TTATT 2016

Proceedings of the Workshop

Workshop on Trends in Tree Automata and Tree Transducers

Seoul, South Korea
July 18, 2016

Preface

These proceedings contain the papers presented at TTATT 2016, the 4th International Workshop on Trends in Tree Automata and Tree Transducers, which was held on July 18, 2016 in Seoul, South Korea in collocation with CIAA 2016, the 21st International Conference on Implementation and Application of Automata. In total, the workshop received 11 submissions, which were reviewed by 3 program committee members. Most submission were appropriate, but due to the time constraints of the 1-day workshop, the committee decided to accept only 7 papers. In addition, to the contributions present in these proceedings, the program also included the keynote lecture *Determinacy and Query Preservation of Tree Transducers* by Kenji Hashimoto (Nagoya University, Japan), who kindly accepted our invitation on short notice after we unfortunately lost our previous confirmed invited speaker, the late Zoltán Ésik (University of Szeged, Hungary). Zoltán was one of the pioneers of the field covered by the workshop and his contributions will remain a lasting influence on the field. In a minute of silence we commemorated Zoltán and his work.

The 4th workshop follows 3 workshops in the series originally estabished under the auspices of Hiroyuki Seki (Nagoya University, Japan). He also organized the first TTATT workshop in 2012 as part of RTA 2012 in Nagoya, Japan. The 2nd workshop followed in the next year and took place in Hanoi, Vietnam in collocation with ATVA 2013. It was organized by Sebastian Maneth (University of Edinburgh, UK). Finally, the 3rd workshop was the first workshop of the series in Europe, and it took place in 2015 as part of the ETAPS joint conferences in London, UK under the auspices of Emmanuel Filiot (Université Libre de Bruxelles, Belgium).

Naturally, even a small workshop like TTATT 2016 relies on the help of several other people. The organizer would like to thank all authors of submissions; their interest and contributions make the workshop possible and demonstrate the significance of such a meeting. Next, I would like to thank all the program committee members, who worked hard during the short evaluation period. Nevertheless the evaluations were achieved in a timely and fair manner. The conference management system EasyChair made the submission management process as smooth as possible, so I gratefully acknowledge their support. Last but not least, I would like to thank Yo-Sub Han (Yonsei University, South Korea), the main organizer of CIAA 2016, and his team, who made the collocation possible and took care of all the local arrangements.

July 10, 2016 Andreas Maletti
Stuttgart

Table of Contents

Program Committee

Arnaud Carayol	IGM, Université Paris Est & CNRS, France
Olivier Carton	LIAFA, Université Paris Diderot, France
Frank Drewes	Umeå University, Sweden
Emmanuel Filiot	Université Libre de Bruxelles, France
Zoltan Fülöp	University of Szeged, Hungary
Andreas Maletti	Universität Stuttgart, Germany
Sebastian Maneth	University of Edinburgh, UK
Wim Martens	University of Bayreuth, Germany
Akimasa Morihata	University of Tokyo, Japan
Keisuke Nakano	The University of Electro-Communications, Japan
Joachim Niehren	Inria Lille, France
Damian Niwinski	Warsaw University, Poland
Helmut Seidl	TU München, Germany
Hiroyuki Seki	Nagoya University, Japan
Jean-Marc Talbot	LIF, Universite d'Aix-Marseille, France
Margus Veanes	Microsoft Research, USA
Heiko Vogler	TU Dresden, Germany
Bruce Watson	Stellenbosch University, South Africa
James Worrell	Oxford University, UK

Additional Reviewers

- Dietze, Toni
- Hashimoto, Kenji
- Ivan, Szabolcs
- Osterholzer, Johannes
- Vagvolgyi, Sandor

On the N-Best Problem for Hypergraphs

Johanna Björklund, Frank Drewes, and Anna Jonsson

Department of Computing Science, Umeå University, 901 87 Umeå, Sweden
{johanna,drewes,aj}@cs.umu.se

Abstract. We propose an algorithm for computing the N best roots of a weighted hypergraph, in which the weight function is given over an idempotent and multiplicatively monotone semiring. We give a set of conditions that ensures that the weight function is well-defined and that solutions exist. Under these conditions, we prove that the proposed algorithm is correct. This generalizes a previous result for weighted tree automata, and in doing so, broadens the practical applications.

1 Introduction

Suppose that we can solve an optimisation problem A by solving, in succession, the problems $A_1, \ldots, A_n$. A simple way of approaching the joint optimization over the cascade $A_1, \ldots, A_n$ is to find the N best solutions to A_1, and take these as input to A_2. We then compute the N best solutions to A_2 for each of these inputs, and prune the combined output down to the N best alternatives. The computation continues in this fashion until we have the outputs for A_n, at which point we take the top-ranking one as the best solution to A. In general, this approach will not yield an optimal solution, but it is often a viable heuristic.

The problem of finding N elements that are optimal with respect to some ranking device is referred to as the N-best problem. Typical ranking devices are weighted automata, in which case N distinct elements with as small weights as possible are sought. If the automaton is nondeterministic, having several distinct runs on some inputs, the N-best problem is harder than the related *N-best derivations problem*, which asks for the N best individual runs of the automaton.

Mohri and Riley [4] provide an algorithm for solving the N-best problem for weighted string automata over the tropical semiring. To keep the run time polynomial, they use a combination of lazy determinisation and Dijkstra's N-shortest paths algorithm. In [1], we generalise this algorithm to work for weighted tree automata over an extremal semiring. In doing so, we simplify the search technique by working directly with the input automaton rather than an on-the-fly determinisation. To mitigate the added dimensionality caused by working with trees rather than strings, we propose a pruning technique that leads to an efficient algorithm. The running time is comparable with that in [2] for computing the N-best derivations.

In this paper, which describes on-going work, we consider the N-best problem for weighted hypergraphs over idempotent and multiplicatively monotone semirings. The hypergraphs may be infinite, but may not contain cycles. Intuitively, this provides an abstraction and generalisation of the approach in [1] if the hypergraph is chosen to represent the set of all trees over a given ranked alphabet. The more general setting means that less is known about the structure of the input graph, and this makes the computations potentially more demanding. However, we believe that the running time of the proposed algorithm is comparable with that of the previous algorithms under the same domain restrictions, and will attempt to prove this in our continued work.

2 Preliminaries

We write $\mathbb{N}$ for the set of non-negative integers, $\mathbb{N}^\infty$ for $\mathbb{N} \cup \{\infty\}$, $\mathbb{R}_+$ for the set of non-negative reals, and $\mathbb{R}_+^\infty$ for $\mathbb{R}_+ \cup \{\infty\}$. Given $n \in \mathbb{N}$, we let $[n] = \{1, \ldots, n\}$, and $[\infty] = \mathbb{N}$. In particular, $[0] = \emptyset$. The powerset of a set S is denoted by $pow(S)$. Given a k-tuple $v = (a_1, \ldots, a_k)$ we may denote its ith component a_i ($i \in [k]$) by $v(i)$. A sequence is *non-repetitive* if each element occurs at most once. A function $\pi \colon S \to S'$ is implicitly extended to a function from sequences over S to sequences over S' and from $pow(S)$ to $pow(S')$ in the usual elementwise fashion, without making a notational distinction between π and these canonical extensions. Given a sequence w, $[w]$ denotes the smallest set S such that w is a sequence over S. The set of all strings, i.e. finite sequences, over S is denoted by S^*; it includes the empty string λ.

A (commutative) *semiring* is a tuple $\mathcal{A} = (A, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ such that $(A, \oplus, \mathbb{0})$ and $(A, \otimes, \mathbb{1})$ are commutative monoids, $\otimes$ distributes both-sided over $\oplus$, and $\mathbb{0}$ is an absorbing element with respect to $\otimes$.

A *quasi-order* on S is a reflexive, transitive binary relation $\leq$. We write $a < b$ to express that $a \leq b$ but $b \not\leq a$. As usual, $\geq$ and $>$ denote the inverses of $\leq$ and $<$, resp. A quasi-order is a *partial order* if it is antisymmetric, and it is *well-founded* if there are no infinite descending chains, i.e., there is no infinite sequence $a_1 > a_2 > a_3 > \cdots$. A stronger notion than that of well-foundedness is that of a *well quasi-order* (wqo). A quasi-order is a wqo if every infinite sequence $a_1, a_2, \ldots$ eventually increases, i.e., there are $i < j$ such that $a_i \leq a_j$.

A semiring $\mathcal{A}$ is *idempotent* if $a \oplus a = a$ for all $a \in A$. In this case, there is a partial order $\leq_\mathcal{A}$ on $\mathcal{A}$, called the *natural order* of $\mathcal{A}$, which is given by $a \leq_\mathcal{A} b \iff a \oplus b = a$. Idempotent semirings are *monotonic* with respect to their natural order [3, Lemma 2], in other words, $a \leq_\mathcal{A} b$ implies $a \odot c \leq_\mathcal{A} b \odot c$ for all $c \in A$ and $\odot \in \{\oplus, \otimes\}$.

The semiring $\mathcal{A}$ is *finitely generated* if there is a finite subset A' of A such that every $a \in A$ can be written as a sum of products of elements in A'. A finitely generated idempotent semiring in which $\mathbb{1}$ is the minimal element is *nice.*

Example 1. The min-plus semiring $\mathbb{R}_+^\infty$, with $\min(a, b)$ serving as addition and ordinary addition as multiplication, is idempotent. It is even extremal, i.e. $a \oplus b \in \{a, b\}$. By generalising the domain to vectors of length $k \in \mathbb{N}$ over $\mathbb{R}_+^\infty$ and applying semiring addition and multiplication component-wise, we get an idempotent semiring that is not extremal. If we restrict the domain of the min-plus semiring to $\mathbb{N}^\infty$, it becomes finitely generated (by $\{0, 1, \infty\}$) and, in fact, nice because $\mathbb{1} = 0$ is its smallest element. The extension to vectors is still nice.

Rather than working on ordinary graphs, in which edges have a single source and a single target, we consider hypergraphs in which hyperedges may have several sources (but still only one target). This is particularly convenient for representing sets of trees: a hyperedge labelled f with n sources corresponds to an occurrence of the symbol f of rank n in a tree. The sources and the target represent the roots of the direct subtrees and of the tree itself, respectively.

Definition 2 (Hypergraph). *A* hypergraph *is a tuple* $G = (V, E, \mathit{src}, \mathit{tar})$ *such that* V *and* E *are disjoint sets of* nodes *and* hyperedges*, respectively,* $\mathit{src}\colon E \to V^*$ *assigns to each* $e \in E$ *a sequence of* sources $\mathit{src}(e)$*, and* $\mathit{tar}\colon E \to V$ *assigns to each* $e \in E$ *a* target $\mathit{tar}(e)$*.*

A node $v \in V$ *is an* end *if* $v \notin [\mathit{src}(e)]$ *for all* $e \in E$*. A* path *in* G *is a nonempty sequence* $\pi = e_0 \cdots e_n \in E^*$ *such that* $\mathit{tar}(e_{i-1}) \in [\mathit{src}(e_i)]$ *for all* $i \in [n]$*. Its* target $\mathit{tar}(\pi)$ *is* $\mathit{tar}(e_n)$ *and we say that* $\mathit{tar}(\pi)$ *is* reachable *from* e_0*.*

With this definition, hypergraphs can be infinite structures and may have parallel hyperedges. In the following, we shall simply speak of graphs and edges instead of hypergraphs and hyperedges.

Given a graph G (which will be understood from the context), we define $\mathit{hull}\colon \mathit{pow}(V) \to \mathit{pow}(E)$ by $\mathit{hull}(U) = \{e \in E \mid \mathit{src}(e) \in U^* \text{ and } \mathit{tar}(e) \notin U\}$ for all $U \subseteq V$. Hence, $\mathit{hull}(U)$ yields the set of all edges that lead from nodes in U to nodes outside U. In particular, $\mathit{hull}(\emptyset)$ is the set of edges that have no sources. For a finite set $F \subseteq E$ of edges, $\mathit{hull}(F)$ abbreviates $\mathit{hull}(\mathit{tar}(F))$. Moreover, $\mathit{hull}^{\leq 0}(F) = F$ and $\mathit{hull}^{\leq n+1}(F) = \mathit{hull}(\mathit{hull}^{\leq n}(F)) \cup \mathit{hull}^{\leq n}(F)$ for $n \in \mathbb{N}$.

Definition 3 (Layered graph). *A graph* G *is* layered *if (a)* $\mathit{hull}^{\leq n}(\emptyset)$ *is finite for every* $n \in \mathbb{N}$*, (b) for every node* $v \in V$ *there are only finitely many paths* π *such that* $\mathit{tar}(\pi) = v$*, and (c)* $V = \mathit{tar}(E)$*.*

In particular, layered graphs are acyclic by requirement (b). Requirements (b) and (c) make sure that all of G can gradually be built up from the "bottom" by starting with the empty set of edges and repeatedly applying *hull*. In this process requirement (a) guarantees that the subgraph obtained always stays finite.

Definition 4 (Weighted graph). *Let* $\mathcal{A} = (A, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ *be a semiring. A* graph with weights in $\mathcal{A}$*, also called a* weighted graph*, is a tuple* $G = (V, E, \mathit{src}, \mathit{tar}, \hat{w})$ *such that* $(V, E, \mathit{src}, \mathit{tar})$ *is a layered graph and* $\hat{w}\colon E \to A$ *is its* weight function*. We let* $w\colon V \cup E \to A$ *be determined by the following conditions:*

- $w(v) = \bigoplus_{e \in \mathit{tar}^{-1}(v)} w(e)$ *for every node* v *and*
- $w(e) = \hat{w}(e) \otimes \bigotimes_{i \in [n]} w(v_i)$ *for every edge* e*, where* $\mathit{src}(e) = v_1 \cdots v_n$*.*

Since G is layered, for every node v there exists a path of maximum length whose target is v. By induction on the length of this path, it follows that w is uniquely determined.

Henceforth, given a graph G with weights, we denote its components by V_G, E_G, src_G, tar_G, and $\hat{w}_G$, respectively, and its induced weight function by w_G.

3 Computing N best nodes

Definition 5 (the N-best nodes problem). *The* N-best nodes problem *is defined as follows. An instance is a pair (G, N) consisting of*

- *a weighted graph G with weights in a well-founded semiring,*[1]
- *a set of* target nodes $V^T \subseteq V_G$ *such that each $v \in V^T$ is an end, and*
- *an integer $N \in \mathbb{N}^\infty$ such that $N \leq |V^T|$.*

A solution is a sequence of N pairwise distinct elements $v_1, v_2, \cdots$ of V^T such that there do not exist $i \in [N]$ and $v \in V^T \setminus \{v_1, \ldots, v_i\}$ with $w(v) <_{\mathcal{A}} w(v_i)$.

In practical instantiations of Definition 5, the set V^T may correspond to accepting configurations.

Before continuing, let us verify that the N-best nodes problem (with the semiring $\mathcal{A}$ being well-founded) always has a solution. For this, choose any element u_0 of V^T and build a sequence of nodes $u_0, u_1, u_2, \ldots$ in V^T with strictly decreasing weights. As $\mathcal{A}$ is well-founded, every such sequence is finite. Thus, the process eventually arrives at a node v_1 such that no node of strictly lesser weight exists in V^T. Now, fix v_1 and repeat the argument (with $V^T \setminus \{v_1\}$ instead of V^T). Continue until N elements $v_1, \ldots, v_N$ have been found (or *ad infinitum* if $N = \infty$). Clearly, $v_1, \ldots, v_N$ is a solution.

In almost all applications, weights will be generated by a finite subset of $\mathcal{A}$, such as the finite set of weights of the rules of a weighted tree automaton. Next, we show that such a situation provides us with a well-founded semiring. For this, we need some additional notation and terminology for reasoning about vectors.

Let $m, m' \in \mathbb{N}^k$. We let $m \leq m'$ if $(\forall i \in [k])\, m(i) \leq m'(i)$ and, as usual, $m < m'$ if $m \leq m'$ and $m \neq m'$. An element $m \in M$ of a finite set $M \subseteq \mathbb{N}^k$, is *minimal* if there is no $m' \in m$ such that $m' < m$. We denote the subset of M of minimal elements by $min(M)$.

Lemma 6. *Every nice semiring $\mathcal{A}$ is well-founded with respect to the natural order.*

Proof. The proof makes use of Dickson's lemma. Recall that this lemma states that $(\mathbb{N}^k, \leq)$ is a wqo.

Let $\mathcal{A}$ be generated by $\{a_1, \ldots, a_k\}$. For $m \in \mathbb{N}^k$ let $\varphi(m) = \bigotimes_{j \in [k]} a_j^{m(j)}$, and for a finite set $M \subseteq \mathbb{N}^k$ let $\Phi(M) = \bigoplus_{m \in M} \varphi(m)$ (where $\Phi(\emptyset) = \mathbb{1}$). Then, since $\otimes$ distributes over $\oplus$, every $a \in A$ is represented by at least one finite set $M \subseteq \mathbb{N}^k$ in the sense that $a = \Phi(M)$. We call such a set a *representative* of a.

We first prove that the value of $\Phi(M)$ is determined by the minimal elements of M, i.e., that the following statement holds.

$$\text{For finite } M \subseteq \mathbb{N}^k,\ \Phi(M) = \Phi(min(M)). \tag{1}$$

Indeed, if $m, m' \in M$ are such that $m < m'$, then we have $\varphi(m') = \varphi(m) \otimes a$ for some a (namely for $a = \varphi(m' - m)$). Hence $\varphi(m) = \varphi(m) \otimes \mathbb{1} \leq_{\mathcal{A}} \varphi(m) \otimes a =$

[1] A semiring is well-founded if its natural order is a well-founded quasi-order.

Algorithm 1 N-best nodes

```
 1: procedure $BestVertices(G, N)$
 2:     $U \leftarrow \emptyset$
 3:     for $i = 1, \ldots, N$ do
 4:         $H \leftarrow hull(U)$
 5:         select $e \in H$ such that $minWeight(e)$ is minimal
 6:         $v \leftarrow bestAncestor(e)$
 7:         output $v$
 8:         $U \leftarrow U \cup descendants(v)$
 9:     end for
10: end procedure
```

$\varphi(m')$ and thus $\varphi(m) \oplus \varphi(m') = \varphi(m)$ by the definition of $\leq_{\mathcal{A}}$. Every such non-minimal element $m' \in M$ can therefore be removed from M with no effect on $\Phi(M)$, which proves (1).

Suppose there is a descending sequence $s = b_0 >_{\mathcal{A}} b_1 >_{\mathcal{A}} b_2 >_{\mathcal{A}} \cdots$ and let M_i be a representative of b_i for all $i \in \mathbb{N}$. Since $\Phi(M_i \cup M_{i+1}) = \Phi(M_i) \oplus \Phi(M_{i+1}) = \Phi(M_{i+1})$ we may assume that $M_0 \subseteq M_1 \subseteq \cdots$. Now, pick $m_i \in min(M_i) \setminus min(M_{i-1})$ for every $i > 0$. This set cannot be empty, by (1) and the fact that $\Phi(M_i) \neq \Phi(M_{i-1})$. Furthermore, we have $m_{i-1} \not\leq m_i$, since $m_{i-1} \leq m_i$ would imply $m_i \notin min(M_i) \setminus min(M_{i-1})$. In other words, the sequence $m_0, m_1, \ldots$ is non-increasing and must thus be finite because $(\mathbb{N}^k, \leq)$ is a wqo. This shows that s is finite, and thus that $\mathcal{A}$ is well-founded. □

We devise a simple algorithm solving the N-best nodes problem, given an instance (G, N). As G is infinite and thus cannot be represented explicitly, we assume that it can be explored through a few procedures. These are as follows:

- There is a procedure that computes $hull(U)$ for every finite set $U \subseteq V_G$.
- For $e \in E_G$, $minWeight(e)$ returns a minimal element of all $w_G(v)$ such that $v \in V^T$ and v is reachable from e on some path π. If no such v exists then an error element $\top$ is returned, which is considered to be larger than all elements of $\mathcal{A}$. Note that $minWeight(e)$ is well-defined since $\mathcal{A}$ is well-founded.
- For every edge e such that $minWeight(e)$ is defined, $bestAncestor(e)$ returns some node $tar_G(\pi)$ such that π is as in the definition of $minWeight(e)$ above.
- For every node $u \in V_G$, $descendants(u)$ returns the set of nodes v of which u is an ancestor (including u itself). Note that this set is finite as G is layered.

The pseudocode is given in Algorithm 1.

Remark 7. Were we to apply Algorithm 1 to finding not only ends with minimal weight, but nodes in general, then we would be faced with difficulties. Part of what makes the algorithm fast is that it only visits each node once. However, when a node v is on a minimal path to a node v', and v has greater weight than v', then the algorithm would have to visit v before outputting v', and then return to v. This problem is however easily done away with by, e.g., introducing dummy ends similarly to the introduction of a dummy root symbol in [1, p. 101].

Let us now verify that the Algorithm 1 is correct.

Theorem 8. *After $i \leq N$ executions of the loop body in Algorithm 1, it will have outputted i nodes $v_1, \dots, v_i \in V^T$ such that*

(1) there are no $v \in V^T \setminus \{v_1, \dots, v_i\}$ and $j \in [i]$ such that $v <_{\mathcal{A}} v_i$, and
(2) $U = \bigcup_{j \in [i]} descendants(v_j)$.

