A Missing Link in Root-to-Frontier Tree Pattern Matching

Loek G. W. A. Cleophas, Kees Hemerik and Gerard Zwaan

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

e-mail: loek@loekcleophas.com, c.hemerik@tue.nl, g.zwaan@tue.nl

Abstract. Tree pattern matching (TPM) algorithms play an important role in practical applications such as compilers and XML document validation. Many TPM algorithms based on tree automata have appeared in the literature. For reasons of efficiency, these automata are preferably deterministic. Deterministic root-to-frontier tree automata (DRFTAs) are less powerful than nondeterministic ones, and no root-to-frontier TPM algorithm using DRFTAs has appeared so far. Hoffmann & O'Donnell [HO82] presented a root-to-frontier TPM algorithm based on an Aho-Corasick automaton recognizing tree stringpaths, but no relationship between this algorithm and algorithms using tree automata has been described. In this paper, we show that a specific DRFTA can be used for stringpath matching in a root-to-frontier TPM algorithm. This algorithm has not appeared in the literature before, and provides a missing link between TPM algorithms using stringpath automata and those using tree automata.

1 Introduction

Tree pattern matching (TPM) is an important problem from regular tree theory. It can be described as finding all occurrences of one or more given pattern trees (patterns) in a given subject tree. Algorithms solving this problem form the basis for tree acceptance and tree parsing algorithms, which play an important role in practical applications such as compilers and XML document validation.

The problems of pattern matching, acceptance and parsing for trees are related, and so are the algorithms solving them (referred to as *tree algorithms* from this point onward). Either implicitly or explicitly, many use some form of tree or string automata, combined with a frontier-to-root (bottom-up) or root-to-frontier (top-down) tree traversal [FSW94, HK89, AGT89, vdM88, vD87, Cha87, HC86, AG85, HO82, Kro75]. For efficiency reasons, the automata used should preferably be deterministic.

In frontier-to-root tree algorithms, deterministic frontier-to-root tree automata can be and indeed often are used [FSW94, HK89, Cha87, HC86, HO82].

Deterministic root-to-frontier tree automata (DRFTAs) however have not been used in root-to-frontier tree algorithms, since it is known from regular tree theory that in general they are less powerful than their nondeterministic counterparts (NRFTAs) [Eng75, GS97]. Such algorithms therefore use NRFTAs [vD87].

Hoffmann & O'Donnell [HO82] and Aho, Ganapathi & Tjiang [AGT89, AG85] presented root-to-frontier tree algorithms based on deterministic *string* automata instead. These algorithms use a deterministic Aho-Corasick (AC) automaton [AC75] and its output function to recognize tree *stringpaths*. Since a tree is uniquely determined by its stringpaths, this automaton can be used to detect tree matches. The presentation in those papers is somewhat informal and complicated by optimizations, but van de Meerakker [vdM88] gave a stepwise account of how to obtain the algorithms. Unfortunately, no relationship between tree algorithms using stringpath automata and those using tree automata seems to have appeared in the literature.

In this paper, we show that even though a DRFTA cannot be used as a tree acceptor or matcher by itself, a specific DRFTA can be used (together with an output function) for stringpath matching in a root-to-frontier tree traversal. We present a version of Hoffmann & O'Donnell's root-to-frontier TPM algorithm that uses an AC automaton and output function, and then present a modified TPM algorithm that uses this DRFTA and associated output function. To the best of our knowledge, this algorithm has not appeared in the literature before. It provides a missing link between TPM algorithms using stringpath automata and those using tree automata.

Our algorithm is not necessarily efficient. It has the same worst-case bound of $\mathcal{O}(m \cdot n)$ as the other papers mentioned (where m and n are the pattern and subject tree size). In recent years, many papers providing algorithms with better worst-case bounds have appeared [Kos89, DGM94, CH97, CHI99]. These algorithms improve the worst-case bound at the cost of somewhat more elaborate algorithms and/or the construction of larger auxiliary data structures. It is unclear (and a potential subject of future research) what the practical performance of the algorithms is.

Section 2 introduces basic definitions and notations. Tree pattern matching is introduced in Section 3. In Section 4 we present a version of Hoffmann & O'Donnell's root-to-frontier TPM algorithm, while Section 5 shows that a particular kind of DRFTA can be constructed and used for stringpath matching, and presents a root-to-frontier TPM algorithm using this DRFTA. Section 6 gives some conclusions as well as suggestions for future work, in particular a more detailed comparison of the two algorithms and automata kinds.

2 Preliminaries

We use $\mathbb{B}, \mathbb{N}$ and $\mathbb{N}_+$ to denote the domain of the booleans, the natural numbers and the natural numbers excluding 0 respectively.

A basic understanding of the meaning of *quantifications* is assumed. We use the notation $\langle \oplus a : R(a) : E(a) \rangle$ where $\oplus$ is the associative and commutative *quantification operator* (with unit $e_\oplus$), a is the *quantified variable* introduced, R is the *range predicate* on a, and E is the *quantified expression*. By definition, we have $\langle \oplus a : false : E(a) \rangle = e_\oplus$. The following table lists some of the most commonly quantified operators, their quantified symbols, and their units:

Operator	$\vee$	$\wedge$	$\cup$
Symbol	$\exists$	$\forall$	$\bigcup$
Unit	$false$	$true$	$\varnothing$

We use $\langle \mathbf{Set}\, a :\ R(a)\ :\ E(a)\ \rangle$ for $\langle \bigcup a :\ R(a)\ :\ \{E(a)\}\ \rangle$.

2.1 Trees

Definition 1. An *ordered tree domain* is a finite non-empty subset D of $\mathbb{N}_+^*$ such that

- $\mathbf{pref}(D) \subseteq D$, i.e. D is prefix-closed, and
- for all $\mathfrak{n} \in D$ and $i \in \mathbb{N}_+$, $\mathfrak{n} \cdot i \in D \Rightarrow \langle \forall\, j :\ j < i\ :\ \mathfrak{n} \cdot j \in D \rangle$.

