Random Generation of Deterministic Tree (Walking) Automata

Pierre-Cyrille Héam, Cyril Nicaud, Sylvain Schmitz

Random Generation of Deterministic Tree (Walking) Automata*

Pierre-Cyrille Héam[1,3] Cyril Nicaud[2] Sylvain Schmitz[3]

[1]LIFC, Université de Franche-Comté & INRIA, Besançon, France
[2]LIGM, Université Paris Est & CNRS, Marne-la-Vallée, France
[3]LSV, ENS Cachan & CNRS & INRIA, Cachan, France

Abstract

Uniform random generators deliver a simple empirical means to estimate the average complexity of an algorithm. We present a general rejection algorithm that generates sequential letter-to-letter transducers up to isomorphism. We tailor this general scheme to randomly generate deterministic tree walking automata and deterministic top-down tree automata. We apply our implementation of the generator to the estimation of the average complexity of a deterministic tree walking automata to nondeterministic top-down tree automata construction we also implemented.

1 Introduction

The widespread use of automata as primitive bricks in computer science motivates an ever renewed search for efficient algorithms taking automata as input (see for some recent examples [16, 15, 12]). Developing new algorithms and heuristics raises crucial evaluation issues, as improved worst-case complexity upper-bounds do not always transcribe into clear practical gains [4].

A suite for software performance evaluation can usually gather three types of entries:[1]

1. benchmarks, i.e. large sets of typical samples, which can be prohibitively difficult to collect, and thus only exist for a few general problems,
2. hard instances, that provide good estimations of the worst case behaviour, but are not always relevant for average case evaluations,
3. random inputs, that deliver average complexity estimations, for which the catch resides in obtaining a meaningful random distribution (for instance

*Published in Maneth, S., editor, *CIAA'09*, volume 5642 of *Lecture Notes in Computer Science*, pages 115–124. Springer, 2009. doi:10.1007/978-3-642-02979-0_15. This work was supported in part by ANR GAMMA - project BLAN07-2_195422, ANR RAVAJ - project SETIN-2006, and ANR AVeriSS.

[1]All of the three types are used in SAT-solver competitions like `http://www.satcompetition.org/`.

a uniform random distribution). As the mathematical computation of the average complexity of an algorithm is an intricate task that cannot be undertaken in general, random inputs can prove themselves invaluable for its empirical estimation.

This paper is dedicated to the random generation of deterministic top-down tree automata and of deterministic tree-walking automata. Tree automata have witnessed a recent surge of interest in connection with XML applications [11, 10], fostering a wealth of theoretical results (e.g. [9, 5, 14]). This paper makes the following contributions:

- Section 2 proposes a generic rejection algorithm for uniformly generating sequential letter-to-letter transducers. Thanks to the structural properties of these transducers, the algorithm can be used for the generation of various kinds of finite automata.
- We apply this algorithm in Sect. 3 to the generation of deterministic tree walking automata. The approach was implemented, and we provide in Sect. 3.3 an empirical estimation of the average size of the nondeterministic top-down tree automaton equivalent to a given deterministic tree walking automaton.
- Section 4 presents a bijection between a class of letter-to-letter transducers and deterministic top-down tree automata, providing a uniform random generator for this class of tree automata.

Our approach consists in reducing the problem to the uniform random generation of deterministic word automata, as developed by Bassino et al. [1, 3].

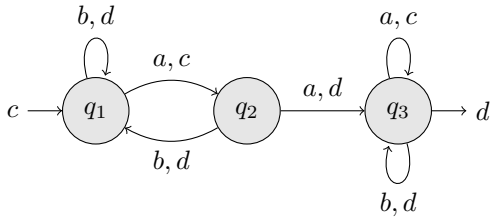
Related Work In the case of deterministic accessible word automata, two main approaches to the random generation with uniform distribution on complete automata stand out: one based on a recursive decomposition [6] and one using Boltzmann samplers [1]. The latter algorithm has been extended to possibly incomplete automata by Bassino et al. [3]. An implementation of these algorithms is available in the C++ package `REGAL` [2].[2]

The random generation of non deterministic finite word automata is still mostly open. Two recent papers propose such random generation algorithms: Tabakov and Vardi [13] apply theirs to the evaluation of inclusion testing procedures, whereas Chen et al. [7] evaluate the performance of a learning algorithm. Both algorithms are *ad hoc* and fail to provide statistically exploitable distributions.

