A Bottom-up Automaton for Tree-Adjoining Languages

Petter Ericson[(A)]

[(A)]Computing Science Department
Umeå University
901 87 Umeå, Sweden
{pettter}@cs.umu.se

Abstract
Current tree parsing algorithms for nonregular tree languages all have superlinear running times, possibly limiting their practical applicability. We present a bottom-up tree automaton that captures exactly the tree-adjoining languages in the non-deterministic case. The determinstic case captures a strict superset of the regular tree languages, while preserving running times linear in the size of the tree.

1. Introduction

Though much recent research in tree automata theory has focused on various subsets of the regular tree languages (RTL), there have also been some interest in strictly more powerful formalisms. This technical report deals with one of the latter. In particular, it defines a class of languages recognisable in linear time that lies between RTL and the class of Tree Adjoining Languages (TAL), analogous to how the deterministic context-free languages is an intermediate class between REG and CFL.

The class of Tree Adjoining Languages [4] have seen use in various contexts, notably natural language processing (syntax trees) [3] and bioinformatics (RNA structure prediction) [8]. However, while its theoretical properties have been well-studied in the string case, the tree structure is less so. Specifically, one of the main aims of the current work is to investigate alternate formalisms that define the same class of tree languages, including a new automaton model.

2. Related work

Previous research into defining automata for strict supersets of the regular tree languages have mostly focused at the context-free tree languages. In particular, the work by Guessarian [2] and Schimpf and Gallier [7] on various definitions of pushdown tree automata both define this class.

However, for various reasons these automata are somewhat unsatisfactory. In particular, recognising the complete class of context-free tree languages requires quite a lot of computational power, meaning the complexity of parsing will be necessarily quite high.

A much more relevant construction is the linear pushdown automata defined by Fujiyoshi and Kasai [1]. In particular, the class of tree languages accepted by linear pushdown automata have been proven by Kepser and Rogers [6] to be strongly equivalent to the tree-adjoining grammars. Our contribution is relatively simple given this background: we use bottom-up instead of top-down automata. However, this does give us the opportunity to define an intermediate class between RTL and TAL by requiring the automata to be deterministic.

3. Preliminaries

Notation In general, we will use lowercase Latin letters for terminal symbols, and uppercase for nonterminals. Lowercase Greek letters represent stack symbols, while q with various subscripts represent states. In depicting trees, soft parentheses (()) are used. Hard parentheses ([]) are used for stack operations. The symbols λ and ε represent the empty tree and string respectively.

Given a finite alphabet Σ, we write Σ^* for the set of all finite strings over Σ. For $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, \ldots, n\}$. When we use the word "tree", we mean rooted, ordered, node-labelled trees. Furthermore, trees (and the alphabets used to construct them) are ranked. That is, there is a function $rk : \Sigma \to \mathbb{N}$ associating every symbol with a rank, and every node labelled with the symbol a has exactly $rk(a)$ subtrees. Given a ranked alphabet Σ, the set Σ_k denotes the set of symbols of Σ with rank k. T_Σ is the set of all trees over the (ranked) alphabet Σ.

Tree formalisms In linguistic applications, Tree-adjoining Grammars (TAGs) are in general used and analysed as working towards an output string, rather than an output tree. Specifically, parsing algorithms for TAGs have been focused on the string case; see, e.g., [5]. In exploring the adjoined trees themselves further, the notion of *tree parsing* is more relevant. However, in this report, we treat the slightly simpler *tree membership* problem, i.e., given a tree t and a grammar G, determine whether it belongs to $\mathcal{L}(G)$. It is known that for every TAG, the corresponding tree membership problem is solvable in time $\mathcal{O}(n^3)$, where n is the size of the input tree; see, e.g., [4].

Whereas such tree verification is relatively simple (and finite) for Regular Tree Languages (RTL), TAGs require some bookkeeping to identify where trees have been adjoined. In order to explore this phenomenon further, it is helpful to first have a model of a finite tree automaton which can be used to recognise trees.

Though some knowledge of tree automata and tree language theory is assumed, we will restate various definitions and assumptions that are useful to clarify.