Note that we use $\cdot$ to separate elements of $\mathbb{N}_+$ in tree domain elements. Tree domain elements are called *nodes*. *Root node* $\varepsilon \in D$ for any D. □

Example 2. Set $\{\varepsilon, 1, 1 \cdot 1, 2\}$ is an ordered tree domain. □

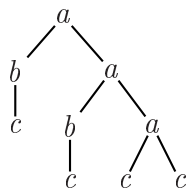
Definition 3. Let D be an ordered tree domain, V a finite non-empty set of symbols (alphabet) and $r \in V \to \mathbb{N}$ a *ranking function*. An *ordered ranked tree* t is a function $t \in D \to V$ for which for every node $\mathfrak{n} \in D$, $r(t(\mathfrak{n}))$ equals the number of $i \in \mathbb{N}_+$ such that $\mathfrak{n} \cdot i \in D$. □

For a tree t, we will use D_t to refer to the tree domain underlying t. Given an alphabet V and ranking function r, we call the pair (V, r) a ranked alphabet. For any $a \in V$, we call $r(a)$ the rank of a. In this paper, we assume (V, r) to be the fixed ranked alphabet $\{(a, 2), (b, 1), (c, 0)\}$, i.e. with symbols a, b, c of rank $2, 1, 0$.

We denote the set of all ordered ranked trees over (V, r) by $\mathit{Tree}(V, r)$. For $t \in \mathit{Tree}(V, r)$ and $\mathfrak{n} \in D_t$, $t@\mathfrak{n}$ is t's subtree starting at $\mathfrak{n}$. Note that $t@\varepsilon = t$.

We may represent a tree by a set of pairs of tree domain values and symbols, graphically, or using a term representation, as in the following example.

Example 4 (Tree). Given (V, r), set $t = \{(\varepsilon, a), (1, b), (1 \cdot 1, c), (2, a), (2 \cdot 1, b), (2 \cdot 1 \cdot 1, c), (2 \cdot 2, a), (2 \cdot 2 \cdot 1, c), (2 \cdot 2 \cdot 2, c)\}$ forms an ordered, ranked tree. It can be represented as a term by $a(b(c), a(b(c), a(c, c)))$, while tree $t@(2 \cdot 2)$ for example corresponds to set $\{(\varepsilon, a), (1, c), (2, c)\}$ and term $a(c, c)$. Graphically, t is represented as



□

2.2 Tree automata

Definition 5. A *tree automaton* (TA) M is a 6-tuple $(Q, V, r, R, Q_{ra}, Q_{la})$ such that

- Q is a finite set, the *state set*
- (V, r) is a ranked alphabet
- $R = \langle \mathbf{Set}\, a :\ a \in V\ :\ R_a \rangle$ is the set of transition relations, where $R_a \subseteq Q \times Q^{r(a)}$ for all $a \in V$
- $Q_{ra} \subseteq Q$, the *root accepting states*

- $Q_{la} \subseteq Q$, the *leaf accepting states*,
  defined by $Q_{la} = \langle \mathbf{Set}\, a, q : \ a \in V \wedge r(a) = 0 \wedge (q, ()) \in R_a \ : \ q \ \rangle$

□

Remark 6. An explicit set Q_{la} for *leaf accepting states* is not needed, but is included to facilitate notation. Note that for $a \in V$ with $r(a) = 0$, $R_a \subseteq Q \times Q^0$, i.e. the second component corresponds to a domain whose single element is the empty tuple $()$. Some definitions of tree automata use $R_a \subseteq Q \times Q$ for such symbols a instead. □

Definition 7. An NRFTA (nondeterministic root-to-frontier tree automaton) $M = (Q, V, r, R, Q_{ra}, Q_{la})$ is a TA where $R_a \in Q \to \mathcal{P}(Q^{r(a)})$ for all $a \in V$, i.e. R_a is considered to be directed. □

Considering the relations R_a in this way is not a restriction, and therefore the classes of NRFTA and TA are equivalent. By directing the relations, the root accepting states become start states. By restricting the relations R_a of the NRFTA to be functions yielding a single state tuple instead of a set of such tuples, we obtain the deterministic root-to-frontier tree automata:

Definition 8. A DRFTA $M = (Q, V, r, R, Q_{ra}, Q_{la})$ is an NRFTA where $R_a \in Q \to Q^{r(a)}$ for all $a \in V$—i.e. the R_a are functions—and $Q_{ra} = \{q_{ra}\}$—i.e. there is a unique root accepting state (start state). □

We define *tree acceptance* using tree state assignments, i.e. assignments of a state to each tree node. Consider the set of tree state assignments that respect the automaton transition relations (or functions in case of directed automata) and that assign a (the, for DRFTAs) root accepting state to the subject tree root. A subject tree is accepted by an automaton if and only if this set is non-empty.

Lemma 9. There are NRFTAs for which no DRFTA accepting the same language can be constructed.

Proof. We give an example of a language which is not recognizable by a DRFTA.

Let $L = \{a(c, d), a(d, c)\}$. We try to construct a DRFTA accepting L. There must be exactly one pair of states (q_1, q_2) such that $R_a(q_{ra}) = (q_1, q_2)$. To recognize both trees, $R_c(q_1) = R_d(q_2) = R_d(q_1) = R_c(q_2) = ()$ must hold, but this means that trees $a(c, c)$ and $a(d, d)$ are accepted as well. A DRFTA accepting L therefore cannot exist, but an NRFTA for L can be constructed (see Figure 1, the notation of which is explained below). □

Finite string automata are often represented visually by a state diagram. We adapt this notation to finite tree automata. Each state is represented by a circle, with double circles indicating root accepting states, while a transition relating state q and states $q_1 \ldots q_n$ on a symbol a is represented by

1. a (directed) edge connecting q to a small unlabeled circle, labeled by a and
2. n (directed) edges connecting the unlabeled circle to q_i (for $1 \leq i \leq n$)

Finally, we introduce dotted trees, which are used in Section 5. A dotted tree is a tree with a distinguished position, as in the following definition.

