots\, s_j[\circ\circ] \,\cdots\, s_m[\,])\ \langle w \rangle \\ k \in \{1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_m\} \text{ s.t. } s' = s_k}} w \cdot out(s_0, s) \cdot in(s_j, s) \cdot \prod_{p \notin \{j,k\}} in(s_p) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to A(s_1[\,] \,\cdots\, s_j[\circ\circ] \,\cdots\, s_m[\,])\ \langle w \rangle \\ k \in \{1, \ldots, s_m\} \text{ s.t. } s' = s_k}} w \cdot out(s_0) \cdot \prod_{p \neq k} in(s_p) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to s_1[\iota\circ\circ]\ \langle w \rangle \\ s_2[\iota\circ\circ] \to s'[\circ\circ]\ \langle v \rangle}} w \cdot v \cdot out(s_0) \cdot in(s_1, s_2) \\
out(s', s) \;=\;& \sum_{\substack{s_0[\circ\circ] \to A(s_1[\,] \,\cdots\, s_j[\circ\circ] \,\cdots\, s_m[\,])\ \langle w \rangle \\ s' = s_j}} w \cdot out(s_0, s) \cdot \prod_{p \neq j} in(s_p) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to s'[\iota\circ\circ]\ \langle w \rangle \\ s[\iota\circ\circ] \to s_3[\circ\circ]\ \langle v \rangle}} w \cdot v \cdot out(s_0, s_4) \cdot in(s_3, s_4) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to s_1[\iota\circ\circ]\ \langle w \rangle \\ s_2[\iota\circ\circ] \to s'[\circ\circ]\ \langle v \rangle}} w \cdot v \cdot out(s_0, s) \cdot in(s_1, s_2) \;+ \\
& \sum_{\substack{s_0[\circ\circ] \to s'[\iota\circ\circ]\ \langle w \rangle \\ s[\iota\circ\circ] \to s_3[\circ\circ]\ \langle v \rangle}} w \cdot v \cdot out(s_0) \cdot in(s_3)
\end{aligned}
$$

Figure 3: The outside values in a WLIG.

we ignore epsilon transitions, and assume there is a unique initial state $q^{\vdash}$ and a unique final state $q^{\dashv}$.

We can construct a new WLIG $\mathcal{G}'$ whose generated language is the intersection of the language generated by $\mathcal{G}$ and the language accepted by $\mathcal{A}$. The rules of $\mathcal{G}'$ are:

1. $(q_0, s_0, q_m)[\circ\circ] \to$
   $A($ $(q_0, s_1, q_1)[\,] \cdots$
   $(q_{j-2}, s_{j-1}, q_{j-1})[\,]$
   $(q_{j-1}, s_j, q_j)[\circ\circ]$
   $(q_j, s_{j+1}, q_{j+1})[\,] \cdots$
   $(q_{m-1}, s_m, q_m)[\,]$ $)$ $\langle w \rangle$,
   for each rule $s_0[\circ\circ] \to A(s_1[\,] \cdots s_{j-1}[\,]$ $s_j[\circ\circ]$ $s_{j+1}[\,] \cdots s_m[\,])$ $\langle w \rangle$ from $\mathcal{G}$ and sequence $q_0, \ldots, q_m$ of states from $\mathcal{A}$;

2. $(q, s, q)[\,] \to A()\ \langle w \rangle$, for each rule $s[\,] \to A()\ \langle w \rangle$ from $\mathcal{G}$ and state q from $\mathcal{A}$;

3. $(q, s, q')[\,] \to a\ \langle w \cdot v \rangle$, for each rule $s[\,] \to a\ \langle w \rangle$ from $\mathcal{G}$ and transition $q \overset{a}{\mapsto} q'\ \langle v \rangle$ from $\mathcal{A}$;

4. $(q, s, q')[\circ\circ] \to (q, s', q')[\iota\circ\circ]\ \langle w \rangle$, for each rule $s[\circ\circ] \to s'[\iota\circ\circ]\ \langle w \rangle$ from $\mathcal{G}$ and states q, q' from $\mathcal{A}$;

5. $(q, s, q')[\iota\circ\circ] \to (q, s', q')[\circ\circ]\ \langle w \rangle$, for each rule $s[\iota\circ\circ] \to s'[\circ\circ]\ \langle w \rangle$ from $\mathcal{G}$ and states q, q' from $\mathcal{A}$.