Definition 3.1 (Nondeterministic tree automaton) *Recall that a* nondeterministic tree automaton (NTA) *is a structure* $A = (\Sigma, Q, R, F)$*, where* Σ *is a ranked alphabet,* Q *and* $F \subset Q$ *a set of states and final states respectively, and* R *is a set of rules on the form*

$$a(q_1, \ldots, q_k) \to q$$

where $a \in \Sigma_k$*,* $q_1, \ldots, q_k, q \in Q$*.*

The semantics of NTA are assumed to be well-known to the reader, as are regular tree grammars (RTG), but as the latter are not relevant to the current subject, we instead consider the more powerful formalism:

Definition 3.2 (Context-free tree grammar) *A* context-free tree grammar *(CFTG) is a quadruple* $G = (\Sigma, N, R, S)$*, where* Σ *is the (ranked) terminal alphabet,* N *is a ranked alphabet of nonterminals,* $S \in N$ *is the starting nonterminal, and* R *is a set of rules on the form*

$$A(x_1, \ldots, x_k) \to t$$

where $A \in N_k$*,* $\{x_1, \ldots, x_k\} = X_k$ *are variables (of rank 0) and* $t \in T_{\Sigma \bigcup N \bigcup X_k}$ *is the output tree.*

A context-free tree grammar is called linear *(LCFTG) if no variable occurs more than once in any right-hand side* t*. A* k*-CFTG is a context-free tree grammar where, for all* $l > k$*,* $N_l = \emptyset$*.*

The semantics of CFTG are similar to RTG in that nonterminals are successively replaced by their right-hand sides until a tree in T_Σ is obtained. However, as nonterminals are no longer only of rank 0, they may be internal nodes, and have subtrees of their own. In this case, a derivation step involves replacing a nonterminal A with the output side t, and then replacing all occurences of x_i with what was the ith subtree of A.

The *yield* of a tree is the string of symbols acquired by reading the leaves from left to right. We extend this notion to (tree) languages in the usual way: $yield(L) = \{w : t \in L, w = yield(t)\}$ for L a tree language.

4. TAG equivalent formalisms

While the TAG formalism is the one most used in practise (in NLP), it is somewhat unwieldy when regarded as a tree generating device. We will instead rely on the result by Kepser and Rogers [6] that monadic linear context-free grammars are strongly equivalent to (non-strict) TAGs, i.e. with some minor relaxations, TAGs define the same class of tree languages as 1-LCFTG.

The main contribution in this paper is the automaton defined next. We will later prove that this is computationally equivalent to 1-LCFTG, and thus useful in order to gain a deeper understanding into the tree adjoining languages.

Definition 4.1 (1-stack bottom-up pushdown tree automaton) *A 1-stack bottom-up pushdown tree automaton (1-PTA) is a structure $A = (\Sigma, \Gamma, Q, R, F)$ where Σ is the (ranked) tree alphabet, $\Gamma = \Gamma_0 \cup \Gamma_1$ is the stack alphabet, $Q = Q_1$ (states have exactly one subtree - the stack) is the set of states where $F \subset (Q \times \Gamma_0)$ is a set of final state-stack combinations, and R is a set of rules on either of these forms:*

1. $a(q_1[\pi_1], \ldots, q_i[\pi_i], \ldots, q_k[\pi_k]) \to q'[\pi']$
2. $q[\pi] \to q'[\pi']$

where $q, q', q_1 \ldots q_k \in Q$, $a \in \Sigma_k$, $\pi_j \in \Gamma_0$ for $j \neq i$, and either $\pi, \pi_i \in \Gamma_0$, $\pi' \in \Gamma_1^*\Gamma_0$, or $\pi, \pi_i \in \Gamma_1$, $\pi' \in \Gamma_1^*$.

The semantics of 1-PTA are relatively closely related to those of regular bottom-up tree automata, with the addition of a stack. However, note that while several states are involved whenever a symbol of rank greater than 1 is involved, at no point is more than one stack of height greater than 1 considered. Indeed, in any computation, all but one of the child stacks need to have a symbol in Γ_0 at its head, i.e. there must have been an active transition emptying the stack down to its last symbol, if one was used in the computations in that subtree. As F is defined as a subset of $(Q \times \Gamma_0)$ this is also true for the complete tree. That is, before accepting a tree, the automaton must completely discard everything but the last symbol of its final stack. This is more or less equivalent to having a specific class (Q_0) of non-stack carrying states, as $(Q \times \Gamma_0)$ is a finite set. It is also possible to define Γ_0 as having a single symbol with no loss of computational power.