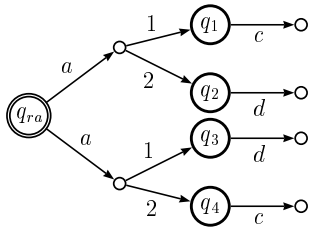


Figure 1: NRFTA accepting $L = \{a(c,d), a(d,c)\}$

Definition 10. Let $t \in \mathit{Tree}(V,r)$ and $\mathfrak{n} \in D_t$, then the pair $(t,\mathfrak{n})$ is a *dotted tree*. We use $DT(t)$ to indicate the set of all dotted trees for a tree t. □

Example 11. Let $u = a(b(c),c)$, then set $DT(u)$ corresponds to $\{(u,\varepsilon),(u,1),(u,1\cdot 1),(u,2)\}$. □

3 Tree pattern matching

The leaves of trees in $\mathit{Tree}(V,r)$ always have symbols of rank 0, but for pattern matching, something more general is needed. We extend the alphabet with a special variable or 'wildcard' symbol, indicating a match of any tree from $\mathit{Tree}(V,r)$. We extend (V,r) into (V',r') by adding symbol ν with $r'(\nu) = 0$, and letting $r'(a) = r(a)$ for all $a \in V$. Trees in $\mathit{Tree}(V',r')$ are called *pattern trees* or *patterns*. Note that ν can only label leaf nodes. Notation $t@\mathfrak{n}$ and $DT(t)$ are extended to trees in $\mathit{Tree}(V',r')$.

We can now define what it means for a subtree of a tree to match a pattern, defining a function *Match* as follows.

Definition 12. Function $\mathit{Match} \in \mathit{Tree}(V',r') \times \mathit{Tree}(V,r) \times D \to \mathbb{B}$ is defined for every pattern $p \in \mathit{Tree}(V',r')$, subject $t \in \mathit{Tree}(V,r)$ and node $\mathfrak{n} \in D_t$ by $\mathit{Match}(p,t,\mathfrak{n}) =$

$$\left\langle \exists\, s_1,\ldots,s_k :\ s_1,\ldots,s_k \in \mathit{Tree}(V,r)\ :\ p[s_1,\ldots,s_k] = t@\mathfrak{n} \right\rangle$$

where $p[s_1,\ldots,s_k]$ is the tree obtained by substituting $s_1,\ldots,s_k$ respectively for the k instances of ν in p. □

Example 13. Given trees $t = a(b(c),a(b(c),a(c,c)))$ and $p = a(b(c),\nu)$, $\mathit{Match}(p,t,\mathfrak{n})$ holds for $\mathfrak{n} = \varepsilon$ and $\mathfrak{n} = 2$ (and not for any other nodes). $\mathit{Match}(p,t,\varepsilon)$ holds since $t@\varepsilon = p[a(b(c),a(c,c))]$. $\mathit{Match}(p,t,2)$ holds since $t@2 = p[a(c,c)]$. □

Apart from the tree domain, term and graphical notations used before, a tree is also uniquely characterized by its set of stringpaths, which represent all its root to leaf paths.

Definition 14 (Tree stringpaths). Let $t \in \mathit{Tree}(V',r')$, then function $\mathit{SPaths} \in \mathit{Tree}(V',r') \to \mathcal{P}((V' \cdot \mathbb{N}_+)^* \cdot V')$ is defined by

$$\begin{aligned} \mathit{SPaths}(t) &= \{t(\varepsilon)\} && \text{if } r(t(\varepsilon)) = 0 \\ \mathit{SPaths}(t) &= \{t(\varepsilon)\} \\ &\quad \cdot \left\langle \bigcup i :\ 1 \le i \le r(t(\varepsilon))\ :\ \{i\} \cdot \mathit{SPaths}(t@i) \right\rangle && \text{if } r(t(\varepsilon)) > 0 \end{aligned}$$

(where string concatenation operator $\cdot$ is extended to operate on sets of strings). □

Example 15. For $t = a(b(c), a(b(c), a(c, c)))$, $SPaths(t@2) = \{a1b1c, a2a1c, a2a2c\}$ and $SPaths(t) = \{a1b1c,\ a2a1b1c, a2a2a1c, a2a2a2c\}$. □

A stringpath of a pattern p matches in a given subject tree t starting at $\mathfrak{n}$ if and only if either the stringpath is in $SPaths(t@\mathfrak{n})$ or the stringpath ends in ν and the stringpath minus this ν is a prefix of some stringpath in $SPaths(t@\mathfrak{n})$. It follows that p matches in t at node $\mathfrak{n}$ if and only if each stringpath in $SPaths(p)$ matches in t starting at $\mathfrak{n}$.

We introduce infix operators $\restriction$ and $\downharpoonright$ (*right take* and *right drop*). For any string s of length $\geq m \in \mathbb{N}_+$, $s \restriction m$ equals the rightmost m symbols of s, while $s \downharpoonright m$ equals s except its rightmost m symbols.

Example 16. Given tree $t = a(b(c), a(b(c), a(c, c)))$ and pattern $p = a(b(c), \nu)$, $Match(p, t, \mathfrak{n})$ holds for $\mathfrak{n} = \varepsilon$ and $\mathfrak{n} = 2$ only. $Match(p, t, \varepsilon)$ holds since $a1b1c \in SPaths(t@\varepsilon)$ and $a2\nu \restriction 1 = \nu \wedge a2\nu \downharpoonright 1 \in \mathbf{pref}(SPaths(t@\varepsilon))$. $Match(p, t, 2)$ holds since $a1b1c \in SPaths(t@2)$ and $a2\nu \restriction 1 = \nu \wedge a2\nu \downharpoonright 1 \in \mathbf{pref}(SPaths(t@2))$. □

To solve the TPM problem using a root-to-frontier approach, stringpath matching can be used. Stringpath matches are most easily registered at their endpoints, but algorithms can be adapted to register stringpath matches at their beginpoints, and by doing so, tree pattern matches can be determined. In the rest of this paper, we consider tree pattern matching as stringpath matching.

