HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-2000-07

Regularly Extended Two-Way Nondeterministic Tree Automata*

Anne Brüggemann-Klein[†] Derick Wood[‡]

July 17, 2000

Abstract

We establish that regularly extended two-way nondeterministic tree automata with unranked alphabets have the same expressive power as regularly extended nondeterministic tree automata with unranked alphabets. We obtain this result by establishing regularly extended versions of a congruence on trees and of a congruence on, so called, views. Our motivation for the study of these tree models is the Extensible Markup Language (XML), a metalanguage for defining document grammars. Such grammars have regular sets of right-hand sides for their productions and tree automata provide an alternative and useful modeling tool for them. In particular, we believe that they provide a useful computational model for what we call caterpillar expressions.

1 Introduction

We became interested in regularly extended two-way tree automata (tree automata that have a regular set of transitions instead of a finite set and, thus, unbounded degree nodes) because of our work [3] in which we show that tree languages recognized by caterpillar expressions are tree regular. Initially, we planned to prove this result by using regularly extended two-way tree automata to emulate caterpillar expressions and then applying the main theorem of this paper; namely, we generalize Moriya's result [8] that demonstrates finite two-way tree automata have the same expressive power as finite bottom-up tree automata to regularly extended tree automata. Our proof of this result is, however, very

*The work of the authors was supported partially by a joint DAAD-HK grant. In addition, the work of the second author was supported under a grant from the Research Grants Council of Hong Kong SAR.

†Institut für Informatik, Technische Universität München, Arcisstr. 21, 80290 München, Germany. Email: brueggem@informatik.tu-muenchen.de

‡Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong SAR. Email: dwood@cs.ust.hk

different from Moriya's. We first establish an algebraic characterization of the languages of regularly extended two-way tree automata and then show that the languages of regularly extended two-way tree automata satisfy the characterization. Unfortunately, we were unable to design a generic emulation of caterpillar expressions with regularly extended two-way tree automata. Therefore, we ended up using the algebraic characterization to to prove that caterpillar expressions recognize tree regular languages.

Regularly extended two-way tree automata are also of interest in their own right since they provide greater programming flexibility than do regularly extended one-way tree automata in much the same way that two-way finite-state automata do when compared to one-way finite-state automata. This choice is motivated by the Standard Generalized Markup Language (SGML) [7] and the Extensible Markup Language (XML) [2], which are metalanguages for document grammars that rely on these requirements. Although most work on classes of documents is grammatical in nature, grammars are not always the most appropriate tool for modeling applications. Murata [9] has argued that regularly extended tree automata often provide a more appropriate framework for investigating tree transformations, tree query languages, layout generation for trees, and context specification and evaluation.

The research on tree automata and regular languages of trees can be divided into two categories: one dealing with ranked and the other with unranked alphabets. The bulk of the literature deals with finite, ranked alphabets. Gécseg and Steinby [5] have written a comprehensive book on tree automata and tree transducers over ranked alphabets (an updated survey by the same authors appeared recently [6]); see also the text of Comon and his collaborators [4]. Although ranked and unranked alphabets are both finite, the transition relations of the corresponding tree automata for ranked alphabets can only be finite whereas the transition relations of the corresponding tree automata for unranked alphabets need not be finite. We consider the transition relation to be either regular or finite in the unranked case. We write **finite tree automaton** to mean that the tree automaton has a finite transition relation and we write **(regularly) extended tree automaton** to mean that the tree automaton has a regular transition relation.

Tree automata for unranked alphabets appear to have been first developed by Thatcher [11, 12, 13, 14]. He states a number of results on finite tree automata that carry over directly from the theory of string automata. In particular, he developed the basic theory of finite tree automata and also introduced and investigated extended tree automata.

Other researchers studied various aspects of finite and extended tree automata; see the work of Barrero [1], Moriya [8], Murata [9] and Takahashi [10].