Proof. We proceed by induction on i. For $i = 0$ the assertions are trivially true as U is initialized to $\emptyset$. Thus, assume that (1) and (2) hold for some $i - 1 < N$ and consider the ith execution of the loop. Let $H^{\uparrow}$ be the set of all nodes that are reachable from some edge in $H = hull(U)$. We show that $(U, H^{\uparrow})$ is a partition of V_G, i.e., that $H^{\uparrow} = V_G \setminus U$. By (2) all descendants of a node $u \in U$ are in U as well. But by the definition of *hull*, $tar_G(H) \cap U = \emptyset$, and so $U \cap H^{\uparrow} = \emptyset$. Moreover, for $v \in V_G \setminus U$, as G is layered, there are only finitely many paths π such that $tar_G(\pi) = v$, and hence there is such a path, say $\pi = e_0 \cdots e_n$, of maximal length, which means that $src_G(e_0) = \lambda$. Now either $tar_G(e_0) \notin U$ and hence e_0 is in H, or there is a largest index $j \in [n]$ with $tar_G(e_{j-1}) \in U$, which means that $e_j \in H$. In both cases $v \in H^{\uparrow}$. This proves that, indeed, $H^{\uparrow} = V_G \setminus U$. Moreover, since all nodes in V^T are ends, $V^T \setminus \{v_1, \dots, v_{i-1}\} \subseteq V_G \setminus U = H^{\uparrow}$. Since a solution does exist, this means that there is a $v_i \in V^T \cap H^{\uparrow}$ such that $\{v_1, \dots, v_i\}$ satisfies (1), and it must be of minimal weight in $V^T \cap H^{\uparrow}$ because $\{v_1, \dots, v_{i-1}\} \notin H^{\uparrow}$.

It follows that lines 5 and 6 select v_i (or another node in V^T of the same weight) and v and Line 7 outputs it. Thus, (2) is now satisfied for i. □

Since the set U in Algorithm 1 is always of the form $\bigcup_{j \in [i]} descendants(v_j)$ for some ends $v_1, \dots, v_i$, the procedures $hull(U)$, $minWeight(e)$, and $bestAncestor(e)$ actually only need to be implemented for this special case. Further, the individual steps of the algorithm can be implemented by maintaining a priority queue for the edges $e \in H$ for which $minWeight(e) \neq \top$, the priority being given by $minWeight(e)$. Then Line 5 becomes a dequeueing operation. In Line 8 elements of the queue whose target nodes are included in U would be deleted from the queue, and Line 4 would enqueue those edges having all of their sources in U and at least one among those recently added to U (except initially, where H becomes the set of all edges not having any sources).

4 Conclusion and future work

The presented work is still in progress. It remains to recast the original algorithm for tree automata in the current setting, and to complement the theoretical analysis with an empirical evaluation. In the future, we are also interested in investigating other combinations of semirings and graph families, so as to further generalize the algorithm, or make it more efficient for restricted domains.

Acknowledgment. We thank the reviewers for valuable suggestions.

References

1. Björklund, J., Drewes, F., Zechner, N.: An efficient best-trees algorithm for weighted tree automata over the tropical semiring. In: Proceedings of the 9th International Conference on Language and Automata Theory and Applications, Nice, France. LNCS, vol. 8977, pp. 97–108 (2015)
2. Büchse, M., Geisler, D., Stüber, T., Vogler, H.: n-best parsing revisited. In: Applications of Tree Automata in Natural Language Processing, Uppsala, Sweden. pp. 46–54. Association for Computational Linguistics (2010)
3. Mohri, M.: Semiring frameworks and algorithms for shortest-distance problems. Journal of Automata, Languages and Combinatorics 7(3), 321–350 (2002)
4. Mohri, M., Riley, M.: An efficient algorithm for the n-best-strings problem. In: Proceedings of the Conference on Spoken Language Processing, Denver, Colorado (2002), `http://www.cs.nyu.edu/~mohri/pub/nbest.ps`, Digital publication.

A Lower Bound for the Length of the Wadge-Wagner Hierarchy of Regular Tree Languages

Jacques Duparc[1] and Kevin Fournier[1,2]

[1]Department of Information Systems
Faculty of Business and Economics
University of Lausanne
CH-1015 Lausanne, Switzerland

[2]Équipe de Logique Mathématique
Université Paris Diderot
UFR de Mathématiques case 7012
75205 Paris Cedex 13, France

Abstract. We investigate the complexity of the *Wadge-Wagner hierarchy* of regular infinite tree languages that relies on the following relation: L is less complicated than a language L' if L continuously reduce to L' We provide a hierarchy of such languages whose length requires infinitely many Veblen ordinal functions to be computed – a drastic extension of both the deterministic and the word cases and previously known results.

Introduction

This paper is a contribution to the fine understanding of the topological complexity of regular tree languages. It thoroughly extends some partial results on regular tree languages of index $[0, 2]$ previously obtained by the same authors in [4]. We make use of descriptive set theory to measure the complexity of these languages which is already highly involved for even the ones recognised by deterministic parity tree automata do not fit inside the Borel hierarchy – indeed $\mathbf{\Pi}^1_1$-complete languages naturally arise even with only two priorities. On the contrary, nondeterministic automata recognise languages that are neither analytic, nor coanalytic, nor in any difference of such sets. However, the expressive power of nondeterministic automata is bounded by the second level of the projective hierarchy, and, by Rabin's complementation result [7], all nondeterministic languages lie in the $\mathbf{\Delta}^1_2$ class.

The tool we rely on to investigate the complexity of these languages is the reduction by continuous functions (the so-called Wadge-reducibility). Granted with such a reduction relation, the complexity classes – called Wadge degrees – are formed of sets that are Wadge-reducible to each other. These Wadge degrees compose a hierarchy whose many levels – denoted *ranks*, can be specified in details with the use of ordinals.

In this extended abstract, we describe a succession of operations on infinite tree automata that haul up the Wadge degrees of the languages that they recognise[1]. By composing these operations together, one generates a hierarchy of

[1] We insist on that the whole study is completed without mention of any particular determinacy principle. In particular, $\mathbf{\Delta}^1_2$-determinacy is not required at all to show that regular infinite tree languages yield the hierarchy we exhibit.

regular tree languages of very high topological complexity for in order to compute its length, one needs to consider all the finite Veblen functions φ_n (any $n \in \mathbb{N}$). These Veblen functions are defined inductively. The first Veblen function $\varphi_0 : On \to On$ is defined as the exponentiation of base ω: $\varphi_0(\alpha) = \omega^\alpha$. $\varphi_{n+1} : On \to On$ is defined as the function that enumerates[2] the fixed points of φ_{n+1}.

The length of the hierarchy that these operations provide corresponds to the ordinal $\varphi_\omega(0) = \sup_{n\in\mathbb{N}} \varphi_n(0)$.

In comparison, when one replaces infinite trees by infinite words, the length of the Wadge hierarchy shrinks drastically since it becomes ω^ω (a result by Klaus Wagner in [10]). This hierarchy also dwarfs down to $(\omega^\omega)^3 + 3$ (a result by Filip Murlak in [6]) when only infinite tree languages recognised by *deterministic* automata are considered.

1 Preliminaries

1.1 The Wadge game

As usual, for Γ a *pointclass*[3] we denote by $\check{\Gamma}$ its *dual* class containing all the subsets whose complements are in Γ, and by $\Delta(\Gamma)$ the ambiguous class $\Gamma \cap \check{\Gamma}$. Given any topological space X, the *Wadge preorder* $\leq_W$ on $\mathscr{P}(X)$ is defined for $A, B \subseteq X$ by $A \leq_W B$ if and only if there exists $f : X \xrightarrow{cont.} X$ such that $f^{-1}(B) = A$. It is a preorder which induces an equivalence relation $\equiv_W$ whose equivalence classes – denoted by $[A]_W$ – are called the *Wadge degrees*. A set $A \subseteq X$ is *self-dual* if $[A]_W = [A^\complement]_W$, and *non-self-dual* otherwise.

The space T_Σ of infinite binary trees over the alphabet Σ equipped with the standard Cantor topology is homeomorphic to the Cantor space [2]. For any $L, M \subseteq T_\Sigma$, the Wadge game $W(L, M)$ is a two player infinite game in which each player builds a tree, say t_{I} and t_{II}. At every round, player I plays first, and both players add a finite number of children to the terminal nodes of their tree. Player II is allowed to skip her turn, but has to produce a tree in T_Σ throughout a game. Player II wins the game if and only if $t_I \in L \Leftrightarrow t_{II} \in M$.

[2] More precisely, the inductive definition of $\varphi_{n+1}(\alpha)$ is

- $\varphi_{n+1}(0) = \sup_{k\in\mathbb{N}} \underbrace{\varphi_n \circ \ldots \circ \varphi_n}_{k}(0)$;
- $\varphi_{n+1}(\alpha + 1) = \sup_{k\in\mathbb{N}} \underbrace{\varphi_n \circ \ldots \circ \varphi_n}_{k}\big(\varphi_{n+1}(\alpha) + 1\big)$;
- $\varphi_{n+1}(\lambda) = \sup_{\lambda_k\in\mathbb{N}} \varphi_{n+1}(\lambda_k)$ when λ is a limit ordinal with $\lambda = \sup_{\lambda_k\in\mathbb{N}}$.

For instance, $\varphi_1(0)$ is the ordinal known as $\varepsilon_0 = \sup_{n<\omega} \underbrace{\omega^{\cdot^{\cdot^{\omega^0}}}}_{n}$; and $\varphi_2(0)$ is the ordinal $\sup_{n<\omega} \underbrace{\varepsilon_{\cdot_{\cdot_{\varepsilon_0}}}}_{n}$.

[3] a pointclass is a collections of subsets of a topological space that is closed under continuous preimages (see [1]).

Lemma 1 ([9]). *Let $L, M \subseteq T_\Sigma$. Then $L \leq_W M$ if and only if player* II *has a winning strategy in the game $W(L, M)$.*

We write $A <_W B$ when II has a winning strategy in $W(A, B)$ *and* I has a winning strategy in $W(B, A)$[4]. Given a pointclass Γ of T_Σ with suitable closure properties, the assumption of the determinacy of Γ is sufficient to prove that Γ is semi-linearly ordered by $\leq_W$, denoted SLO(Γ), i.e. that for all $L, M \in \Gamma$,

$$L \leq_W M \qquad \text{or} \qquad M \leq_W L^{\complement},$$

and that $\leq_W$ is well founded when restricted to sets in Γ ([8,1]). Under these conditions, the Wadge degrees of sets in Γ with the induced order is thus a hierarchy called the *Wadge hierarchy.* Therefore, there exists a unique ordinal, called the height of the Γ-Wadge hierarchy, and a mapping d_W^Γ from the Γ-Wadge hierarchy onto its height, called the *Wadge rank*[5], such that, for every L, M non-self-dual in Γ, $d_W^\Gamma(L) < d_W^\Gamma(M)$ if and only if $L <_W M$ and $d_W^\Gamma(L) = d_W^\Gamma(M)$ if and only if $L \equiv_W M$ or $L \equiv_W M^{\complement}$. The wellfoundedness of the Γ-Wadge hierarchy ensures that the Wadge rank can be defined by induction as follows:

- $d_W^\Gamma(\emptyset) = d_W^\Gamma(\emptyset^{\complement}) = 1$
- $d_W^\Gamma(L) = \sup\left\{d_W^\Gamma(M) + 1 : M \text{ is non-self-dual}, M <_W L\right\}$ for $L >_W \emptyset$.

1.2 The Conciliatory Hierarchy

A *conciliatory* binary tree over a finite set Σ is a partial function $t : \{0,1\}^* \to \Sigma$ with a prefix closed domain. Such trees can have both infinite and finite branches. A tree is called *full* if $\text{dom}(t) = \{0,1\}^*$. Let $\mathcal{T}_\Sigma^{\leq\omega}$ and T_Σ denote, respectively, the set of all conciliatory binary trees and the set of full binary trees over Σ. Given $x \in \text{dom}(t)$, we denote by t_x the subtree of t rooted in x. Let $\{0,1\}^n$ denote the set of words over $\{0,1\}$ of length n, and let t be a conciliatory tree over Σ. We denote by $t[n]$ the finite initial binary tree of height $n+1$ given by the restriction of t to $\bigcup_{0 \leq i \leq n}\{0,1\}^i$.

For conciliatory languages L, M we define the *conciliatory* version of the Wadge game: $C(L, M)$ ([3,5]). The rules are similar, except for the fact that both player are now allowed to skip and to produce trees with finite branches - or even finite trees. For conciliatory languages L, M we use the notation $L \leq_c M$ if and only if II has a winning strategy in the game $C(L, M)$. If $L \leq_c M$ and $M \leq_c L$, we will write $L \equiv_c M$. The conciliatory hierarchy is thus the partial order induced by $\leq_c$ on the equivalence classes given by $\equiv_c$. We write $A <_c B$ when II has a winning strategy in $C(A, B)$ *and* I has a winning strategy in $C(B, A)$.

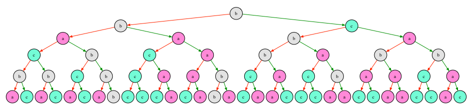
[4] This is in general stronger than the usual $A <_W B$ if and only if $A \leq_W B$ and $B \not\leq_W A$, but the two definitions coincide when the classes considered are determined.

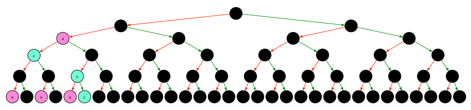
[5] However the main result of this article does not provide any Wadge rank for the canonical languages that are constructed, because we do not make use of any determinacy principle.

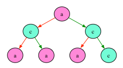
From a conciliatory language L over Σ, one defines the corresponding language L^b of full trees over $\Sigma \cup \{b\}$ by:

$$L^b = \left\{ t \in T_{\Sigma \cup \{b\}} : t_{[\ /b]} \in L \right\},$$

where b is an extra symbol that stands for “blank”, and $t_{[\ /b]}$, the *undressing* of t, is informally the conciliatory tree over Σ obtained once all the occurences of b have been removed in a top-down manner. More precisely, if there is a node v such that $t(v) = b$, we ignore this node and replace it with $v0$. If, for each integer n, $t(v0^n) = b$, then $v \notin \mathrm{dom}(t_{[\ /b]})$. This process is illustrated below


(a) A tree t with blanks


(b) blanks deleted in a top-down manner.


(c) The resulting tree $t_{[\ /b]}$.

We say a conciliatory language L is in a pointclass of *full trees* Γ if $L^b \subseteq \Gamma$.

Lemma 2. *For L and M any conciliatory languages,*

$$L \leq_c M \textit{ if and only if } L^b \leq_W M^b.$$

The mapping $L \mapsto L^b$ gives thus a natural embedding of the preorder $\leq_c$ restricted to conciliatory sets in Γ into the Γ-Wadge hierarchy. Hence, for Γ with suitable closure and determinacy properties, the conciliatory degrees of sets in Γ with the induced order constitute a hierarchy called the *conciliatory hierarchy*. We define, by induction, the corresponding *conciliatory rank* of a language:

- $d_c^\Gamma(\emptyset) = d_c^\Gamma(\emptyset^\complement) = 1$
- $d_c^\Gamma(L) = \sup\{d_c^\Gamma(M) + 1 : M <_c L\}$ for $L >_c \emptyset$.

Similarly to the Wadge case, given two pointclasses Γ and Γ', for every conciliatory $L \in \Gamma \cap \Gamma'$, $d_c^\Gamma(L) = d_c^{\Gamma'}(L)$. Observe that the conciliatory hierarchy does not contain self-dual languages: a strategy for I in $C(L, L^\complement)$ is to skip in the first round, and then copy moves of II.

1.3 Automata and conciliatory trees

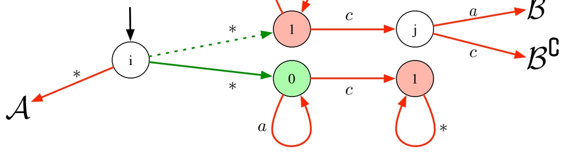
A *nondeterministic parity tree automaton* $\mathcal{A} = \langle \Sigma, Q, I, \delta, \mathrm{r} \rangle$ consists of a finite input alphabet Σ, a finite set Q of states, a set of initial states $I \subseteq Q$, a transition relation $\delta \subseteq Q \times \Sigma \times Q \times Q$ and a priority function $\mathrm{r} : Q \to \omega$. A run of automaton $\mathcal{A}$ on a binary conciliatory input tree $t \in \mathcal{T}_\Sigma^{\leq\omega}$ is a conciliatory tree $\rho_t \in \mathcal{T}_Q^{\leq\omega}$ with $\mathrm{dom}(\rho_t) = \{\varepsilon\} \cup \{va : v \in \mathrm{dom}(t) \wedge a \in \{0,1\}\}$ such that the root of this tree is labeled with a state $q \in I$, and for each $v \in \mathrm{dom}(t)$, transition $(\rho_t(v), t(v), \rho_t(v_1), \rho_t(v_1)) \in \delta$. We say that $\mathcal{A}$ *accepts* t if there exists a run ρ_t that is *accepting*: either the highest priority of a state occurring infinitely often on the branch is even or the priority of each leaf node in ρ_t is even.

2 Operations on infinite tree automata

We briefly describe the operations on tree automata that yield a very long hierarchy of conciliatory tree languages. We let $\Sigma = \{a, c\}$ and use the following conventions in the diagrams. Nodes represent states of the automaton. Node labels correspond to state ranks. A red edge shows the state assigned to the left successor node of a transition, a green edge goes to the right successor node. [6]

2.1 The sum (+)

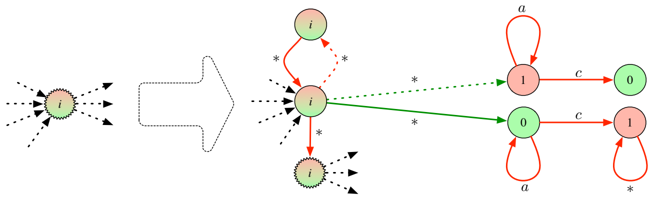
Given any automata $\mathcal{A}$ and $\mathcal{B}$ we form[7] the automaton $\mathcal{B}+\mathcal{A}$:



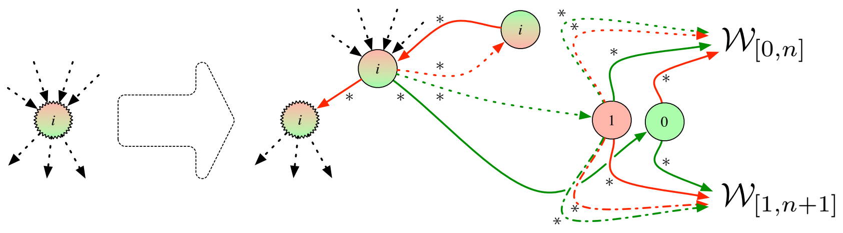
We write $\mathcal{A}\bullet n$ for $\underbrace{\mathcal{A}+\ldots+\mathcal{A}}_{n}$.

2.2 The operation φ_0

Given any automaton $\mathcal{A}$ we form $\varphi_0(\mathcal{A})$ by replacing each state in $\mathcal{A}$ with



2.3 The operations φ_{n+1}



[6] In order to lighten the notation, transitions that are not depicted on a diagram lead to some all-accepting state.

[7]

– $i = 0$ if the empty tree is accepted by $\mathcal{A}$, and $i = 1$ otherwise;
– $j = 1$ if $L(\mathcal{A})$ is equivalent to $L(\mathcal{A}) \to \ominus$, where $\ominus$ denotes any automaton that rejects all trees, and $j = 0$ otherwise.

Given any automaton $\mathcal{A}$ and any automaton $\mathcal{W}_{[0,n]}$ with priorities inside $[0,n]$ whose language Wadge-reduces every regular language recognised by an automaton with priorities inside $[0,n]$. We form $\varphi_{n+1}(\mathcal{A})$ by replacing each state in $\mathcal{A}$ with

Given automata $\mathcal{A}$ and $\mathcal{B}$, we write $\mathcal{A} \leq_c \mathcal{B}$ for $L(\mathcal{A}) \leq_c L(\mathcal{B})$, and same with $<_c, \leq_W, <_W$. These operations satisfy the following properties.

Lemma 3. *Let $\mathcal{A}$, $\mathcal{A}'$, $\mathcal{B}$ and $\mathcal{B}'$ be conciliatory languages such that $\mathcal{A} <_c \mathcal{A}'$ and $\mathcal{B} \leq_c \mathcal{B}'$, and $0 \leq n < m < \omega$. Then, the following hold.*

1. $(\mathcal{B}+\mathcal{A})^{\complement} \equiv_c \mathcal{B}+(\mathcal{A})^{\complement}$
2. $\mathcal{B}+\mathcal{A} <_c \mathcal{B}'+\mathcal{A}'$;
3. $\mathcal{B} <_c \mathcal{B}+\mathcal{A}$.
4. $\mathcal{A}\bullet m <_c \varphi_0(\mathcal{A}+\ominus)$
5. $\varphi_n(\mathcal{A})^{\complement} \equiv_c \varphi_n(\mathcal{A}^{\complement})$.
6. $\varphi_n(\mathcal{B}) \leq_c \varphi_n(\mathcal{B}')$.
7. $\varphi_n(\mathcal{A}) <_c \varphi_n(\mathcal{A}')$.
8. $\varphi_n \circ \varphi_m(\mathcal{A}) \equiv_c \varphi_m(\mathcal{A})$.