Notations If i and j are positive integers, we denote by $[i, j]$ the set of integers k such that $i \leq k$ and $k \leq j$. If K is a set, $\mathcal{P}(K)$ (resp. $\mathcal{P}^*(K)$) denotes the set of subsets (resp. the set of non empty subsets) of K. The domain of a function φ is denoted $\mathrm{Dom}(\varphi)$.

A *sequential letter-to-letter transducer* (SLT) from input alphabet Σ_1 to output alphabet Σ_2 is a tuple $\mathcal{T} = (\Sigma_1, \Sigma_2, Q, q_{\mathrm{init}}, \delta, \gamma, \rho, a_{\mathrm{init}})$ where Q is the finite set of *states*, $q_{\mathrm{init}} \in Q$ is the *initial state*, δ is a partial *transition function* from $Q \times \Sigma_1$ into Q, γ is a partial *output function* from $Q \times \Sigma_1$ into Σ_2 such that

[2] `http://regal.univ-mlv.fr/`



$$\begin{array}{lll} \delta(q_1,a)=q_2 & \delta(q_2,a)=q_3 & \delta(q_3,a)=q_3 \\ \delta(q_1,b)=q_1 & \delta(q_2,b)=q_1 & \delta(q_3,b)=q_3 \\ \gamma(q_1,a)=c & \gamma(q_2,a)=d & \gamma(q_3,a)=c \\ \gamma(q_1,b)=d & \gamma(q_2,b)=d & \gamma(q_3,b)=d \\ & & \rho(q_3)=d \end{array}$$

$$a_{\text{init}}=c$$

Figure 1: A sequential letter-to-letter transducer.

$\text{Dom}(\delta) = \text{Dom}(\gamma)$, ρ is a partial *final function* from Q into Σ_2, and $a_{\text{init}} \in \Sigma_2$ is the *initial output*. An SLT is *complete* if $\text{Dom}(\delta) = Q \times \Sigma_1$. *Accessible* states of an SLT are inductively defined by: q_{init} is accessible and if q is accessible, for every $a \in \Sigma_1$, $\delta(q, a)$ is accessible. An SLT is *accessible* if all its states are accessible. An example of complete and accessible SLT is depicted in Fig. 1.

Let $\mathcal{T}_1 = (\Sigma_1, \Sigma_2, Q_1, q_{\text{init}1}, \delta_1, \gamma_1, \rho_1, a_{\text{init}1})$ and $\mathcal{T}_2 = (\Sigma_1, \Sigma_2, Q_2, q_{\text{init}2}, \delta_2, \gamma_2, \rho_2, a_{\text{init}2})$ be two SLTs. A function φ from Q_1 to Q_2 is an *isomorphism* from $\mathcal{T}_1$ to $\mathcal{T}_2$ if it satisfies the following conditions: (1) φ is bijective, (2) $\varphi(q_{\text{init}1}) = q_{\text{init}2}$, (3) $\delta_1(q, a) = p$ iff $\delta_2(\varphi(q), a) = \varphi(p)$, (4) $\gamma_1(q, a) = b$ iff $\gamma_2(\varphi(q), a) = b$, (5) $\rho_1(q) = \rho_2(\varphi(q))$ and (6) $\varphi(a_{\text{init}1}) = a_{\text{init}2}$. If such an isomorphism exists, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are isomorphic. Informally, $\mathcal{T}_1$ and $\mathcal{T}_2$ are isomorphic if they encode the same SLT, up to state names. The relation *is isomorphic to* is trivially an equivalence relation.

In this paper, we are interested in the uniform random generation of SLTs up to isomorphism, i.e. we want to equiprobably generate equivalence classes for the isomorphic relation (and for a given number of states). Since the approach is purely syntactic and will be applied to different classes of finite automata, we do not need to define a semantic for SLTs.

2 Generating Sequential Transducers

We propose in this section a general method to generate randomly and uniformly deterministic and accessible automata-like structures with n states. For this purpose, we develop an algorithm that generates sequential letter-to-letter accessible transducers with n states, that can be further parametrized by giving some *restrictions* on the possible outputs for each input letter. The idea thereafter, for each given problem, is to find an effective *bijection* φ between the structures one wants to generate and such a family of transducers. The algorithm is in fact more general, since by Proposition 1, one can build an effective random generator even if φ is only an injection, provided that all the complete transducers are in the image of φ. This method will be applied in Sect. 3 and Sect. 4 to build random generators for deterministic tree walking automata and deterministic top-down tree automata.