Related to the definition of stringpaths, we define a function representing the rootpath to a given node, i.e. the labeled path from the tree root to the given node:

Definition 17. Function $RPath \in Tree(V', r') \times D \to (V' \cdot \mathbb{N}_+)^* \cdot V'$ is defined by

$$\begin{aligned} RPath(t, \varepsilon) &= t(\varepsilon) \\ RPath(t, \mathfrak{n} \cdot i) &= RPath(t, \mathfrak{n}) \cdot i \cdot t(\mathfrak{n} \cdot i) \text{ for } \mathfrak{n} \cdot i \in D_t \end{aligned}$$

□

Note that a rootpath $RPath(t, \mathfrak{n})$ always ends with symbol $t(\mathfrak{n})$.

For every pattern p, there is a correspondence between dotted trees and rootpaths: $RPath(p, \mathfrak{n})$ is defined if and only if $(p, \mathfrak{n}) \in DT(p)$.

4 Using AC stringpath automata

The basic idea of Hoffmann & O'Donnell's root-to-frontier TPM algorithm [HO82] is to use an optimal AC automaton for matching pattern stringpaths, combined with a root-to-frontier traversal of the subject tree.

An optimal AC automaton is a version of the AC automaton without failure transitions. Construction algorithms for AC automata have been described in numerous references [CR03, NR02, Wat95, AC75], and we do not discuss any in detail.

Given the state reached by the AC automaton by processing an input string upto a given position, the output function determines the set of keyword occurrences ending at this position.

The AC automaton built from stringpath set $SPaths(p)$ for a given pattern $p \in Tree(V', r')$ is a 5-tuple $M_{AC} = (Q, V' \cup \mathbb{N}_+, \delta, q_0, output)$ in which Q is the state set,

$V' \cup \mathbb{N}_+$ the alphabet, $\delta \in Q \times (V' \cup \mathbb{N}_+) \to Q$ the transition function, q_0 the start state, and $output \in Q \to \mathcal{P}(SPaths(p))$ the output function.

On a high level, the construction of this automaton can be described as follows:

1. Construct a *trie* recognizing the set of stringpaths
2. For every state corresponding to a stringpath match, define the output of the state equal to the stringpath; for other states, the output is empty
3. Add a 'self-loop' transition on every alphabet symbol to the start state
4. Determinize the resulting automaton and adapt the output function accordingly

The resulting optimal AC automaton for the set of pattern stringpaths can be used in a root-to-frontier subject tree traversal to find all pattern stringpath matches.

Example 18 (AC stringpath automaton for pattern p). The trie with 'self-loop' constructed for pattern $p = a(b(c), \nu)$ by steps 1–3 of the above construction is depicted in Figure 2. The output function values corresponding to final states are defined as $output(q_c) = a1b1c, output(q_d) = a2\nu$.

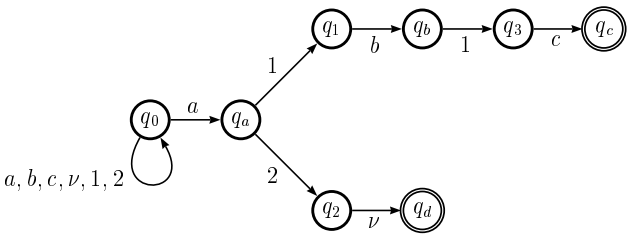


Figure 2: Trie with 'self-loop' for p

Applying step 4 of the above construction leads to the AC stringpath automaton depicted in Figure 3 (in which transitions not shown lead to q_0). The output function values corresponding to final states are defined as $output(q_c) = \{a1b1c\}, output(q_d) = \{a2\nu\}$. Note that states in the AC automaton are different from those in the trie with 'self-loop', since states of the AC automaton correspond to sets of states of the trie with 'self-loop'. □

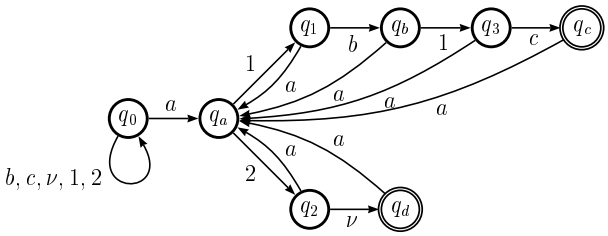


Figure 3: AC stringpath automaton for p.

4.1 An AC-based TPM algorithm

In this section, we present a version of Hoffmann & O'Donnell's root-to-frontier TPM algorithm. The algorithm presentation is similar to that by van de Meerakker [vdM88]. It uses explicit recursion instead of a stack as in the original algorithm. As an invariant, when visiting a node $\mathfrak{n}$ of the given subject tree t, the AC

automaton is in the state reached on input equal to the rootpath $RPath(t, \mathfrak{n})$ except its last symbol, $t(\mathfrak{n})$, i.e. on input $RPath(t, \mathfrak{n})\lfloor 1$.

To traverse the tree, the algorithm should be called on every child node i of the current node, if any. To maintain the invariant, the AC automaton should be in the state reached from the current state by a transition on $t(\mathfrak{n})$ followed by one on i, the number of the branch leading to the child node.