We recall that every ordinal $\alpha > 0$ admits a unique Cantor Normal Form of base ω (CNF) which is an expression of the form $\alpha = \omega^{\alpha_k} \cdot \nu_k + \cdots + \omega^{\alpha_0} \cdot \nu_0$ where $k < \omega$, $0 < \nu_i < \omega^\omega$ for any $i \leq k$, and $\alpha_0 < \ldots < \alpha_k < \alpha$. For every ordinal $0 < \alpha < \varphi_\omega(0)$, we inductively define a pair of automata $(\mathcal{A}_\alpha, \bar{\mathcal{A}}_\alpha)$ whose languages are $\leq_c$-incomparable by setting:

$$\mathcal{A}_\alpha = \mathcal{A}_{\omega^{\alpha_k}} \bullet \nu_k + \cdots + \mathcal{A}_{\omega^{\alpha_0}} \bullet \nu_0, \qquad \bar{\mathcal{A}}_\alpha = \mathcal{A}_{\omega^{\alpha_k}} \bullet \nu_k + \cdots + \bar{\mathcal{A}}_{\omega^{\alpha_0}} \bullet \nu_0,$$

where $\mathcal{A}_{\omega^{\alpha_i}}$ and $\bar{\mathcal{A}}_{\omega^{\alpha_i}}$ are respectively:

- $\ominus$ and $\ominus^{\complement}$ if $\alpha_i = 0$;
- $\varphi_0(\mathcal{A}_{\alpha_i})$ and $\varphi_0(\bar{\mathcal{A}}_{\alpha_i})$ if $\alpha_i < \omega^{\alpha_i}$;
- $\varphi_{n+1}(\mathcal{A}_\beta)$ and $\varphi_{n+1}(\bar{\mathcal{A}}_\beta)$ if $\beta < \alpha_i = \omega^{\alpha_i} = \varphi_n(\alpha_i) = \varphi_{n+1}(\beta)$.

Lemma 4. *For $0 < \alpha < \beta < \varphi_\omega(0)$, we have*

1. $\mathcal{A}_\alpha \not\leq_c \bar{\mathcal{A}}_\alpha$ *and* $\bar{\mathcal{A}}_\alpha \not\leq_c \mathcal{A}_\alpha$.
2. $\mathcal{A}_\alpha <_c \mathcal{A}_\beta$; $\bar{\mathcal{A}}_\alpha <_c \mathcal{A}_\beta$; $\mathcal{A}_\alpha <_c \bar{\mathcal{A}}_\beta$ *and* $\bar{\mathcal{A}}_\alpha <_c \bar{\mathcal{A}}_\beta$.

Applying the embedding $L \mapsto L^b$, we obtain the main result[8].

Theorem 1. *There exists a family $\left({\mathcal{A}_\alpha}^b\right)_{\alpha<\varphi_\omega(0)}$ of parity tree automata s.t.*

1. *each ${\mathcal{A}_\alpha}^b$ recognises languages of full trees over the alphabet $\{a, b, c\}$;*
2. *$\alpha < \beta$ holds if and only if ${\mathcal{A}_\alpha}^b <_W {\mathcal{A}_\beta}^b$ holds.*

[8] It is both new and hard to prove, yielding an unexpectedly high lower bound for the height of the Wadge hierarchy on ω-regular tree languages.

References

1. Andretta, A., Louveau, A.: Wadge degrees and pointclasses. In: Kechris, A.S., Löwe, B., Steel, J.R. (eds.) Wadge Degrees and Projective Ordinals: The Cabal Seminar, Volume II. Cambridge University Press (2012)
2. Arnold, A., Duparc, J., Murlak, F., Niwiński, D.: On the topological complexity of tree languages. Logic and automata: History and Perspectives 2, 9–29 (2007)
3. Duparc, J.: Wadge hierarchy and Veblen hierarchy, Part I : Borel sets of finite rank. The Journal of Symbolic Logic 66(1), 56 – 86 (2001)
4. Duparc, J., Fournier, K.: A tentative approach for the wadge-wagner hierarchy of regular tree languages of index [0, 2]. In: Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings. pp. 81–92 (2015)
5. Duparc, J., Murlak, F.: On the topological complexity of weakly recognizable tree languages. In: FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings. pp. 261–273 (2007)
6. Murlak, F.: The wadge hierarchy of deterministic tree languages. Logical Methods in Computer Science 4(4) (2008)
7. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the AMS 141, 1–23 (1969)
8. Van Wesep, R.: Wadge degrees and descriptive set theory. In: Cabal Seminar 76–77. pp. 151–170. Springer (1978)
9. Wadge, W.W.: Reducibility and determinateness on the Baire space. Ph.D. thesis, University of California, Berkeley (1984)
10. Wagner, K.W.: On omega-regular sets. Information and Control 43(2), 123–177 (1979)

Rigid Tree Automata With Isolation

Nathaniel Wesley Filardo and Jason Eisner

Johns Hopkins University

Abstract. Rigid Tree Automata (RTAs) are a strict super-class of Regular Tree Automata (TAs), additionally capable of recognizing certain nonlinear patterns such as $\{\mathtt{f}\langle x, x\rangle \mid x \in X\}$. RTAs were developed for use in tree-automata-based model checking; we hope to use them as part of a static analysis system for a logic programming language. In developing that system, we noted that RTAs are not closed under Kleene-star or pre-concatenation with a regular language. We now introduce a strict super-class of RTA, called Isolating Rigid Tree Automata, which can accept rigid structures with arbitrarily many *isolated* rigid substructures, such as "lists of equal pairs," by allowing rigidity to be confined within subtrees. This class is Kleene-star and concatenation closed and retains many features of RTAs, including linear-time emptiness testing and NP-complete membership testing. However, it gives up closure under intersection.

1 Rigid Tree Automata

Rigid Tree Automata (RTAs) [2] extend regular bottom-up nondeterministic Tree Automata by imposing *global* constraints on accepting runs. They are well-suited to describe regular structures containing finitely many typed variables, such as $\{\mathtt{f}\langle \mathtt{g}\langle x\rangle, \mathtt{h}\langle x, y\rangle\rangle \mid x \in L, y \in L'\}$ where L, L' are *regular* tree languages representing types. They can also describe families of "all-equal lists" $\{[], [x], [x,x], [x,x,x], \ldots \mid x \in L\}$.[1] As these examples show, variables may be reused, each occurrence *co-varying* with the others. RTAs may also express *unions* of such nonlinear structures, including infinite unions via recursion, as in the case of all-equal lists.

An RTA is very much like a TA. Each has an underlying language signature $\mathcal{F}$; a set of states Q; a set of accepting states $Q_F \subseteq Q$; and a transition map Δ, which is a set of rules of the form $\mathtt{f}\langle q_1, \ldots, q_n\rangle \to q_0$ where $\forall_i q_i \in Q$ and $\mathtt{f}/n \in \mathcal{F}$. A **run** of an RTA A on a tree t is exactly like that of a TA: a map that annotates each node ν of t with a state from Q in a way that **respects** Δ. That is, if node ν has label $\mathtt{g}/m \in \mathcal{F}$ and its m children are annotated with $q_1, \ldots, q_m \in Q$, then ν may be annotated with q_0 if $(\mathtt{g}\langle q_1, \ldots, q_m\rangle \to q_0) \in \Delta$.

The novelty of the RTA class is that an RTA designates a set of **rigid** states, $Q_R \subseteq Q$, and runs are accepted more selectively. A tree is **accepted** by the RTA $A = \langle \mathcal{F}, Q, Q_F, Q_R, \Delta\rangle$ iff there exists a run in which the root position is annotated by $q \in Q_F$ (this is the TA acceptance criterion) *and*, for each $q \in Q_R$, all

[1] We adopt some standard shorthand: $[] = \mathtt{nil}\langle\rangle$ and $[a, b, \ldots] = \mathtt{cons}\langle a, \mathtt{cons}\langle b, \ldots\rangle\rangle$.

subtrees whose roots are annotated by q are *equal*.[2] For example, $\{\mathtt{h}\langle x,x\rangle \mid x \in L\}$ is recognized by an RTA $\langle \mathcal{F} \cup \{\mathtt{h}/2\}, Q \cup \{q^*\}, \{q^*\}, \{q_F\}, \Delta \cup \{\mathtt{h}\langle q_F, q_F\rangle \to q^*\}\rangle$ if $q^* \notin Q$ and L is recognized by a regular TA $A = \langle \mathcal{F}, Q, \{q_F\}, \Delta\rangle$ whose sole accepting state q_F is **non-reentrant** (i.e., only occurs on the right of rules in Δ).[3] The set of languages described by RTAs are a *strict superset* of those described by regular TAs [2, Theorem 5]: the RTA language above is not regular, but any regular TA is an RTA with $Q_R = \varnothing$.

2 Kleene Non-Closure of Rigid Tree Automata

RTA cannot, however, describe (finite) structures with arbitrary numbers of variables, as each variable corresponds to a state in Q_R. Let us look at two examples. We use the notations $\cdot_\square$, $L^{*,\square}$, and $L^{n,\square}$ as defined in [1, §2.2.1].

First, consider $P = \{[], [\mathtt{p}\langle x_1, x_1\rangle], [\mathtt{p}\langle x_1, x_1\rangle, \mathtt{p}\langle x_2, x_2\rangle], \cdots \mid x_i \in L_\mathrm{x}\}$, with L_x regular and $|L_\mathrm{x}| = \infty$.[4] The RTA pumping lemma [2, Lemma 1] says that no RTA can recognize P. (The essential obstacle is that P needs to enforce separate equalities on unboundedly many pairs, which cannot be done with only finitely many rigid states.) This implies that the RTA family is not closed under pre-concatenation with a regular language, since $P = L \cdot_\square M$ where $L = \{\mathtt{nil}\langle\rangle, \mathtt{cons}\langle\square, l\rangle \mid l \in L\}$ is regular (note the recursive definition, allowing trees with arbitrarily many $\square$ leaves) and $M = \{\mathtt{p}\langle x,x\rangle \mid x \in L_\mathrm{x}\}$ is rigid. RTAs *are* trivially closed under *post*-concatenation with a regular language: $L \cdot_\square M$ is an RTA language over $\mathcal{F}$ if L is rigid over $\mathcal{F} \cup \{\square\}$ and M is regular over $\mathcal{F}$, as the rigidity in L will not be able to test the structure induced by concatenation with M, making concatenation behave locally as if L were regular.[5]

Second, consider the set of lists $D = \{[], [x_1, x_1], [x_1, x_1, x_2, x_2], \cdots \mid x_i \in L_\mathrm{x}\}$ for some regular L_x with $|L_\mathrm{x}| = \infty$. Again, the RTA pumping lemma implies that D cannot be recognized by an RTA. This shows that RTAs are not closed under Kleene-star, since $D = E^{*,\square}$ for the RTA language $E = \{\mathtt{nil}\} \cup \{\mathtt{cons}\langle x, \mathtt{cons}\langle x, \square\rangle\rangle : x \in L_\mathrm{x}\}$, Note that $E^{k,\square}$ is an RTA language for any finite k and any regular (or even rigid) language L_x.

3 Isolation

We augment RTA transition rules with the ability to discard rigidity constraints across subtrees, introducing **Isolating Rigid Tree Automata** (IRTA), a proper

[2] The states Q_R are thus "rigid" as each expands in one way throughout the tree.

[3] These requirements on accepting states of A are needed for our RTA construction, in which q_F becomes a rigid state. However, they involve no loss of generality, since if L is recognized by any regular TA $A' = \langle \mathcal{F}, Q, Q_F, \Delta\rangle$, it is also recognized by an equivalent one that uses a single, non-reentrant accepting state, as required: $A = \langle \mathcal{F}, Q \cup \{q_F\}, \{q_F\}, \Delta \cup \{\mathtt{f}\langle q_1, \ldots, q_k\rangle \to q_F \mid (\mathtt{f}\langle q_1, \ldots, q_k\rangle \to q) \in \Delta, q \in Q_F\}\rangle$.

[4] For concreteness and to avoid any ability of the lemma to find pumping opportunities in L_x, restrict to runs over "short" trees from L_x for this and the next example.

[5] One could define a notion of concatenation that was more specialized to RTAs, where $\square$ itself was interpreted rigidly. On this definition, RTAs would be closed under both pre- and post-concatenation with regular languages.

super-class of RTA.[6] Each transition rule is decorated with a set of rigid states to isolate, making it of the form $\mathtt{f}\langle q_1,\ldots,q_n\rangle \xrightarrow{!I} q_0$ with $\mathtt{f}/n \in \mathcal{F}$, $\forall_i . q_i \in Q$, and $I \subseteq Q_R$.[7] Intuitively, when such a rule is used in a run to reach a node ν, the equality constraint for a rigid state $q \in I$ is no longer enforced between q-annotated nodes strictly dominated by ν and q-annotated nodes elsewhere. Every RTA is an IRTA with $I = \varnothing$ everywhere.

The non-RTA examples from before are easily captured (see Figure 1 in the appendix for illustrations). As before, suppose that L_x is recognized by the TA $A = \langle \mathcal{F}, Q, \{q_F\}, \Delta\rangle$ with non-reentrant accepting state q_F. Then taking $\mathcal{F}' = \mathcal{F} \cup \{\mathtt{p}/2, \mathtt{cons}/2, \mathtt{nil}/0\}$,

- The language P is recognized by the IRTA $\langle \mathcal{F}', Q \cup \{q^*\}, \{q^*\}, \{q_F\}, \Delta'\rangle$ with $\Delta' = \Delta \cup \{\mathtt{p}\langle q_F, q_F\rangle \xrightarrow{!\{q_F\}} q_p, \mathtt{cons}\langle q_p, q^*\rangle \to q^*, \mathtt{nil}\langle\rangle \to q^*\}$.
- The language D is recognized by the IRTA $\langle \mathcal{F}', Q \cup \{q_1^*, q_2^*\}, \{q_1^*\}, \{q_F\}, \Delta'\rangle$ with $\Delta' = \Delta \cup \{\mathtt{cons}\langle q_F, q_1^*\rangle \to q_2^*, \mathtt{cons}\langle q_F, q_2^*\rangle \xrightarrow{!\{q_F\}} q_1^*, \mathtt{nil}\langle\rangle \to q_1^*\}$.

The use of $\varnothing \subsetneq I \subsetneq Q_R$ allows for hybrid structures with both global and local equalities, such as $D' = \{[], [x_0, x_1, x_1], [x_0, x_1, x_1, x_0, x_2, x_2], \cdots \mid x_i \in L_\mathrm{x}\}$. Here the equality of every third entry (x_0) would be enforced throughout the entire list using a rigid state that is not isolated (à la RTA), while the other entries are only equal in adjacent pairs, using a rigid state that is periodically isolated as in D.

To describe the semantics of IRTA rules more formally, we first restate the acceptance condition for TAs and RTAs as a bottom-up algorithm for generating accepting runs, if any, on an input tree. A simple change then will suffice to make this algorithm construct IRTA runs.

Membership testing for a *deterministic* TA can be accomplished by bottom-up annotation of the given tree t. A step of this algorithm visits any unannotated node of t whose children have already been annotated, and annotates it with the *only* state that respects Δ (given the child annotations), or rejects t if there is no such state. t is accepted if the root is annotated by a final state. In the *nondeterministic* case, each node of t is simultaneously annotated with *all* states that can respect Δ (given some choice of the child annotations), and t is accepted if its root node is annotated with at least one final state.

We can extend this approach to RTAs by augmenting the annotations. Let t_ν denote the subtree of t rooted at node ν. Each annotation of ν, rather than being a state in Q, is now a pair $(q, r) \in Q \times \wp(Q_R \times \mathcal{T}(\mathcal{F}))$. Intuitively, this pair records the existence of some run on t_ν that annotates ν with q, where $r : Q_R \rightharpoonup \{\text{subtrees of } t_\nu\}$ is a partial function (represented as a set of ordered

[6] In this work, we consider the family of nondeterministic (I)RTAs. Of course there is also a class of deterministic IRTAs that generalize deterministic RTAs.

[7] We choose the isolating set I as part of the transition rule. In the case of *deterministic* IRTAs, however, it might increase power to change the form of the rules to defer the choice of I until the next rule is selected. The next rule would then have the form $\mathtt{g}\langle \ldots, q_0!I, \ldots\rangle \to q_{-1}$, allowing the choice of I at the q_0-annotated node ν depend on the annotations at ν's siblings, and on the functor $\mathtt{g}$ and annotation q_{-1} at ν's parent.

pairs) that maps each rigid state q' used in the run to the tree t' such that q' was used in the run only to annotate the roots of copies of t'. When visiting a node ν with label g/m, if $(\mathsf{g}\langle q_1, \ldots, q_m\rangle \to q) \in \Delta$ and the m children are annotated with $(q_1, r_1), \ldots, (q_m, r_m)$, the algorithm annotates this node with (q, r), provided that $r = \bigcup_{i=0}^{m} r_i$ is a partial function, where $r_0 = \{(q, t_\nu)\}$ if $q \in Q_R$ and otherwise $r_0 = \varnothing$. The full tree t is accepted if its root has a label (q, r) for some $q \in Q_F$.

The generalization to IRTAs is now straightforward: the algorithm simply "forgets" subtrees when directed to do so by the transition rules. When visiting a node ν with label g/m, if $(\mathsf{g}\langle q_1, \ldots, q_m\rangle \xrightarrow{!I} q) \in \Delta$ and the m children are annotated with $(q_1, r_1), \ldots, (q_m, r_m)$, the algorithm computes $r' = r_0 \cup \{(q', t') \in r \mid q' \notin I\}$, where $r = \bigcup_{i=1}^{m} r_i$ and r_0 is as before, and annotates this node with (q, r'), provided that r' is a partial function.

4 Pumping Lemma

The pumping lemma construction for RTAs given in [2, §2.4] relies heavily on the fact that any path from a the root of an accepted run to a leaf thereof will contain each rigid state at most once. Thus if there is an accepting run with a path of length $|Q_R|(1 + |Q|)$, there must exist a nontrivial sub-path with all nodes there-on labeled with states from $Q \smallsetminus Q_R$ (i.e., not rigidly) and with both endpoints equally labeled. This is no longer true in IRTA: a root-leaf path in an accepted run can contain a rigid state at most once *between isolations* of that state, but isolations may occur arbitrarily often.

Nevertheless, a pumping-style construction is still possible (see Figure 2 for an illustration). Given an accepted tree t of height $|Q| \cdot 2^{|Q_R|} + 1$, a root-leaf path of that length is guaranteed to have two distinct nodes analyzed with the same (possibly rigid) state and with the same *set of* rigid states having not been isolated. Let two such colliding nodes be δ and α, respectively labeled as (q, r) and (r, r') with r and r' having equal domains. We can then partition the tree into three regions by writing it as $B[D[A]]$, where B ("before") and D ("during") are 1-contexts, with D rooted at δ, and $A = t_\alpha$ ("after") is a tree rooted at α. We can construct a new 1-context D' from D by "rewriting": use the values from r', rather than r, to satisfy rigid states in D, traversing bottom up and manipulating r' as directed by the automaton's rules. The result will be a revised label of (q, r'') for the root of D'; use the same rewrite procedure to turn B, which used rigid trees from r', into B' using r''. Now $B'[D'[D[A]]]$ is another accepted tree satisfying the pumping preconditions. One could, alternatively, rewrite B to B'' using r to obtain $B''[A]$, another accepted tree.

This pumping construction merely builds other trees; it does not *repeat* parts of the tree structure exactly. Still, it shows that *if* an IRTA accepts a sufficiently tall tree, it accepts infinitely many trees. It also shows an argument (different from that of § 5.1 below) that emptiness of an IRTA's language is decidable: one could exhaustively enumerate and test trees of height up to $|Q| \cdot 2^{|Q_R|}$ only, since the shortest accepted tree cannot be taller than that—any such tree could be pumped down using the $B''[A]$ construction.

5 Decision Problems

5.1 Emptiness

RTAs may be tested for non-emptiness using a state-marking algorithm [2, §6.1]. The RTA algorithm constructs *acyclic* runs, demonstrating occupancy of the RTA's states by visiting them in a "depth-first" order. If a state is non-empty, then this algorithm will construct a witness tree for it of height at most n, where n is the number of states in the RTA. The RTA is non-empty iff at least one of its final states is non-empty.

To find a witness of an IRTA's non-emptiness, it suffices to find a witness for the corresponding RTA (which drops the $!I$ decoration, and thus enforces even more equality than the IRTA requires). This works because if the IRTA has any witness t, then it has a witness t' that would be accepted by the RTA, which can be found by rewriting subtrees to be equal much as in section 4.

5.2 Membership Testing

As with RTAs [2, §6.2], membership testing of a tree t (with n nodes) against an IRTA $\langle \mathcal{F}, Q, Q_F, Q_R, \Delta \rangle$ is NP-complete. The proof for RTA reduces 3-SAT to membership testing. We need only show that an annotation of t's nodes can be checked in polynomial time to determine whether it constitutes a valid run (section 3). This involves checking each node of t separately to ensure that its annotation (q, r) can be derived from the annotations of its children by one of the rules in Δ. Given such a rule, checking the r annotation (which dominates the runtime) involves comparing at most $a|Q_R|$ pairs of subtrees of t, each having at most n nodes, where a is an upper bound on the number of children (the largest arity of any symbol in $\mathcal{F}$). Thus, the total runtime is $O(an^2|Q_R||\Delta|)$.[8]

5.3 Universality

As all RTAs are IRTAs, tests for universality ($\mathcal{L}(A) = \mathcal{TF}$?), equality ($\mathcal{L}(A) = \mathcal{L}(A')$?), and inclusion ($\mathcal{L}(A) \subseteq \mathcal{L}(A')$?) all remain non-computable for our new class: the proof from [2, §6.4] continues to hold. For practical purposes, we envision the possibility of a *3-way* inclusion test that spends limited computational power to prove or disprove inclusion, but sometimes fails to do either.

6 Closure Properties

Pre-concatenation with a Regular Language IRTAs are, by design, trivially closed under this operation. When constructing an IRTA for $L \cdot_{\square} M$ from an IRTA for M, where L is regular over $\mathcal{F} \cup \{\square\}$, isolate all rigid states in M on any transition to the sole L state that labels $\square$.

