Enumeration and Generation with a String Automata Representation [1]

Marco Almeida Nelma Moreira * Rogério Reis

DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

Abstract

The representation of combinatorial objects is decisive for the feasibility of several enumerative tasks. In this work, we present a unique string representation for complete initially-connected deterministic automata (ICDFAs) with n states over an alphabet of k symbols. For these strings we give a regular expression and show how they are adequate for exact and random generation, allow an alternative way for enumeration and lead to an upper bound for the number of ICDFAs. The exact generation algorithm can be used to partition the set of ICDFAs in order to parallelize the counting of minimal automata, and thus of regular languages. A uniform random generator for ICDFAs is presented that uses a table of pre-calculated values. Based on the same table, an optimal coding for ICDFAs is obtained.

Key words: finite automata, initially connected deterministic finite automata, exact enumeration, random generation, minimal automata

1 Introduction

The enumeration of languages based on their model representations is useful for several language characterizations, as well as for random generation and average case analysis. Adequate representations are also a main issue in symbolic manipulation environments. In this paper, we present a canonical form for initially connected deterministic finite automata (ICDFAs) with n states over an alphabet of k symbols and show how it can be used for counting, exact enumeration, sampling and optimal coding, not only the set

[1] Work partially funded by FCT and Program POSI.

* Corresponding author.
Email addresses: mfa@ncc.up.pt (Marco Almeida), nam@ncc.up.pt (Nelma Moreira), rvr@ncc.up.pt (Rogério Reis).

of ICDFAs but, to some extent, the set of regular languages. This canonical form is based on a usual representation of ICDFAs and was used in the **FAdo** project [MR05a,FAdo] to test if two minimal DFAs are *isomorphic*. However a precise characterization of these representations as regular languages of $\{0,\ldots,n-1\}^\star$ allows an exact and ordered generator of ICDFAs and leads to an alternative way to enumerate them. The enumeration of different kinds of finite automata was considered by several authors since late 1950s. For more complete surveys we refer the reader to Domaratzki *et al.* [DKS02] and to Domaratzki [Dom06]. Harary and Palmer [HP67,HP73] enumerate isomorphic automata with output functions as certain ordered pairs of functions. Harrison [Har65] considered the enumeration of non-isomorphic DFAs (and connected DFAs) up to a permutation of alphabetic symbols. With the same criteria, Narushima [Nar77] enumerated minimal DFAs. Liskovets [Lis69] and Robinson [Rob85] counted strongly connected DFAs and also non-isomorphic ICDFAs. The work of Korshunov, surveyed in [Kor78], enumerates minimal automata and gives estimates of ICDFAs without an initial state.

More recently, several authors examined related problems. Domaratzki *et al.* [DKS02] studied the (exact and asymptotic) enumeration of distinct languages accepted by finite automata with n states. Liskovets [Lis06] and Domaratzki [Dom04] gave (exact and asymptotic) enumerations of acyclic DFAs and of finite languages. Nicaud [Nic00], and Champarnaud and Paranthoën [CP05] presented a method for randomly generating ICDFAs. Bassino and Nicaud [BN] showed that the number of ICDFAs is $\Theta(n2^n\left\{{kn \atop n}\right\})$, where $\left\{{kn \atop n}\right\}$ is a Stirling number of the second kind.

In this paper we obtain a new formula for the number of non-isomorphic ICDFAs and we precisely relate our methods to those used by Nicaud and Champarnaud *et al.*, in the cited works. The exact generation algorithm developed can be used to partition the set of ICDFAs in order to parallelize the process of counting minimal automata, and thus counting regular languages. We also designed a uniform random generator for ICDFAs that uses a table of pre-calculated values (as usual in combinatorial decomposition approaches). Based on the same table it is also possible to obtain an optimal coding for ICDFAs.

The work reported in this paper was already partially presented in Reis *et al.* [RMA05] and Almeida *et.al* [AMR06], and is organized as follows. In the next section, some definitions and notation are introduced. Section 3 presents and characterizes canonical strings for non-isomorphic ICDFA$_\emptyset$s. Section 4 gives an upper bound and a new formula for ICDFA$_\emptyset$s' enumeration and relates our methods to some others in the literature. Section 5 briefly describes the implementation of a generator and methods for parallelizing the counting of regular languages. Using a table of pre-calculated values, in Section 6 is designed a uniform random generator and in Section 7 an optimal coding for

ICDFA$_\emptyset$s. Section 8 concludes and addresses some future work.