When visiting a node, the algorithm should register matches indicated by the AC automaton after a transition on symbol $t(\mathfrak{n})$, but also matches indicated after a transition on symbol ν, since ν matches any subtree. This results in:

{ Pre: $q = \delta^*(q_0, RPath(t, \mathfrak{n})\lfloor 1)$ }
proc $Traverse(q : Q, \mathfrak{n} : D) =$
[[**var** $q_{next} : Q$; $i : \mathbb{N}_+$; $sp : (V \cdot \mathbb{N}_+)^* \cdot V$
| **for** $i : 1 \leq i \leq r'(t(\mathfrak{n})) \rightarrow$
$q_{next} := \delta(\delta(q, t(\mathfrak{n})), i)$;
$Traverse(q_{next}, \mathfrak{n} \cdot i)$
rof;
for $sp : sp \in output(\delta(q, t(\mathfrak{n}))) \rightarrow$
"register sp match at its endpoint $\mathfrak{n}$";
rof
for $sp : sp \in output(\delta(q, \nu)) \rightarrow$
"register sp match at its endpoint $\mathfrak{n}$";
rof
]];
{ Post: every stringpath match in t whose endpoint is in the subtree $t@\mathfrak{n}$
has been registered at its endpoint }

{ Pre: $M_{\text{AC}} = (Q, V' \cup \mathbb{N}_+, \delta, q_0, output)$ is the AC automaton
built on the stringpaths of the pattern tree }
$Traverse(q_0, \varepsilon)$
{ Post: every stringpath match in t has been registered at its endpoint }

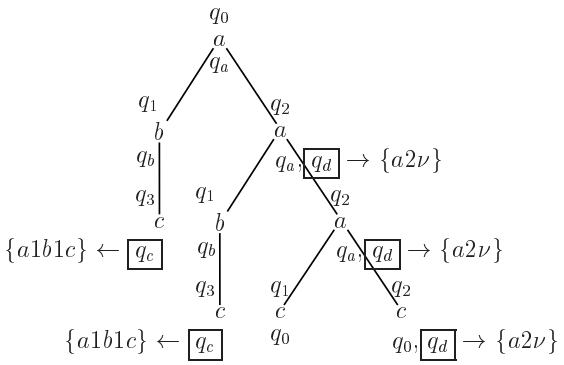


Figure 4: AC automaton state assignment and stringpath matches

Example 19. As an example, Figure 4 shows the states associated with every node and matches detected by the algorithm for subject tree $t = a(b(c), a(b(c), a(c, c)))$.

Note that even though the AC automaton used is deterministic, two states may be associated with a tree node $\mathfrak{n}$: the states corresponding to $\delta(q, t(\mathfrak{n}))$ and to $\delta(q, \nu)$. States corresponding to stringpath matches are framed. □

The algorithm can be extended to deal with multiple patterns as well, and can be used as the basis for tree acceptance and tree parsing algorithms [vdM88, AGT89].

5 Using stringpath DRFTAs

In this section, we present our new TPM algorithm. It uses a particular DRFTA and associated output function, combined with a root-to-frontier subject tree traversal.

On a high level, the DRFTA and output function construction works as follows:

1. Construct a DRFTA recognizing the pattern tree
2. For every state and alphabet symbol indicating a stringpath match, define the output of the state and symbol equal to this stringpath; for other combinations of state and alphabet symbol, define the output to be empty
3. Add 'self-loop' transitions on every symbol of rank > 0 to the start state
4. Determinize the resulting automaton and adapt the output function accordingly

The construction bears a lot of resemblance to the AC automaton construction process enumerated in the preceding section. A detailed investigation of the correspondence between the two constructions will be the subject of future work.

We discuss the above construction in more detail and show that the results can be used for root-to-frontier TPM, before presenting the new algorithm. Steps 1–3 result in a TPM NRFTA and are discussed first. In Section 5.2, step 4 is applied to obtain a DRFTA. Although this automaton cannot be used as a TPM automaton by itself, we show that it can be used for stringpath matching.

5.1 TPM NRFTA construction

Given a pattern, we can construct a DRFTA M accepting this pattern, in which the set of states is the set of dotted trees:

Construction 20. Let $p \in \mathit{Tree}(V', r')$, then $M = (Q, V', r', R, Q_{ra}, Q_{la})$ where

$$
\begin{array}{lcl}
Q & = & DT(p) \\
Q_{ra} & = & \{(p, \varepsilon)\} \\
R_a & = & \left\langle \mathbf{Set}\,\mathfrak{n} : \begin{array}{ll} (p,\mathfrak{n}) \in Q & : ((p,\mathfrak{n}), \\ \wedge\, p(\mathfrak{n}) = a & ((p,\mathfrak{n}\cdot 1), \\ & \ldots, \\ & (p,\mathfrak{n}\cdot r(a)))) \end{array} \right\rangle \text{ for all } a \in V'
\end{array}
$$

□

This construction results in a *deterministic* root-to-frontier tree automaton, but when extending it to deal with multiple patterns this may no longer be the case. Note that for $\mathfrak{n}$ such that $p(\mathfrak{n})$ has rank 0, elements of R_a have the form $((p, \mathfrak{n}), ())$ i.e. relate a state (a dotted tree) to the empty tuple of states.

Example 21 (DRFTA accepting pattern p). Applying the construction to pattern $p = a(b(c), \nu)$ leads to the DRFTA depicted in Figure 5.

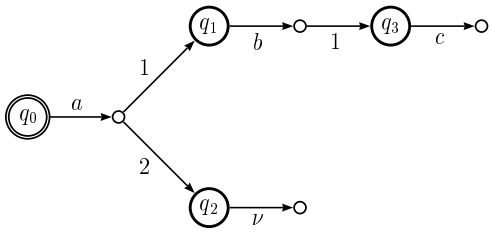


Figure 5: DRFTA resulting from Construction 20

The correspondence between the state labels used and the dotted trees they represent is as follows:

$$\begin{array}{lcl|lcl} q_0 & = & (p, \varepsilon) & q_2 & = & (p, 2) \\ q_1 & = & (p, 1) & q_3 & = & (p, 1 \cdot 1) \end{array}$$

The state assignment for every node of p in an accepting computation is shown in Figure 6. Tree p is accepted since $R_c(q_3) = ()$ and $R_\nu(q_2) = ()$. □

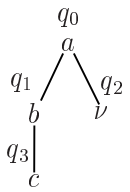


Figure 6: State assignment leading to acceptance of pattern tree p

Note how the DRFTA constructed for a tree pattern is similar to a trie constructed for the corresponding set of stringpaths.