This paper has four further sections. In Section 2, we introduce the basic notation and terminology for extended tree automata, in Section 3, we introduce the notion of a top congruence and of views and, in Section 4, we use these notions to prove that extended two-way tree automata are only as expressive as extended bottom-up tree automata. Last, in Section 5, we state some conclusions and provide some research problems.

2 Notation and definitions

We first recall tree and tree automata concepts before introducing the new concepts that we need.

Definition 2.1 **Trees** have at least one node; their node labels are taken from a finite alphabet Σ. We represent trees by expressions that use the symbols in Σ as operators.

Operators have no rank, so they can have any number of operands, including none. For example, the term $a(a(a()a())a(a()a()))$ represents a complete binary tree of height two, whose nodes all have the label a. Observe that external nodes or leaves correspond exactly to those subterms of the form $a()$.

We denote symbols in Σ with a, strings over Σ with w and sets of strings over Σ (we call them string languages) with L. The Greek letter λ denotes the empty string. We denote trees with t and sets of trees (we call them tree languages) with T. Subscripted and superscripted variables have the same types as their base names.

Definition 2.2 We define the set $nodes(t)$ of **nodes of a tree** t as a set of strings of natural numbers. Its definition is by induction on t:

For a tree $a(t_1 \cdots t_n)$, $n \geq 0$, we define

$$nodes(a(t_1 \cdots t_n) = \bigcup_{1 \leq i \leq n} i \cdot nodes(t_i) \cup \{\lambda\}.$$

The nodes of a tree viewed as terms correspond to subterms. We denote nodes of trees with ν.

Definition 2.3 The **root node** $root(t)$ of a tree t is defined as λ. For each node ν of t we define the set $children(\nu)$ of ν's **children** as the set of all nodes $\nu \cdot i$ in $nodes(t)$.

Definition 2.4 A node ν of a tree t is a **leaf** if and only if $children(\nu) = \emptyset$. The set of leaves of t is denoted by $leaves(t)$.

Definition 2.5 For each node ν of a tree t, we denote the label of ν in Σ by $label(\nu)$. More precisely, for a tree $t = a(t_1 \cdots t_n)$, $n \geq 0$, we define:

1. The label of the root node λ in t is a.
2. The label of the node $i \cdot s$ in t is the label of the node s in t_i.

We are now in a position to define the class of tree automata that we investigate.

Definition 2.6 A **(regularly) extended two-way (nondeterministic) tree automaton** M is specified by a triple (Q, δ, F), where Q is a finite set of states, $F \subseteq Q$ is a set of final or accepting states, and $\delta \subseteq \Sigma \times Q^* \times Q \times \{u, d, s\}$ is a transition relation that satisfies the condition that, for all a in Σ, q in Q and m in $\{u, d, s\}$, the set $\{w \in Q^* \mid (a, w, q, m) \in \delta\}$ is a regular set of strings over the alphabet Q.

If, for all a in Σ, q in Q and m in $\{u, d, s\}$, the set $\{w \in Q^* \mid (a, w, q, m) \in \delta\}$ is a finite set of strings over the alphabet Q, then M is a **finite two-way tree automaton.**

Finite two-way tree automata have been investigated by Moriya [8], whereas our results are on regularly extended two-way tree automata.

We define the computations of a two-way tree automaton on a tree by sequences of configurations. A configuration assigns a state of the automaton to each node in a cut of the tree.

Definition 2.7 A **cut** C **of a tree** t is a subset of $nodes(t)$ such that, for each leaf node ν of t, there is exactly one node in C on the path from the root to ν; in other words, there is exactly one node in C given by a prefix of ν.

Definition 2.8 A **configuration** c **of a two-way tree automaton** $M = (Q, \delta, F)$ operating on a tree t is a map $c : C \longrightarrow Q$ from a cut C of t to the set of states Q of M.