2 Preliminaries

Given two integers, m and n, let $[m, n]$ be the set $\{i \in \mathbb{Z} \mid m \leq i \ \wedge \ i \leq n\}$. In a similar way, we consider the variants $]n, m]$, $[n, m[$ and $]n, m[$.

A *deterministic finite automaton* (DFA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ the alphabet, *i.e.*, a non-empty finite set of symbols, $\delta : Q \times \Sigma \to Q$ is the transition function, q_0 the initial state and $F \subseteq Q$ the set of final states. Let the *size of* $\mathcal{A}$ be $|Q|$. We assume that the transition function is total, so we consider only *complete* DFAs. As we are not interested in the labels of the states, we can represent them by an integer $i \in [0, |Q| - 1]$.

A DFA is *initially-connected* [1] (IDFA) if for each state $q \in Q$ there exists a sequence $(q'_i)_{i \in [0,j]}$ of states and a sequence $(\sigma_i)_{i \in [0,j[}$ of symbols, for some $j < |Q|$, such that $\delta(q'_m, \sigma_m) = q'_{m+1}$ for $m \in [0, j[$, $q'_0 = q_0$ and $q'_j = q$. We denote by ICDFA a complete IDFA. The *structure* of an automaton (Q, Σ, δ, q_0) denotes a DFA without its final state information and is referred to as a DFA$_\emptyset$. Each structure, if $|Q| = n$, will be shared by 2^n DFAs. We denote by ICDFA$_\emptyset$ (IDFA$_\emptyset$) the structure of an ICDFA (IDFA).

Two DFAs $(Q, \Sigma, \delta, q_0, F)$ and $(Q', \Sigma', \delta', q'_0, F')$ are called *isomorphic* (by states) if $|\Sigma| = |\Sigma'| = k$, there exist bijections $\Pi_1 : \Sigma \to [0, k[$, $\Pi_2 : \Sigma' \to [0, k[$ and a bijection $\iota : Q \to Q'$ such that $\iota(q_0) = q'_0$ and, for all $\sigma \in \Sigma$ and $q \in Q$, $\iota(\delta(q, \sigma)) = \delta'(\iota(q), \Pi_2^{-1}(\Pi_1(\sigma)))$, and $\iota(F) = F'$.

The *language* accepted by a DFA $\mathcal{A}$ is $L(\mathcal{A}) = \{x \in \Sigma^\star \mid \delta(q_0, x) \in F\}$ with δ extended to $\Sigma^\star$. Two DFA are *equivalent* if they accept the same language. Obviously, two isomorphic automata are equivalent, but two non-isomorphic automata may also be equivalent. A DFA $\mathcal{A}$ is *minimal* if there is no DFA $\mathcal{A}'$, with fewer states, equivalent to $\mathcal{A}$. Trivially, if a DFA is minimal then it must be an ICDFA. Minimal DFAs are unique up to isomorphism. Domaratzki *et al.* gave some asymptotic estimates and explicit computations of the number of distinct languages accepted by finite automata with n states over an alphabet of k symbols. Given n and k, they denote by $f_k(n)$ the number of pairwise non-isomorphic minimal DFAs and by $g_k(n)$ the number of distinct languages accepted by DFAs, where $g_k(n) = \sum_{i=1}^{n} f_k(i)$.

[1] Also called *accessible*.

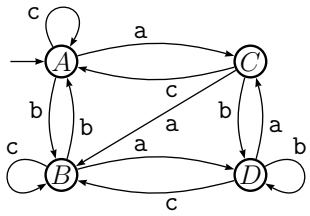
3 String representation for ICDFAs

The method used to represent a DFA has a significant role in the amount of computer work needed to manipulate that information, and can give an important insight about this set of objects, both in its characterization and enumeration.

Let us disregard the set of *final states* of a DFA. A *naive* representation of a $\mathrm{DFA}_\emptyset$ can be obtained by the enumeration of its states and for each state the list of its transitions for each symbol. But this representation is not unique. To obtain a canonical representation we can consider an order in the alphabet and an induced order in the states and transitions.