Kleene Closure Similarly, when constructing an IRTA for $L^{*,\square}$ from an IRTA for L over $\mathcal{F} \cup \{\square\}$, isolate all rigid states of L on transitions to the $\square$ state.

Projection Closure If L_{x} is an IRTA language, then the set of trees that appear at a given address α (e.g., 1st child of 2nd child of root) within trees of L_{x} is also an IRTA language. After eliminating unreachable rules (rules that contain empty IRTA states as determined by § 5.1) to obtain a "trimmed" IRTA

[8] Hash consing can eliminate a factor of n by allowing $O(1)$-time subtree comparison.

$\langle \mathcal{F}, Q, Q_F, Q_R, \Delta \rangle$, a simple recursive algorithm can nondeterministically follow transitions of Δ backwards from Q_F to find the collection Q_q of states that can appear at address α. The desired IRTA is then $\langle \mathcal{F}, Q, \bigcup_{q \in Q_F} Q_q, Q_R, \Delta \rangle$.

Complementation Non-closure We conjecture that IRTAs are, like RTAs, not closed under complementation. The existing demonstration from [2, Example 7 and §4.2] is, however, no longer sufficient: the set B of balanced binary trees over $\mathcal{F} = \{\mathtt{a}/0, \mathtt{f}/2\}$ *is* an IRTA language. Let $Q = \{q_0, q_1\}$; then B is recognized by $\langle \mathcal{F}, Q, Q, Q, \{\mathtt{a}\langle\rangle \to q_0, \mathtt{f}\langle q_0, q_0 \rangle \xrightarrow{!\{q_0\}} q_1, \mathtt{f}\langle q_1, q_1 \rangle \xrightarrow{!\{q_1\}} q_0\}\rangle$. Unfortunately, finding a replacement has proven tricky!

Intersection Non-closure It is possible to construct a series of IRTA machines whose intersection would give the language of accepting runs of a two-counter machine, as in [1, Thm. 4.4.7]. Therefore, as IRTA has a decidable emptiness test, it must not be intersection-closed. Despite that, we conjecture that some special cases of intersection may still be possible; in particular, we speculate that intersecting an IRTA language with either a regular language or an RTA language will tractably yield an IRTA language.

Union Closure IRTAs are trivially closed under union, by nondeterminism.

7 Comparison to TAC+ / TA$_=$

The IRTA class is neither more general nor more specific than tree automata with local equality constraints (TAC+ or TA$_=$, [3]). The non-inclusion of IRTA in TAC+ follows from the non-inclusion of RTA. RTA's ability to enforce constraints globally rather than solely at fixed relative positions allow it to recognize, e.g., the class of trees t in which every two subterms $\mathtt{g}\langle t_1 \rangle$ and $\mathtt{g}\langle t_2 \rangle$ satisfy $t_1 = t_2$, even if they are arbitrarily far apart in the tree [2, Example 3]. To show conversely that TAC+ is not included in IRTA, consider the language $L = \{[0], [1, 0], \ldots, [n, n-1, \ldots, 1, 0], \ldots\}$ (with integers represented as their Peano encodings). L is recognized by the TAC+ $\langle \{\mathtt{z}/0, \mathtt{s}/1, \mathtt{nil}/0, \mathtt{cons}/2\}, \{q_z, q_s, q_n, q_c\}, \{q_c\}, \Delta \rangle$, where $\Delta = \{\mathtt{cons}\langle q_s, q_c \rangle \xrightarrow{11=21} q_c, \mathtt{z}\langle\rangle \to q_z, \mathtt{s}\langle q_z \rangle \to q_s, \mathtt{s}\langle q_s \rangle \to q_s, \mathtt{nil}\langle\rangle \to q_n, \mathtt{cons}\langle q_z, q_n \rangle \to q_l\}$. The first rule in Δ is the centerpiece. L is not an IRTA language: suppose that L is recognized by an IRTA A with k states, and consider an accepting run of A on $t = [k, \ldots, 1, 0]$. Let ν be a minimum-height Peano node of t such that its state annotation q_ν is reused for some ν' in t with $t_\nu \neq t_{\nu'}$. ν exists by pigeonhole. By minimality, each proper descendant of ν uses a state that annotates equal trees throughout the run on t. Substituting t_ν in for all q_ν-annotated nodes yields another accepting run on a new tree t'. However, $t' \notin L$: either t' is not a list, or t' has the same length as t but different elements.

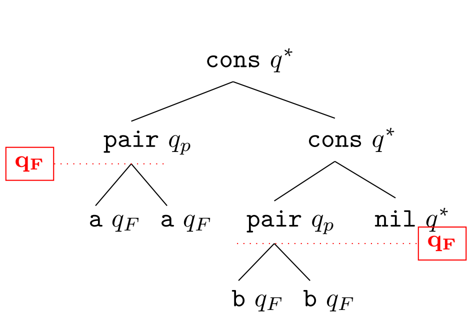
8 Conclusion

We have introduced a new class of automata, Isolating Rigid Tree Automata, which are a Kleene-closed super-class of Rigid Tree Automata. We hope, despite the loss of intersection closure, that IRTA will be useful for modeling inductive (i.e., recursive) data types for programming languages where a data constructor may make non-linear use of its (finitely many) arguments (e.g., Prolog).

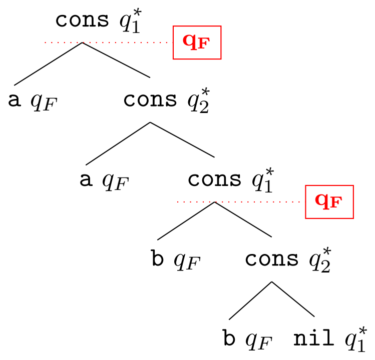
References

1. Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Loding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications.*
2. Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata and applications. *Information and Computation*, 209(3):486–512.
3. Jocelyne Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, 1981.

A Additional Figures



(a) An example tree from P.



(b) An example tree from D.

Fig. 1: Runs of IRTAs, as given in § 3, for languages defined in § 2. Horizontal dotted lines indicate isolation: any two nodes labeled by the same rigid state must dominate equal trees, unless separated by a line labeled by that state.

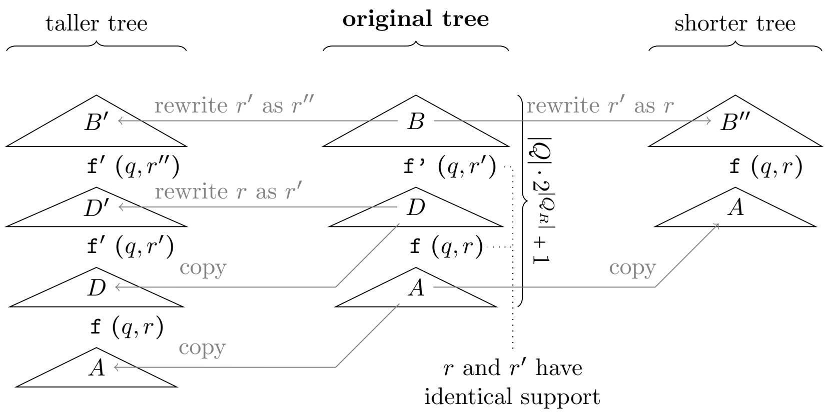


Fig. 2: Graphic depiction of the IRTA pumping construction of § 4, showing how to derive both a shorter and taller tree from a tree of height $|Q| \cdot 2^{|Q_R|} + 1$.

Direct Evaluation of Selecting Tree Automata on XML Documents Compressed with Top Trees

Kenji Hashimoto, Suguru Nishimura, and Hiroyuki Seki

Nagoya University, Japan
{k-hasimt,seki}@is.nagoya-u.ac.jp, nishimura@apal.i.is.nagoya-u.ac.jp

Abstract. Tree compression methods that utilize the structural information have the advantage that a compressed document can be directly scanned without decompression. We propose a method for manipulating an XML document compressed with a top tree without uncompressing the document. Deterministic selecting top-down tree automata are used for specifying positions in a tree. Experimental results show the avoidance of duplicated computation for common substructures can greatly reduce the computation time.

1 Introduction

An XML document is easily handled and has high versatility, but the size of a real-world XML document is apt to become large and various compression methods for XML have been proposed. Among them, tree compression methods have advantage that a compressed document preserves the structure of the original document and we can retrieve or update the document without decompressing it in principle. A direct evaluation method for structured documents compressed with tree grammars has shown good performance [5], which naturally raises a question whether a direct evaluation is also effective for other tree compression methods. Among others, a compression method with top trees [4] has been paid attention because of advantages in processing some type of navigational queries, while it has been reported in [7] that the compression ratio achieved by TreeRepair [8] is better than that by the method with top trees.

This paper proposes a method of directly querying a tree compressed with a top tree. A query is given by a deterministic top-down selecting tree automaton (DSTA) that computes a set of nodes of an input tree via state transitions. The proposed method applies a given DSTA directly to a tree compressed with a top tree. When the same states of the automaton are detected to be assigned to a subtree, the algorithm avoids the duplicated computation by "memoization."

2 Preliminaries

2.1 Tree

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and Σ be an alphabet. The set $T(\Sigma)$ of (unranked) trees over Σ is the smallest set that satisfies $f(t_1, \ldots, t_n) \in T(\Sigma)$ whenever $f \in \Sigma$ and $t_i \in T(\Sigma)$ for $1 \leq i \leq n$. The set $Pos(t)$ of positions of $t = f(t_1, \ldots, t_n)$ is defined by $Pos(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{ip_i \mid p_i \in Pos(t_i)\}$. For $u, v \in Pos(t)$, if $u = vw$ for some $w \in \mathbb{N}^*$, we write $v \preceq u$. If $v \preceq u$ and $v \neq u$, we write

$v \prec u$. If $u = vi$ for $i \in \mathbb{N}$, (v, u) is an edge of t. The root of t is ε. If there is no $u \in Pos(t)$ such that $v \prec u$, v is a leaf of t. $\mathrm{Leaf}(t)$ is the set of all leaves of t. For $p \in Pos(t)$, the subtree of $t = f(t_1, \ldots, t_n)$ at p, denoted as $t|_p$, is defined by $t|_\varepsilon = t$ and $t|_{ip'} = t_i|_{p'}$ $(1 \le i \le n)$ and the label of p of t, $\lambda(t, p)$, is defined by $\lambda(t, \varepsilon) = f$ and $\lambda(t, ip') = \lambda(t_i, p')$ $(1 \le i \le n)$. For trees $t, t' \in T(\Sigma)$ and $p \in Pos(t)$, let $t[t']_p$ denote the tree obtained from t by replacing $t|_p$ with t'. For a tree $t = f(t_1, \ldots, t_n)$, let $|t| = 1 + \sum_{i=1}^{n} |t_i|$ and $\mathrm{height}(t) = 1 + \max(\{0\} \cup \{\mathrm{height}(t_i) \mid 1 \le i \le n\})$.

A deterministic selecting top-down path automaton (abbreviated as DSTA) over Σ is a triple $\mathcal{A} = (Q, q_0, \Delta)$ where (1) Q is a finite set of states, (2) $q_0 \in Q$ is the initial state, and (3) Δ is a set of transition rules that have the shape of $(q, f)Rq_c$ where $q, q_c \in Q$, $f \in \{\%\} \cup \Sigma$, $R \in \{\rightarrow, \Rightarrow\}$ such that for every $q \in Q$, there is a rule whose left-hand side is $(q, \%)$ in Δ, and for every pair of $q \in Q$ and $f \in \{\%\} \cup \Sigma$, there is at most one rule whose left-hand side is (q, f). A rule in Δ whose left-hand side is $(q, \%)$ for $q \in Q$ is called a default rule and a rule $(q, f) \Rightarrow q_c$ is called a selecting rule.

Let $\mathcal{A} = (Q, q_0, \Delta)$ be a DSTA. For a tree $t \in T(\Sigma)$, a tree $r \in T(Q)$ is a run of t by $\mathcal{A}$ if (1) $Pos(r) = Pos(t)$, (2) if p is not a leaf, $\lambda(t, p) = f$, $\lambda(r, p) = q$, $\lambda(r, pi) = q_i$ and $\lambda(r, pj) = q_j$ for any $i, j \in \mathbb{N}$, then $q_i = q_j$, and for some $R \in \{\rightarrow, \Rightarrow\}$, either (a) $(f, q)Rq_i \in \Delta$ or (b) $(\%, q)Rq_i$ (and $(f, q)Rq' \notin \Delta$ for any q'), which is called the rule applied at p, and (3) $\lambda(r, \varepsilon) = q_0$. For a tree t, a run of t by $\mathcal{A}$ is unique if exists. A position p of t is selected by $\mathcal{A}$ if there is a run r of t by $\mathcal{A}$ such that a selecting rule is applied at p. DSTAs express the fragment of XPath with only child, descendant, test on labels. Let $\mathcal{A}_{pre}(t)$ be the set of pre-order numbers of positions of t selected by $\mathcal{A}$, where the pre-order number n of p means that p is the nth position when traversing t in pre-order.

Example 1. Consider an XPath expression `//f/a`. The DSTA $\mathcal{A} = (Q, q_0, \Delta)$ selects the set of nodes specified by this XPath expression where $\Delta = \{(q_0, f) \rightarrow q_1, (q_0, \%) \rightarrow q_0, (q_1, f) \rightarrow q_1, (q_1, a) \Rightarrow q_0, (q_1, \%) \rightarrow q_0\}$.

2.2 Top Tree and Top DAG

Definition 2. *Let $t \in T(\Sigma)$ and $v, u \in Pos(t)$ such that $v \prec u$ with $u = viw$ for some $i \in \mathbb{N}$ and $w \in \mathbb{N}^*$. Also let $t|_v = f(t_1, \ldots, t_n)$, $t|_u = g(t'_1, \ldots, t'_m)$, and assume $1 \le r \le i \le s \le n$. The cluster of t determined by v, r, s, u is defined as $\mathcal{C}_t(v, r, s, u) = f(t_r, \ldots, t_s)[g]_z$ where $z = (i - r + 1)w$. Especially when $u \in \mathrm{Leaf}(t)$, the cluster is denoted as $\mathcal{C}_t(v, r, s) = f(t_r, \ldots, t_s)$. Also, v and u (if $u \notin \mathrm{Leaf}(t)$) are called boundaries of the cluster; v is the top boundary and u is the bottom boundary.* □

Definition 3. *Let $C_1 = \mathcal{C}_t(v_1, r_1, s_1, u_1)$ and $C_2 = \mathcal{C}_t(v_2, r_2, s_2, u_2)$ be clusters of a tree t. If C_1 and C_2 share one of their boundaries and they are either horizontally or vertically adjacent in one of the following five ways, define the merge C of C_1 and C_2 as follows:*

(1) Case $u_1 = v_2$, $r_2 = 1$, and $s_2 = \max\{i \mid v_2 i \in Pos(t)\}$:
(A) if $u_2 \notin \mathrm{Leaf}(t)$, $C = \mathcal{C}_t(v_1, r_1, s_1, u_2)$;

(B) if $u_2 \in \mathrm{Leaf}(t)$, $C = \mathcal{C}_t(v_1, r_1, s_1)$.

(2) Case $v_1 = v_2$ *and* $r_2 = s_1 + 1$*:*

(C) if $u_1 \notin \mathrm{Leaf}(t)$ *and* $u_2 \in \mathrm{Leaf}(t)$, $C = \mathcal{C}_t(v_1, r_1, s_2, u_1)$*;*

(D) if $u_1 \in \mathrm{Leaf}(t)$ *and* $u_2 \notin \mathrm{Leaf}(t)$, $C = \mathcal{C}_t(v_1, r_1, s_2, u_2)$*;*

(E) if $u_1, u_2 \in \mathrm{Leaf}(t)$, $C = \mathcal{C}_t(v_1, r_1, s_2)$.

(A), . . . , (E) *are called merge types, or simply, types. A merge by* (A) *or* (B) *is called a vertical merge and a merge by* (C), (D) *or* (E) *is called a horizontal merge (see Figure 1).* □

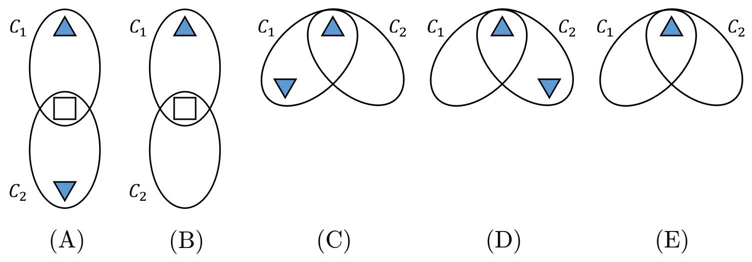


Fig. 1. Merge types of clusters

Definition 4. *A top tree* τ *of a tree* t *is a (binary) tree that satisfies all of the following conditions:*

(1) Every $\pi \in Pos(\tau)$ *represents a cluster of* $\perp(t)$ *, denoted by* $Cl(\pi)$.

(2) If $\pi \in \mathrm{Leaf}(\tau)$, π *represents an edge of* t*, i.e.,* $\mathcal{C}_{\perp(t)}(v, i, i, u)$ *(*$i \in \mathbb{N}$ *and* $u = vi$*) and* $\lambda(\tau, \pi) = \lambda(t, u)$.

(3) If $\pi \notin \mathrm{Leaf}(\tau)$, π *represents the merge of* $Cl(\pi 1)$ *and* $Cl(\pi 2)$ *and* $\lambda(\tau, \pi)$ *is the merge type of* C. □

A top tree of a given tree t is not always uniquely determined because there may be more than one way of constructing the cluster corresponding to t. For a tree t, a top DAG (directed acyclic graph) τ_D is the minimal DAG representation of a top tree τ of t where a node of τ_D may represent more than one position of τ. We say that τ_D is a compression of t and t is the decompression of τ_D. We abuse notions of a top tree, e.g., for a node ν of τ_D, we will write $Cl(\nu)$ to mean $Cl(\pi)$ where π is any position of the top tree represented by ν. The size $|\tau_D|$ of τ_D is the number of edges of τ_D. The compression ratio $CR(t, \tau_D)$ of τ_D against t is defined as $CR(t, \tau_D) = |\tau_D|/|t|$.

In the above definitions, we do not keep the label of the top boundary of each cluster, following the idea in [4, 7]. This increases the likelihood of identical subtrees in the top tree, and thus we can get a smaller top DAG. We wrap a given tree t by a dummy symbol $\perp$ so that the leftmost leaf of the top tree τ keeps the label of the root of t.

3 Direct Processing of Selecting Tree Automaton

In this section, we propose algorithms for two tasks, counting and materialization [10], given a top DAG τ_D of a tree t and a DSTA $\mathcal{A}$. To begin with, we present an underlying procedure for computing a run of $\mathcal{A}$. Algorithm 2 shows eval(ν, q) that computes, for a node ν of τ_D and $q \in Q$, the state in Q assigned to each position of $Cl(\nu)$ and returns the state assigned to all the children (in t) of the bottom boundary of $Cl(\nu)$ (if exists) assuming that q is assigned to each child of the top boundary of $Cl(\nu)$. Note that for the root node δ of τ_D, eval(δ, q_0) in effect constructs the run of t by $\mathcal{A}$ (if exists). More precisely, when a leaf node ν with $\lambda(\tau_D, \nu) = f$ is visited with a state $q \in Q$ and $(q, f)Rq_c \in \Delta$, the state q_c is returned, and if $R = \Rightarrow$ in addition, the position 1 in $Cl(\nu)$ is selected. This is because ν represents an edge (v, u) of t and u (its position in $Cl(\nu)$ is 1) is selected by $\mathcal{A}$.

Algorithm 2 eval(ν, q)

```
input: node $\nu$ (of top DAG $\tau_D$), state $q \in Q$
output: state $q'$
 1: if $\nu$ is a leaf node then
 2:     $f := \lambda(\tau_D, \nu)$
 3:     if $(q, f)\ R\ q_c \in \Delta$ then
 4:         $q' := q_c$
 5:     else
 6:         $q' := q_c$ where $(q, \%)\ R\ q_c \in \Delta$
 7:     end if
 8: else
 9:     Let $\nu_1$ and $\nu_2$ be the left and right children of $\nu$, respectively.
10:     switch the type of $Cl(\nu)$ do
11:         case (A) or(B):
12:             $q' := \text{eval}(\nu_2, \text{eval}(\nu_1, q))$
13:         case (C):
14:             $q' := \text{eval}(\nu_1, q);\ \text{eval}(\nu_2, q)$
15:         case (D):
16:             $\text{eval}(\nu_1, q);\ q' := \text{eval}(\nu_2, q)$
17:         case (E):
18:             $\text{eval}(\nu_1, q);\ \text{eval}(\nu_2, q)$
19:     end switch
20: end if
```

Since $\mathcal{A}$ is deterministic, the result of eval(ν, q) is unique when ν and q are given. To avoid a duplicated computation, a table (hash map) is used for memoization. When the algorithm needs to compute eval(ν, q), it first retrieves the hash map with (ν, q) as a key, and if no information has been recorded, eval(ν, q) is executed and the result is stored in the hash map with key (ν, q).

Counting. We consider the task of counting positions of t selected by $\mathcal{A}$ recursively along with the structure of τ_D. We can compute the number $|\mathcal{A}(t)|$ in $O(|\mathcal{A}||\tau_D|)$ time. Our algorithm for counting just computes the number $s(\nu)$ of selected positions in $Cl(\nu)$ for each node ν. The number $s(\nu)$ can be recursively computed in parallel with the computation of eval(ν, q) by adding $s(\nu)$ to the return value of eval(ν, q).

(1) ν is a leaf node of τ_D and $Cl(\nu)$ is an edge (v, u) of t: $s(\nu) = 1$ if a selecting rule in Δ is applied to u, $s(\nu) = 0$ otherwise.
(2) ν is an internal node of τ_D: Let ν_1 and ν_2 be the left and right children of ν, respectively. $s(\nu) = s(\nu_1) + s(\nu_2)$.