The new states (q, s, q') give (left-most) derivations in $\mathcal{G}'$ that each simultaneously represent one (left-most) derivation in $\mathcal{G}$ of a certain substring, starting from state s, and one sequence of transitions taking the automaton $\mathcal{A}$ from state q to state q' while scanning the same substring. The initial state of $\mathcal{G}'$ is naturally $(q^{\vdash}, s^{\vdash}, q^{\dashv})$, which derives strings in the intersection of the original two languages.

Further note that each derivation in $\mathcal{G}'$ has a weight that is the product of the weight of the cor-

responding derivation in $\mathcal{G}$ and the weight of the corresponding sequence of transitions in $\mathcal{A}$. This allows a range of useful applications. For example, if $\mathcal{A}$ is deterministic (the minimum requirement is in fact absence of ambiguity) and if it assigns the weight one to all transitions, then $\mathcal{G}'$ generates a set of trees that is exactly the subset of trees generated by $\mathcal{G}$ whose yields are accepted by $\mathcal{A}$. Furthermore, the weights of those derivations are preserved. If $\mathcal{G}$ is a consistent PLIG in TAG-normal form, and if $\mathcal{A}$ accepts the language of all strings containing a fixed substring x, then the sum of probabilities of all derivations in $\mathcal{G}'$ gives the substring probability of x. The effective computation of this probability was addressed in Section 3.

An even more restricted, but perhaps more familiar case is if $\mathcal{A}$ is a linear structure that accepts a single input string y of length n. Then $\mathcal{G}'$ generates exactly the set of trees generated by $\mathcal{G}$ that have y as yield. In other words, the string y is thereby parsed.

If $\mathcal{G}$ is *binary*, i.e. all rules have at most two states in the right-hand side, then $\mathcal{G}'$ has a size that is cubic in n. This may seem surprising, in the light of the awareness that practical parsing algorithms for tree adjoining grammars have a time complexity of no less than $\mathcal{O}(n^6)$. However, in order to solve the recognition problem, an analysis is needed to determine whether $\mathcal{G}'$ allows at least one derivation.

The analysis from Figure 1 requires $\mathcal{O}(|S'| + |R'_1| \cdot |S'| + |R'_2| + |R'_3| \cdot |R'_4| \cdot |S'|)$ steps, where $|S'| = \mathcal{O}(n^2)$ is the number of states of $\mathcal{G}'$, and $|R'_1| = \mathcal{O}(n^3)$, $|R'_2| = |R'_3| = |R'_4| = \mathcal{O}(n^2)$ are the numbers of rules of $\mathcal{G}'$, divided into the four main types. This leads to an overall time complexity of $\mathcal{O}(n^6)$, as expected.

The observation that recognition can be harder than parsing was made before by (Lang, 1994). The central new insight this provided was that the notion of 'parsing' is ill-defined in the literature. One may choose a form in which to capture all parses of an input allowed by a grammar, but different such forms may incur different costs of extracting individual parse trees.

In Section 6.2 we will consider the complexity of parsing and recognition if $\mathcal{G}$ is not binary.

4.2 Intersection of tree languages

We now shift our attention from strings to trees, and consider the intersection of the tree language generated by a weighted linear indexed grammar $\mathcal{G}_1$ and the tree language generated by a weighted regular tree grammar $\mathcal{G}_2$. This intersection is generated by another weighted linear indexed grammar $\mathcal{G}$, which has the following rules:

1. $(s_0, q_0)[\circ\circ] \rightarrow A(\ (s_1, q_1)[\,] \cdots (s_{j-1}, q_{j-1})[\,] (s_j, q_j)[\circ\circ] (s_{j+1}, q_{j+1})[\,] \cdots (s_m, q_m)[\,]\ )\ \langle w \cdot v \rangle$, for each rule $s_0[\circ\circ] \rightarrow A(s_1[\,] \cdots s_{j-1}[\,]\ s_j[\circ\circ]\ s_{j+1}[\,] \cdots s_m[\,])\ \langle w \rangle$ from $\mathcal{G}_1$ and each rule $q_0 \rightarrow A(q_1 \cdots q_m)\ \langle v \rangle$ from $\mathcal{G}_2$;