Given a complete $\mathrm{DFA}_\emptyset$ (Q, Σ, δ, q_0) with $|Q| = n$ and $|\Sigma| = k$, consider a total order over Σ. We can define a canonical order over the set of the states by traversing the automaton in a breadth-first way choosing at each node the outgoing edges using the order of Σ. The procedure is as following: let the first state 0 be the initial state q_0, the second state the first one to be referred to (excepting q_0) by a transition from q_0, the third state the next referred in transitions from one of the first two states, and so on...

If we restrict this representation to $\mathrm{ICDFA}_\emptyset$s, then this representation is unique and defines an order over the set of its states. For instance, consider the following $\mathrm{ICDFA}_\emptyset$ and consider the alphabetic order in $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.



The states ordering is A,C,B,D and $[1, 2, 0, 2, 3, 0, 3, 0, 2, 1, 3, 2]$ is its string representation.

We stress that this kind of representation for the transition table of an automaton is not new in the literature, but is new its characterization and application to enumeration and generation.

Formally, let Σ be an alphabet with $|\Sigma| = k$, and $\Pi : \Sigma \to [0, k[$ a bijection. Given an $\mathrm{ICDFA}_\emptyset$ (Q, Σ, δ, q_0) with $|Q| = n$, let $\varphi : Q \to [0, n[$ be defined by the following algorithm:

$\varphi(q_0) \leftarrow 0$
i $\leftarrow 0$
s $\leftarrow 0$

```
do
   for  j ∈ [0, k[
        if  δ(φ^{-1}(s), Π^{-1}(j)) ∉ φ^{-1}([0, i])  then
            φ(δ(φ^{-1}(s), Π^{-1}(j)) ← i + 1
            i ← i + 1
   s ← s + 1
while  s <= i
```

Lemma 1 *The function φ is bijective.*

PROOF. That φ is injective is trivial, because whenever, in the definition above, a new extension to φ is defined a different value is assigned. Let us prove that φ is surjective. Let $q \in Q$. As (Q, Σ, δ, q_0) is an ICDFA$_\emptyset$ there exist sequences $(q'_i)_{i\in[0,j]}$ and $(\sigma_i)_{i\in[0,j[}$ with $j < n$ such that $\delta(q'_m, \sigma_m) = q'_{m+1}$ for $m \in [0, j[$, $q'_0 = q_0$ and $q'_j = q$. We have $\varphi(q'_0) = 0$. For $m \in [0, j[$, if $q'_m \in \varphi^{-1}([0, n[)$ then $q'_{m+1} \in \varphi^{-1}([0, n[)$. Then $\varphi^{-1}([0, n[) = Q$, and thus φ is a bijection.

We have the following with trivial proof.

Lemma 2 *The function φ defines an isomorphism between (Q, Σ, δ, q_0) and $([0, n[, \Sigma, \delta', 0)$ with $\delta'(i, \sigma) = \varphi(\delta(\varphi^{-1}(i), \sigma))$ for $i \in [0, n[$. Moreover the* canonical string *that represents this automaton, as described before, is defined by: $(s_i)_{i\in[0,kn[}$ with $s_i \in [0, n[$ and $s_i = \delta'(\lfloor i/k \rfloor, \Pi^{-1}(i \bmod k))$, for $i \in [0, kn[$.*

Lemma 3 *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0)$ be an* ICDFA$_\emptyset$*, with $|Q| = n$ and $|\Sigma| = k$, and let $(s_i)_{i\in[0,kn[}$ be its canonical string. Then,*

$$(\exists j \in [0, kn[)\ s_j = n - 1, \qquad \textbf{(R0)}$$

$$(\forall m \in [2, n[)(\forall i \in [0, kn[)(s_i = m \ \Rightarrow\ (\exists j \in [0, i[)\ s_j = m - 1), \qquad \textbf{(R1)}$$

$$(\forall m \in [1, n[)(\exists j \in [0, km[)\ s_j = m. \qquad \textbf{(R2)}$$

PROOF. As **R0** is a consequence of **R2**, we will omit it whenever **R2** is enforced. Rule **R1** establishes that a state label (greater than 0) can only occur after some occurrence of its predecessors. This is a direct consequence of φ definition where the extensions to φ are defined in ascending order.