Let ν be a node of a tree t and let $c : C \longrightarrow Q$ be a configuration of the two-way tree automaton M operating on t. If $children(\nu) \subseteq C$, then formally $c(children(\nu))$ is a subset of Q. We overload this notation such that $c(children(\nu))$ also denotes the sequence of states in Q which arises from the order of ν's children in t.

Definition 2.9

1. A **starting configuration** of a two-way tree automaton $M = (Q, \delta, F)$ operating on a tree t is a configuration $c : leaves(t) \longrightarrow Q$ such that $c(\nu)$ is any state q in Q such that $(label(\nu), \lambda, c(\nu), u) \in \delta$.
2. A **halting configuration** is a configuration $c : C \longrightarrow Q$ such that $C = \{root(t)\}$.
3. An **accepting configuration** is a configuration $c : C \longrightarrow Q$ such that $C = \{root(t)\}$ and $c(root(t)) \in F$.

Definition 2.10

1. A two-way tree automaton $M = (Q, \delta, F)$ operating on a tree t makes a **transition** from a configuration $c_1 : C_1 \longrightarrow Q$ to a configuration $c_2 : C_2 \longrightarrow Q$ (symbolically $c_1 \longrightarrow c_2$) if and only if it makes an up transition, a down transition or a no-move transition each of which we now define.
2. M makes an **up transition** from c_1 to c_2 if and only if t has a node ν such that the following four conditions hold:

   (a) $children(\nu) \subseteq C_1$.
   (b) $C_2 = (C_1 \setminus children(\nu)) \cup \{\nu\}$.
   (c) $(label(\nu), c_1(children(\nu)), c_2(\nu), u) \in \delta$.
   (d) c_1 is identical to c_2 on their domains' common subset $C_1 \cap C_2$.

3. M makes a **down transition** from c_1 to c_2 if and only if t has a node ν such that the following four conditions hold:
   (a) $\nu \in C_1$.
   (b) $C_2 = (C_1 \setminus \{\nu\} \cup children(\nu))$.
   (c) $(label(\nu), c_2(children(\nu)), c_1(\nu), d) \in \delta$.
   (d) c_1 is identical to c_2 on their domains' common subset $C_1 \cap C_2$.
4. M makes a **no-move transition** from c_1 to c_2 if and only if t has a node ν such that the following four conditions hold:
   (a) $\nu \in C_1$.
   (b) $C_2 = C_1$.
   (c) $(label(\nu), c_1(\nu), c_2(\nu), s) \in \delta$.
   (d) c_1 is identical to c_2 on $C_1 \setminus \{\nu\}$, which is equal to $C_2 \setminus \{\nu\}$.

Definition 2.11

1. A **computation** of a two-way tree automaton M on a tree t from configuration c to configuration c' is a sequence of configurations $c_1, \ldots, c_n$, $n \geq 1$, such that $c = c_1 \longrightarrow \cdots \longrightarrow c_n = c'$.
2. An **accepting computation** of M on t is a computation from a starting configuration to an accepting configuration.

Definition 2.12

1. A tree t is **recognized** by a two-way tree automaton M if and only if there is an accepting computation of M on t.
2. The *tree language* $T(M)$ of a two-way tree automaton M is the set of trees that are recognized by M.

Definition 2.13 A **(regularly) extended (nondeterministic) bottom-up tree automaton** is an extended (nondeterministic) two-way tree automaton $M = (Q, \delta, F)$ such that δ contains only transitions whose last component is u. For a bottom-up tree automaton M, we consider δ to be a subset of $\Sigma \times Q^* \times Q$ by dropping the fourth, constant component in the transition relation of a two-way tree automaton.

Note that nondeterministic bottom-up tree automata are only as expressive as deterministic bottom-up tree automata [9].

Definition 2.14 A tree language is **regular** if and only if it is the language of an extended bottom-up tree automaton.