2. $(s, q)[\,] \rightarrow A()\ \langle w \cdot v \rangle$, for each rule $s[\,] \rightarrow A()\ \langle w \rangle$ from $\mathcal{G}_1$ and each rule $q \rightarrow A()\ \langle v \rangle$ from $\mathcal{G}_2$;

3. $(s, q)[\circ\circ] \rightarrow (s', q)[\iota\circ\circ]\ \langle w \rangle$, for each rule $s[\circ\circ] \rightarrow s'[\iota\circ\circ]\ \langle w \rangle$ from $\mathcal{G}_1$ and state q from $\mathcal{G}_2$;

4. $(s, q)[\iota\circ\circ] \rightarrow (s', q)[\circ\circ]\ \langle w \rangle$, for each rule $s[\iota\circ\circ] \rightarrow s'[\circ\circ]\ \langle w \rangle$ from $\mathcal{G}_1$ and state q from $\mathcal{G}_2$.

Much as in the previous section, each (leftmost) derivation in $\mathcal{G}$ corresponds to one (leftmost) derivation in $\mathcal{G}_1$ and one in $\mathcal{G}_2$. Furthermore, these three derivations derive the same labelled tree, and a derivation in $\mathcal{G}$ has a weight that is the product of the weights of the corresponding derivations in $\mathcal{G}_1$ and $\mathcal{G}_2$.

It can be instructive to look at special cases. Suppose that $\mathcal{G}_2$ is an unambiguous regular tree grammar of size $\mathcal{O}(n)$ generating a single tree t with n vertices, assigning weight one to all its rules. Then the above construction can be seen as *parsing* of that tree t. The sum of weights of derivations in $\mathcal{G}$ then gives the weight of the tree in $\mathcal{G}_1$. See Section 3 once more for a general way to compute this weight, as the inside value of the initial state of $\mathcal{G}$, which is naturally $(s^{\vdash}, q^{\vdash})$.

In order to do recognition of t, or in other words, to determine whether $\mathcal{G}$ allows at least one derivation, the analysis from Figure 1 can be used, which has time complexity $\mathcal{O}(|S| + |R_1| \cdot |S| + |R_2| + |R_3| \cdot |R_4| \cdot |S|)$, where $|S| = \mathcal{O}(n)$ is the number of states of $\mathcal{G}$, and the numbers of rules are $|R_1| = \mathcal{O}(n)$, $|R_2| = |R_3| = |R_4| = \mathcal{O}(n)$. Note that $|R_1| = \mathcal{O}(n)$ because we have assumed that $\mathcal{G}_2$ allows only one derivation of one tree t,

hence q_0 uniquely determines $q_1, \ldots, q_m$. Overall, we obtain $\mathcal{O}(n^3)$ steps, which concurs with a known result about the complexity of TAG parsing of trees, as opposed to strings (Poller and Becker, 1998).

Another special case is if WLIG $\mathcal{G}_1$ simplifies to a WRTG (i.e. the stacks of indices remain always empty), which means we compute the intersection of two weighted regular tree grammars $\mathcal{G}_1$ and $\mathcal{G}_2$. For recognition, or in other words to decide non-emptiness of the intersection, we can still use Figure 1, although now only inference rules (b) and (d) are applicable (with a small refinement to the algorithm we can block spurious application of (a) where no rules exist that pop indices.) The complexity is determined by (d), which requires $\mathcal{O}(|\mathcal{G}_1| \cdot |\mathcal{G}_2|)$ steps.

5 Parameter estimation

PLIGs allow finer description of probability distributions than PRTG, both over string languages and over tree languages. However, the (string) parsing complexity of regular tree grammars is $\mathcal{O}(n^3)$ and that of LIGs is $\mathcal{O}(n^6)$. It may therefore be preferable for reasons of performance to do parsing with a PRTG even when a PTAGs or PLIG is available with accurately trained probabilities. Alternatively, one may do both, with a PRTG used in a first phase to heuristically reduce the search space.