Note that we are only interested here in the combinatorial structures of transducers, not on what their models are. Indeed, our approach will be used in order to generate several kinds of finite automata. Also note that we are interested in the uniform random generation of isomorphic classes of SLTs. The

algorithms proposed in this section fulfill this criterion. However, in order to simplify the exposition, we will write about random generation of SLTs rather than of equivalence classes of SLT, but keep in mind that we randomly generate witnesses of equivalence classes.

2.1 Rejection Algorithms

Before we describe the generation algorithm, let us recall the definition of a *rejection algorithm*: Suppose we want to generate elements of a set X, according to a probability distribution p_X. Furthermore, suppose that X is a subset of Y, and that we have a probability distribution p_Y on Y, whose restriction to X is p_X. If we have an algorithm to generate elements of Y according to p_Y, we may use this algorithm to generate elements of X as follows: repeatedly draw an element of Y, reject it if it is not in X, and stop if it is in X.

The average complexity of this rejection algorithm depends on the complexity of the generation algorithm on Y, on the complexity of the test whether an element of Y is in X, and on the average number of rejects. One can show that if $p_Y(X)$ is the probability for an element of Y to be in X, the average number of iterations is $1/p_Y(X)$.

2.2 Families of Transducers

Let us consider the family $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ of accessible SLTs with n states, where Σ_1 is the input alphabet, Σ_2 is the output alphabet, $r : \Sigma_1 \to \mathcal{P}^*(\Sigma_2)$ is the restriction on transitions, $r_i \in \mathcal{P}^*(\Sigma_2)$ is the restriction on initialization and $r_F \in \mathcal{P}^*(\Sigma_2)$ is the restriction on finalizations. An n-states accessible SLT $(\Sigma_1, \Sigma_2, Q, i, \delta, \gamma, \rho, a_i)$ belongs to $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ if the following conditions are met: (i) $a_i \in r_i$, (ii) $\rho(Q) \subseteq r_F$, and (iii) for all $a \in \Sigma_1$, $\gamma(Q, a) \subseteq r(a)$.

We denote by $\mathcal{C}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ the subset of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ that contains all the complete transducers. In order to generate a random element of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ or $\mathcal{C}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$, we split the problem into three parts: the underlying graph with input symbols, the transitions outputs, and the set of final states. For complete transducers, one can perform these parts independently and still ensure equiprobability. A rejection algorithm is used to adapt this method to possibly incomplete ones.

2.3 Generation Algorithm

The idea to generate deterministic and accessible word automata developed by Bassino et al. [1, 3] is to exhibit an effective injection ι from automata with n states on a k-letter alphabet to partitions of $[1, kn+1]$ in n parts in the complete case and of $[1, kn+2]$ in $n+1$ parts in the possibly incomplete case. The inverse ι^{-1} can also be computed, and though all partitions are not the image of an automaton, there are enough of them to guarantee that a rejection algorithm is efficient. The algorithm therefore consists in randomly generating a partition, using a Boltzmann sampler, until the partition is the image of an automaton, and then compute its preimage. Its average complexity is $\mathcal{O}(n^{3/2})$.

The algorithm to generate a random element of $\mathcal{C}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ consists in the following three steps:

1. Randomly generate a complete deterministic and accessible automaton with n states on Σ_1.

2. For each $q \in Q$ and each $a \in \Sigma_1$, randomly and uniformly choose $\gamma(q, a)$ in $r(a)$.

3. For each $q \in Q$, randomly and uniformly choose an element x of $r_F \uplus \{\#\}$, where $\#$ is a new symbol indicating that the state is not final; then define $\rho(q) = x$ if $x \neq \#$ and leave $\rho(q)$ otherwise undefined.

One can give the number of final states as a parameter f and change Step 3 into: Choose a random subset F with f elements of Q, and for each $q \in F$, choose $\rho(q)$ in r_F. The average complexity of the algorithm remains in $\mathcal{O}(n^{3/2})$.

To generate a random element of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$, we proceed as before, except that we generate a possibly incomplete automaton at Step 1. The problem here is that the distribution is not uniform anymore, since we consider multiple choices of $\gamma(q, a)$ when the transition does not exist, leading to the same transducer. In order to obtain uniformity, we arbitrarily order Σ_2 and only keep, using a rejection algorithm, transducers such that $\gamma(q, a)$ is set to the minimum in $r(a)$ for every undefined transition. Corollary 1 of [3] ensures that a proportion greater than c, where $c > 0$ is a real number, of possibly incomplete automata are complete. The average number of rejects of this method is therefore in $\mathcal{O}(1)$, as complete structures are not rejected and are numerous enough. The average complexity is in $\mathcal{O}(n^{3/2})$ as well. Observe that if we had generated the image of $\gamma(q, a)$ for defined transitions only, we would have lost the uniformity.

