Z-automata for Compact and Direct Representation of Unranked Tree Languages

Johanna Björklund[1], Frank Drewes[1], and Giorgio Satta[2]

[1] Dept. Computing Science, Umeå University
{johanna,drewes}@cs.umu.se
[2] Department of Information Engineering, University of Padua
satta@dei.unipd.it

Abstract. Unranked tree languages are valuable for modelling structured objects such as XML documents, database entries, and dependency trees. We introduce a new type of automaton for unranked tree languages, called Z-automaton. The model is closely related to stepwise tree automata, thus offering a compact form of representation, but it avoids obfuscating encoding schemes. We discuss alternative semantics and normal forms, and finally prove the membership problem to be in $O(mn)$, where m is the size of the transition table, an n is the size of the input tree.

1 Introduction

Unranked tree languages (UTLs) have been studied since the 60s, most notably as a formal model for the document markup language XML [1, 6]. The current work is motivated by the use of UTLs as a representation for *dependency trees* in natural language processing [8]. A dependency tree for a sentence is, simply put, an arrangement of the lexical items of the sentence into a hierarchy of heads and their dependents. Our long-term objective is to provide transducers that translate dependency syntactic trees into graph-based structures, representing the semantics of the input, in which nodes encode objects and edges encode relations. In this paper, we take a first step by introducing a new type of automaton to capture the domain language of well-formed dependency trees.

Computer Science literature contains a number of formalisms for representing unranked tree languages. The best known is probably the unranked tree automaton (UTA) of Brüggemann-Klein, Murata, and Wood [1]. These UTA use regular string languages to express the left-hand sides of transitions rules. Depending on what device is chosen to represent these string languages, different types of UTA arise. Natural choices include regular expressions and nondeterministic or deterministic finite state automata. However, as shown in [11], none of these yields unique minimal UTA in the deterministic case. Moreover, even when the string languages are represented by DFAs, the minimization problem is NP-complete. Martens and Niehren therefore proposed *stepwise tree automata* (STA) which

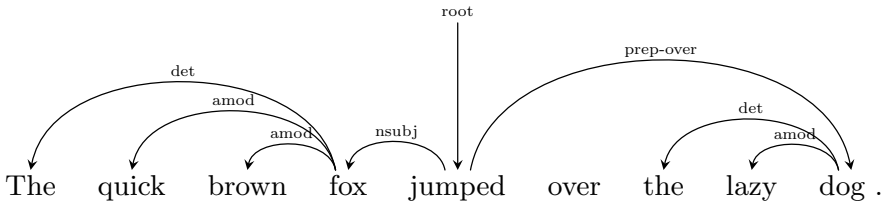

Fig. 1. The dependency analysis assigned by the Stanford parser to the sentence 'The quick brown fox jumped over the lazy dog.'

process binarized encodings of unranked tree automata. Bottom-up (bu-) deterministic stepwise automata have the advantage of having a canonical form, and being exponentially more succinct than bu-deterministic tree automata over the first-child next-sibling encoding.

There are also logic formalisms for (weighted) unranked tree languages. In [5], Droste and Vogler provide a weighted monadic second order logic for unranked trees and introduce the notion of weighted UTA. Again, the theories of ranked and unranked tree languages differ, this time in that weighted UTA and a syntactically restricted weighted MSO-logic for unranked trees have the same expressive power in case the semiring is commutative, but not in general.

In its canonical computations, the Z-automaton proposed here visits the input tree bottom-up in an order resembling a zed-shaped motion, alternating horizontal moves and vertical moves. The Z-automaton is as expressive as UTA and STA. It combines the best aspects of both: The representation is arguably as natural as UTA, and as compact as STA. Furthermore, Z-automata use transitions whose left-hand sides may refer to components at differing tree depths. Similarly to the tree transducers developed in [9], this feature allows for a direct implementation of so-called local rotations when Z-automata are used as a basis for the development of transducers for unranked tree languages.

2 Preliminaries

General Notation. The set of natural numbers (including zero) is denoted by $\mathbb{N}$, and $\mathbb{N}_+ = \mathbb{N}\backslash\{0\}$. For $n \in \mathbb{N}$ the set $\{1, \ldots, n\}$ is abbreviated to $[n]$. In particular, $[0] = \emptyset$. The set of all finite sequences of elements of a set S is written S^*, ε is the empty sequence, $S^+ = S^* \setminus \{\varepsilon\}$, and 2^S is the powerset of S. Given a sequence w, we write $[w]$ for the set of its elements, i.e., the smallest set S such that $w \in S^*$. Given a string s and a set of strings S, we denote by $s \cdot S$ the set of strings $\{ss' \mid s' \in S\}$.