This section outlines how a suitable PRTG $\mathcal{G}_2$ can be extracted out of a PLIG $\mathcal{G}_1$, assuming the underlying RTG $\mathcal{G}_2'$ without weights is already given. The tree language generated by $\mathcal{G}_2'$ may be an approximation of that generated by $\mathcal{G}_1$. The objective is to make $\mathcal{G}_2$ as close as possible to $\mathcal{G}_1$ in terms of probability distributions over trees. We assume that $\mathcal{G}_2'$ is unambiguous, that is, for each tree it generates, there is at most one derivation.

The procedure is a variant of the one described by (Nederhof, 2005). The idea is that derivations in $\mathcal{G}_1$ are mapped to those in $\mathcal{G}_2'$, via the trees in the intersection of the two tree languages. The probability distribution of states and rules in $\mathcal{G}_2$ is estimated based on the expected frequencies of states and rules from $\mathcal{G}_2'$ in the intersection.

Concretely, we turn the RTG $\mathcal{G}_2'$ into a PRTG $\mathcal{G}_2''$ that is obtained simply be assigning weight one to all rules. We then compute the intersection grammar $\mathcal{G}$ as in Section 4.2. Subsequently, the inside and outside values are computed for $\mathcal{G}$, as explained in Section 3. The expected number of occurrences of a rule in $\mathcal{G}$ of the form:

$$\begin{aligned}(s_0, q_0)[\circ\circ] \rightarrow A(\ & (s_1, q_1)[\,] \cdots \\ & (s_{j-1}, q_{j-1})[\,] \\ & (s_j, q_j)[\circ\circ] \\ & (s_{j+1}, q_{j+1})[\,] \cdots \\ & (s_m, q_m)[\,]\ )\ \langle w \cdot v \rangle,\end{aligned}$$

is given by multiplying the outside and inside probabilities and the rule probability, as usual. We get two terms however that we need to sum. The intuition is that we must count both rule occurrences used for building initial TAG trees and those used for building auxiliary TAG trees. This gives:

$$\begin{aligned}& w \cdot v \cdot out((s_0, q_0)) \cdot \prod_k in((s_k, q_k)) \ + \\ & w \cdot v \cdot \sum_{s,q} out((s_0, q_0), (s, q)) \cdot \\ & \qquad in((s_j, q_j), (s, q)) \cdot \prod_{k \neq j} in((s_k, q_k))\end{aligned}$$

By summing these expected numbers for different rules $s_0[\circ\circ] \rightarrow A(s_1[\,] \cdots s_{j-1}[\,]\ s_j[\circ\circ]\ s_{j+1}[\,] \cdots s_m[\,])$, we obtain the expected number of occurrences of $q_0 \rightarrow A(q_1 \cdots q_m)$, Let us denote this sum by $E(q_0 \rightarrow A(q_1 \cdots q_m))$. By summing these for fixed q_0, we obtain the expected number of occurrences of q_0, which we denote by $E(q_0)$. The probability of $q_0 \rightarrow A(q_1 \cdots q_m)$ in $\mathcal{G}_2$ is then set to be the ratio of $E(q_0 \rightarrow A(q_1 \cdots q_m))$ and $E(q_0)$.

By this procedure, the Kullback-Leibler distance between $\mathcal{G}_1$ and $\mathcal{G}_2$ is minimized. Although the present paper deals with very different formalisms, the proof of correctness is identical to that in (Nederhof, 2005). The reason is that in both cases the mathematical analysis must focus on the objects in the intersection (strings or trees) which may correspond to multiple derivations in the original model (here $\mathcal{G}_1$) but to a single derivation in the unambiguous model to be trained (here $\mathcal{G}_2$), and each derivation is composed of rules, whose probabilities are to be multiplied.