Using the same argument about the proportion of complete automata given in Corollary 1 of [3], we can prove the following fairly general proposition:

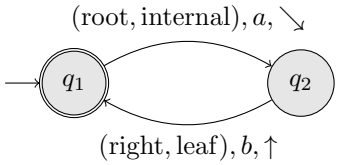
Proposition 1 *Let E_n be a subset of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ such that E_n contains $\mathcal{C}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$. The rejection algorithm consisting in generating uniformly an element of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ until it is in E_n performs $\mathcal{O}(1)$ iterations on average.*

Therefore, we have a straightforward method to build a random generator for such a class E_n, which is efficient if one can quickly test if a given transducer is in E_n. In particular, if the membership test can be done in linear time, then the average complexity of this method is in $\mathcal{O}(n^{3/2})$. Note that the constant factor might grow quickly, e.g. when $|\Sigma_1|$ grows.

3 Application to Tree Walking Automata

3.1 Deterministic Tree Walking Automata

A *deterministic tree walking automaton* (DTWA) on binary trees is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, q_{\text{init}}, F)$ where Q is a finite set of states, $q_{\text{init}} \in Q$ is the initial state, $F \subseteq Q$ the set of final states and Δ is a partial transition function from $Q \times \text{TYPE} \times \Sigma$ to $\{\varepsilon, \uparrow, \swarrow, \searrow\} \times Q$, where $\text{TYPE} = \{\text{root}, \text{left}, \text{right}\} \times \{\text{internal}, \text{leaf}\}$. A deterministic tree walking automaton is *complete* if Δ is a complete function. Accessible states of a DTWA are defined inductively: q_{init} is accessible, and if q is accessible and $\Delta(q, t, a) = (d, p)$ for some $(t, a) \in \text{TYPE} \times \Sigma$, then p is accessible. An example of a DTWA is shown in Fig. 2.



$$\Delta(q_1, (\text{root}, \text{internal}), a) = (\searrow, q_2)$$
$$\Delta(q_2, (\text{right}, \text{leaf}), b) = (\uparrow, q_1)$$

Figure 2: A deterministic tree walking automaton.

An *isomorphism* from a DTWA $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, q_{\text{init}1}, F_1)$ to a DTWA $\mathcal{A}_2 = (Q_2, \Sigma, \Delta_2, q_{\text{init}2}, F_2)$ is a bijective function from Q_1 to Q_2 satisfying the three conditions (1) $\varphi(q_{\text{init}1}) = q_{\text{init}2}$, (2) $\varphi(q) \in F_2$ iff $q \in F_1$, and (3) $\Delta_1(q, t, a) = (d, p)$ iff $\Delta_2(\varphi(q), t, a) = (d, \varphi(p))$.

3.2 From SLTs to DTWAs

We define in this section a rather straightforward bijection τ between DTWAs and a class of SLTs, called *DTWA-coherent* SLTs, that contains all the complete SLTs. We obtain thereafter a random generation algorithm for DTWAs thanks to the restriction mechanisms introduced in Sect. 2.

We first observe that a tree walking automaton can be viewed as a "classical" finite automaton on the alphabet $\Sigma_1 \times \Sigma_2$ defined by $\Sigma_1 = \text{TYPE} \times \Sigma$ and $\Sigma_2 = \{\varepsilon, \uparrow, \swarrow, \searrow\}$. Let $\mathcal{A} = (Q, \Sigma, \Delta, q_{\text{init}}, F)$ be a DTWA; we define $\tau(\mathcal{A})$ by

$$\tau(\mathcal{A}) = (\Sigma_1, \Sigma_2 \uplus \{\$, 1\}, Q, q_{\text{init}}, \delta, \gamma, \rho, \$) \ ,$$

with $\delta(q, (t, a)) = p$ and $\gamma(q, (t, a)) = d$ iff $\Delta(q, t, a) = (d, p)$, and $\text{Dom}(\rho) = F$ with $\rho(q) = 1$ iff $q \in F$. For the example depicted in Fig. 2,

$$\delta(q_1, ((\text{root}, \text{intern}), a)) = q_2 \quad \gamma(q_1, ((\text{root}, \text{intern}), a)) = \searrow$$
$$\delta(q_2, ((\text{right}, \text{leaf}), b) = q_1 \quad \gamma(q_2, ((\text{right}, \text{leaf}), b) = \uparrow \quad \rho(q_1) = 1 \ .$$

An SLT on $\Sigma_1, \Sigma_2 \uplus \{\$, 1\}$ is *DTWA-coherent* if its initial output symbol is \$.