Materialization. Here we focus on computing $\mathcal{A}_{pre}(t)$ from a given top DAG τ_D of a tree t. We can construct an ordered list of $\mathcal{A}_{pre}(t)$ in $O(|\mathcal{A}||\tau_D| + |\mathcal{A}(t)|)$ time. Our algorithm for materialization computes recursively the following values and sets for each node ν, in parallel with the computation of eval(ν, q): $n(\nu)$ is the number of positions of $Cl(\nu)$ except its top boundary; $b(\nu)$ is the pre-order number of the bottom boundary of $Cl(\nu)$ if exists; $P_l(\nu)$ (resp. $P_r(\nu)$) is the set of pre-order numbers of the selected positions in $Cl(\nu)$ smaller than or equal to (resp. greater than) that of its bottom boundary. $n(\nu)$, $b(\nu)$, $P_l(\nu)$, and $P_r(\nu)$ can be recursively defined as follows.

(1) ν is a leaf node of τ_D and $Cl(\nu)$ is an edge (v, u) of t: $n(\nu) = 1$, $b(\nu) = 1$, $P_r(\nu) = \emptyset$, and $P_l(\nu) = \{1\}$ if a selecting rule in Δ is applied to u, $P_l(\nu) = \emptyset$ otherwise.
(2) ν is an internal node of τ_D: Let ν_1 and ν_2 be the left and right children of ν, respectively. Let $n_i = n(\nu_i)$, $b_i = b(\nu_i)$, and $P_{li} = P_l(\nu_i)$ and $P_{ri} = P_r(\nu_i)$ for $i = 1, 2$. Let us denote $P + n = \{n' + n \mid n' \in P\}$. $n(\nu) = n_1 + n_2$.

$$b(\nu) = \begin{cases} b_1 + b_2 & \text{if } Cl(\nu) \text{ is of type (A)},\\ b_1 & \text{if } Cl(\nu) \text{ is of type (C)},\\ n_1 + b_2 & \text{if } Cl(\nu) \text{ is of type (D)},\\ \emptyset & \text{if } Cl(\nu) \text{ is of type (B) or (E)}. \end{cases}$$

$$P_l(\nu) = \begin{cases} P_{l1} \cup (P_{l2} + b_1) & \text{if } Cl(\nu) \text{ is of type (A)},\\ P_{l1} \cup (P_{r1} + n_2) \cup (P_{l2} + b_1) & \text{if } Cl(\nu) \text{ is of type (B)},\\ P_{l1} & \text{if } Cl(\nu) \text{ is of type (C)},\\ P_{l1} \cup (P_{l2} + n_1) & \text{if } Cl(\nu) \text{ is of type (D) or (E)}. \end{cases}$$

$$P_r(\nu) = \begin{cases} (P_{r1} + n_2) \cup (P_{r2} + b_1) & \text{if } Cl(\nu) \text{ is of type (A)},\\ P_{l2} \cup (P_{r1} + n_1) & \text{if } Cl(\nu) \text{ is of type (C)},\\ P_{r2} + n_1 & \text{if } Cl(\nu) \text{ is of type (D)},\\ \emptyset & \text{if } Cl(\nu) \text{ is of type (B) or (E)}. \end{cases}$$

4 Experiments

We implemented a prototype tool for direct evaluation of DSTAs on XML documents compressed with top DAGs. We used the XML documents from [1, 2, 3, 6, 11] listed in Table 1. Table 2 shows (the XPath expression equivalent with) a DSTA we performed for a top DAG of each document. We implemented a tool for compressing XML documents to top DAGs. As the same as [7], our tool applies the idea of RePair to the horizontal merge step of top tree compression. The top DAGs we used in the experiment were generated by the tool. Experiments were performed in the following environment: Intel Xeon CPU E5-2407 v2 2.40GHz, 32GB RAM, FreeBSD 10.1.

Table 3 shows experimental results of our prototype evaluator for the two tasks, counting and materialization, with/without memoization. We observe that meemoization was much effective for both tasks, in particular when the compression ratio of a top DAG is high, e.g., XMark.

Table 1. XML documents and the compression ration of their top DAGs

File name	size(KB)	#edges	height	compression ratio
Mondial	408	22,422	5	0.2537
Nasa	8,267	476,645	8	0.1042
Treebank	25,508	2,437,665	36	0.4924
DBLP	1,770,844	42,699,876	6	0.0616
jawiki-article	491,552	32,615,072	5	0.0350
OpenStreetMap-spain	968,367	159,126,336	3	0.0385
XMark	5,738,500	87,538,032	12	0.0124

Table 2. Queries

	File name	XPath expression	#rules of DSTA
(1)	Mondial	//*	1
(2)	Nasa	/*/*/*/*/*/*/*/*	9
(3)	Treebank	//NP//VP//PP//PP//IN	11
(4)	DBLP	/dblp/mastersthesis/school	7
(5)	jawiki-article	/mediawiki/page/revision/parentid	9
(6)	OpenStreetMap-spain	/osm/way/nd	7
(7)	XMark	//closed_auction//keyword//*	5

Table 3. Results on direct query evaluation

	$\|\mathcal{A}_{pre}(t)\|$	counting (ms)		materialization (ms)	
		w/o memo	memo	w/o memo	memo
(1)	22,423	13	3	49	14
(2)	25,060	65	6	115	20
(3)	6,388	386	232	488	310
(4)	9	5,112	349	6,204	535
(5)	1,481,076	3,426	154	7,113	665
(6)	70,241,127	13,666	905	95,629	10,914
(7)	757,335	10,809	147	14,830	353

5 Conclusion

In this paper, we have presented a method of running a DSTA on a tree represented with a top DAG. Future work includes evaluation of queries with filtering by using bottom-up automata, and updating directly a document represented with a top DAG. Moreover, we need to compare our evaluator with other evaluators working over compressed documents such as TinyT [9].

References

[1] dblp – computer science bibliography, `http://dblp.uni-trier.de/`
[2] jawiki, `https://dumps.wikimedia.org/jawiki/20160501/`
[3] XMLCompBench. http://xmlcompbench.sourceforge.net/Dataset.html

[4] Bille, P., Gørtz, I.L., Landau, G.M., Weimann, O.: Tree compression with top trees. Information and Computation 243(C), 166–177 (2015)

[5] Böttcher, S., Hartel, R., Jacobs, T.: Fast multi-update operations on compressed XML data. In: Proceedings of the 29th British National Conference on Databases (BNCOD 2013), LNCS 7968. pp. 149–164 (2013)

[6] Geofabrik: OpenStreetMap, `http://download.geofabrik.de/`

[7] Hübschle-Schneider, L., Raman, R.: Tree compression with top trees revisited. In: Proceedings of the 14th International Symposium on Experimental Algorithms (SEA 2015), LNCS 9125. pp. 15–27 (2015)

[8] Lohrey, M., Maneth, S., Mennicke, R.: XML tree structure compression using RePair. Information Systems 38(8), 1150–1167 (2013)

[9] Maneth, S., Sebastian, T.: Fast and tiny structural self-indexes for XML. CoRR abs/1012.5696 (2010), `http://arxiv.org/abs/1012.5696`

[10] Maneth, S., Sebastian, T.: XPath node selection over grammar-compressed trees. In: Proceedings of the 2nd International Workshop on Trends in Tree Automata and Tree Transducers (TTATT 2013), EPTCS 134. pp. 38–48 (2013)

[11] Schmidt, A., Waas, F., Kersten, M., Carey, M.J., Manolescu, I., Busse, R.: XMark: A benchmark for XML data management. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002). pp. 974–985 (2002)

The Square Trees in the Tribonacci Sequence

Yu-Ke Huang* and Zhi-Ying Wen

School of Mathematics and Systems Science, Beihang University (BUAA), Beijing, 100191, P. R. China; Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, P. R. China
`huangyuke@buaa.edu.cn,hyg03ster@163.com;wenzy@tsinghua.edu.cn`

Abstract. The Tribonacci sequence $\mathbb{T}$ is the fixed point of the substitution $\sigma(a) = ab$, $\sigma(b) = ac$ and $\sigma(c) = a$. In this note, we get the explicit expressions of all squares, and then establish the tree structure of the positions of repeated squares in $\mathbb{T}$, called square trees. Using the square trees, we give a fast algorithm for counting the number of repeated squares in $\mathbb{T}[1, n]$ for all n, where $\mathbb{T}[1, n]$ is the prefix of $\mathbb{T}$ of length n. Moreover we get explicit expressions for some special n such as $n = t_m$ (the Tribonacci number) etc., which including some known results such as in Mousavi-Shallit[6].

Keywords: kernel, square, gap sequence.

1 Introduction

Let $\mathcal{A} = \{a, b, c\}$ be a three-letter alphabet. The Tribonacci sequence $\mathbb{T}$ is the fixed point beginning with the letter a of the substitution $\sigma(a) = ab$, $\sigma(b) = ac$ and $\sigma(c) = a$. As a natural generalization of the Fibonacci sequence, $\mathbb{T}$ has been studied extensively by many authors, see [1,6,7,8].

Let ω be a factor of $\mathbb{T}$, denoted by $\omega \prec \mathbb{T}$. Let ω_p be the p-th occurrence of ω. If the factor ω and integers p, q such that $\omega_p\omega_q \prec \mathbb{T}$, we call $\omega_p\omega_q$ a square of $\mathbb{T}$. As we know, $\mathbb{T}$ contains no fourth powers. The properties of squares and cubes are objects of a great interest in mathematics and computer science etc.

We denote by $|\omega|$ the length of ω. Denote $|\omega|_\alpha$ the number of letter α in ω, where $\alpha \in \mathcal{A}$. Let $\tau = x_1 \cdots x_n$ be a finite word (or $\tau = x_1x_2\cdots$ be a sequence). We define $\tau[i, i-1] = \varepsilon$ (empty word), $\tau[i] = \tau[i, i] = x_i$ for $1 \leq i \leq n$; and $\tau[i, j] = x_ix_{i+1}\cdots x_{j-1}x_j$ for $i \leq j \leq n$. Denote $T_m = \sigma^m(a)$ for $m \geq 0$, $T_{-2} = \varepsilon$, $T_{-1} = c$. Denote $t_m = |T_m|$ for $m \geq -2$, called the m-th Tribonacci number. Denote by δ_m the last letter of T_m for $m \geq -1$.

Denote $A(n) = \#\{(\omega, p) \mid \omega_p\omega_q \prec \mathbb{T}[1, n]\}$ the number of repeated squares in $\mathbb{T}[1, n]$. In 2014, Mousavi-Shallit[6] gave expression of $A(t_m)$, which they proved by mechanical way. In [4], we give an algorithm for counting the number of repeated squares in each prefix of the Fibonacci sequence. In this note, we give an

* Corresponding author. The research is supported by the Grant NSFC No.11431007, No.11271223 and No.11371210. The authors thank sincerely the referees for many helpful suggestions.

algorithm for counting $A(n)$ for all n. In Section 2, we establish the tree structure of the positions of repeated squares in $\mathbb{T}$, called the square trees. Section 3 is devoted to give an algorithm for counting $A(n)$. As a special case, we get expression of $A(t_m)$ in Section 4.

The main tools of the paper are "kernel word" and "gap sequence", which introduced and studied in [2]. We define the kernel numbers that $k_0 = 0$, $k_1 = k_2 = 1$, $k_m = k_{m-1} + k_{m-2} + k_{m-3} - 1$ for $m \geq 3$. The kernel word with order m is defined as $K_1 = a$, $K_2 = b$, $K_3 = c$, $K_m = \delta_{m-1} T_{m-3}[1, k_m - 1]$ for $m \geq 4$. Using the property of gap sequence, we can determine the positions of all ω_p, and then establish the square trees.

2 The square trees

In [3], we determined the three cases of squares with kernel K_m (i.e., the maximal kernel word occurring in these squares is K_m). For $m \geq 4$ and $p \geq 1$, we denote

$$\Lambda(m,p) = pt_{m-1} + |\mathbb{T}[1,p-1]|_a(t_{m-2} + t_{m-3}) + |\mathbb{T}[1,p-1]|_b t_{m-2}.$$

By Property 6.1 in [5], we have the position of the last letter of the p-th occurrence of K_m that $P(K_m, p) = \Lambda(m,p) + k_m - 1$. Thus we can define three sets for $m \geq 4$ and $p \geq 1$, which contain all positions of the last letters of the three cases of squares, respectively.

$$\begin{cases} \langle 1, K_m, p\rangle & = \{P(\omega\omega, p) \mid Ker(\omega\omega) = K_m, |\omega| = t_{m-1}, \omega\omega \prec \mathbb{T}\} \\ & = \{\Lambda(m,p) + t_{m-1}, \ldots, \Lambda(m,p) + k_{m+3} - 2\}; \\ \langle 2, K_m, p\rangle & = \{P(\omega\omega, p) \mid Ker(\omega\omega) = K_m, |\omega| = t_{m-4} + t_{m-3}, \omega\omega \prec \mathbb{T}\} \\ & = \{\Lambda(m,p) + t_{m-3} + t_{m-4}, \ldots, \Lambda(m,p) + k_{m+2} - 2\}; \\ \langle 3, K_m, p\rangle & = \{P(\omega\omega, p) \mid Ker(\omega\omega) = K_m, |\omega| = t_{m-4} - k_{m-3}, \omega\omega \prec \mathbb{T}\} \\ & = \{\Lambda(m,p) + k_m - 1, \ldots, \Lambda(m,p) + 2t_{m-4} - 1\}. \end{cases}$$

In the sequel, a finite set of integers S is identified with the vector T obtained from S by arranging its entires in increasing order. For $m \geq 4$ and $p \geq 1$, we consider the vectors

$$\begin{cases} \Gamma_{1,m,p} = [\Lambda(m,p) + k_{m+2} - 1, \ldots, \Lambda(m,p) + k_{m+3} - 2]; \\ \Gamma_{2,m,p} = [\Lambda(m,p) + k_{m+1} - 1, \ldots, \Lambda(m,p) + k_{m+2} - 2]; \\ \Gamma_{3,m,p} = [\Lambda(m,p) + k_m - 1, \ldots, \Lambda(m,p) + k_{m+1} - 2]. \end{cases}$$

Obviously, $|\Gamma_{i,m,p}| = k_{m+4-i} - k_{m+3-i} = t_{m-i-1}$ for $i \in \{1,2,3\}$. And $\langle 1, K_m, p\rangle$ (resp. $\langle 2, K_m, p\rangle$, $\langle 3, K_m, p\rangle$) contains the several maximal (resp. maximal, minimal) elements of $\Gamma_{1,m,p}$ (resp. $\Gamma_{2,m,p}$, $\Gamma_{3,m,p}$). Moreover $\max \Gamma_{2,m,p} + 1 = \min \Gamma_{1,m,p}$ and $\max \Gamma_{3,m,p} + 1 = \min \Gamma_{2,m,p}$.

Now we denote $P(\alpha, p) + 1$ by $\hat{\alpha}$ for short, $\alpha \in \{a,b,c\}$. Using Lemma 6.4 in [5], comparing minimal and maximal elements in these sets below, we have $\Gamma_{1,m+1,p} = [\Gamma_{3,m,\hat{a}}, \Gamma_{2,m,\hat{a}}, \Gamma_{1,m,\hat{a}}]$, $\Gamma_{2,m+2,p} = [\Gamma_{3,m,\hat{b}}, \Gamma_{2,m,\hat{b}}, \Gamma_{1,m,\hat{b}}]$ and $\Gamma_{3,m+3,p} = [\Gamma_{3,m,\hat{c}}, \Gamma_{2,m,\hat{c}}, \Gamma_{1,m,\hat{c}}]$ for $m \geq 4$. By the relation between $\Gamma_{i,m,p}$ and

$\langle i, K_m, p\rangle$, we get the trees structure of the positions of repeated squares in $\mathbb{T}$, called the square trees. The square trees is a directed graph $\mathcal{G} = (V, A)$ where:

$$V = \{nodes\} = \{\langle i, K_m, p\rangle \mid i = 1, 2, 3;\ m \geq 4;\ p \geq 1\};$$

$$A = \{edges\} = \begin{cases} \langle 1, K_{m+1}, p\rangle \to \langle i, K_m, \hat{a}\rangle; \\ \langle 2, K_{m+2}, p\rangle \to \langle i, K_m, \hat{b}\rangle; \\ \langle 3, K_{m+3}, p\rangle \to \langle i, K_m, \hat{c}\rangle; \end{cases} \quad \text{for } i = 1, 2, 3;\ m \geq 4;\ p \geq 1.$$

Here the notation "$v \to x, y, z$" means three directed edges from the node v to nodes x, y and z, respectively.

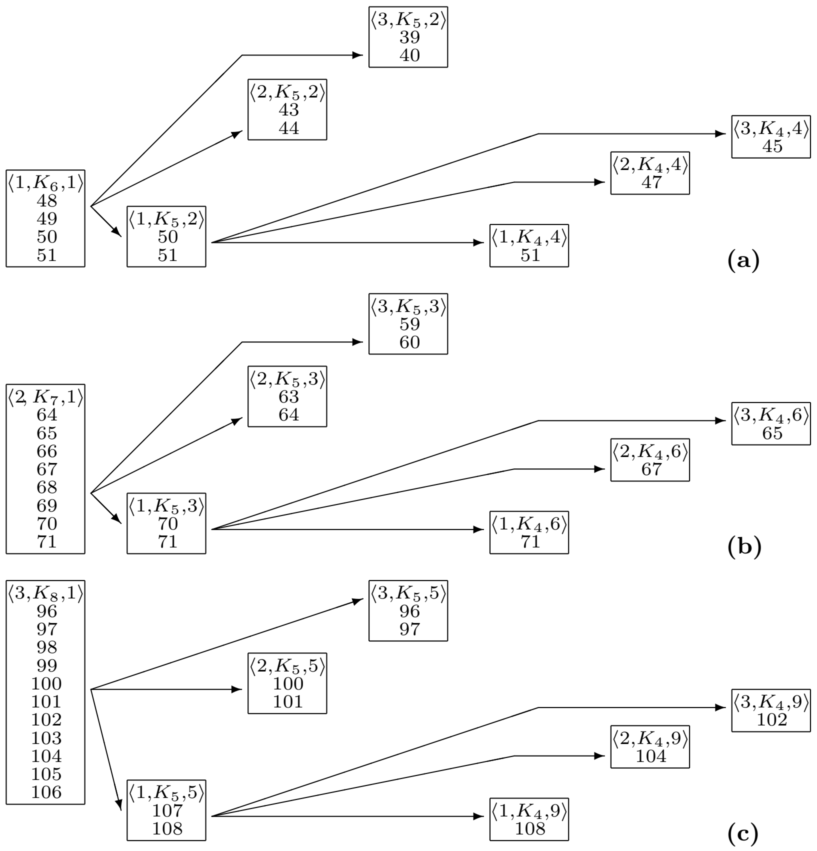


Fig. 1. (a)-(c) are square trees from root $\langle 1, K_6, 1\rangle$, $\langle 2, K_7, 1\rangle$, $\langle 3, K_8, 1\rangle$, respectively.

By the definition of $\mathcal{G}$ above, we have $\mathcal{G}$ is a family of finite trees with nodes $\{\langle i, K_m, p\rangle \mid i = 1, 2, 3;\ m \geq 4;\ p \geq 1\}$ satisfying the following conditions:

(1) The roots are $\langle i, K_m, 1\rangle$ for all $i = 1, 2, 3$ and $m \geq 4$;

(2) The leaves are $\langle i, K_4, p\rangle$ for all $i = 1, 2, 3$ and $p \geq 1$.

(3) We define the *range* of the tree with root $\langle i, K_m, 1\rangle$ to be the finite set of integers which belong to at least one of the nodes of the tree. The ranges of all the trees are pairwise disjoint.

3 Algorithm: the numbers of repeated squares in $\mathbb{T}[1, n]$

The notation $\nu \triangleright \omega$ means that the word ν is a suffix of the word ω. Denote $a(n) = \#\{(\omega, p) \mid \omega_p\omega_q \triangleright \mathbb{T}[1, n]\}$ the number of squares ending at position n. By Proposition 1 below, we can calculate $a(n)$, and obversely calculate $A(n)$ by $A(n) = \sum_{i=4}^{n} a(i)$. For $m \geq 4$, since $k_m = \frac{t_m - 2t_{m-1} + t_{m-2} + 1}{2}$,

$$\begin{cases} \Gamma_{1,m,1} = [\frac{t_m+2t_{m-1}-t_{m-2}-1}{2}, \ldots, \frac{t_m+2t_{m-1}+t_{m-2}-3}{2}];\\ \Gamma_{2,m,1} = [\frac{-t_m+4t_{m-1}+t_{m-2}-1}{2}, \ldots, \frac{t_m+2t_{m-1}-t_{m-2}-3}{2}];\\ \Gamma_{3,m,1} = [\frac{t_m+t_{m-2}-1}{2}, \ldots, \frac{-t_m+4t_{m-1}+t_{m-2}-3}{2}]. \end{cases}$$

We extend the expressions on $m = 3$. The first few values of $a(n)$ are:
$a([1, 2, 3]) = [0, 0, 0]$, $a(\Gamma_{3,3,1}) = a([4]) = [0]$, $a(\Gamma_{2,3,1}) = a([5]) = [0]$,
$a(\Gamma_{1,3,1}) = a([6,7]) = [0,0]$, $a(\Gamma_{3,4,1}) = a([8]) = [1]$, $a(\Gamma_{2,4,1}) = a([9,10]) = [0,1]$,
$a(\Gamma_{1,4,1}) = a([11, 12, 13, 14]) = [0, 0, 0, 1]$, $a(\Gamma_{3,5,1}) = a([15, 16]) = [1, 1]$.