We restate the definition of linear pushdown tree automata used in [1], in order to show its equivalence to 1-PTA. However, we do it directly, instead of starting with the complete PTA as defined by Guessarian in [2] and restricting it.

Definition 4.2 (Linear pushdown tree automaton) *A linear pushdown tree automaton is a structure $A = (\Sigma, \Gamma, Q, R, q_0, \pi_0)$ where Σ is the (ranked) tree alphabet, $\Gamma = \Gamma_0 \cup \Gamma_1$ is the stack alphabet, $Q = Q_2$ is a ranked alphabet of states (each with two subtrees - the subtree left to process and the stack) where q_0 is the initial state, π_0 is the initial stack symbol and R is a set of rules on either of these forms:*

1. $q(a(x_1, \ldots, x_k), \pi) \to a(q_1(x_1, \pi_1), \ldots, q_i(x_i, \pi_i), \ldots, q_k(x_k, \pi_k))$
2. $q(x, \pi) \to q'(x, \pi')$
3. $q(b, \pi_f) \to b$

where $q, q', q_1 \ldots q_k \in Q$, $a \in \Sigma_k$, $b \in \Sigma_0$, $\pi_f \in \Gamma_0$, $\pi_j \in \Gamma_1^*\Gamma_0$ for $j \neq i$, and either $\pi \in \Gamma_0$, $\pi', \pi_i \in \Gamma_1^*\Gamma_0$ or $\pi \in \Gamma_1$, $\pi', \pi_i \in \Gamma_1^*$

Lemma 4.3 *1-PTA are strongly equivalent to extended TAGs*

Proof. From the above definitions, it should be fairly clear that simply inverting the rules takes us a long way towards having the two automata models being equal. Standard automata theoretic tools will suffice to handle the remaining detail: the strings of stack symbols that

appear in the right-hand sides. Simply add a sequence of rules on form (2) that read each symbol of the string, and push the appropriate symbol onto the stack as a final action. This gives us a complete symmetry between 1-PTA and linear pushdown tree automata. The lemma follows from the proofs in [6] and [1]

□

5. Deterministic tree languages

Deterministic linear pushdown tree automata cannot recognise e.g. the (regular) tree language $\{f(a,b), f(b,a)\}$, which makes their usefulness somewhat limited as tree recognising devices. Deterministic 1-PTA in contrast capture all regular tree languages, and a subset of TAL which may be called the "deterministic tree adjoining languages" (DTAL), analogous to the deterministic context-free (string) languages. As deterministic 1-PTA runs in linear time, this is also a class of efficiently recognizable languages.

The motivation for using mildly context-sensitive, rather than just context-free, grammars in natural language processing applications is that context-free grammars cannot handle all features of natural languages. The most commonly cited such features are ones that have the basic structure of the copy language $\{ww \mid w \in \Sigma^+\}$ or of the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$. In this section, we observe that there are 1-LCFTGs whose yield correspond to these string languages, and such that their tree languages are recognised by deterministic 1-PTA.

We define a 1-LCFTG G_{copy} such that the yield language of $\mathcal{L}(G_{copy})$ is the copy language over $\{a,b\}$, i.e.

$$yield(\mathcal{L}(G_{copy})) = \{ww \mid w \in \{a,b\}^+\}.$$

The grammar G_{copy} is the tuple (Σ, N, R, S) where

- $\Sigma_0 = \{a,b\}$, $\Sigma_1 = \{f\}$, and $\Sigma_2 = \{g\}$,
- $N = (S, A)$, and
- R is the following set of rules:

$$\begin{array}{cccc} S \to g(a, f(a)) & S \to g(b, f(b)) & S \to g(a, A(f(a))) & S \to g(b, A(f(b))) \\ A(x) \to g(a, (g(x,a))) & A(x) \to g(b, (g(x,b))) & A(x) \to g(a, A(g(x,a))) & A(x) \to g(b, A(g(x,b))) \end{array}$$

An example derivation of G_{copy} is shown in Figure 1. The tree language $\mathcal{L}(G_{copy})$ is easily recognised by a deterministic 1-PTA. All it has to do is to keep a stack that works along the longest path of the derivation trees, pushes symbols at the unique f-labelled node and at g-labelled nodes with leafs as right children, and pops symbols at g-labelled nodes with leafs as left children.