Let us now provide an algorithm for random generation up to isomorphism of DTWAs. We reuse for this purpose the SLT generation algorithm, and need the following two propositions.

Proposition 2 *The function τ is a bijection from DTWAs to DTWA-coherent SLTs. Moreover, for every DTWA $\mathcal{A}$, $\tau(\mathcal{A})$ is complete (resp. accessible) if and only if $\mathcal{A}$ is complete (resp. accessible).*

Proposition 3 *Two DTWAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are isomorphic if and only if $\tau(\mathcal{A}_1)$ and $\tau(\mathcal{A}_2)$ are isomorphic.*

Proof. It suffices to note that the same isomorphism holds between $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\tau(\mathcal{A}_1)$ and $\tau(\mathcal{A}_2)$. □

Moreover, the restrictions introduced in Sect. 2 are helpful in order to generate *nicer* tree walking automata. Indeed, in a tree walking automaton, a transition labeled by $((t, a), d)$, with $(t, a) \in \Sigma_1$ and $d \in \Sigma_2$ is *useless* (i.e. can never be fired) in either of the following two cases:

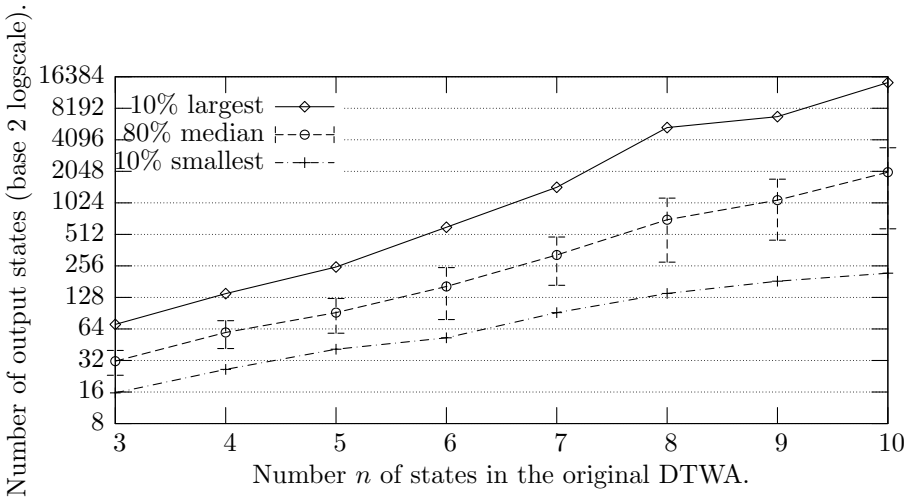


Figure 3: Average number of states in the 10 smallest, the 10 largest, and the 80 median top-down tree automata obtained from transforming 100 2-letter DTWAs with n states.

1. t is in $\{\text{root}\} \times \{\text{internal}, \text{leaf}\}$ and $d = \uparrow$, or
2. t is in $\{\text{root}, \text{left}, \text{right}\} \times \{\text{leaf}\}$ and $d \in \{\swarrow, \searrow\}$.

Let us denote by r^{DTWA} the subset of $\Sigma_1 \times \Sigma_2$ of the pairs (a, b) that do not match any of the above two cases. The class E_n^{DTWA} of useful DTWA-coherent SLTs with n states then contains $\mathcal{C}_n(\Sigma_1, \Sigma_2 \uplus \{\$, 1\}, r^{\text{DTWA}}, \{\$\}, \{1\})$ and is included in $\mathcal{D}_n(\Sigma_1, \Sigma_2 \uplus \{\$, 1\}, r^{\text{DTWA}}, \{\$\}, \{1\})$. Thus, random generation of DTWAs can be performed by first using Proposition 1 to obtain a SLT $\mathcal{T}$ and then by computing $\tau^{-1}(\mathcal{T})$.

3.3 Experimentation: From DTWAs to Top-Down Tree Automata

Tree walking automata enjoy a tight connection with several logical formalisms [9, 14], including some XPath fragments. Formula satisfiability then reduces to the emptiness of the language of a tree walking automaton. Nevertheless, the latter problem is rather hard to decide: it is an ExpTime-complete problem, for which the known algorithms consist essentially in constructing an exponentially larger equivalent top-down tree automaton, and (on the fly) checking this automaton for emptiness in linear time.