Proposition 1. *For $m \geq 3$,*

$$\begin{cases} a(\Gamma_{1,m+1,1}) = [a(\Gamma_{3,m,1}), a(\Gamma_{2,m,1}), a(\Gamma_{1,m,1})] + [\underbrace{0, \ldots, 0}_{t_{m-1}-k_{m+1}+1}, \underbrace{1, \ldots, 1}_{k_{m+1}-1}];\\ a(\Gamma_{2,m+2,1}) = [a(\Gamma_{3,m,1}), a(\Gamma_{2,m,1}), a(\Gamma_{1,m,1})] + [\underbrace{0, \ldots, 0}_{t_{m-1}-k_{m+2}+1}, \underbrace{1, \ldots, 1}_{k_{m+2}-1}];\\ a(\Gamma_{3,m+3,1}) = [a(\Gamma_{3,m,1}), a(\Gamma_{2,m,1}), a(\Gamma_{1,m,1})] + [\underbrace{1, \ldots, 1}_{t_{m-1}-k_m+1}, \underbrace{0, \ldots, 0}_{k_m-1}]. \end{cases}$$

Denote $\Phi_m = \sum a(\Gamma_{3,m,1}) + \sum a(\Gamma_{2,m,1}) + \sum a(\Gamma_{1,m,1})$. The immediately corollaries of Proposition 1 are $\sum a(\Gamma_{1,m,1}) = \Phi_{m-1} + k_m - 1$, $\sum a(\Gamma_{2,m,1}) = \Phi_{m-2} + k_m - 1$, $\sum a(\Gamma_{3,m,1}) = \Phi_{m-3} + t_{m-4} - k_{m-3} + 1$. Moreover, for $m \geq 6$,

$$\Phi_m = \Phi_{m-1} + \Phi_{m-2} + \Phi_{m-3} + \frac{-3t_m + 6t_{m-1} + t_{m-2} - 1}{2}.$$

By induction, we can prove the four properties in Fig 2.

Algorithm. Step 1. For $n \geq 52$, find the integers m and i such that $n \in \Gamma_{i,m,1}$. Then $A(n) = A(\min \Gamma_{i,m,1} - 1) + \sum_{i=\min \Gamma_{i,m,1}}^{n} a(i)$.

Step 2. Since $A(\min \Gamma_{i,m,1}{-}1) = A(\max \Gamma_{i+1,m,1})$, $i = 1, 2$; $A(\min \Gamma_{3,m,1}{-}1) = A(\max \Gamma_{1,m-1,1})$. We calculate $A(\min \Gamma_{i,m,1} - 1)$ by Property (d) in Fig.2.

Step 3. We calculate $\sum_{i=\min \Gamma_{i,m,1}}^{n} a(i)$ by the properties in Fig.3.

4 Expression: the numbers of repeated squares in T_m

Since $\theta_m^8 \leq t_m < \theta_m^9$ and $\theta_{m-1}^6 \leq t_m - t_{m-1} < \theta_{m-1}^7$ for $m \geq 7$, see Fig.3,

$$\begin{aligned} &\textstyle\sum_{i=\min \Gamma_{1,m,1}}^{t_m} a(i) - \sum_{i=\min \Gamma_{1,m-3,1}}^{t_{m-3}} a(i)\\ &= \textstyle\sum a(\Gamma_{3,m-1,1}) + \sum a(\Gamma_{3,m-3,1}) + \sum a(\Gamma_{2,m-3,1}) + 2t_m - 2t_{m-1} - 3t_{m-2} + 1\\ &= \tfrac{m}{22}(-19t_m + 29t_{m-1} + 13t_{m-2}) + \tfrac{1}{44}(347t_m - 622t_{m-1} - 47t_{m-2}) + \tfrac{9}{4}. \end{aligned}$$

(a) $\Phi_m = \frac{m}{22}(-5t_m + 14t_{m-1} + 4t_{m-2}) + \frac{1}{44}(67t_m - 166t_{m-1} + 5t_{m-2}) + \frac{1}{4}$.
(b) $\sum a(\Gamma_{i,4,1}) = 1$ where $i \in \{1,2,3\}$, and

$$\begin{cases} \sum a(\Gamma_{1,m,1}) = \frac{m}{22}(4t_m - 9t_{m-1} + 10t_{m-2}) + \frac{1}{44}(19t_m + 36t_{m-1} - 169t_{m-2}) - \frac{1}{4}; \\ \sum a(\Gamma_{2,m,1}) = \frac{m}{22}(10t_m - 6t_{m-1} - 19t_{m-2}) + \frac{1}{44}(-189t_m + 156t_{m-1} + 331t_{m-2}) - \frac{1}{4}; \\ \sum a(\Gamma_{3,m,1}) = \frac{m}{22}(-19t_m + 29t_{m-1} + 13t_{m-2}) + \frac{1}{44}(237t_m - 358t_{m-1} - 157t_{m-2}) + \frac{3}{4}. \end{cases}$$

(c) $\sum_{j=4}^{m-1} \Phi_j = \frac{m}{44}(13t_m - 10t_{m-1} + 5t_{m-2}) + \frac{2}{11}(-8t_m + 8t_{m-1} - 7t_{m-2}) + \frac{m}{4} + 2$.

(d) $\begin{cases} A(\max \Gamma_{3,m,1}) = \frac{m}{44}(-25t_m + 48t_{m-1} + 31t_{m-2}) + \frac{1}{44}(173t_m - 294t_{m-1} - 213t_{m-2}) + \frac{m+11}{4}; \\ A(\max \Gamma_{2,m,1}) = \frac{m}{44}(-5t_m + 36t_{m-1} - 7t_{m-2}) + \frac{1}{22}(-8t_m - 69t_{m-1} + 59t_{m-2}) + \frac{m+10}{4}; \\ A(\max \Gamma_{1,m,1}) = \frac{m}{44}(3t_m + 18t_{m-1} + 13t_{m-2}) + \frac{1}{44}(3t_m - 102t_{m-1} - 51t_{m-2}) + \frac{m+9}{4}. \end{cases}$

Fig. 2. These properties can be proved easily by induction, where (a) and (c) hold for $m \geq 4$, (b) and (d) hold for $m \geq 5$.

For $m \geq 7$, by induction, $\sum_{i=\min \Gamma_{1,m,1}}^{t_m} a(i)$ is equal to

$$\frac{m}{44}(23t_m - 38t_{m-1} - 3t_{m-2}) + \frac{1}{44}(-65t_m + 164t_{m-1} - 105t_{m-2}) + \frac{3m}{4} - \frac{9}{4}.$$

Since $\min \Gamma_{1,m,1} - 1 = \max \Gamma_{2,m,1}$, $A(t_m) = A(\max \Gamma_{2,m,1}) + \sum_{i=\min \Gamma_{1,m,1}}^{t_m} a(i)$. By the properties in Fig.3, we prove Theorem 21 in [6] in a novel way: for $m \geq 3$, $A(t_m) = \frac{m}{22}(9t_m - t_{m-1} - 5t_{m-2}) + \frac{1}{44}(-81t_m + 26t_{m-1} + 13t_{m-2}) + m + \frac{1}{4}$.

References

1. Delecroix V., Hejda T., Steiner W.: Balancedness of Arnoux-Rauzy and Brun words. Lecture Notes in Computer Science. 8079, 119–131 (2013).
2. Huang Y K., Wen Z Y.: Kernel words and gap sequence of the Tribonacci sequence, Acta Mathematica Scientia (Series B). 36.1, 173–194 (2016).
3. Huang Y K., Wen Z Y.: The numbers of distinct squares and cubes in the Tribonacci sequence. Numeration 2016, Prague, (2016).
4. Huang Y K., Wen Z Y.: The number of distinct and repeated squares and cubes in the Fibonacci sequence. arXiv:1603.04211.
5. Huang Y K., Wen Z Y.: The numbers of repeated palindromes in the Fibonacci and Tribonacci sequences. arXiv: 1604.05021.
6. Mousavi H., Shallit J.: Mechanical proofs of properties of the Tribonacci word. Combinatorics on Words. Springer International Publishing. 170–190 (2014).
7. Richomme G., Saari K., Zamboni L.Q.: Balance and Abelian complexity of the Tribonacci word. Advance Applied Mathematic. 45, 212–231 (2010).
8. Tan B., Wen Z Y.: Some properties of the Tribonacci sequence. European J Combin. 28, 1703–1719 (2007).

(a) $n \in \mathit{\Gamma}_{3,m,1} = [\frac{t_m+t_{m-2}-1}{2}, \cdots, \frac{-t_m+4t_{m-1}+t_{m-2}-3}{2}]$ for $m \geq 7$. Denote

$$\begin{cases} \theta_m^1 &= \min \mathit{\Gamma}_{3,m,1} = \frac{t_m+t_{m-2}-1}{2}; \\ \theta_m^2 &= \min \mathit{\Gamma}_{3,m,1} + |\mathit{\Gamma}_{3,m-3,1}| = \frac{-5t_m+10t_{m-1}+3t_{m-2}-1}{2}; \\ \theta_m^3 &= \min \mathit{\Gamma}_{3,m,1} + |\mathit{\Gamma}_{3,m-3,1}| + |\mathit{\Gamma}_{2,m-3,1}| = \frac{-t_m+6t_{m-1}-3t_{m-2}-1}{2}; \\ \eta_m^1 &= \min \mathit{\Gamma}_{3,m,1} + t_{m-4} - k_{m-3} + 1 = -2t_m + 5t_{m-1}. \\ \theta_m^4 &= \max \mathit{\Gamma}_{3,m,1} + 1 = \min \mathit{\Gamma}_{2,m,1} = \frac{-t_m+4t_{m-1}+t_{m-2}-1}{2}. \end{cases}$$

Obviously, $\theta_m^3 < \eta_m^1 < \theta_m^4$ for $m \geq 7$, and $\min \mathit{\Gamma}_{3,m,1} - \min \mathit{\Gamma}_{3,m-3,1} = t_{m-1}$. By Property 1, we have: for $n \geq 52$, let m such that $n \in \mathit{\Gamma}_{3,m,1}$, then $m \geq 7$ and

$$\sum_{i=\min \mathit{\Gamma}_{3,m,1}}^{n} a(i)$$
$$= \begin{cases} \sum_{i=\min \mathit{\Gamma}_{3,m-3,1}}^{n-t_{m-1}} a(i) + n - \min \mathit{\Gamma}_{3,m,1} + 1, & \theta_m^1 \leq n < \theta_m^2; \\ \sum_{i=\min \mathit{\Gamma}_{2,m-3,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-3,1}) + n - \min \mathit{\Gamma}_{3,m,1} + 1, & \theta_m^2 \leq n < \theta_m^3; \\ \sum_{i=\min \mathit{\Gamma}_{1,m-3,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-3,1}) + \sum a(\mathit{\Gamma}_{2,m-3,1}) + n - \min \mathit{\Gamma}_{3,m,1} + 1, & \theta_m^3 \leq n < \eta_m^1; \\ \sum_{i=\min \mathit{\Gamma}_{1,m-3,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-3,1}) + \sum a(\mathit{\Gamma}_{2,m-3,1}) + \frac{-5t_m+10t_{m-1}-t_{m-2}+1}{2}, & otherwise. \end{cases}$$

(b) $n \in \mathit{\Gamma}_{2,m,1} = [\frac{-t_m+4t_{m-1}+t_{m-2}-1}{2}, \cdots, \frac{t_m+2t_{m-1}-t_{m-2}-3}{2}]$ for $m \geq 6$. Denote

$$\begin{cases} \theta_m^5 &= \min \mathit{\Gamma}_{2,m,1} + |\mathit{\Gamma}_{3,m-2,1}| = \frac{3t_m-5t_{m-2}-1}{2}; \\ \eta_m^2 &= \min \mathit{\Gamma}_{2,m,1} + t_{m-3} - k_m + 1 = 2t_{m-1} - t_{m-2}. \\ \theta_m^6 &= \min \mathit{\Gamma}_{3,m,1} + |\mathit{\Gamma}_{3,m-3,1}| + |\mathit{\Gamma}_{2,m-3,1}| = \frac{3t_m-2t_{m-1}-t_{m-2}-1}{2}; \\ \theta_m^7 &= \max \mathit{\Gamma}_{2,m,1} + 1 = \min \mathit{\Gamma}_{1,m,1} = \frac{t_m+2t_{m-1}-t_{m-2}-1}{2}. \end{cases}$$

Obviously, $\theta_m^5 < \eta_m^2 \leq \theta_m^6$ for $m \geq 6$, and $\min \mathit{\Gamma}_{2,m,1} - \min \mathit{\Gamma}_{3,m-2,1} = t_{m-1}$. By Property 1, we have: for $n \geq 32$, let m such that $n \in \mathit{\Gamma}_{2,m,1}$, then $m \geq 6$ and

$$\sum_{i=\min \mathit{\Gamma}_{2,m,1}}^{n} a(i)$$
$$= \begin{cases} \sum_{i=\min \mathit{\Gamma}_{3,m-2,1}}^{n-t_{m-1}} a(i), & \theta_m^4 \leq n < \theta_m^5; \\ \sum_{i=\min \mathit{\Gamma}_{2,m-2,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-2,1}), & \theta_m^5 \leq n < \eta_m^2; \\ \sum_{i=\min \mathit{\Gamma}_{2,m-2,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-2,1}) + n - \eta_m^2 + 1, & \eta_m^2 \leq n < \theta_m^6; \\ \sum_{i=\min \mathit{\Gamma}_{1,m-2,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-2,1}) + \sum a(\mathit{\Gamma}_{2,m-2,1}) + n - \eta_m^2 + 1, & otherwise. \end{cases}$$

(c) $n \in \mathit{\Gamma}_{1,m,1} = [\frac{t_m+2t_{m-1}-t_{m-2}-1}{2}, \cdots, \frac{t_m+2t_{m-1}+t_{m-2}-3}{2}]$ for $m \geq 5$. Denote

$$\begin{cases} \theta_m^8 &= \min \mathit{\Gamma}_{1,m,1} + |\mathit{\Gamma}_{3,m-1,1}| = \frac{t_m+3t_{m-2}-1}{2}; \\ \theta_m^9 &= \min \mathit{\Gamma}_{1,m,1} + |\mathit{\Gamma}_{3,m-1,1}| + |\mathit{\Gamma}_{2,m-1,1}| = \frac{-t_m+4t_{m-1}+3t_{m-2}-1}{2}; \\ \eta_m^3 &= \min \mathit{\Gamma}_{1,m,1} + t_{m-2} - k_m + 1 = 2t_{m-1}. \\ \theta_m^{10} &= \max \mathit{\Gamma}_{1,m,1} + 1 = \min \mathit{\Gamma}_{3,m+1,1} = \frac{t_m+2t_{m-1}+t_{m-2}-1}{2}. \end{cases}$$

Obviously, $\theta_m^9 < \eta_m^3 < \theta_m^{10}$ for $m \geq 5$, and $\min \mathit{\Gamma}_{1,m,1} - \min \mathit{\Gamma}_{3,m-1,1} = t_{m-1}$. By Property 1, we have: for $n \geq 21$, let m such that $n \in \mathit{\Gamma}_{1,m,1}$, then $m \geq 5$ and

$$\sum_{i=\min \mathit{\Gamma}_{1,m,1}}^{n} a(i)$$
$$= \begin{cases} \sum_{i=\min \mathit{\Gamma}_{3,m-1,1}}^{n-t_{m-1}} a(i), & \theta_m^7 \leq n < \theta_m^8; \\ \sum_{i=\min \mathit{\Gamma}_{2,m-1,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-1,1}), & \theta_m^8 \leq n < \theta_m^9; \\ \sum_{i=\min \mathit{\Gamma}_{1,m-1,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-1,1}) + \sum a(\mathit{\Gamma}_{2,m-1,1}), & \theta_m^9 \leq n < \eta_m^3; \\ \sum_{i=\min \mathit{\Gamma}_{1,m-1,1}}^{n-t_{m-1}} a(i) + \sum a(\mathit{\Gamma}_{3,m-1,1}) + \sum a(\mathit{\Gamma}_{2,m-1,1}) + n - \eta_m^3 + 1, & otherwise. \end{cases}$$

Fig. 3. (a)-(c) show the three cases of recursive relations between $\sum_{i=\min \mathit{\Gamma}_{k,m,1}}^{n} a(i)$ and $\sum_{i=\min \mathit{\Gamma}_{t,m-k,1}}^{n} a(i)$, where $k, t \in \{1, 2, 3\}$, respectively. These relations are derived directly from the square trees (the tree structure of the positions of repeated squares). Using them, we can calculate $\sum_{i=\min \mathit{\Gamma}_{k,m,1}}^{n} a(i)$ fast, and give a fast algorithm for $A(n)$.

The Output Size Problem for String-to-Tree Transducers

Brink van der Merwe[1], Nicolaas Weideman[1,2], and Frank Drewes[3]

[1] Department of Computer Science, Stellenbosch University, South Africa
[2] Center for AI Research, CSIR, Stellenbosch University, South Africa
[3] Department of Computer Science, Umeå University, Sweden

Abstract. The output size problem, for a string-to-tree transducer, is to determine the rate of growth of the function describing the maximum size of output trees, with respect to the length of the input strings. We study the complexity of this problem. Our motivation is that a solution to the output size problem can be used in order to determine the worst-case matching time (for a given regular expression) of a backtracking regular expression matcher, with respect to the length of the input strings.

Keywords: string-to-tree transducers, output size, backtracking regular expression matchers

1 Introduction

The complexity of determining the asymptotic behavior of the maximum output size for trees produced by a given top-down tree transducer, as a function of the size of input trees, was initiated in [4]. It was shown that the exponential output size problem is NL-complete for total top-down tree transducers, and DEXPTIME-complete for top-down tree transducers in general. We investigate the complexity of determining the degree of the polynomial, in cases where the maximum output size is polynomial in the size of the input, listed as future work in [4]. Due to space limitations, we restrict our attention to total string-to-tree transducers, but in contrast to [4], we allow transducer rules that, when applied, will not consume any input (so-called ε-input rules).

The motivation of this research is that the results can be used to estimate the matching time of backtracking regular expression matchers [2,3], by constructing transducers that produce as output the computation tree of a backtracking regular expression matcher, for a given input string that the regular expression is attempting to match. Thus in this case, the maximum output size provides the worst-case matching time behavior of a backtracking regular expression matcher, for a given regular expression. For lack of space, we cannot detail this in the present paper, but we will come back to it in the conclusion.

The outline of the paper is as follows. In the next section we introduce the required notation and definitions. After this, we describe how results on ambiguity of non-deterministic finite automata (NFA) can be applied to solve the output size problem, followed by our conclusions.

2 Definitions

In this section we introduce the notation and some of the definitions required for the remainder of the paper.

For a finite alphabet Σ, we denote the set of all strings (or sequences) over Σ, by Σ^*. In particular, Σ^* contains the empty string ε. To avoid confusion, it is assumed that $\varepsilon \notin \Sigma$, and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. For $\Sigma_1 \subseteq \Sigma$ and $w = a_1 \dots a_n \in \Sigma^*$, with $a_i \in \Sigma$, we let $\pi_{\Sigma_1}(w)$ be the word $b_1 \dots b_n \in \Sigma_1^*$, with $b_i = a_i$ if $a_i \in \Sigma_1$, and $b_i = \varepsilon$ otherwise. The length of a string w is denoted by $|w|$. We use $\mathbb{N}_+$ for the positive integers, and $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$.

A *tree*, with labels in a finite set Δ, is a function $t \colon t_D \to \Delta$, where $t_D \subseteq \mathbb{N}_+^*$ is a non-empty, finite set of vertices (or nodes) such that (i) t_D is prefix-closed, i.e., for all $v \in \mathbb{N}_+^*$ and $i \in \mathbb{N}_+$, $vi \in t_D$ implies $v \in t_D$, and (ii) t_D is closed to the left, i.e., for all $v \in \mathbb{N}_+^*$ and $i \in \mathbb{N}_+$, $v(i+1) \in t_D$ implies $vi \in t_D$. The vertex ε is the root of the tree and vertex vi is the ith child of v. We assume that Δ is a ranked alphabet, i.e. Δ is a union of (not necessarily disjoint) sets $\Delta^{(0)} \cup \Delta^{(1)} \cup \Delta^{(2)} \dots$ (with finitely of the $\Delta^{(i)}$ non-empty). When $f \in \Delta^{(k)}$, we say f has rank k, and we allow symbols in Δ to have more than one possible rank, which is required for our application to regular expressions. If we want to indicate explicitly that we consider f as a rank k symbol, we replace f by $f^{(k)}$. Also, all trees are ranked, thus if $v \in t_D$, then $vi \in t_D$, for all $1 \le i \le k$, where k is one of the possible ranks of $t(v)$. A node v such that $v1 \notin t_D$, is a leaf node. The yield of a tree t, denoted by yield(t), is the concatenation of the labels of the leaf nodes from left to right, i.e. yield$(t) = \alpha_1 \dots \alpha_m$, where $\alpha_i = t(v_i)$, and $v_1 \dots v_m$ are the leaf nodes in lexicographic order. We let $|t| := |t_D|$ denote the size of t and $|t|_S = |\{v \in t_D \mid t(v) \in S\}|$, for $S \subseteq \Delta$, the number of occurrences of symbols from S in t. Moreover, t/v ($v \in t_D$), denotes the tree t', with $t'_D = \{w \in \mathbb{N}_+^* \mid vw \in t_D\}$, where $t'(w) = t(vw)$ for all $w \in t'_D$. Given trees $t_1, \dots, t_n$ and $\alpha \in \Delta^{(n)}$, we let $\alpha[t_1, \dots, t_n]$ denote the tree t with $t(\varepsilon) = \alpha$ and $t/i = t_i$ for all $i \in \{1, \dots, n\}$. The tree $\alpha[\,]$ may be abbreviated as α. Trees t with $t(v) \in \Delta^{(0)} \cup \Delta^{(1)}$, for all $v \in t_D$, may be written as $\alpha_1\alpha_2 \dots \alpha_n$ (instead of $\alpha_1[\alpha_2[\dots[\alpha_n]]]$), where $\alpha_i \in \Delta^{(1)}$ for $i < n$ and $\alpha_n \in \Delta^{(0)}$, and also being regarded as strings (with rank 0 end markers). Given a ranked alphabet Δ, the set of all ranked trees $t \colon t_D \to \Delta$ is denoted by T_Δ. Moreover, if Q is an alphabet disjoint from Δ, we let $T_\Delta(Q) := T_{\Delta \cup Q}$ where $Q = Q^{(0)}$, i.e., the symbols in Q appear only at the leaves.