Suppose that **R2** does not verify, thus there exists a state $r \in Q$, such that for $m = \varphi(r)$, m does not occur in the first km symbols of the string (the m first state descriptions). But $m \notin \{s_i \mid i \in [0, mk[\} = \{\delta'(i, \sigma) \mid i \in [0, m[, \sigma \in \Sigma\}$ means that m in not accessible from state 0 in $([0, n[, \Sigma, \delta', 0)$, and this automaton is isomorphic to $\mathcal{A}$ (by φ). This contradicts the fact that $\mathcal{A}$ is initially connected. Thus **R2** is verified.

Lemma 4 *Every string* $(s_i)_{i\in[0,kn[}$ *with* $s_i \in [0, n[$ *satisfying* ***R1*** *and* ***R2*** *represents an* ICDFA$_\emptyset$ *with* n *states over an alphabet of* k *symbols.*

PROOF. Let $S = \{s_i \mid i \in [0, kn[\}$. Because of **R2**, $(n-1) \in S$, and using **R1**, we have $S = [0, n[$. Thus let us consider the automaton $([0, n[, [0, k[, \delta, 0)$ where $\delta(r, \sigma) = s_{kr+\sigma}$. Trivially this defines a DFA$_\emptyset$, it only remains to show that it is initially connected. Let m be a state of the automaton. Because of **R2** there must exist $j < mk$ such that $s_j = m$. This means that $\delta(\lfloor j/k \rfloor, j \bmod k) = m$. If $j = 0$ then we can stop, if not we can repeat the process, the number of times necessary (not more than m) to get to the initial state and thus prove that m is accessible from the initial state.

From these lemmas (Lemma 1–4), follows immediately that:

Theorem 5 *There is a one-to-one mapping between* $(s_i)_{i\in[0,kn[}$ *with* $s_i \in [0, n[$ *satisfying rules* ***R1*** *and* ***R2****, and the non-isomorphic* ICDFA$_\emptyset$*s with* n *states, over an alphabet of* k *symbols.*

For each canonical string representing an ICDFA$_\emptyset$, if we add a sequence of *final states*, we obtain a *canonical form* for ICDFAs.

This canonical representation can be extended to general initially-connected IDFA$_\emptyset$s, by representing all missing transitions with the value -1. In this case, rules **R1** and **R2** remain valid, and we can assume that the transitions from this state are into itself. Moreover, the enumeration formulae and the generation algorithms we are going to present can also be extended to IDFA$_\emptyset$s, and thus to IDFAs.

4 Enumeration of ICDFAs

In order to have an algorithm for the enumeration and generation of ICDFA$_\emptyset$s, instead of rules **R1** and **R2** an alternative set of rules were used. For $n = 1$ there is only one (non-isomorphic) ICDFA$_\emptyset$ for each $k \geq 1$, so we assume in the following that $n > 1$. In a canonical string for an ICDFA$_\emptyset$, let $(f_j)_{j\in[1,n[}$ be the sequence of indexes of the first occurrence of each state label j. For explanation purposes, we call those indexes *flags*.

It is easy to see that (**R0**,**R1**) and (**R2**) correspond, respectively, to (**G1**) and (**G2**):

$$(\forall j \in [2, n[)(f_j > f_{j-1}), \qquad (\mathbf{G1})$$

$$(\forall m \in [1, n[)\ (f_m < km). \qquad (\mathbf{G2})$$

This means that $f_1 \in [0, k[$, and $f_{j-1} < f_j < kj$ for $j \in [2, n[$. We begin by counting the number of sequences of flags allowed.

Theorem 6 *Given k and n, the number of sequences $(f_j)_{j\in[1,n[}$, $F_{k,n}$, is given by*

$$F_{k,n} = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} 1 = \binom{kn}{n} \frac{1}{(k-1)n+1} = C_n^{(k)},$$

where $C_n^{(k)}$ are the (generalized) Fuss-Catalan numbers.