Trees. Let Σ be an alphabet. We define the set T_Σ of *(unranked) trees* over Σ as usual. It is the smallest set such that, for all $f \in \Sigma$ and $t_1, \ldots, t_n \in \mathrm{T}_\Sigma$

($n \in \mathbb{N}$), we have $f(t_1, \ldots, t_n) \in \mathrm{T}_\Sigma$. In particular $f()$, which we abbreviate by f, is in T_Σ. (This is the base case of the inductive definition.)

A *ranked alphabet* Σ is an alphabet additionally equipped with a function $\#\colon \Sigma \to \mathbb{N}$. For $f \in \Sigma$, the value $\#(f)$ is called the rank of f. For any $n \geq 0$, we denote by Σ_n the set of all symbols of rank n from Σ. If Σ is ranked, then T_Σ is restricted so that $f(t_1, \ldots, t_n) \in \mathrm{T}_\Sigma$ only if $n = \#(f)$. Thus, in this case T_Σ becomes a set of *ranked trees*.

In both cases, we speak of the *nodes* of a tree in the usual way, and identify them by their Gorn addresses, which are strings in $\mathbb{N}_+^*$: the root has the address ε, and if α is the address of a node in t_i then $i\alpha$ is the address of that node in $f(t_1, \ldots, t_n)$. The label of node α in t is denoted by $t(\alpha)$, and the set of all nodes of t is $N(t)$. For $\Sigma' \subseteq \Sigma$, the set of all nodes $\alpha \in N(t)$ with $t(\alpha) \in \Sigma'$ is denoted by $N_{\Sigma'}(t)$. A node $\alpha \in N(t)$ is a *leaf* if $\alpha 1 \notin N(t)$, and is *internal* otherwise. The *size* of t is $|t| = |N(t)|$.

We denote a subset $\{\alpha_1, \ldots, \alpha_k\}$ of the set of nodes of a tree t as $(\alpha_1, \ldots, \alpha_k)$ if we wish to indicate that $\alpha_1, \ldots, \alpha_k$ are listed in lexicographic order.

Let $\Box$ be a special symbol never used as an ordinary symbol. A *context* is a tree $c \in \mathrm{T}_{\Sigma \cup \{\Box\}}$ such that c contains exactly one occurrence of $\Box$, and this occurrence is a leaf. Given such a context and a tree t, we let $c[t]$ denote the tree obtained from c by replacing $\Box$ with t. Formally, $c[t] = t$ if $c = \Box$, and otherwise $c[t] = f(s_1, \ldots, s_{i-1}, s_i[t], s_{i+1}, \ldots, s_n)$, where $c = f(s_1, \ldots, s_n)$ and $s_i \in \mathrm{T}_{\Sigma \cup \{\Box\}}$ is the context among $s_1, \ldots, s_n$. For contexts $c \neq \Box$, the notation $c[t]$ is extended in the obvious way to $c[t_1, \ldots, t_k]$ for trees $t_1, \ldots, t_k$ ($k \in \mathbb{N}$). It yields the tree obtained by inserting the sequence of subtrees $t_1, \ldots, t_k$ at the position marked by $\Box$. (Note that this yields a tree, since we only use it if $c \neq \Box$.) To be precise, if $c = f(s_1, \ldots, s_n)$ and $i \in [n]$ is the index such that $\Box$ occurs in s_i, then

$$c[t_1, \ldots, t_k] = \begin{cases} f(s_1, \ldots, s_{i-1}, t_1, \ldots, t_k, s_{i+1}, \ldots, s_n) & \text{if } s_i = \Box \\ f(s_1, \ldots, s_{i-1}, s_i[t_1, \ldots, t_n], s_{i+1}, \ldots, s_n) & \text{otherwise.} \end{cases}$$

Ranked tree automata A *ranked bottom-up tree automaton* (TA) is a tuple $A = (Q, \Sigma, R, F)$ where

- Q is a finite set of states,
- Σ is a ranked input alphabet,
- R is a finite set of transition rules, and
- $F \subseteq Q$ is a set of accepting (final) states.