Theorem 22. Given a subject tree t, pattern tree p, nodes $\mathfrak{m} \in D_t$ and $\mathfrak{n} \in D_p$, and a DRFTA as in Construction 20,

$$\begin{array}{l} (p, \mathfrak{n}) \text{ is assigned to node } \mathfrak{m} \text{ by} \\ \text{the DRFTA computation} \\ \wedge\ (t(\mathfrak{m}) = p(\mathfrak{n}) \vee \nu = p(\mathfrak{n})) \end{array} \Rightarrow \begin{array}{l} RPath(p, \mathfrak{n}) \text{ matches} \\ \text{ending at node } \mathfrak{m} \end{array}$$

Proof : We prove this theorem by structural induction on $\mathfrak{n}$.

Case $\mathfrak{n} = \varepsilon$: $t(\mathfrak{m}) = p(\varepsilon) \vee \nu = p(\varepsilon)$ implies that $p(\varepsilon) = RPath(p, \varepsilon)$ matches ending at node $\mathfrak{m}$.

Case $\mathfrak{n} = \mathfrak{l} \cdot i$: Using the definition of the DRFTA's transition relation, $(p, \mathfrak{n})$ is assigned to node $\mathfrak{m}$ by the DRFTA computation $\wedge\ (t(\mathfrak{m}) = p(\mathfrak{n}) \vee \nu = p(\mathfrak{n}))$ implies that $(p, \mathfrak{l})$ is assigned to node $\mathfrak{m}\downarrow 1$ and $t(\mathfrak{m}\downarrow 1) = p(\mathfrak{l})$. Using the induction hypothesis, $RPath(p, \mathfrak{l})$ matches ending at node $\mathfrak{m}\downarrow 1$. Since $t(\mathfrak{m}) = p(\mathfrak{n}) \vee \nu = p(\mathfrak{n})$, $RPath(p, \mathfrak{n}) = RPath(p, \mathfrak{l}) \cdot i \cdot p(\mathfrak{n})$ matches ending at node $\mathfrak{m}$. □

Since stringpaths are rootpaths ending in symbols of rank 0, matches can only end in such symbols, and using Theorem 22 we obtain the following definition:

Definition 23. For automata as in Construction 20, partial function $output \in Q \times V' \to (V' \cdot \mathbb{N}_+)^* \cdot V'$ is defined for $(p, \mathfrak{n}) \in Q$ and $a \in V'$ such that $a = p(\mathfrak{n}) \wedge r(a) = 0$ by $output((p, \mathfrak{n}), a) = RPath(p, \mathfrak{n})$. □

Note that function *output* is used with the symbol $t(\mathfrak{m})$ for $\mathfrak{m}$ the node of t that a state is assigned to, and with symbol ν. The inverse of the implication in Theorem 22 does not hold; the automaton is a tree *acceptor*, and can only be used to detect a pattern match that starts at the subject tree root. To enable pattern matches starting at other input tree nodes to be detected, an extension similar to the addition of the 'self-loop' transitions of the AC automaton is necessary, as follows:

Construction 24. Let $p \in Tree(V', r')$, then $M' = (Q, V', r', R', Q_{ra}, Q_{la})$ where

$$R'_a = R_a \cup \begin{cases} \{((p, \varepsilon), ((p, \varepsilon)^{r(a)}))\} & \text{for all } a \in V' \text{ with } r(a) > 0 \\ \varnothing & \text{for all } a \in V' \text{ with } r(a) = 0 \end{cases}$$

□

The result is an NRFTA accepting all trees ending in pattern occurrences.

Example 25 (Stringpath NRFTA for pattern p). The NRFTA with 'self-loops' constructed for pattern $p = a(b(c), \nu)$ by Construction 24—corresponding to steps 1–3 of the high-level construction at the beginning of Section 5—is depicted in Figure 7. The output function is defined by $output(q_3, c) = a1b1c$, $output(q_2, \nu) = a2\nu$ and undefined for other input range values. □

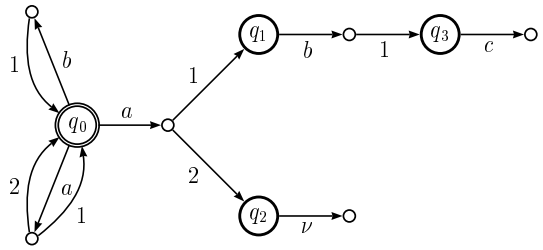


Figure 7: NRFTA with 'self-loops' resulting from Construction 24

Theorem 26. Given a subject tree t, pattern tree p, nodes $\mathfrak{m} \in D_t$ and $\mathfrak{n} \in D_p$, and an NRFTA as in Construction 24,

$$\begin{array}{l} (p, \mathfrak{n}) \text{ is assigned to node } \mathfrak{m} \text{ by an} \\ \text{NRFTA computation} \\ \wedge\ (t(\mathfrak{m}) = p(\mathfrak{n}) \vee \nu = p(\mathfrak{n})) \end{array} \equiv \begin{array}{l} RPath(p, \mathfrak{n}) \text{ matches} \\ \text{ending at node } \mathfrak{m} \end{array}$$

Proof : $\Rightarrow$: As in the proof of Theorem 22. $\Leftarrow$: By structural induction on $\mathfrak{n}$.

Case $\mathfrak{n} = \varepsilon$: $RPath(p, \varepsilon) = p(\varepsilon)$ matches ending at node $\mathfrak{m}$ implies that $t(\mathfrak{m}) = p(\varepsilon) \vee \nu = p(\varepsilon)$. From the definition of the NRFTA's transition function, (p, ε) is assigned to any given node in some computation of the NRFTA.