Clearly, since every extended bottom-up tree automaton is an extended two-way tree automaton, every regular tree language is recognized by some regular two-way tree automaton. Our goal is to prove that the converse also holds; namely, every tree language recognized by an extended two-way tree automaton is regular. We establish this result indirectly by developing an algebraic characterization of regular tree languages and then proving that the tree languages recognized by extended two-way tree automata satisfy this characterization.

3 Top congruences

Definition 3.1 A **pointed tree** (also called a tree with a handle or a handled tree) is a tree over an extended alphabet $\Sigma \cup \{X\}$ such that precisely one node is labeled with the variable X and that node is a leaf.

Definition 3.2 If t is a pointed tree and t' is a (pointed or nonpointed) tree, we can **catenate** t and t' by replacing the node labeled X in t with the root of t'. The result is the (pointed or nonpointed) tree tt'.

Definition 3.3 Let T be a tree language. Trees t_1 and t_2 are **top congruent with respect to** T ($t_1 \sim_T t_2$) if and only if, for each pointed tree t, the following condition holds:

$$tt_1 \in T \text{ if and only if } tt_2 \in T.$$

The top congruence for trees is the tree analog of the left congruence for strings.

Lemma 3.1 *The top congruence is an equivalence relation on trees; it is a congruence with respect to catenations of pointed trees with nonpointed trees.*

Definition 3.4 The **top index of a tree language** T is the number of $\sim_T$-equivalence classes.

Lemma 3.2 *Each regular tree language has finite top index.*

A string language is regular if and only if it has finite index; however, that a tree language has finite top index is insufficient for it to be regular. For example, consider the tree language

$$L = \{a(b^i c^i) : i \geq 1\}.$$

Clearly, L has finite top index, but it is not regular. A second condition, regularity of local views, must also be satisfied.

Definition 3.5 Let T be a tree language, a be a symbol in Σ, t be a pointed tree and T_f be a finite set of trees. Then, the **local view of T with respect to t, a and T_f** is the string language

$$V_{t,a,T_f}(T) = \{t_1 \cdots t_n \in T_f^* \mid ta(t_1 \cdots t_n) \in T\}$$

over the alphabet T_f. For the purposes of local views we treat the trees in the finite set T_f as symbols in the alphabet T_f; the trees in T_f are primitive entities that can be catenated to give strings over T_f. Note that we are not catenating trees.

Lemma 3.3 *All local views of each regular tree language are regular string languages.*

Example Let

$$T = \{c(t_1 \cdots t_n) \mid label(root(t_1)) \cdots label(root(t_n)) \in \{a^l b^l \mid l \geq 1\}\}.$$

The tree language T has top index four. Two of its equivalence classes are the sets of trees whose root labels are a or b; the other two are T and the set of trees that are not in T, but have the root label c. The local view of T with respect to the pointed tree $X()$, symbol c and the finite set of trees $\{a(), b()\}$ is the non-regular set of strings $\{a^l b^l \mid l \geq 1\}$. Hence, T has finite top index but it is not regular.

Theorem A *A tree language is regular if and only if it has finite top index and all its local views are regular string languages.*

At first glance it may appear that the local-view condition for regular tree languages is a condition on an infinite number of trees. But, if we exchange a tree t_1 in a finite set T_f by an equivalent—with respect to top congruence—tree t_2, then $V_{a,t,(T_f \setminus \{t_1\}) \cup \{t_2\}}(T)$ is the homomorphic image of $V_{a,t,T_f}(T)$ under a string isomorphism. Hence, if T has finite top index, we need to check the local-view condition for only a finite number of tree sets T_f.

4 Regularly extended two-way tree automata languages

Lemma 4.1 *The language of every extended two-way tree automaton has finite top index.*

Lemma 4.2 *The languages of all extended two-way tree automata have only regular local views.*

Proof Let t be a pointed tree, a be a symbol in Σ, and T_f be a finite set of trees. We demonstrate that the local view $V_{t,a,T_f}(T)$ of T with respect to t, a, and T_f, namely the string language

$$\{t_1 \cdots t_n \in T_f^* \mid ta(t_1 \cdots t_n) \in T\},$$

is regular.