Each transition rule is a triple of the form $f(q_1, \ldots, q_n) \to q$ where $q_1, \ldots, q_n, q \in Q$, $f \in \Sigma$, and $\#(f) = n$. The TA is *determinstic* if all distinct rules have distinct left-hand sides.

Let $t \in \mathrm{T}_{\Sigma \cup Q}$. A transition $f[q_1, \ldots, q_n) \to q$ is *applicable* to t, if t can be written as $t = c[f(q_1, \ldots, q_n)]$. If so, then there is a computation step $t \to_A \overline{t} = c[q]$. A tree $t \in \mathrm{T}_\Sigma$ is *accepted* by A if there is a sequence of computation steps $t \to_A^* q$, for some $q \in F$. The *language* accepted by A, denoted $\mathcal{L}(A)$, is the set of all trees in T_Σ that A accepts.

3 Z-automata

A *Z-automaton* is a quadruple $A = (\Sigma, Q, R, F)$ consisting of

- a finite input alphabet Σ;
- a finite set Q of *states* which is disjoint with Σ;
- a finite set R of *transitions*, each of the form $s \to r$ consisting of a left-hand side $s \in \mathrm{T}_{\Sigma \cup Q}$ and a right-hand side $q \in Q$;
- a finite set $F \subseteq Q$ of accepting states.

Let $t \in \mathrm{T}_{\Sigma \cup Q}$. A transition $s \to q$ is *applicable* to t, if t can be written as $t = c[f(t_1, \ldots, t_n)]$, such that $s = f(t_1, \ldots, t_k)$ for some $k \leq n$. If so, then there is a computation step $t \to_A \overline{t} = c[q(t_{k+1}, \ldots, t_n)]$. A tree $t \in \mathrm{T}_\Sigma$ is *accepted* by A if there is a sequence of computation steps $t \to_A^* q$, for some $q \in F$. The *language* accepted by A, denoted $\mathcal{L}(A)$, is the set of all trees in T_Σ that A accepts. The automaton A is *deterministic* if there does not exist distinct transitions $s \to q$ and $t \to p$ in R such that $s \to q$ is applicable to t.

Example 1. Consider the Z-automaton $A = (Q, \Sigma, R, f)$, where $\Sigma = \{a, b, c\}$, $Q = \bigcup_{x \in \Sigma} \{p_x, q_x, r_x\}$, $F = \{p_x \mid x \in \Sigma\}$, and R is given as followed, where x and y range over Σ:

$$\begin{array}{llll} x(y) & \to q_x & q_x(y) & \to q_x \quad q_x(x) \to r_x \\ x(p_y) & \to q_x & q_x(p_y) & \to q_x \quad q_x(p_x) \to r_x \\ q_x(y) & \to q_x & q_x(x) & \to r_x \quad r_x(y) \to p_x \ . \end{array}$$

The language accepted by A is the set of all unranked trees over the alphabet Σ, in which the second-to-last child of every internal node carries the same label as the node itself.

This can be seen as follows. The first line of the table lets us process a node α labelled $x \in \Sigma$ by nondeterministically guessing if it is a leaf. If it is, the automaton should assign it the state p_x indicating that the subtree below it is valid. Otherwise, it should assign the state q_x to it. If the automaton guessed wrong, then it will not be able to complete the processing of the subtree rooted at α, because R contains no transitions with a left-hand side of the form $p(q)$, where $p = p_x$ or $q = q_x$ for an $x \in \Sigma$.

The rules in the second line of the table verify that the second-to-last child of a node labelled x was also labelled x (and was thus eventually assigned the state p_x). To accomplish this, the automaton advances over its input while looping on the same state q_x, until it nondeterministically decides guesses that it has reached the child p_x which is the second to last. It then turns q_x into the intermediate state r_x and to p_x in the next step. Again, if there should still be children of n left after that, then the computation will not be able to proceed and the input tree is rejected.