6 Extensions

6.1 Transduction

For various formalisms describing (string or tree) languages, there are straightforward generalizations that describe a relation between two or more

languages, which is known as a transduction. The idea is that the underlying control mechanism, such as the states in regular tree grammars or linear indexed grammars, is now coupled to two or more surface forms that are synchronously produced. For example, a rule in a weighted *synchronous* regular tree grammar (WSRTG) has the form:

$$s_0 \rightarrow A(s_1 \cdots s_m), B(s_{\pi(1)} \cdots s_{\pi(m)})\ \langle w \rangle,$$

where π is a permutation of $1, \ldots, m$. We can generalize this to having a third label C and a second permutation π', in order to describe simultaneous relations between three tree languages, etc. In this section we will restrict ourselves to binary relations however, and call the first surface form the input and the second surface form the output. For synchronous tree adjoining grammars, see for example (Shieber, 1994).

If we apply intersection on the input or on the output of a synchronous grammar formalism, then this is best seen as composition. This is well-known in the case of finite-state transducers and some forms of context-free transduction (Berstel, 1979), and application to a wider range of formalisms is gaining interest in the area of machine translation (Knight, 2007).

With the intersection from Section 4.2 trivially extended to composition, we can now implement composition of the form:

$$\tau_1 \circ \ldots \circ \tau_k,$$

where the different τ_j are transducers, of which $k - 1$ are (W)SRTGs and at most one is a (weighted) synchronous LIG ((W)SLIG). The result of the composition is another (W)SLIG. It should be noted that a (W)RTS (or (W)LIG) can be seen as a (W)SRTG (or (W)SLIG, respectively) that represents the identity relation on its tree language.

6.2 Binarization

In the discussion of complexity in Section 4.1, we assumed that rules are binary, that is, that they have at most two states in each right-hand side. However, whereas any context-free grammar can be transformed into a binary form (e.g. Chomsky normal form), the grammars as we have defined them cannot be. We will show that this is to a large extent a consequence of our definitions, which were motivated by presentational ease, rather than by generality.

The main problem is formed by rules of the form $s_0 \rightarrow A(s_1 \cdots s_m)$, where $m > 2$. Such long rules cannot be broken up into shorter rules of the same form, as this would require an additional labelled vertex, changing the tree language. An apparent solution lies in allowing branching rules without any label, for example $s_1 \rightarrow s_2\, s_3$. Regrettably this could create substantial computational problems for intersection of the described tree languages. As labels provide the mechanism through which to intersect tree languages, rules of the above form are somewhat similar to unit rules or epsilon rules in context-free grammars, in that they are not bound to observable elements. Branching rules furthermore have the potential to generate context-free languages, and therefore they are more pernicious to intersection, considering that emptiness of intersection of context-free languages is undecidable.

It therefore seems better to restrict branching rules $s_1 \rightarrow s_2\, s_3$ to finite-state power, for example by making these rules exclusively left-linear or right-linear. A more elegant but equivalent way of looking at this may be to have rules of the form:

$$s_0 \rightarrow A(\mathcal{R}),$$

where $\mathcal{R}$ is a regular language over states. In the case of linear indexed grammars, we would have rules of the form:

$$s[\circ\circ] \rightarrow A(\mathcal{L}\ s'[\circ\circ]\ \mathcal{R})$$

where $\mathcal{L}$ and $\mathcal{R}$ are regular languages over expressions of the form $s[\,]$. Appropriate weighted finite automata can be used to assign weights to sequences of such expressions in $\mathcal{L}$ and $\mathcal{R}$. With these extended types of rules, our construction from Section 4.2 still works. The key observation here is that regular languages are closed under intersection.

One of the implications of the above extended definitions is that labels appear not only without fixed ranks, as we have assumed from the start in Section 2, but even without a bound on the rank. Concretely, a vertex may appear with any number of children in a tree. Whereas this may be unconventional in certain areas of formal language theory, it is a well-accepted practice in the parsing of natural language to make the number of constituents of syntactic categories flexible and conceptually unbounded; see for example

(Collins, 1997). Also the literature on unranked tree automata is very relevant; see for example (Schwentick, 2007). Binarization for LIGs was considered before by (Vijay-Shanker and Weir, 1993a).