Case $\mathfrak{n} = \mathfrak{l} \cdot i$: $RPath(p, \mathfrak{n})$ matches ending at node $\mathfrak{m}$ implies that $RPath(p, \mathfrak{l})$ matches ending at node $\mathfrak{m}\lfloor 1$ and $t(\mathfrak{m}) = p(\mathfrak{n}) \vee \nu = p(\mathfrak{n})$. Using the induction hypothesis, $(p, \mathfrak{l})$ is assigned to node $\mathfrak{m}\lfloor 1$ by a computation of the NRFTA $\wedge\ (t(\mathfrak{m}\lfloor 1) = p(\mathfrak{l}) \vee \nu = p(\mathfrak{l}))$. Since $r(\nu) = 0$, the second conjunct reduces to $t(\mathfrak{m}\lfloor 1) = p(\mathfrak{l})$, and using the transition function definition we have that $(p, \mathfrak{l} \cdot i) = (p, \mathfrak{n})$ is assigned to node $\mathfrak{m}$ by a computation of the NRFTA. Since we already had $t(\mathfrak{m}) = p(\mathfrak{n}) \vee \nu = p(\mathfrak{n})$ this completes the proof of this case. □

5.2 Determinization

Similarly to the determinization of the trie with 'self-loops' to obtain a deterministic AC automaton, the NRFTA resulting from steps 1–3 can be determinized. The subset construction for NRFTAs is a straightforward generalization of that for string automata and is not elaborated here. It is known from regular tree theory however that the resulting DRFTA in general recognizes a superset of the NRFTA's language: the set of trees of which every stringpath occurs as a stringpath in a tree from the NRFTA's language. We aim at using the resulting DRFTA for tree stringpath pattern matching however, and it turns out to be suitable for this purpose.

Example 27 (Stringpath DRFTA for pattern p). Applying the subset construction to the NRFTA of Example 25 (corresponding to step 4 at the beginning of Section 5) leads to the stringpath DRFTA depicted in Figure 8. Output function values for this example DRFTA are singleton set versions of the values for the NRFTA of Example 25. As in Example 18, states of the automaton are different from those with the same label in the automaton of Example 25. □

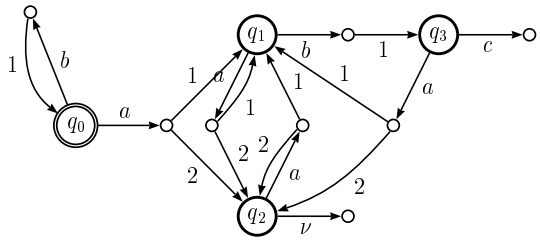


Figure 8: Stringpath DRFTA for p. Missing transitions on symbols of rank > 0 lead to (tuples of size equal to the symbol's rank of) state q_0

Using Theorem 26 and the subset construction, we obtain:

Corollary 28. Given a subject tree t, pattern tree p, nodes $\mathfrak{m} \in D_t$ and $\mathfrak{n} \in D_p$, and a DRFTA obtained from Construction 24 by a subset construction,

$$\begin{array}{l}(p, \mathfrak{n}) \text{ is part of the state assigned} \\ \text{to node } \mathfrak{m} \text{ by the DRFTA computation} \\ \wedge\ ((t(\mathfrak{m}) = p(\mathfrak{n})) \vee (\nu = p(\mathfrak{n})))\end{array} \equiv \begin{array}{l}RPath(p, \mathfrak{n}) \text{ matches} \\ \text{ending at node } \mathfrak{m}\end{array}$$

□

In other words, the state assigned to a node and the symbol at that node and ν together determine the set of all matching stringpaths ending at that node. As indicated before, the DRFTA and associated output function can thus be used in a root-to-frontier subject tree traversal to detect all stringpath matches.

5.3 A DRFTA-based TPM algorithm

As an invariant, when visiting a node $\mathfrak{n}$ of the given subject tree t, the DRFTA is in the state assigned to the node based on the symbols on the rootpath $RPath(t, \mathfrak{n})$ with the exception of the last symbol of this rootpath—symbol $t(\mathfrak{n})$.

As in the previous algorithm, the recursive procedure is called on every child i of the current node, with a state obtained by projecting away all except the ith component of the state tuple reached by a transition from the current state on $t(\mathfrak{n})$.

When visiting a node, the algorithm should register any matches indicated by the DRFTA's *output* function for the current state and either symbol $t(\mathfrak{n})$ or ν, since ν matches any subtree. This leads to the following algorithm:

{ Pre: $q = p_n$ where $n = |\mathfrak{n}| \wedge p_0 = q_0 \wedge$
$\langle \forall\, i :\ 1 \le i \le n\ :\ p_i = \pi_{RPath(t,\mathfrak{n})_{2i}}(R_{RPath(t,\mathfrak{n})_{2i-1}}(p_{i-1})) \rangle$ }
proc $Traverse(q : Q, \mathfrak{n} : D) =$
[[**var** $q_{next} : Q;\ i : \mathbb{N}_+ ; sp : (V \cdot \mathbb{N}_+)^* \cdot V$
| **for** $i : 1 \le i \le r'(t(\mathfrak{n})) \rightarrow$
$q_{next} := \pi_i(R_{t(\mathfrak{n})}(q));$
$Traverse(q_{next}, \mathfrak{n} \cdot i)$
rof;
for $sp : sp \in output(q, t(\mathfrak{n})) \rightarrow$
"register sp match at its endpoint $\mathfrak{n}$";
rof
for $sp : sp \in output(q, \nu) \rightarrow$
"register sp match at its endpoint $\mathfrak{n}$";
rof
]];
{ Post: every stringpath match in t whose endpoint is in the subtree $t@\mathfrak{n}$
has been registered at its endpoint }

{ Pre: $M_{\text{DRFTA}} = (Q, V', r', R, q_0, Q_{la})$ is the DRFTA built on the
pattern tree, and *output* is the associated output function }
$Traverse(q_0, \varepsilon)$
{ Post: every stringpath match in t has been registered at its endpoint }