Figure 2 shows an accepting computation of the automaton A on an input tree. The automaton starts by processing the right-most subtree, guessing correctly which nodes are leaves, and which are internal. The states labelling an internal node α, take the role of an interal state in a string automaton processing

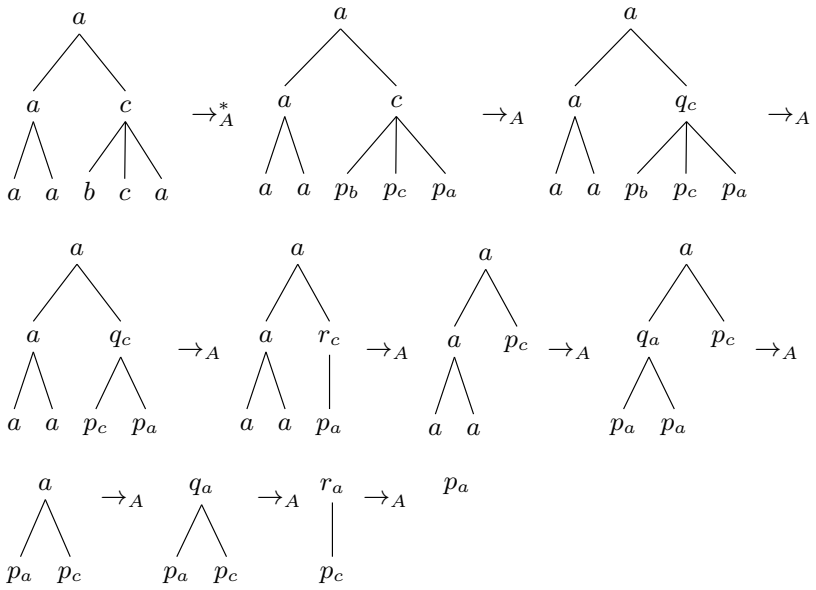

Fig. 2. A sample computation of the Z-automaton A of Example 1. The automaton accepts all unranked trees over the alphabet $\{a, b, c\}$, in which every internal node has the same label as its second-to-last child.

the children of α. As this subtree is well-formed with respect to $\mathcal{L}(A)$ and its root is labelled by c, it is mapped to p_c. At this point, the computation must also process the left-most subtree, before it can continue upwards. As also this subtree is well-formed and its root is labelled a, it is mapped to the state p_a. In the final steps of the computation, the automaton repeats the same type of verification at the highest level in the tree, and as the labels of the root of the entire tree and the left-hand subtree agree, the whole tree is accepted.

An alternative set of transitions exploits the fact that we can turn individual input symbols into states:

$$\begin{array}{llll} x & \to p_x & x & \to q_x \\ q_x(p_y) & \to q_x & q_x(p_x) \to r_x & r_x(p_y) \to p_x \ . \end{array}$$

These transitions are in fact in a particular normal form, that simplifies many of the upcoming arguments.

Definition 1 (Arc-factored normal form). *Let $A = (\Sigma, Q, R, F)$ be a Z-automaton. A transition is in* arc-factored normal form *if its left-hand side is in $\Sigma \cup Q(Q)$, and A is in arc-factored normal form if every transition in R is in arc-factored normal form.*

We shall now show that every Z-automaton can indeed be transformed into arc-factored normal form. As can be expected, this makes a larger number of

states and transitions necessary to recognize the same language. To make this precise, let us say that the *size* of a transition r is the size of its left-hand side, denoted by $|r|$. The size of A is $|A| = \sum_{r \in R} |r|$.

Theorem 1. *Every Z-automaton $A = (\Sigma, Q, R, F)$ can effectively be transformed into a Z-automaton B in arc-factored normal form such that $\mathcal{L}(B) = \mathcal{L}(A)$ and $|B| \leq 4|A|$.*

Proof. We transform A in three steps:

In the first step, we remove rules of the form $p \to q$ with $p, q \in Q$. We do this by the usual procedure: simply replace R by the set of all rules $t \to q'$ such that $t \to q$ is in R, $t \notin Q$, and $q \to_A^* q'$. Clearly, this does not affect the set of trees accepted by A.

In the second step, we add a transition $f \to q_f$ for every symbol $f \in \Sigma$, where q_f is a fresh state added to Q. Furthermore, we replace f by q_f in the left-hand side of every ordinary transition in R of size larger than 1. Clearly, this does not affect the language recognized by A. Of course, the introduction of new states and transitions can be restricted to those symbols which actually occur in a left-hand side, which means that at most $|A|$ transitions are added.

The third and final step is slightly more technical, and easiest to describe in an iterative manner. However, the intuition is rather straightforward: instead of consuming an entire left-hand side $s \in \mathrm{T}_Q$ in one step, the arcs are consumed one by one, using auxiliary states.