In the definition of an NFA below, the transition function δ is defined to allow for parallel transitions on the same symbol between a pair of states. By $\delta(p, \alpha, q) = i > 0$, we indicate that there are i transitions on α between p and q, numbered from 1 to i. By writing $p \xrightarrow{\alpha(j)} q$ or $p\alpha(j)q$, where $1 \le j \le \delta(p, \alpha, q)$, we refer to the jth-transition on α from p to q (but if $\delta(p, \alpha, q) = 1$, we may still use the standard notation $p \xrightarrow{\alpha} q$ or $p\alpha q$). Although parallel transitions do not influence the language accepted by an NFA, they do influence the number of accepting runs of a given input string, and thus play a role in our setting.

Definition 1. *A* non-deterministic finite automaton *(NFA) is a tuple $A = (Q, \Sigma, q_0, \delta, F)$ where: (i) Q is a finite set of* states; *(ii) Σ is the* input alphabet; *(iii) $I \subseteq Q$ is the set of* initial states; *(iv) the partial function $\delta : Q \times \Sigma_\varepsilon \times Q \to \mathbb{N}_+$ is the* transition function; and *(v) $F \subseteq Q$ is the set of* final states.

Also, $|A|_\delta := \sum_{q_1,q_2 \in Q, \alpha \in \Sigma_\varepsilon} \delta(q_1, \alpha, q_2)$ is the transition size *of A.*

Next we define (accepting) runs and the language accepted by an NFA.

Definition 2. *For an NFA $A = (Q, \Sigma, q_0, \delta, F)$ and $w \in \Sigma^*$, a* run *on w is a string $r = s_0\alpha_1(j_1)s_1 \cdots s_{n-1}\alpha_n(j_n)s_n$, with $s_0 \in I$, $s_i \in Q$ and $\alpha_i \in \Sigma_\varepsilon$ such that $\delta(s_i, \alpha_{i+1}, s_{i+1}) \geq j_{i+1}$ for $0 \leq i < n$, and the string $\pi_\Sigma(r) = \alpha_1 \cdots \alpha_n$ is equal to w. A run is accepting if $s_n \in F$. The language accepted by A, denoted by $\mathcal{L}(A)$, is the subset $\{\pi_\Sigma(r) \mid r \text{ is an accepting run in } A\}$ of Σ^*.*

Remark 1. Instead of our definition of NFA, one could also use weighted automata over the semiring $\mathbb{N}$, and replace the i transitions on α between p and q, when $\delta(p, \alpha, q) = i > 0$, by a single transition of weight i. We defined NFA as above, to keep Definition 6, in Section 3, as close as possible to the the corresponding definition from [1].

We now recall string-to-tree transducers, followed by the definition of the set output trees produced by a string-to-tree transducer, when applied to a given input string.

Definition 3. *A* string-to-tree transducer *is a tuple $td = (Q, \Gamma, \Delta, I, \delta)$, where $\Gamma = \Gamma^{(0)} \cup \Gamma^{(1)}$ and Δ are finite ranked input and output alphabets respectively, Q a finite set of states disjoint with Δ, $I \subseteq Q$ the initial states, and $\delta \subseteq (Q \times \Gamma^{(0)} \times T_\Delta) \cup (Q \times \Gamma_\varepsilon^{(1)} \times T_\Delta(Q))$ is the transition relation. When $(q, \alpha, t) \in \delta$, we also write $q \xrightarrow{\alpha} t$. Also, $|td|_\delta := \sum_{(q,\alpha,t)\in\delta} |t|$ is the* transition size *of td.*

We assume in the remainder of the paper that all transducers are string-to-tree transducers.

For $w \in T_\Gamma$, the set of output trees, when applying td to w, is denoted by $td(w) \subseteq T_\Delta$, and defined as follows. We have that $t \in td(w)$ if w can be written as $\alpha_1 \cdots \alpha_n$, with $\alpha_i \in \Gamma_\varepsilon^{(1)}$ for $i < n$ and $\alpha_n \in \Gamma^{(0)}$, such that there exists a sequence of trees $t_0, \ldots, t_n \in T_\Delta(Q)$ with $t_0 \in I$ and $t_n = t$; and for every $i \in \{1, \ldots, n\}$, t_i is obtained from t_{i-1} by replacing every leaf v for which $t_{i-1}(v) \in Q$ with any tree t' such that $t_{i-1}(v) \xrightarrow{\alpha_i} t'$.

We now define when transducers are total and deterministic. The presence of ε-input rules leads to non-standard definitions. A transducer td is total if $td_q(w) \neq \emptyset$ for all $q \in Q$ and $w \in T_\Gamma$ (where td_q is td with its initial state replaced by q), and deterministic if for each $q \in Q$ and $\alpha \in \Gamma$, we have at most one rule of the form $q \xrightarrow{\varepsilon} t$ or $q \xrightarrow{\alpha} t'$ in δ.

Definition 4. *A transducer $td = (Q, \Gamma, \Delta, I, \delta)$ is* total *if $q \in Q, a^{(1)}, b^{(0)} \in \Gamma$ implies $td_q(ab) \neq \emptyset$ and $td_q(b) \neq \emptyset$, where td_q is $(Q, \Gamma, \Delta, \{q\}, \delta)$. Also, td is* deterministic *if for all $q \in Q$ and $\alpha \in \Gamma$, there is at most one rule of the form $q \xrightarrow{\varepsilon} t$ or $q \xrightarrow{\alpha} t'$ in δ.*

Next we give the output size definition from [4], and also an output size definition based on the length of the yield of the output trees.

Definition 5. *The* full output size *and the yield output size of a transducer td is given respectively by functions $os^F_{td}, os^Y_{td} : \mathbb{N}_+ \to (\mathbb{N} \cup \{\infty\})$, such that $os^F_{td}(n) = \max\{|t| \mid t \in td(s)$ and $|s| \leq n\}$ (with $\max \emptyset = 0$), and $os^Y_{td}(n) = \max\{|yield(t)| \mid t \in td(s)$ and $|s| \leq n\}$. The* exponential output size problem *is to decide if os^F_{td} has exponential rate of growth, and the* polynomial output size problem *is to determine the degree of the asymptotic polynomial growth of os^F_{td} and os^Y_{td} (if it is polynomial).*

Note that since we allow ε-input transducer rules, it may happen that output trees of arbitrary size are produced for input trees of a given fixed size.

In the following example, transducers with exponential and with polynomial (of arbitrary degree) output size, are given.

Example 1. First we define transducers td_k with $os^F_{td_k}$ (and $os^Y_{td_k}$) exponential. Let $td_k = (Q, \Gamma, \Delta_k, \{q_0\}, \delta_k)$ with $Q = \{q_0\}$, $\Gamma = \{a^{(0)}, f^{(1)}\}$, $\Delta_k = \{a^{(0)}, g^{(k)}\}$, for some fixed integer $k \geq 2$, and $\delta_k = \{q_0 \xrightarrow{f} g[q_0, \ldots, q_0], q_0 \xrightarrow{a} a\}$. Then $td_k(f^n a)$ is a perfect k-ary tree of height n, and $|td_k(f^n a)|$ is thus exponential, with base k, in n.

Next we give transducers $\overline{td}_k$, for $k \geq 1$, such that $os^F_{\overline{td}_k}$ and $os^Y_{\overline{td}_k}$ are polynomial, of degrees k and $(k-1)$, respectively. In general, in the polynomial case, the degree of os^F_{td} is at most 1 larger than that of os^Y_{td}. We let $\overline{td}_k = (Q, \Gamma, \Delta, \{q_k\}, \delta_k)$ with $Q_k = \{q_1, \ldots, q_k\}$, $\Gamma = \{a^{(0)}, f^{(1)}\}$, $\Delta = \{a^{(0)}, f^{(1)}, g^{(2)}\}$, and $\delta_k = \{q_i \xrightarrow{f} g[q_{i-1}, q_i]$ for $1 < i \leq k\} \cup \{q_1 \xrightarrow{f} f[q_1]\} \cup \{q_i \xrightarrow{a} a$ for $0 \leq i \leq k\}$. We have that $|\overline{td}_k(f^n a)| \in \Theta(n^k)$ and $|\text{yield}(\overline{td}_k(f^n a))| \in \Theta(n^{k-1})$, which is obtained by induction, using $\overline{td}_k(f^n a) = g[\overline{td}_{k-1}[f^{n-1}a], \overline{td}_k[f^{n-1}a]]$, for $k > 1$. □

3 Linking output size to NFA ambiguity

In this section we show how results on ambiguity of NFA, from [1], can be used to approximate os^F_{td} and os^Y_{td}, when td is total. We note here that, although in this paper we use NFA with possibly parallel transitions, the results from [1] (which were proved for ordinary NFAs) still hold. This can easily be seen by replacing every transition by a succession of two transitions on the same symbol, with a fresh state in between. We begin by introducing definitions related to NFA ambiguity.

Definition 6. *The degree of ambiguity for $w \in \Sigma^*$, with respect to the NFA A, denoted by $d_A(w)$, is the number of accepting runs on w in A. The degree of ambiguity of A is the maximum degree of ambiguity over all $w \in \Sigma^*$, which might be infinite, in which case we say A has* infinite degree of ambiguity *(IDA). When A has IDA (and A does not have ε-loops), we consider the rate at which the maximum number of accepting runs grows in proportion to the length of the input strings, which might be exponential, described by saying A has* exponential degree of ambiguity *(EDA), or polynomial, described as A being* polynomially ambiguous.

With every total deterministic transducer $td = (Q, \Gamma, \Delta, I, \delta)$, we associate NFAs nfa^F_{td} and nfa^Y_{td}, for which the ambiguity of a given input string w is equal to $|td(w)|$ and $|\mathrm{yield}(td(w))|$, respectively. If td is not total, the ambiguity values only provide upper bounds for $|td(w)|$ and $|\mathrm{yield}(td(w))|$.

The NFAs nfa^Y_{td} and nfa^F_{td} are constructed in a very similar way. Their input alphabet is Γ. The set of initial states I is the same as for td and the set of states is $Q \cup \{q_p, q_f\}$, where q_p and q_f are fresh states and q_f the only final state. The transition functions $\delta_Y$and δ_F of nfa^Y_{td} and nfa^F_{td} are built according to the following intuition: If td processes a symbol of rank 1 in a state q, producing the partial output t, then one "process" q_p is spawned for each occurrence of a symbol in $\Delta^{(0)}$ in the case of nfa^Y_{td}. In the case of nfa^F_{td}, the same is done for each occurrence of a symbol in Δ. Each of these will give rise to a single accepting computation without branching any further. In addition, each occurrence of a state in t gives rise to a corresponding sub-computation of nfa^Y_{td} and nfa^F_{td}, resp. Formally, we define $\delta_X(q_p, \alpha, q_p) = \delta_X(q_p, \beta, q_f) = 1$ for $\alpha \in \Gamma^{(1)}, \beta \in \Gamma^{(0)}$ and $X \in \{F, Y\}$. For every $(q, \alpha, t) \in \delta$ we let

$$\delta_X(q, \alpha, q') = \begin{cases} |t|_\Delta & \text{if } \alpha \in \Gamma^{(0)}, q' = q_f \text{ and } X = F; \\ |t|_\Delta & \text{if } \alpha \in \Gamma^{(1)}, q' = q_p \text{ and } X = F; \\ |t|_{\Delta^{(0)}} & \text{if } \alpha \in \Gamma^{(0)}, q' = q_f \text{ and } X = Y; \\ |t|_{\Delta^{(0)}} & \text{if } \alpha \in \Gamma^{(1)}, q' = q_p \text{ and } X = Y; \\ |t|_{\{q'\}} & \text{if } q' \in Q \text{ and } X \in \{F, Y\}. \end{cases}$$

Following the intuition explained above, one can easily prove the following.

Lemma 1. *Let td be a total deterministic transducer and let nfa^F_{td} and nfa^Y_{td} be as above. Then for all $w \in T_\Gamma$ we have that $d_{nfa^X_{td}}(w) = os^X_{td}(w)$, and for $w \in \Gamma^* \setminus T_\Gamma$, we have $d_{nfa^X_{td}}(w) = 0$, with $X \in \{F, Y\}$. Also, let δ, δ_F and δ_Y be the transitions functions of td, nfa^F_{td} and nfa^Y_{td}, resp. Then $|nfa^Y_{td}|_{\delta_Y} \leq |nfa^F_{td}|_{\delta_F} = |td|_\delta$.*

By [4, Lemma 3.2], for every transducer td one can construct a deterministic transducer td' such that, for some constant $a \in \mathbb{N}_+$, $os^X_{td}(n/a) \leq os^X_{td'}(n) \leq os^X_{td}(n)$ for all $n \in \mathbb{N}$ and $X \in \{F, Y\}$. Also, the construction preserves totality and can be carried out on logarithmic space. Altough this lemma is only stated and proved for full output size and transducers without ε-input rules, it is straightforward to extend the result to our more general setting. Thus, Lemma 1 can also be used in the case where td is nondeterministic. (Alternatively, the construction used in [4] can easily be incorporated into the way nfa^Y_{td} and nfa^F_{td} are built, thus avoiding the need to modify td.) Together with results from [1], on the complexity of determining various types of ambiguity, we obtain the following result on full and yield output size for transducers.

Theorem 1. *Let td be a total deterministic transducer. Then:*

1. *It is decidable in time $O(|td|^2_\delta)$ if td has exponential full (and yield) output size.*

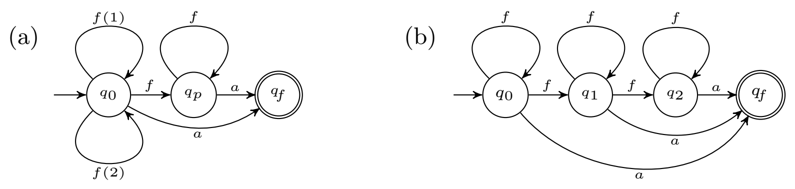


Fig. 1. (a) $nfa^{F}_{td_2}$ and (b) $nfa^{Y}_{\overline{td}_3}$, with td_2 and $\overline{td}_3$ as defined in Example 1.

2. If the full (and yield) output size of td is not exponential (and not ∞, corresponding to nfa^{F}_{td} or nfa^{Y}_{td} having ε-loops), then it is polynomial, and the degree of the polynomial growth of the full and yield output size of td can be computed in time $O(|td|^3_\delta)$.

Example 2. In Figure 3, we show $nfa^{F}_{td_2}$ and $nfa^{Y}_{\overline{td_3}}$, with td_2 and $\overline{td_3}$ as in Example 1. Clearly, both $nfa^{F}_{td_2}$ and $nfa^{Y}_{\overline{td_3}}$ has IDA, and $nfa^{F}_{td_2}$ has EDA, whereas $nfa^{Y}_{\overline{td_3}}$ is polynomial ambiguous of degree 2 (also see [1], for NFA properties characterizing various types of ambiguity). Thus td_2 has exponential full (and yield) output size, and the yield output size of $\overline{td_3}$ grows quadratically. □

4 Conclusions and Future Work

In [2,3] the time complexity of backtracking regular expression matchers was investigated. Starting from a regular expression E, the question asked is how efficient (or inefficient) the corresponding matcher is. It was shown that, given E, one can construct a transducer td_E such that, for every input string w, $td_E(w)$ (or more precisely, $td_E(w\$)$, with \$ a rank 0 symbol) represents the computation tree of the matcher. Hence, to know the full output size of td_E is to know the running time of the matcher. Unfortunately, td_E is not total, which means that we cannot directly apply Theorem 1 but have to extend it to non-total transducers first. This, together with a detailed exposition of the facts mentioned in the previous paragraph, is our agenda for future work.

Acknowledgment. We thank the reviewers for suggesting some improvements of the presentation.

References

1. Allauzen, C., Mohri, M., Rastogi, A.: General algorithms for testing the ambiguity of finite automata. In: Ito, M., Toyama, M. (eds.) Proc. 12th Intl. Conf. on Developments in Language Theory (DLT 2008). LNCS, vol. 5257, pp. 108–120 (2008)
2. Berglund, M., Drewes, F., van der Merwe, B.: Analyzing catastrophic backtracking behavior in practical regular expression matching. In: Ésik, Z., Fülöp, Z. (eds.) Proce. 14th Intl. Conf. on Automata and Formal Languages (AFL 2014). EPTCS, vol. 151, pp. 109–123 (2014)

3. Berglund, M., van der Merwe, B.: On the semantics of regular expression parsing in the wild. In: Drewes, F. (ed.) 20th Intl. Conf. on Implementation and Application of Automata (CIAA 2015). LNCS, vol. 9223, pp. 292–304 (2015)
4. Drewes, F.: The complexity of the exponential output size problem for top-down and bottom-up tree transducers. Inf. Comput. 169(2), 264–283 (2001)

Aligned Multistring Languages

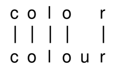
Anssi Yli-Jyrä

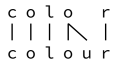
University of Helsinki, Finland
Anssi.Yli-Jyra@helsinki.fi

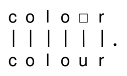
Abstract. Aligned bistrings and multistrings generalize Solomon Marcus' structured strings to configurations of multiple strings related with alignment structures. Their languages form an interesting set of objects that are related to n-way same-length relations, parallel corpora, aligned gene sequences, syntactic trees, phonological derivations, and autosegmental representations. The current work-in-progress aims to extend the recently discovered class of inertial bistrings for which a coding morphism is available. The code words generate a prime decomposition of inertial bistrings and yield a way to recognize bistring languages with string automata. It is now proven that a coding morphism is also available for so called intervallic bistrings that may contain some crossing associations. Relational join of bistrings gives rise to multistrings that may also have a coding morphism when their join configuration is a tree. For tree-shaped inertial and intervallic multistrings, the recognizable languages are closed under concatenation, star, and Boolean operations, as well as join and projection, but not necessarily path contraction. It remains open if path contraction of a recognizable intervallic multistring language always results in a recognizable intervallic multistring language and if there is a constructive proof for this.

1 Aligned Bistring Languages

An **aligned bistring** is defined in [11] as a triple $S = (\sigma, \tau, E)$ where $\sigma = s_1 s_2 \ldots s_{|\sigma|}$ and $\tau = t_1 t_2 \ldots t_{|\tau|}$ are component strings over respective finite alphabets and E is a subset of $[1..|\sigma|] \times [1..|\tau|]$ called the **association relation**. Our slightly modified definition adds the possibility to use a padding symbol and a shared alphabet. The modified definition is illustrated by the diagrams

(a) 

(b) 

(c) 

(1)

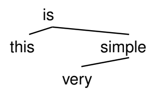
These diagrams represent, respectively, the bistrings

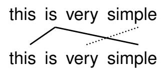
$$(\text{color}, \text{colour}, \{(1,1),(2,2),(3,3),(4,4),(5,6)\})$$
$$(\text{color}, \text{colour}, \{(1,1),(2,2),(3,3),(4,4),(4,5),(5,6)\})$$
$$(\text{colo□r}, \text{colour}, \{(1,1),(2,2),(3,3),(4,4),(5,5),(6,6)\}).$$

In (1a-b), the component strings are not of the same length whereas diagram (1c) describes only very simple letter-to-letter correspondences. The core alphabet of the bistrings is Σ but an additional padding symbol $\Box \notin \Sigma$ gives the option to associate insertions and deletions in the component strings to the symbol. *Bistrings*(Σ) is the set of all aligned bistrings with component strings over the total alphabet $\Sigma_\Box = \Sigma \cup \{\Box\}$. Its subsets are called **aligned bistring languages**.

The empty string is denoted by ϵ and the empty bistring with $(\epsilon, \epsilon, \{\})$. **Concatenation** of bistrings is defined in the natural way, with no added associations. Special concatenations that add new associations can also be defined [11,9].

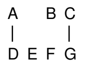
Aligned bistrings generalize the notion of structured strings, due to Solomon Marcus (1925-2016) [13,14]. These are pairs (s, E) where $s = s_1...s_n \in \Sigma^*$ is a string over alphabet Σ and E is an anti-reflexive dependency relation over string positions $\{1, ..., n\}$. The rooted tree-structured string in diagram (2a) can be viewed as an aligned bistring (s, s, E) (2b) with a **copy property**.

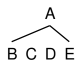
(a) 

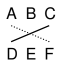
(b) 

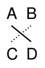
(2)

Since all bistrings are at least trivially aligned with $E = \{\}$, we drop the qualifier *aligned* in the presence of further qualifiers. Languages of **fully associated bistrings** [11] are sets of bijectively associated bistrings such as (1b-c) and (3d), in contrast to (1a), (2b) and (3a-c). In (3a), the respective order of *unassociated* elements B, E and F is not specified and the bistring cannot be unambiguously factorized. Languages of **semi-inertial bistrings** restrict the unassociated elements of bistrings to *interrupting* ones, unlike (3a) that contains unassociated elements that are not interruptions. Languages of **proper bistrings** [11] consists of such bistrings as (1a-c) and (3a,d) where no interruptions occur. Languages of **well-formed bistrings** consists of bistrings like (1a-c,3a-b) where no crossing associations occur. Languages of **one-to-one bistrings** consist of fully associated bistrings such as (1c) and (3d). Languages of **inertial bistrings** [18] consist of well-formed semi-inertial bistrings, such as (1b-c,3b).