The proof is in three steps.

The first step is to recognize that V_{t,a,T_f} is a finite union of finite intersections of the following sets X_p and X_{pq}, $p, q \in Q$:

$$\begin{aligned} X_p \;=\; \{t_1 \cdots t_n \in T_f^* \mid & \\ & c_1 \longrightarrow c_2, \\ & c_1 \text{ is a starting configuration of } M \text{ on } a(t_1 \cdots t_n), \\ & c_2 \text{ is a halting configuration of } M \text{ on } a(t_1 \cdots t_n), \\ & c_2(\mathit{root}(a(t_1 \cdots t_n))) = p \text{ and} \\ & \text{there is no other halting configuration in the computation } c_1 \longrightarrow c_2\} \end{aligned}$$

and

$$\begin{aligned} X_{pq} \;=\; \{t_1 \cdots t_n \in T_f^* \mid & \\ & c_1 \longrightarrow c_2, \\ & c_1 \text{ and } c_2 \text{ are halting configurations of } M \text{ on } a(t_1 \cdots t_n), \\ & c_1(\mathit{root}(a(t_1 \cdots t_n))) = p, \\ & c_2(\mathit{root}(a(t_1 \cdots t_n))) = q \text{ and} \\ & \text{there is no other halting configuration in the computation } c_1 \longrightarrow c_2\}. \end{aligned}$$

Any computation on $ta(t_1 \cdots t_n)$ from a starting configuration to a halting configuration can be partitioned into those parts that concern only t and those parts that concern only $a(t_1 \cdots t_n)$. The parts that concern only $a(t_1 \cdots t_n)$ form a computation from a starting configuration to a halting configuration, followed by a number of computations from halting configurations to halting configurations.

Hence, the $a(t_1 \cdots t_n)$-related parts of any accepting computation of M on $ta(t_1 \cdots t_n)$ first go from a starting configuration to a halting configuration, having M in some state p at $a(t_1 \cdots t_n)$'s root, and then from halting configuration to halting configuration, leading M from some state p_i to some state q_i on $a(t_1 \cdots t_n)$'s root until M finally leaves $a(t_1 \cdots t_n)$ and does not return. This implies that $t_1 \cdots t_n$ is in $X_p \cap X_{p_1q_1} \cap \cdots \cap X_{p_rq_r}$. The state sequence $p, p_1, q_1, \ldots, p_r, q_r$ documents the behaviour of M at the root of $a(t_1 \cdots t_n)$ during an accepting computation of M on the complete tree $ta(t_1 \cdots t_n)$.

For any other sequence of trees $t'_1 \cdots t'_m$ in $X_p \cap X_{p_1q_1} \cap \cdots \cap X_{p_rq_r}$, we can construct an accepting computation of M on $ta(t'_1 \cdots t'_m)$ by patching the $a(t_1 \cdots t_n)$-related parts of the original computation with computations on $a(t'_1 \cdots t'_m)$ that have the same state-behaviour at the root as $a(t_1 \cdots t_n)$ had. Since $t'_1 \cdots t'_m$ is in $X_p \cap X_{p_1q_1} \cap \cdots \cap X_{p_rq_r}$, we can find such patches.

We conclude that the whole set $X_p \cap X_{p_1 q_1} \cap \cdots \cap X_{p_r q_r}$ is a subset of V_{t,a,T_f}.

Since there are only finitely many sets X_p and X_{pq}, the set V_{t,a,T_f} is a finite union of finite intersections of these.

The next two steps establish that X_p and X_{pq} are regular string languages.