Formally, as long as A is not in arc-factored normal form, select any transition $s \to q$ such that $|s| > 2$. Then s has the form $c[q_1(q_2, t_1, \ldots, t_n)]$ for some context c, states q_1, q_2, and trees $t_1, \ldots, t_n$ ($n \geq 0$). We decompose the transition into one that consumes $q_1(q_2)$, resulting in a new state $q_{1;2}$, and one that consumes $c[q_{1;2}(t_1, \ldots, t_n)]$. Thus, the first of these transitions is in arc-factored normal form and the second is of size one less than the original transition. Let n be the type of q_1 and α its address in s (i.e., α is the address of $\Box$ in c). Then the first transition is $q_1(q_2) \to q_{1;2}$. The second transition is $c[q_{1;2}(t_1, \ldots, t_n)] \to q$.

It should be clear that this procedure of splitting the original transition $s \to q$ into two does not change the language recognized by A. Moreover, $\sum_{r \in R,\, |r|>2} |r|$ is reduced by one each time a transition is split in this way. Thus, the process will terminate, resulting in a Z-automaton which is in arc-factored normal form and of size at most $4|A|$. □

Once the automaton is in arc-factored normal form, we can use the standard powerset construction to make it deterministic.

Lemma 1. *There is an algorithm that turns a Z-automaton A into a deterministic Z-automaton B in arc-factored normal form such that $\mathcal{L}(A) = \mathcal{L}(B)$.*

Proof (sketch). Let $A = (\Sigma, Q, R, F)$ be a Z-automaton. Without loss of generality, we may assume that A is in arc-factored normal form. For $t \in \Sigma \cup Q(Q)$, let $R(t) = \{q \in Q \mid (t \to q) \in R\}$. We let $B = (\Sigma, 2^Q, R_1 \cup R_2, F')$, where

$$R_1 = \bigcup_{f \in \Sigma} \{f \to R(f)\} \ ,$$

$$R_2 = \bigcup_{P_1,P_2\subseteq Q} \{P_1(P_2) \to \bigcup_{p_1\in P_1,p_2\in P_2} R(p_1(p_2))\} \ ,$$

and $F' = \{P \subseteq Q \mid P \cap F \neq \emptyset\}$. Clearly, B is in arc-factored normal form, and hence also deterministic because rules in arc-factored normal form violate the determinism requirement only if their left-hand sides are equal.

It is furthermore straightforward to verify that A and B are language equivalent, which completes the proof sketch. □

Naturally, the determinisation according to the previews lemma may take exponential time, simply because the size of the output Z-automaton B has exponentially many states.

4 Equivalence to Stepwise Tree Automata

As we shall see, Z-automata accept the same family of unranked regular tree languages as unranked tree automata [2] and stepwise automata [3]. The latter is syntactically similar to the Z-automaton, but operates on binary encodings of the input tree. This encoding makes use of an auxiliary symbol @, used as a binary operator over trees, that extends the tree t_1 in its first argument by adding the tree t_2 in its second argument as the right-most direct child of the root of t_1. A sample encoding is shown in Figure 3.

Definition 2 (Binary encoding). *Given an (unranked) alphabet Σ, we denote by $\Sigma_@$ the alphabet $\Sigma \cup \{@\}$, viewed as a ranked alphabet in which the symbols in Σ are taken to have rank 0, and the symbol @ to have rank 2. For every tree $t = f(t_1, \ldots, t_n) \in \mathrm{T}_\Sigma$, the function $tree_@(t) : \mathrm{T}_\Sigma \to \mathrm{T}_{\Sigma_@}$ is given by*

$$tree_@(t) = \begin{cases} f & \text{if } n = 0 \\ @(tree_@(f(t_1, \ldots, t_{n-1})), tree_@(t_n)) & \text{otherwise} \ . \end{cases}$$

We extend $tree_@$ to tree languages: for all $L \subseteq \mathrm{T}_\Sigma$, $tree_@(L) = \{tree_@(t) \mid t \in L\}$.

It is not difficult to check that $tree_@$ is a bijection between T_Σ and $\mathrm{T}_{\Sigma_@}$.

A stepwise automaton is simply a ranked bottom-up tree automaton that inputs binary encodings of unranked trees.

Definition 3 (Stepwise automaton [4]). *A* stepwise tree automaton *(STA) $M = (Q, \Sigma, R, F)$ is a ranked bottom-up tree automaton over $\Sigma_@$. Consequently, every transition in R is of the form $a \to q$ or $@(p, p') \to q$ where $a \in \Sigma$ is a symbol of rank 0, and $q, p, p' \in Q$.*

As we shall see, each Z-automaton A can be translated into an STA that recognizes $tree_@(\mathcal{L}(A))$.