(a) 

(b) 

(c) 

(d) 

(3)

Proposition 1 (e.g. [10]). *Closure of a finite set of well-formed one-to-one bistrings under concatenation, star, union and relative complement is effective.*

Theorem 1 ([18,20] for inertial bistrings; [19] for semi-inertial b.s.). *The finest factorization of semi-inertial bistrings is unambiguous.*

Theorem 1 gives rise to prime decomposition of all semi-inertial bistrings $\mathit{Bistrings}_{\text{semi-inertial}}(\Sigma)$. The possible factors in prime decomposition of semi-inertial bistrings constitute an infinite set of **semi-inertial primes** $\mathcal{P}_{\text{semi-inertial}}$. Its well-formed restriction, $\mathcal{P}_{\text{inertial}}$, generates the set $\mathit{Bistrings}_{\text{inertial}}(\Sigma)$.

A set $X \subseteq A^*$ is a **code**[2] if all strings $w \in X^*$ have an unambiguous factorization $w = w_1 w_2 ... w_n$ into elements of X. A morphism $\mu : H^* \to A^*$ from a free monoid $(H^*, \cdot, \epsilon_\mu)$ into a free monoid $(A^*, \cdot, \epsilon)$ is a **coding morphism** for a code $X \subset A^*$ if μ is injective and saturates X by $X = \mu(H)$.

Theorem 2 ([18]). *There is a bijection between the inertial bistring primes* $\mathcal{P}_{inertial}$ *and an infinite regular language* X *that is a code.*

Corollary 1 ([18]). *There is a coding morphism between the set of inertial bistrings* $\mathcal{P}^*_{inertial}$ *and the regular language* X^* *generated by a code* X.

E.g., (1b) and (3b) could be mapped to code words '$|{}^{\mathtt{c}}{}_{\mathtt{c}}|{}^{\mathtt{o}}{}_{\mathtt{o}}|{}^{\mathtt{l}}{}_{\mathtt{l}}|{}_{\mathtt{o}}|{}^{\mathtt{o}}{}_{\mathtt{u}}|{}^{\mathtt{r}}{}_{\mathtt{r}}$' and '$|{}_{\mathtt{B}}{}^{\circ}{}_{\mathtt{C}}{}^{\circ}{}_{\mathtt{D}}|{}^{\mathtt{A}}{}_{\mathtt{E}}$', respectively, using the Polish code described in [18]. (In comparison to [18], we turn the diagrams upside down for more intuitive presentation.)

A bistring set $L \subseteq \mathcal{P}^*_{\text{inertial}}$ is X**-recognizable** if its image under a coding morphism $\mathcal{P}^*_{\text{inertial}} \to A^*$ for the code $X \subset A^*$ is a recognizable subset of A^*.

Corollary 2. *The closure of* X*-recognizable sets of inertial bistrings under concatenation, star, union and relative complement is effective.*

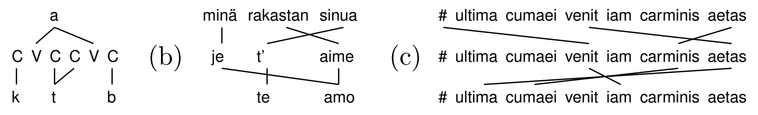
2 Aligned Multistring Languages

Proposition 2. *The recognizable well-formed one-to-one bistring languages (aka same-length regular relations [10]) extend to* n*-way same-length relations that are effectively closed under concatenation, star and the Boolean operations.*

We want to generalize Corollary 2 and Proposition 2 to larger families of aligned bistring languages with multiple component strings and more complex association relations. The motivation comes from autosegmental phonology [6], nonconcatenative morphology [15], multi-level derivations [8], multilingual parallel corpora [3] and multiplanar decomposition of dependency trees [16]. Further motivation comes from bioinformatics and parallel computing.

Example (4a) presents the nonconcatenative structure of the Arabic stem *kattab* meaning 'caused to write', (4b) presents alignments between Finnish-French-Spanish translations of the sentence 'I love you', (4c) presents a multiplanar decomposition of the dependencies in a Latin sentence 'The final age of the Cumaean song has now arrived' (Virgil, *Eclogues*, IV.4).

(a) (b) (c) (4)


a
C V C C V C
k t b
minä rakastan sinua
je t' aime
te amo
# ultima cumaei venit iam carminis aetas
# ultima cumaei venit iam carminis aetas
# ultima cumaei venit iam carminis aetas

Let $K = (V_K, E_K)$ be a graph over vertices $V_K = \{1, ..., k\}$ and an irreflexive relation $E_K \subseteq V \times V$. An **aligned** E_K**-multistring** is a map $m : E_K \to$ **Bistrings**(Σ) satisfying a **constraint** according to which (i) the image of m consists of bistrings $m((i,j)) = \big(\sigma_{m,i}, \sigma_{m,j}, E_{i,j}\big)$ where $(i,j) \in E_k$, and (ii) the component strings of these constitute the set $\{\sigma_{m,1}, ..., \sigma_{m,k}\}$. *Multistrings*$(\Sigma, K)$ is the set of aligned E_K-multistrings with component strings over alphabet $\Sigma_\Box$. Its subsets are called **aligned** E_K**-multistring languages**. The qualifiers (*inertial* etc.) of bistring languages are lifted to these languages in the natural way.

Any bistring (σ, τ, E) can be seen as a multistring $\{(1,2) \mapsto (\sigma, \tau, E)\}$. Examples (4a-c) are $\{(1,2),(2,3)\}$-multistrings but the bistrings in (4c) have, in addition, the copy property mentioned above. An E_K-multistring is **acyclic** if the graph K is acyclic. [6] discuss K-multistrings where K is a tree. Now we want to generalize Theorem 2 to languages of **tree-shaped** inertial multistrings.

For arbitrary alphabets $A, B \subseteq A$, let $\pi_B : A^* \to B^*$ be a morphism that eliminates the letters $A \backslash B$ in strings.

Lemma 1. *Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be regular subsets of A^*. There is a language $L_1 \uparrow L_2 = \{w \in A^* \mid \pi_{\Sigma_1}(w) \in L_1, \pi_{\Sigma_2}(w) \in L_2\}$ such that $\{(x, y) \in L_1 \times L_2 \mid \pi_{\Sigma_2}(x) = \pi_{\Sigma_1}(y)\} = \{(\pi_{\Sigma_1}(w), \pi_{\Sigma_2}(w)) \mid w \in L_1 \uparrow L_2\}$ and, for all $w_1, w_2 \in L_1 \uparrow L_2$, $\pi_{\Sigma_1}(w_1) = \pi_{\Sigma_1}(w_2)$ and $\pi_{\Sigma_2}(w_1) = \pi_{\Sigma_2}(w_2)$ only if $w_1 = w_2$.*

Lemma 1 varies the idea of *composition filters* [1] and *synchronized infiltration* [17] of two regular languages to combine two regular languages in an unambiguous way. If $L_1 = \{abca, cba\} \subseteq \{a, b, c\}^*$ and $L_2 = \{dabda, dbda\} \subseteq \{a, b, d\}^*$, $L_1 \uparrow L_2$ could be defined, for example, as $\{dabdca, dcbda\}$ or $\{dabcda, cdbda\}$.

Lemma 2. *Let L_1 and L_2 be an inertial bistring languages. Then there are coding morphisms $\mu_1 : \mathcal{P}^*_{inertial} \to \Sigma_1^*$, $\mu_2 : \mathcal{P}^*_{inertial} \to \Sigma_2^*$ such that $\pi_{\Sigma_2}(\mu_1((\sigma_1, \tau_2, E))) = \pi_{\Sigma_1}(\mu_2((\sigma_2, \tau_3, F)))$ exactly when $\tau_2 = \sigma_2$.*

For example, if $\mu_1(\overset{A}{{}_B\wedge_C})$='$\mathtt{1}B\mathtt{1}^A C$' and $\mu_2(\underset{D}{{}^B\vee^C})$='$\mathtt{2}B\mathtt{2}C_D$' then $\pi_{\Sigma_2}(\mu_1(\overset{A}{{}_B\wedge_C})) = \pi_{\Sigma_1}(\mu_2(\underset{D}{{}^B\vee^C}))$ = 'BC'. Here we adapted the Polish code from [18].

Theorem 3. *There is a bijection between inertial $\{(1, 2), (2, 3)\}$- multistrings and a regular language L, and there is a code X such that $L = X^*$.*

For example, a specific implementation of the $\uparrow$-operator in Lemma 1 combines the Polish code words of the previous example as the string '$\mathtt{12}B\mathtt{1}^A\mathtt{2}C_D$'. This string is a code word that encodes the $\{(1, 2), (2, 3)\}$-multistring $(\overset{A}{{}_B\wedge_C}, \underset{D}{{}^B\vee^C})$.

Corollary 3. *The closure of X-recognizable sets of inertial $\{(1, 2), (2, 3)\}$- multistrings under concatenation, star, union and relative complement is effective.*

Theorem 3 generalizes to more complex tree-shaped multistrings [4,6], but the necessary steps cannot be elaborated here.

Let $K_1 = (\{1, ..., k\}, E_1)$, $K_2 = (\{1, ..., l\}, E_2)$ be trees with 1 as the root. For $g \in \{1, ..., k\}$ and K_1 and K_2-multistring languages L_1 and L_2, the ***g*-join of** K_2 **into** K_1 is a $(E_1 \cup E_2')$-multistring language $L_1 \bowtie_g L_2$ where $E_2' = \{(g, j{+}k{-}1) \mid (1, j) \in E_2\} \cup \{(i + k - 1, j + k - 1) \mid (i, j) \in E_2, i \neq 1\}$. This is defined by

$$L_1 \bowtie_g L_2 = \left\{ m_1 \cup f(m_2) \;\middle|\; m_1 \in L_1, m_2 \in L_2, \sigma_{m_1,g} = \sigma_{m_2,1} \right\}$$

where $f((i, j) \mapsto \beta) = \begin{cases} (g, j{+}k{-}1) \mapsto \beta & \text{if } i = 1 \\ (i{+}k{-}1, j{+}k{-}1) \mapsto \beta & \text{otherwise.} \end{cases}$

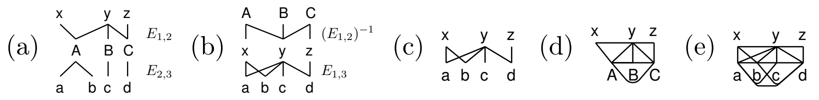
Theorem 4. *Let L_1 and L_2 be X-recognizable languages of inertial tree-shaped multistrings. The join of L_2 into L_1 is an X-recognizable multistring language.*

The join operation and the multistring languages obtained through it can be used to generalize the results of [8] on multi-tape compositions in phonology: If aligned bistring languages implement phonological relations, then join operation does not only compute the composition of the relations but also builds the derivation history of each surface phoneme.

Component projection of a bistring language produces an ordinary string language. This generalizes to **projection (dropping a leaf of K)** in a K-multistring language, aka a *chart reduction* [6]. For many encoding schemes, projection of a leaf string is easy to implement.

Theorem 5. *Projection dropping a leaf vertex of K in a recognizable inertial K-multistring language is effective.*

For 1-rooted trees K, **path contraction** (cf. the *alignment* operation in [11]) lifts an edge in E_K to a higher node along the path towards the root of K and simultaneously updates a K-multistring language by modifying bistrings accordingly: the bistring that is further away from the root is replaced with a new bistring that represents the relational composition of the adjacent bistrings. The contraction of path $(1-2-3)$ in a $\{(1,2),(2,3)\}$-multistring language produces a $\{(1,2),(1,3)\}$-multistring language. After this, projection can be applied to obtain a $\{(1,3)\}$-multistring language.



(5)

Example (5a) is an inertial multistring. The contraction of the path $(1-2-3)$ creates crossing associations (5b). The contracted path is singled out in (5c).

Lemma 3. *Path contraction does not generally preserve well-formedness.*

3 Intervallic Bistrings

A graph G is an **interval graph** [5] if and only if the maximal cliques of the graph can be ordered to a **clique path** $(C_1, C_2, ..., C_n)$ in such a way that if an element belongs to C_i and C_k, $i < k$, then it belongs to every clique C_j such that $i < j < k$ [12]. The edges of such a graph represent intersections of intervals in the real line [7]. A bistring is **intervallic** if

1. it is fully associated (a simplifying restriction due to space constraints here),
2. it can be converted to an interval graph by extending its complete bipartite subgraphs into cliques with additional edges, and
3. the clique path respects the linear order of component strings.

For example, bistring $(\mathsf{abc}, \mathsf{ABC}, \{(1,1),(1,2),(1,3),(2,2),(3,3)\}$ is not intervallic because there is no way to order the cliques $\{\mathsf{a},\mathsf{A},\mathsf{B},\mathsf{C}\}$, $\{\mathsf{a},\mathsf{b},\mathsf{B}\}$, $\{\mathsf{a},\mathsf{c},\mathsf{C}\}$ of its extended graph as required.

Bistring (5a.upper) is intervallic by conversion to (5d) that has the clique path $(\{\mathsf{x},\mathsf{y},\mathsf{A}\}, \{\mathsf{y},\mathsf{A},\mathsf{B},\mathsf{C}\}, \{\mathsf{y},\mathsf{z},\mathsf{C}\})$. The path decomposes into halves ($\{\mathsf{x},\mathsf{y}\}$, $\{\mathsf{y}\}$,

$\{\mathsf{y},\mathsf{z}\}$) and ($\{\mathsf{A}\},\{\mathsf{A},\mathsf{B},\mathsf{C}\},\{\mathsf{C}\}$) that are encoded, respectively, as bracketed strings $[\mathsf{x}[\mathsf{y}\cdot_{\mathsf{xy}}]\cdot_{\mathsf{y}}[\mathsf{z}\cdot_{\mathsf{yz}}]]$ and $\langle\mathsf{A}\cdot_{\mathsf{A}}\langle\mathsf{B}\langle\mathsf{C}\cdot_{\mathsf{ABC}}\rangle\rangle\cdot_{\mathsf{C}}\rangle$ where the letters occur in linear order without repetition. Each clique is indicated by a dot '$\cdot$' whose subscript is added to illustrate the clique content: the intersecting intervals at that point. Observe that bracketed intervals are cross-serial (FIFO) rather than nested (LIFO).

The halves of the decomposition combine into the full clique path by one-by-one matching. For the encoded halves, this is implemented by an interpretation of Lemma 1: it consumes the first string first but synchronizes the strings at the shared dots. The result is $[\mathsf{x}[\mathsf{y}\langle\mathsf{A}\cdot_{\mathsf{xyA}}]\langle\mathsf{B}\langle\mathsf{C}\cdot_{\mathsf{yABC}}[\mathsf{z}\rangle\rangle\cdot_{\mathsf{C}}\rangle\cdot_{\mathsf{yz}}]]$ Similarly, (5a.lower) can be encoded by joining the encoded halves $[\mathsf{A}\cdot][\mathsf{B}\cdot][\mathsf{C}\cdot]$ and $\langle\mathsf{ab}\cdot\rangle\langle\mathsf{c}\cdot\rangle\langle\mathsf{d}\cdot\rangle$ as the combined string $[\mathsf{A}\langle\mathsf{ab}\cdot_{\mathsf{Aab}}]\rangle[\mathsf{B}\langle\mathsf{c}\cdot_{\mathsf{Bc}}]\rangle[\mathsf{C}\langle\mathsf{d}\cdot_{\mathsf{Cd}}]\rangle$.

Bistring (5c) turns out to be intervallic as it corresponds to graph (5e) having the clique path ($\{\mathsf{x},\mathsf{y},\mathsf{a},\mathsf{b}\}$, $\{\mathsf{y},\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\}$, $\{\mathsf{y},\mathsf{z},\mathsf{d}\}$. Its decomposition is encoded by $[\mathsf{x}[\mathsf{y}\cdot_{\mathsf{xy}}]\cdot_{\mathsf{y}}[\mathsf{z}\cdot_{\mathsf{yz}}]]$ and $\langle\mathsf{ab}\cdot_{\mathsf{ab}}\langle\mathsf{c}\langle\mathsf{d}\cdot_{\mathsf{abcd}}\rangle\rangle\cdot_{\mathsf{d}}\rangle$. The combination of the halves can be encoded as $[\mathsf{x}[\mathsf{y}\langle\mathsf{ab}\cdot_{\mathsf{xyab}}]\langle\mathsf{c}\langle\mathsf{d}\cdot_{\mathsf{yabcd}}[\mathsf{z}\rangle\rangle\cdot_{\mathsf{yzd}}]]\rangle$ by an interpretation of Lemma 1. When we have encoded the clique paths of interval graphs (5d,e), we have also implicitly encoded their largest bipartite subgraphs (5a.upper,c).

Conjecture 1. Path contraction of an intervallic multistring is an intervallic m.s.

The **sharing limit** (abbr. **s.l.**) is a parameter h that determines how many cliques can share each letter in the clique path. For inertial bistrings $h = 2$, but a higher h is needed if the bistring is obtained as a result of a path contraction.

Proposition 3. *Every intervallic bistrings is semi-inertial and has prime decomposition.*

Lemma 4. *There is a bijection between the tree-shaped intervallic bistring primes $\mathcal{P}_{intervallic}$ with a fixed sharing limit h and an infinite regular code X.*

Theorem 6. *The X-recognizable intervallic multistring languages with a s.l. are closed under concatenation, star, union, rel. complement, join and projection.*

Conjecture 2. Path contraction of a recognizable intervallic K-multistring language is effective and it produces a recognizable intervallic K'-multistring language (with a changed shape K' and a possibly increased sharing limit).

4 Conclusion

In this extended abstract and ongoing work, the notion of aligned bistrings (aka autosegmental graphs) is generalized towards *aligned multistrings* and the theory of *multistring languages*. *Recognizable languages of tree-shaped inertial multistrings* and their effective closure under the *join* and *projection* operations are shortly presented. The third new operation, *path contraction* motivates the introduction of *intervallic multistrings*, for which, the existence of a coding morphisms is indicated. Conjectures are presented that the intervallic multistrings and their code recognizable languages are actually closed under path contraction.

References

1. Allauzen, C., Riley, M., Schalkwyk, J.: Filters for efficient composition of weighted finite-state transducers. In: Domaratzki, M., Salomaa, K. (eds.) Implementation and Application of Automata: 15th International Conference, CIAA 2010, Winnipeg, MB, Canada, August 12-15, 2010. Revised Selected Papers. pp. 28–38. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
2. Berstel, J., Perrin, D., Reutenauer, C.: Codes and Automata, Encyclopedia of Mathematics and Its Applications, vol. 129. Cambridge University Press (2010)
3. Christodouloupoulos, C., Steedman, M.: A massively parallel corpus: the Bible in 100 languages. Language Resources and Evaluation 49(2) (2014)
4. Clements, G.N.: The geometry of phonological features. Phonology Yearbook 2, 225–252 (1985)
5. Cohen, J.E.: Food Webs and Niche Space. Princeton University Press, Princeton, New Jersey (1978)
6. Coleman, J., Local, J.: The "no crossing constraint" in Autosegmental Phonology. Linguistics and Philosophy 14, 295–338 (1991)
7. Fishburn, P.C.: Interval graphs and interval orders. Discrete Mathematics 55, 135–149 (1985)
8. Hulden, M.: Grammar design with multi-tape automata and composition. In: Hanneforth, T., Wurm, C. (eds.) Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing 2015 (FSMNLP 2015 Düsseldorf) (2015)
9. Jardine, A., Heinz, J.: A concatenation operation to derive autosegmental graphs. In: Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015). pp. 139–151. Association for Computational Linguistics, Chicago, USA (July 2015)
10. Kaplan, R., Kay, M.: Regular models of phonological rule systems. Computational Linguistics 20, 331–378 (1994)
11. Kornai, A.: Formal Phonology. Garland Publishing, New York (1995)
12. Lekkerkerker, C.G., Boland, J.C.: Representation of a finite graph by a set of intervals on the real line. Fundamenta Mathematicae 51, 45–64 (1962)
13. Marcus, S.: Algebraic Linguistics; Analytical Models. Academic Press, New York (1967)
14. Marcus, S.: Words and Languages Everywhere. Polimetrica, New York (2007)
15. McCarthy, J.: A prosodic theory of non-concatenative morphology. Linguistic Inquiry pp. 373–418 (1981)
16. Yli-Jyrä, A.: Multiplanarity – a model for dependency structures in treebanks. In: TLT 2003, Proceedings of the Second Workshop on Treebanks and Linguistic Theories. pp. 189–200 (2003)
17. Yli-Jyrä, A.: An efficient constraint grammar parser based on inward deterministic automata. In: Proceedings of the NODALIDA 2011 Workshop Constraint Grammar Applications. NEALT Proceedings Series, vol. 14, pp. 50–60 (2011)
18. Yli-Jyrä, A.: Three equivalent codes for autosegmental representations. In: Hanneforth, T., Wurm, C. (eds.) Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing 2015 (FSMNLP 2015 Düsseldorf). Düsseldorf (2015)
19. Yli-Jyrä, A.: Prime factorization for a subclass of well-formed bistrings. Manuscript (2016)
20. Yli-Jyrä, A.: Transition systems for well-formed bistrings. Manuscript. To appear in a collection (2016)