We have implemented a prototype tool for converting DTWAs into coaccessible nondeterministic top-down tree automata (under the form of Relax NG grammars [10]). Given a DTWA with n states, the resulting top-down tree automaton can hold as many as $\mathcal{O}(2^{n^2})$ states, that encode which pairs (p, q) of states allow a run of the DTWA to start from state p on a given tree node and return to it in state q without ever visiting its parent node.

We ran the algorithm on 100 randomly generated incomplete DTWA for each n and report the mean number of states in the computed equivalent top-down tree automaton in Fig. 3. Due to very high standard deviation values, we exclude the 10 smallest and 10 largest output automata from the mean computation,

and display their mean number of states on separate plots. All three plots display an exponential behaviour. Overall, the translation results in a $\mathcal{O}(2^n)$ size increase on average, which is significantly better than the worst-case $\mathcal{O}(2^{n^2})$ bound.

4 Application to Top-Down Tree Automata

4.1 Deterministic Top-Down Tree Automata

In this section, $\mathcal{F}$ denotes a finite ranked alphabet, i.e. there is an arity function ar from $\mathcal{F}$ into $\mathbb{N}$. We denote by $\mathcal{F}_i$ the subset of elements C of $\mathcal{F}$ such that $ar(C) = i$. We assume that $\$ \notin \mathcal{F}$. Let $\overline{\mathcal{F}} = \{(f, i) \mid f \in \mathcal{F} \setminus \mathcal{F}_0,\ 1 \leq i \leq ar(f)\}$.

A *deterministic top-down tree automata* (DTDA) is a tuple $(Q, \mathcal{F}, \theta, q_{\text{init}})$ where Q is a finite set of *states* satisfying $0 \notin Q$, $q_{\text{init}} \in Q$ is the *initial state*, and θ is a partial *transition function* mapping elements of $Q \times \mathcal{F}_i$ to Q^i (for all $i \geq 1$) and elements of $Q \times \mathcal{F}_0$ to 0. One can inductively define accessible states of a DTDA by: the initial state q_{init} is accessible and for every $f \notin \mathcal{F}_0$, if q is accessible and $\theta(q, f) = (q_1, \ldots, q_{ar(f)})$ then the q_i's are accessible. A DTDA is *complete* if $Q \times (\mathcal{F} \setminus \mathcal{F}_0) \subseteq \text{Dom}(\theta)$. For more information on top-down tree automata, the reader is referred to [8].

Let $\mathcal{A}_1 = (Q_1, \mathcal{F}, \theta_1, q_{\text{init1}})$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}, \theta_2, q_{\text{init2}})$ be two DTDAs. An *isomorphism* φ is a bijective function φ from Q_1 to Q_2 such that (1) for every state q, every $f \in \mathcal{F} \setminus \mathcal{F}_0$, $\theta_1(q, f) = (q_1, \ldots, q_{ar(f)})$ iff $\theta_2(\varphi(q), f) = (\varphi(q_1), \ldots, \varphi(q_{ar(f)}))$, (2) $\varphi(q_{\text{init1}}) = q_{\text{init2}}$, and (3) for every state q, every $C \in \mathcal{F}_0$, $\theta_1(q, C) = 0$ iff $\theta_2(\varphi(q), C) = 0$.

4.2 From SLTs to DTDAs

We define in this section a bijection ψ from DTDAs to a subclass of SLTs, called *DTDA-coherent* SLTs, that contains all the complete SLTs. For every DTDA $\mathcal{A} = (Q, \mathcal{F}, \theta, q_{\text{init}})$, let $\psi(\mathcal{A})$ be the SLT

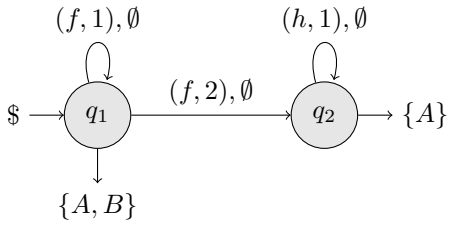
$$\psi(\mathcal{A}) = (\overline{\mathcal{F}}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, Q, q_{\text{init}}, \delta, \gamma, \rho, \$)$$