Definition 4 (Related automata). *A Z-automaton $A = (\Sigma, Q, R, F)$ in arc-factored normal form and an STA $M_A = (Q, \Sigma_@, P, Q)$, are* related *if $P = P' \cup P''$ with*

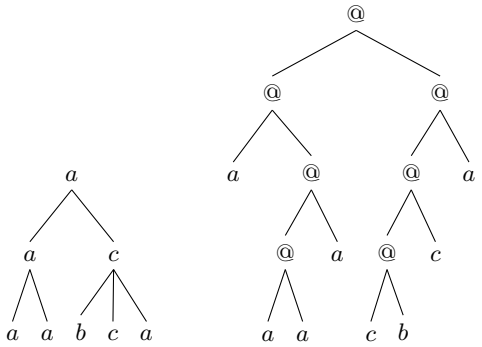

Fig. 3. The input tree t over the alphabet $\{a, b, c\}$ used in Example 1 and its binary encoding $tree_{@}(t)$.

- $P' = \{a \to q \mid a \to q \in R\}$,
- $P'' = \{@(p, p') \to q \mid p(p') \to q \in R\}$.

Let us ensure ourselves that relatedness preserves the accepted language. Given a Z-automaton or STA A with state set Q, and a tree t, we let $eval(A, t) = \{q \in Q \mid t \to_A^* q\}$ denote the set of states reached by A on input t.

Theorem 2. *For every Z-automaton $A = (\Sigma, Q, R, F)$ in arc-factored normal form and its related STA M_A, it holds that $\mathcal{L}(M_A) = tree_{@}(\mathcal{L}(A))$, and M_A is deterministic if and only if A is.*

Proof. Clearly, M_A is deterministic if and only if A is. To prove that $\mathcal{L}(M_A) = tree_{@}(\mathcal{L}(A))$, we show that $eval(M_A, tree_{@}(t)) = eval(A, t)$ for every tree $t \in \mathrm{T}_\Sigma$. We prove this by induction on the structure of the trees, as constructed with the extension operator (as opposed to classical top-concatenation). Let $t = f(t_1, \ldots, t_n) \in \mathrm{T}_\Sigma$.

If $q \in eval(A, t)$, then we have the following cases:

- If $n = 0$, then there is a transition $f \to q \in R$, and by construction, $f \to q \in P' \subseteq P$ so $q \in eval(M_A, tree_{@}(t))$.
- If $n = 1$, then there is a $p \in Q$ and a $p' \in eval(A, t_1)$ such that $f \to p$ and $p(p') \to q \in R$. By the induction hypothesis, $eval(M_A, tree_{@}(t_1)) = eval(A, t_1)$ and by construction, $f \to p \in P'$ and $@(p, p') \to q \in P''$, so $q \in eval(M_A, tree_{@}(t))$.
- If $n > 1$, then there is a $p \in eval(A, f(t_1, \ldots, t_{n-1}))$ and a $p' \in eval(A, t_n)$ such that $p(p') \to q \in R$. By the induction hypothesis,

$$eval(M_A, tree_{@}(f(t_1, \ldots, t_{n-1}))) = eval(A, f(t_1, \ldots, t_{n-1})) \quad ,$$

and $eval(M_A, tree_{@}(t_n)) = eval(A, t_n)$, and by construction $@(p, p') \to q \in P''$, so again $q \in eval(M_A, tree_{@}(t))$.

For the other inclusion, if $q \in \mathit{eval}(M_A, \mathit{tree}_@(t))$, then we similarly have the following three cases:

- If $n = 0$, then there is a transition $f \to q \in P'$, and by construction, a transition $f \to q \in R$, so $q \in \mathit{eval}(A, t)$.
- If $n = 1$, then there is be a $p \in Q$ and $p' \in \mathit{eval}(M_A, \mathit{tree}_@(t_1))$ such that $f \to p \in P'$ and $@[p, p'] \to q \in P''$. By the induction hypothesis, $\mathit{eval}(A, t_1) = \mathit{eval}(M_A, \mathit{tree}_@(t_1))$ and by construction, $f \to p$, and $p(p') \to q \in R$, so $q \in \mathit{eval}(A, t)$.
- If $n > 1$, then there is be a $p \in \mathit{eval}(M_A, \mathit{tree}_@(f(t_1, \ldots, t_{n-1})))$ and a $p' \in \mathit{eval}(M_A, \mathit{tree}_@(t_n))$ such that $@(p, p') \to q \in P$. By the induction hypothesis,

  $$\mathit{eval}(A, f(t_1, \ldots, t_{n-1})) = \mathit{eval}(M_A, \mathit{tree}_@(f(t_1, \ldots, t_{n-1}))) \ ,$$

  and $\mathit{eval}(A, t_n) = \mathit{eval}(M_A, \mathit{tree}_@(t_n))$, and by construction $p(p') \to q \in R$, yielding again $q \in \mathit{eval}(A, t)$. □

This bridge between Z-automata and STA has several immediate implications, summarised in Corollaries 1 and 2.