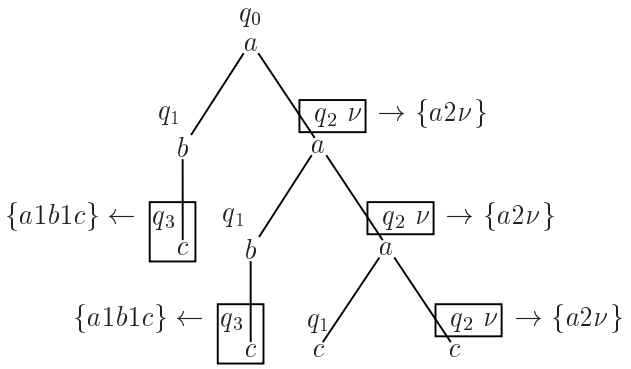


Figure 9: DRFTA state assignment and stringpath matches

Example 29. Figure 9 shows the states associated with every node and matches detected by the algorithm for $t = a(b(c), a(b(c), a(c, c)))$. Combinations of states and symbols corresponding to stringpath matches are framed. Note that symbol ν is only explicitly depicted for nodes at which it occurs in a stringpath match. □

6 Concluding remarks

We presented two algorithms for stringpath-based tree pattern matching. One of these, based on a root-to-frontier tree traversal and using an Aho-Corasick automaton, is already well known from the literature [HO82, AGT89, vdM88, AG85]. The other, based on a root-to-frontier tree traversal and using a DRFTA, is new. By presenting the two in a similar style, we highlighted their similarities and provided a missing link between TPM algorithms using tree automata and those using stringpath automata.

The two TPM algorithms are very similar, their difference being restricted to the different automata and output functions used. As future work, we intend to compare the automata in more detail. We conjecture that they are in some sense equivalent, i.e. can be transformed into one another.

We intend to extend the new algorithm to multiple tree patterns and from there to a tree acceptance and a tree parsing algorithm, providing related solutions to the related problems of tree acceptance and tree parsing. The result will likely be similar to the Aho-Corasick-based tree acceptance and tree parsing algorithms of Aho, Ganapathi & Tjiang [AGT89, vdM88, AG85].

Finally, it would be interesting to investigate the use of different keyword pattern matching automata or algorithms—such as those in [CWZ04, Wat95]—to obtain new tree pattern matching algorithms that are based on stringpath matching. One such algorithm, using Boyer-Moore pattern matching, was presented in [Wat97].

References

[AC75] A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.

[AG85] A.V. Aho and M. Ganapathi. Efficient tree pattern matching: An aid to code generation. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, pages 334–340, 1985.

[AGT89] A.V. Aho, M. Ganapathi, and S.W.K. Tjiang. Code generation using tree matching and dynamic programming. *ACM Transactions on Programming Languages and Systems*, 11(4):491–516, 1989.

[CH97] R. Cole and R. Hariharan. Tree pattern matching and subset matching in randomized o(n log3 m) time. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 66–75, 1997.

[Cha87] David R. Chase. An improvement to bottom-up tree pattern matching. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 168–177. ACM, 1987.

[CHI99] R. Cole, R. Hariharan, and P. Indyk. Tree pattern matching and subset matching in deterministic o($nlog^3n$) time. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 245–254, 1999.

[CR03] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology - Text Algorithms*. World Scientific Publishing, 2003.

[CWZ04] Loek Cleophas, Bruce W. Watson, and Gerard Zwaan. Automaton-based sublinear keyword pattern matching. In *Proceedings of the 11th international conference on String Processing and Information REtrieval (SPIRE 2004)*, volume 3246 of *LNCS*. Springer, October 2004.

[DGM94] M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. *Journal of the ACM*, 41(2):205–213, 1994.

[Eng75] Joost Engelfriet. Tree Automata and Tree Grammars. Lecture Notes DAIMI FN-10, Aarhus University, April 1975.

[FSW94] Christian Ferdinand, Helmut Seidl, and Reinhard Wilhelm. Tree automata for code selection. *Acta Informatica*, 31:741–760, 1994.

[GS97] Ferenc Gécseg and Magnus Steinby. *Tree Languages*, volume 3 of *Handbook of Formal Languages*, pages 1–68. Springer, 1997.

[HC86] Philip J. Hatcher and Thomas W. Christopher. High-quality code generation via bottom-up tree pattern matching. In *Conference Record of the Thirteenth Annual ACM Symposium on Principles of Programming Languages*, pages 119–130. ACM, 1986.

[HK89] C. Hemerik and J.P. Katoen. Bottom-up tree acceptors. *Science of Computer Programming*, 13(1):51–72, 1989.

[HO82] C.M. Hoffmann and M.J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, January 1982.

[Kos89] S.R. Kosaraju. Efficient tree pattern matching. In *Proceedings of the 30th annual IEEE Symposium on Foundations of Computer Science, FOCS'89*, pages 178–183. IEEE Computer Society Press, 1989.

[Kro75] H. Kron. *Tree templates and subtree transformational grammars*. PhD thesis, University of California, Santa Cruz, 1975.

[NR02] Gonzalo Navarro and Mathieu Raffinot. *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

[vD87] Yolanda van Dinther. De systematische afleiding van acceptoren en ontleders voor boom-grammatica's. Master's thesis, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, August 1987. (In Dutch).

[vdM88] H.J.A. van de Meerakker. Een parsing algoritme voor boomgrammatica's. Master's thesis, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, May 1988. (In Dutch).

[Wat95] Bruce W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms*. PhD thesis, Technische Universiteit Eindhoven, September 1995.

[Wat97] Bruce W. Watson. A Boyer-Moore (or Watson-Watson) Type Algorithm for Regular Tree Pattern Matching. In *Proceedings of the Prague Stringology Club Workshop '97*, pages 33–38, 1997.