defined by: $\gamma(q, (f, i)) = \emptyset$ and $\delta(q, (f, i)) = p_i$ iff $\theta(q, f) = (p_1, \ldots, p_n)$, and $\rho(q) = \{A \in \mathcal{F}_0 \mid \theta(q, A) = 0\}$ iff this set is not empty, and $\rho(q)$ is undefined otherwise. For example, let $\mathcal{F}_0 = \{A, B\}$, $\mathcal{F}_1 = \{h\}$ and $\mathcal{F}_2 = \{f\}$ in $\mathcal{A}_{\text{exe}} = (\{q_1, q_2\}, \mathcal{F}, \theta_{\text{exe}}, \{q_1\})$ with $\theta_{\text{exe}}(q_1, f) = (q_1, q_2)$, $\theta_{\text{exe}}(q_2, h) = q_2$, and $\theta_{\text{exe}}(q_1, A) = \theta_{\text{exe}}(q_1, B) = \theta_{\text{exe}}(q_2, A) = 0$. This entails $\overline{\mathcal{F}} = \{(h, 1), (f, 1), (f, 2)\}$ in the SLT $\psi(\mathcal{A}_{\text{exe}})$ depicted in Fig. 4.

A SLT $(\overline{\mathcal{F}}, \mathcal{P}^*(\mathcal{F}_0) \uplus \{\$\}, Q, q_{\text{init}}, \delta, \gamma, \rho, \$)$ is *DTDA-coherent* if (1) for every state q, every $(f, i) \in \overline{\mathcal{F}}$, $\delta(q, (f, i))$ is defined iff $\delta(q, (f, j))$ is defined for all $j \in [1, ar(f)]$, (2) $\gamma(q, (f, i))$ is either undefined or equal to $\emptyset$, and (3) its initial output is \$.

Proposition 4 *The function ψ is a bijection from DTDA to DTDA-coherent SLTs. Moreover, for every DTDA $\mathcal{A}$, $\psi(\mathcal{A})$ is complete (resp. accessible) if and only if $\mathcal{A}$ is complete (resp. accessible).*

Proof. If $\mathcal{A}$ is a DTDA, then it is clear that $\psi(\mathcal{A})$ is DTDA-coherent. Now let $\mathcal{A}_1 = (Q_1, \mathcal{F}, \theta_1, q_{\text{init1}})$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}, \theta_2, q_{\text{init2}})$ be DTDAs such that



$$\delta_{\text{exe}}(q_1,(f,1)) = q_1$$
$$\delta_{\text{exe}}(q_1,(f,2)) = q_2$$
$$\delta_{\text{exe}}(q_2,(h,1)) = q_2$$
$$\rho_{\text{exe}}(q_1) = \{A,B\}$$
$$\rho_{\text{exe}}(q_2) = \{A\}$$

Figure 4: The SLT $\psi(\mathcal{A}_{\text{exe}}) = (\overline{\mathcal{F}}, \mathcal{P}^*(\{A,B\}) \uplus \{\$\}, \{q_1,q_2\}, q_1, \delta_{\text{exe}}, \gamma_{\text{exe}}, \rho_{\text{exe}}, \$)$.

$\psi(\mathcal{A}_1) = \psi(\mathcal{A}_2)$. By definition of ψ, $Q_1 = Q_2$ and $q_{\text{init}1} = q_{\text{init}2}$. Set $\psi(\mathcal{A}_1) = \psi(\mathcal{A}_2) = (\overline{F}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, Q_1, q_{\text{init}1}, \delta, \gamma, \rho, \$)$. Reasoning on δ shows that θ_1 and θ_2 are equal for letters in $\mathcal{F} \setminus \mathcal{F}_0$. Reasoning on ρ shows that θ_1 and θ_2 are equal for letters in $\mathcal{F}_0$. It follows that ψ is injective. The remaining points of the proposition are straightforward verifications. □

Proposition 5 *Two DTDAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are isomorphic if and only if $\psi(\mathcal{A}_1)$ and $\psi(\mathcal{A}_2)$ are isomorphic.*
Proof. It suffices to note that the same isomorphism holds between $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\psi(\mathcal{A}_1)$ and $\psi(\mathcal{A}_2)$. □

Let $r^{\text{DTDA}} = \overline{\mathcal{F}} \times \{\emptyset\}$. The class E_n^{DTDA} of DTDA-coherent SLTs with n states contains $\mathcal{C}_n(\overline{\mathcal{F}}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, r^{\text{DTDA}}, \{\$\}, \mathcal{P}^*(\mathcal{F}_0))$ and is included in $\mathcal{D}_n(\overline{\mathcal{F}}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, r^{\text{DTDA}}, \{\$\}, \mathcal{P}^*(\mathcal{F}_0))$. Thus, random generation of DTDAs can be performed using Proposition 1 to obtain a SLT $\mathcal{T}$ and by computing $\psi^{-1}(\mathcal{T})$.