Corollary 1. *For an unranked tree language L, the following are equivalent:*

1. *L is recognisable by a Z-automaton.*
2. *L is recognisable by a UTA.*
3. *$\mathit{tree}_@(L)$ is recognisable by an STA.*

Corollary 2. *The following properties hold for Z-automata:*

1. *The family of tree languages accepted by Z-automata is closed under union, intersection, and complement.*
2. *Z-automata are an equally succinct representation of unranked tree languages as STA, and exponentially more succinct than UTA [10].*

Theorem 3. *For every Z-automaton A, there is a unique minimal deterministic Z-automaton B in arc-factored normal form such that $\mathcal{L}(A) = \mathcal{L}(B)$. Furthermore, B can be computed in time $O(m \log n)$, where m is the size of the transition table, and n is the number of states of the input automaton.*

Proof. Let $A = (\Sigma, Q, R, F)$ be a Z-automaton, which we may, without loss of generality, assume to be in arc-factored normal form. Let M_A be the STA related to A. We note that both can be viewed as ordered labelled hypergraphs with node set Q and hyperedge set R. A hyperedge corresponding to $r = (a \to q)$ is labelled with a and incident with q. A hyperedge corresponding to $r = (@(p, p') \to q)$ or $r = (p(p') \to q)$ is labelled with $@$ and incident with $pp'q$. Obviously, the hypergraph representations of A and A_M are isomorphic.

Let B be the Z-automaton A_N, where N is the result of minimizing the stepwise automaton A_M using the forward-bisimulation algorithm of [7], which coincides with standard minimization on deterministic tree automata. For binary

trees, the time complexity of this algorithm is $O(m \log n)$. Since relatedness preserves the recognized language, the number of states, and the property of being deterministic, we have that B is a deterministic Z-automaton with the same number of states as A_N, and such that $\mathcal{L}(B) = \mathcal{L}(N) = \mathcal{L}(M_A) = \mathcal{L}(A)$.

Suppose that there is another deterministic Z-automaton C with strictly fewer states than B recognizing the same language. In this case, the deterministic stepwise automaton M_C would be language equivalent to N, but have fewer states. This is not possible by the correctness of the minimization algorithm in [7]. Hence, B has a minimal number of states.

Finally assume that there is another determinstic Z-automaton C with the same number of states as B that is, when viewed as a hypergraph, not isopmorphic to B. Then B_M and C_M are not isomorphic to each other, as they are isomorphic to B and C, respectively. This contradicts the uniquness of the minimal deterministic STA2.

Hence, B is the unique minimal deterministic Z-automaton. □

5 Left-to-Right Bottom-Up Derivations

The Z-automata semantics given in Section 3 allows transitions to be applied whereever they apply in an intermediate tree t. As we shall see, we can restrict applications to the two lowest nodes along the left-most path of t, without loss of expressive power.

Definition 5 (lrbu derivation). *Let $A = (\Sigma, Q, R, F)$ be a Z-automaton in arc-factored normal form, and let $s \in \mathrm{T}_\Sigma$. A computation is* left-to-right-bottom-up *(lrbu) if, in every step $t \to_A t'$ of the computation, one of the following holds. If α is the left-most leaf of t, then either*

1. *$t(\alpha) \in \Sigma$ and the rule is applied at node α or*
2. *$t(\alpha) \in Q$ and the rule is applied at the parent of α.*

We let $\mathcal{L}_{lrbu}(A) = \{t \in \mathrm{T}_\Sigma \mid t \to_A^* q$ by an lrbu computation for some $q \in F\}$.

Note that, in the second case above, the rule applied to the parent of α can either be of the form $a \to q$ or of the form $p(p') \to q$.