First, a string $t_1 \cdots t_n$ is in X_p if and only if there are $p_1, \ldots, p_n$ in Q such that M, when operating on t_i beginning in a starting configuration, eventually reaches a halting configuration c such that $c(root(t_i)) = p_i$ and $(a, p_1 \cdots p_n, p, u) \in \delta$. The regularity of the transition table δ implies that X_p is a regular string language.

Second, let X^s_{pq} be the subset of X_{pq} in which the computation $c_1 \longrightarrow c_2$ (compare the definition of X_{pq}) makes just one computation step and let X^m_{pq} be the subset of X_{pq} in which the computation $c_1 \longrightarrow c_2$ makes more than one computation step. Then, X_{pq} is the (not necessarily disjoint) union of X^s_{pq} and X^m_{pq}. We demonstrate that both subsets are string regular.

First, note that

$$X^s_{pq} = \{t_1 \cdots t_n \in T^*_f \mid (a, p, q, s) \in \delta\}.$$

Hence, X^s_{pq} depends only on a and is either empty or T^*_f. In both cases, X^s_{pq} is regular.

Second, $t_1 \cdots t_n \in X^m_{pq}$ if and only if there are $p_1, \ldots, p_n, q_1, \ldots, q_n$ in Q such that $(a, p_1 \cdots p_n, p, d)$ is in δ and M, when operating on t_i, makes a computation from a halting configuration with root label p_i to a halting configuration with root label q_i and $(a, q_1 \cdots q_n, q, u) \in \delta$.

The regularity of the transition table δ implies that X^m_{pq} is a regular string language. □

Theorem B *The language of every extended two-way tree automaton is tree regular.*

5 Concluding remarks

Moriya [8] uses crossing sequences to prove that finite two-way tree automata are as expressive as finite bottom-up tree automata. Thus, one follow-up question is whether we can prove our result using Moriya's approach.

We may define context-free two-way tree automata and ask whether they are as expressive as context-free bottom-up tree automata. Moriya [8] considers a variation on tree automata for which he demonstrates that the two-way version is indeed more expressive than the bottom-up version.

Takahashi [10], on the other hand, establishes a different characterization of regular tree languages. We would be interested in knowing whether her characterization can be used to derive our algebraic characterization and, conversely, can we use our characterization to prove her's.

As we mention in the introduction, we originally planned to use extended two-way tree automata to emulate (or execute) caterpillar expressions. Our difficulty was that we could not design such an emulation. Therefore, is there an effective emulation of caterpillar expressions with extended two-way tree automata?

References

[1] A. Barrero. Unranked tree languages. *Pattern Recognition*, 24(1):9–18, 1991.

[2] T. Bray, J. P. Paoli, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0. http://www.w3.org/TR/1998/REC-xml-19980210/, February 1998.

[3] A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2000. To appear.

[4] H. Comon, M. Daucher, R. Gilleron, S. Tison, and M. Tommasi. Tree automata techniques and applications, 1998. Available on the Web from l3ux02.univ-lille3.fr in directory tata.

[5] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[6] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3, Beyond Words*, pages 1–68. Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[7] ISO 8879: Information processing—Text and office systems—Standard Generalized Markup Language (SGML), October 1986. International Organization for Standardization.

[8] E. Moriya. On two-way tree automata. *Information Processing Letters*, 50:117–121, 1994.

[9] M. Murata. Forest-regular languages and tree-regular languages. Unpublished manuscript, 1995.

[10] M. Takahashi. Generalization of regular sets and their application to a study of context-free languages. *Information and Control*, 27(1):1–36, January 1975.

[11] J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1:317–322, 1967.

[12] J. W. Thatcher. A further generalization of finite automata. Technical Report RC 1846, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1967.

[13] J. W. Thatcher. There's a lot more to finite automata theory than you would have thought. Technical Report RC 2852 (#13407), IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1970.

[14] J. W. Thatcher and J. B. Wright. Abstract 65T-469. *Notices of the American Mathematical Society*, 12:820, 1965.