6.3 Beyond TAGs

In the light of results by (Kepser and Mönnich, 2006) it is relatively straightforward to consider larger classes of linear context-free tree grammars in place of tree-adjoining grammars, in order to generalize the construction in Section 4.2.

The generalization described in what follows seems less straightforward. Context-free languages can be characterized in terms of parse trees in which path sets (sets of strings of labels on paths from the root to a leaf) are regular. In the case of tree adjoining languages, the path sets are context-free. There is a hierarchy of classes of languages in which the third step is to consider path sets that are tree adjoining languages (Weir, 1992). In this paper, we have considered the parsing-as-intersection paradigm for the first two members of the hierarchy. It may be possible that the paradigm is also applicable to the third and following members. This avenue is yet to be pursued.

7 Conclusions

This paper has extended the parsing-as-intersection paradigm from string languages to tree languages. Probabilities, or weights in general, were incorporated in this framework in a natural way. We have discussed one particular application involving a special case of the extended paradigm.

Acknowledgements

Helpful comments by anonymous reviewers are gratefully acknowledged. The basic result from Section 4.1 as it pertains to RTGs as subclass of LIGs was discussed with Heiko Vogler, who proposed two alternative proofs. Sylvain Schmitz pointed out to me the relevance of literature on linear context-free tree languages.

References

A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. 2007. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley.

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

J. Berstel. 1979. *Transductions and Context-Free Languages*. B.G. Teubner, Stuttgart.

M. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *35th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 16–23, Madrid, Spain, July.

J. Graehl and K. Knight. 2004. Training tree transducers. In *HLT-NAACL 2004, Proceedings of the Main Conference*, Boston, Massachusetts, USA, May.

F. Gcseg and M. Steinby. 1997. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, chapter 1, pages 1–68. Springer, Berlin.

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

A.K. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages. Vol 3: Beyond Words*, chapter 2, pages 69–123. Springer-Verlag, Berlin/Heidelberg/New York.

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

S. Kepser and U. Mönnich. 2006. Closure properties of linear context-free tree languages with an application to optimality theory. *Theoretical Computer Science*, 354:82–97.

K. Knight. 2007. Capturing practical natural language transformations. *Machine Translation*, 21:121–133.

B. Lang. 1994. Recognition can be harder than parsing. *Computational Intelligence*, 10(4):486–494.

M.-J. Nederhof and G. Satta. 2003. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, LORIA, Nancy, France, April.

M.-J. Nederhof. 2005. A general technique to train language models on language models. *Computational Linguistics*, 31(2):173–185.

P. Poller and T. Becker. 1998. Two-step TAG parsing revisited. In *Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 143–146. Institute for Research in Cognitive Science, University of Pennsylvania, August.

P. Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proc. of the fifteenth International Conference on Computational Linguistics*, pages 418–424. Nantes, August.

A. Sarkar. 1998. Conditions on consistency of probabilistic tree adjoining grammars. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 2, pages 1164–1170, Montreal, Quebec, Canada, August.

Y. Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proc. of the fifteenth International Conference on Computational Linguistics*, pages 426–432. Nantes, August.

Thomas Schwentick. 2007. Automata for XML–a survey. *Journal of Computer and System Sciences*, 73:289–315.

S.M. Shieber. 1994. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385.

K. Sima'an. 1997. Efficient disambiguation by means of stochastic tree substitution grammars. In D. Jones and H. Somers, editors, *New Methods in Language Processing*. UCL Press, UK.

S. Sippu and E. Soisalon-Soininen. 1988. *Parsing Theory, Vol. I: Languages and Parsing*, volume 15 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

K. Vijay-Shanker and D.J. Weir. 1993a. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

K. Vijay-Shanker and D.J. Weir. 1993b. The use of shared forests in tree adjoining grammar parsing. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 384–393, Utrecht, The Netherlands, April.

K. Vijay-Shanker and D.J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–546.

D.J. Weir. 1992. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104:235–261.

D. Wojtczak and K. Etessami. 2007. PReMo: an analyzer for Probabilistic Recursive Models. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference*, volume 4424 of *Lecture Notes in Computer Science*, pages 66–71, Braga, Portugal. Springer-Verlag.