Theorem 4. *For every Z-automaton A in arc-factored normal form, $\mathcal{L}(A) = \mathcal{L}_{lrbu}(A)$.*

Proof. Let $A = (\Sigma, Q, R, F)$ be a Z-automaton in arc-factored normal form, and let $t \in \mathrm{T}_\Sigma$. We show that A has an accepting computation on t if and only if it has an accepting lrbu computation on t. The 'if' direction is trivial, since every lrbu computation is a regular computation. The 'only if' direction is proved by induction on the structure of t. We argue that every computation of A ending in a state $q \in Q$ can be turned into an lrbu computation ending in the same state.

If $|t| = 1$ then this is trivially true because the computation already is lrbu. If $|t| \geq 2$, then t can be obtained from two trees $s, s' \in \mathrm{T}_\Sigma$ by adding s' as the

rightmost child of the root of s. Since t is accepted by A, there is an accepting computation π of A on t. This computation, restricted to the rule applications at nodes in s and s' in the obvious way, yields subcomputations $s \to_A^* p$ and $s' \to_A^* p'$, for some $p, p' \in Q$. Since the computation accepts t, its last step is of the form $p(p') \to_A q$ for a state $q \in F$. By the induction hypothesis, there are lrbu computations μ and μ' on s and s', respectively, that accomplish the same. We can thus construct an accepting lrbu computation on t by first applying μ to t, then μ' to the subtree s' of t, and finally the transition $p(p') \to q$ to the remaing tree $p(p')$. □

From Theorem 4, Theorem 5 follows:

Theorem 5. *The membership problem for Z-automata is in $O(mn)$, where m is the number of rules and n is the size of the input tree.*

Proof (Sketch). Let $A = (\Sigma, Q, R, F)$ be a Z-automaton. By Theorem , we can assume that A is in arc-factored normal form. To decide whether an input tree $t \in \mathrm{T}_\Sigma$ is in $\mathcal{L}(A)$, the automaton is applied to t by performing lrbu computations on t. At every step, it uses on-the-fly subset construction to check what transitions in R are applicable at the last two nodes of the leftmost path of the intermediate tree. The time needed to consume one edge of t is thus in $O(|R|)$, and there are $|t| = n$ edges to consume. □

6 Conclusion

We have introduced a new type of automaton for unranked trees, the Z-automaton, and shown it to be equivalent to the UTA and STA. Z-automata offer a more compact form of representation than UTA, and avoid the binary encoding used by STA. We have also provided a normal form and a standard left-to-right bottom-up (lrbu) mode of derivations that although syntactically more restrictive, retain the expressive power of the original device.

Given the close relation between Z-automata and STA, we expect the majority of the results pertaining to the latter to carry over. However, in some situations, the time and space complexities may be affected, however. A further investigation in this direction is left for future work, as is the study of logical characterizations.

References

1. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1. Technical Report HKUST-TCSC-2001-0, 2001.
2. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1. Technical Report Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.

3. Julien Carme, Joachim Niehren, and Marc Tommasi. Querying unranked trees with stepwise tree automata. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications*, pages 105–118, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
4. Julien Carme, Joachim Niehren, and Marc Tommasi. Querying Unranked Trees with Stepwise Tree Automata. In Vincent van Oostrom, editor, *19th International Conference on Rewriting Techniques and Applications*, volume 3091 of *LNCS*, pages 105–118, Aachen, Georgia, 2004. Springer.
5. Manfred Droste and Heiko Vogler. Weighted logics for unranked tree automata. *Theory of Computing Systems*, 48(1):23–47, Jan 2011.
6. Ferenc Gécseg and Magnus Steinby. Tree automata. 1984.
7. Johanna Högberg, Andreas Maletti, and Jonathan May. Backward and forward bisimulation minimization of tree automata. *Theoretical Computer Science*, 410(37):3539 – 3552, 2009. Implementation and Application of Automata (CIAA 2007).
8. Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*. Morgan and Claypool Publishers, 2009.
9. A. Maletti, J. Graehl, M. Hopkins, and K. Knight. The power of extended top-down tree transducers. *SIAM Journal on Computing*, 39(2):410–430, 2009.
10. Wim Martens and Joachim Niehren. Minimizing tree automata for unranked trees. In Gavin Bierman and Christoph Koch, editors, *Database Programming Languages*, pages 232–246, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
11. Wim Martens and Joachim Niehren. On the minimization of xml schemas and tree automata for unranked trees. *Journal of Computer and System Sciences*, 73(4):550 – 583, 2007. Special Issue: Database Theory 2005.