5 Conclusion

In this paper we define a rejection algorithm to randomly and uniformly generate sequential letter-to-letter transducers with some restrictions. We also exhibit two bijections from this class of transducers to the class of deterministic tree walking automata and deterministic top-down tree automata respectively, and report on an empirical evaluation of a $\mathcal{O}(2^n)$ average complexity instead of a $\mathcal{O}(2^{n^2})$ worst-case bound for turning a deterministic tree walking automaton into an equivalent nondeterministic top-down tree automaton.

The approach we propose in this paper can easily be extended to some other classes of finite automata, like deterministic pebble tree walking automata. A much less obvious variation would be needed in order to randomly generate deterministic bottom-up tree automata or hedge automata.

References

[1] Bassino, F. and Nicaud, C., 2007. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1–3):86–104. doi:10.1016/j.tcs.2007.04.001.

[2] Bassino, F., David, J., and Nicaud, C., 2007. `REGAL`: A library to randomly and exhaustively generate automata. In Holub, J. and Žďárek, J., editors, *CIAA'07*, volume 4783, pages 303–305. doi:10.1007/978-3-540-76336-9_28.

[3] Bassino, F., David, J., and Nicaud, C., 2009. Enumeration and random generation of possibly incomplete deterministic automata. *Pure Mathematics and Applications*.

[4] Bassino, F., David, J., and Nicaud, C., 2009. On the average complexity of Moore's state minimization algorithm. In Albers, S. and Marion, J.Y., editors, *STACS'09*, volume 3 of *Leibniz International Proceedings in Informatics*, pages 123–134. Schloss Dagstuhl - LCI. urn:nbn:de:0030-drops-18222.

[5] Bojańczyk, M. and Colcombet, T., 2008. Tree-walking automata do not recognize all regular languages. *SIAM Journal on Computing*, 38(2):658–701. doi:10.1137/050645427.

[6] Champarnaud, J.M. and Paranthoën, T., 2005. Random generation of DFAs. *Theoretical Computer Science*, 330(2):221–235. doi:10.1016/j.tcs.2004.03.072.

[7] Chen, Y.F., Farzan, A., Clarke, E.M., Tsay, Y.K., and Wang, B.Y., 2009. Learning minimal separating DFA's for compositional verification. In Kowalewski, S. and Philippou, A., editors, *TACAS'09*, volume 5505 of *Lecture Notes in Computer Science*, pages 31–45. Springer. doi:10.1007/978-3-642-00768-2_3.

[8] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M., 2007. *Tree Automata Techniques and Applications*. http://www.grappa.univ-lille3.fr/tata.

[9] Engelfriet, J. and Hoogeboom, H.J., 1999. Tree-walking pebble automata. In Karhumäki, J., Maurer, H.A., Paun, G., and Rozenberg, G., editors, *Jewels are Forever*, pages 72–83. Springer. ISBN 3-540-65984-6.

[10] Murata, M., Lee, D., Mani, M., and Kawaguchi, K., 2005. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5(4):660–704. doi:10.1145/1111627.1111631.

[11] Neven, F., 2002. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46. doi:10.1145/601858.601869.

[12] Schewe, S., 2009. Büchi complementation made tight. In Albers, S. and Marion, J.Y., editors, *STACS'09*, volume 3 of *Leibniz International Proceedings in Informatics*, pages 661–672. Schloss Dagstuhl - LCI. urn:nbn:de:0030-drops-18543.

[13] Tabakov, D. and Vardi, M.Y., 2005. Experimental evaluation of classical automata constructions. In Sutcliffe, G. and Voronkov, A., editors, *LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 396–411. Springer. doi:10.1007/11591191_28.

[14] ten Cate, B. and Segoufin, L., 2008. XPath, transitive closure logic, and nested tree walking automata. In Lenzerini, M. and Lembo, D., editors, *PODS'08*, pages 251–260. ACM. doi:10.1145/1376916.1376952.

[15] van Glabbeek, R.J. and Ploeger, B., 2008. Five determinisation algorithms. In Ibarra, O.H. and Ravikumar, B., editors, *CIAA'08*, volume 5148 of *Lecture Notes in Computer Science*, pages 161–170. Springer. doi:10.1007/978-3-540-70844-5_17.

[16] Wulf, M.D., Doyen, L., Henzinger, T.A., and Raskin, J.F., 2006. Antichains: A new algorithm for checking universality of finite automata. In Ball, T. and Jones, R.B., editors, *CAV'06*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer. doi:10.1007/11817963_5.