In a similar fashion, we can define a 1-LCFTG whose yield language is

$$\{a^n b^n c^n \mid n \in \mathbb{N}\}.$$

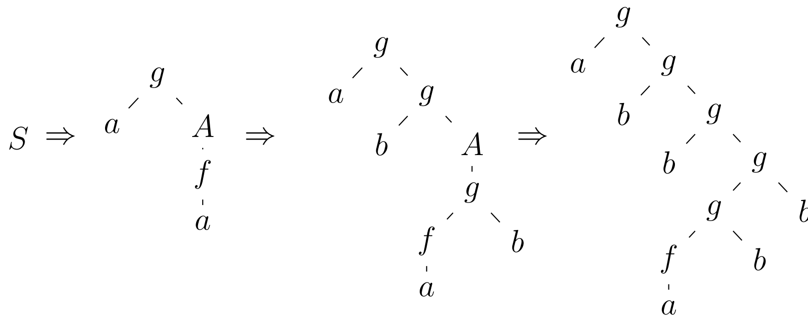

Figure 1: A derivation of G_{copy}. The final tree has yield *abbabb*.

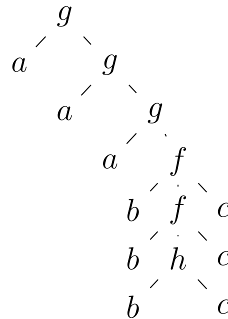

Figure 2: A derivation tree with yield *aaabbbccc*.

Figure 2 shows what a derivation tree of such a grammar might look like. Again, it is easy to see that the corresponding tree language is recognised by a deterministic 1-PTA. In this instance, the automaton pushes on its stack when reading an h- or g-labelled node with a leftmost b child and a rightmost c child. It pops when reading g-labelled nodes with a-labelled children to the left.

6. Future work

Having defined the DTAL class of tree languages, it remains to be explored exactly how useful this class is. Preliminary studies of linguistic treebanks appear to reveal that nonregular features in the actual tree structure are rare to nonexistent. Instead, cross-dependencies are represented by reordering of leaves leading to crossing branches (meaning the structures are no longer strictly trees) or similar measures. Indeed, it seems difficult in general to find actual real-world examples of TAG grammars utilizing non-regularity.

Furthermore, it is conjectured that (similarly to the DCFL case) the question “given a TAG G, is L(G) in DTAL” is undecidable.

7. Acknowledgements

We gratefulle acknowledge valuable comments from the anonymous reviewers of the (unpublished) conference version of this report. Moreover, the advisors of the author deserve many thanks for valuable insight during the writing process. Lastly, the financial support from the Swedish Research Council grant 621-2011-6080 is acknowledged with thanks.

References

[1] Akio Fujiyoshi and Takumi Kasai. Spinal-formed context-free tree grammars. *Theory of Computing Systems*, 33(1):59–83, 2000.

[2] Inène Guessarian. Pushdown tree automata. *Mathematical Systems Theory*, 16(1):237–263, 1983.

[3] A. K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions. In *Natural Language Parsing*, pages 206–250. Cambridge University Press, 1985.

[4] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer Berlin Heidelberg, 1997.

[5] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Springer, 2010.

[6] Stephan Kepser and James Rogers. The equivalence of tree adjoining grammars and monadic linear context-free tree grammars. In *The Mathematics of Language*, pages 129–144. Springer, 2010.

[7] Karl M Schimpf and Jean H Gallier. Tree pushdown automata. *Journal of Computer and System Sciences*, 30(1):25–40, 1985.

[8] Yasuo Uemura, Aki Hasegawa, Satoshi Kobayashi, and Takashi Yokomori. Tree adjoining grammars for rna structure prediction. *Theoretical computer science*, 210(2):277–303, 1999.