PROOF. The first equality follows from the definition of the $(f_j)_{j\in[1,n[}$. For the second, note that $C_n^{(k)}$ enumerates k-ary trees with n internal nodes, $\mathcal{T}_n^k$ (see for instance [SF96]). In particular, for $k = 2$, $C_n^{(2)}$ are exactly the Catalan numbers that count binary trees with n internal nodes. This sequence appears in Sloane [Slo03] as **A00108** and for $k = 3$ and $k = 4$ as sequences **A001764** and **A002293**, respectively. So it suffices to give a bijection between these trees and the sequences of flags. Recall that a k-ary tree is an external node or an internal node attached to an ordered sequence of k, k-ary sub-trees.

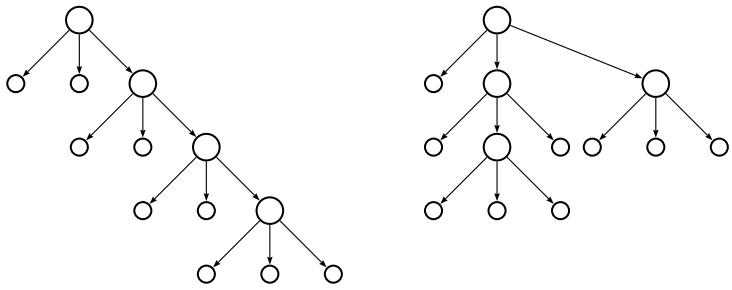


Fig. 1. Two 3-ary trees with 4 internal nodes and the correspondent sequence of flags.

Let $\mathcal{T}_n^k$ be a k-ary tree and let $<$ be a total order over Σ. For each internal node i of $\mathcal{T}_n^k$ its outgoing edges can be ordered left-to-right and attached a unique symbol of Σ according to $<$. Considering a breadth-first, left-to-right, traversal of the tree and ignoring the root node (that is considered the 0-th internal node), we can represent $\mathcal{T}_n^k$, uniquely, by a bitmap where a 0 represents an external node and a 1 represents an internal node. As the number of external nodes are $(k-1)n+1$, the length of the bitmap is kn. Moreover the $(j+1)$-th block of k bits corresponds to the children of the j-th internal node visited, for $j \in [0, n[$. For example, the bitmaps of the trees in Figure 1 are $[0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0]$ and $[0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]$, respectively. The positions of the 1's in the bitmaps correspond to a sequence of flags, $(f_i)_{i\in[1,n[}$, *i.e.*, f_i corresponds to the number of nodes visited before the i-th internal node (excluding the root node). It is obvious that $(f_i)_{i\in[1,n[}$ verifies **G1**. For **G2**,

note that for the each internal node the outdegree of the previous internal nodes is k. Conversely, given a sequence of flags $(f_i)_{i\in[1,n[}$, we construct the bitmap such that b_{f_i}=1 for $i \in [1, n[$ and $b_j = 0$ for the remaining values, for $j \in [0, kn[$. As above, for the representation of the $(j + 1)$-th internal node, $\lfloor f_j/k \rfloor$ gives the parent and $f_j \bmod k$ gives its position between its siblings (in breadth-first, left-to-right traversal).

To generate all the ICDFA$_\emptyset$s, for each allowed sequence of flags $(f_j)_{j\in[1,n[}$, all the remaining symbols, s_i, can be generated according to the following rules:

$$i < f_1 \;\Rightarrow\; s_i = 0, \tag{\textbf{G3}}$$
$$(\forall j \in [1, n-2])(f_j < i < f_{j+1} \;\Rightarrow s_i \in [0, j]), \tag{\textbf{G4}}$$
$$i > f_{n-1} \;\Rightarrow\; s_i \in [0, n[. \tag{\textbf{G5}}$$

Before we give a formula for the number of these strings, we recall that Liskovets [Lis69] and, independently, Robinson [Rob85] gave for the number of non-isomorphic ICDFA$_\emptyset$s, $B_{k,n}$, the formula $B_{k,n} = \frac{b_{k,n}}{(n-1)!}$ where $b_{k,1} = 1$ and $b_{k,n} = n^{kn} - \sum_{1\le j<n} \binom{n-1}{j-1} n^{k(n-j)} b_{k,j}$, for $n > 1$. The total number of transition functions is n^{kn} and from that they subtract the number of those that have $n-1$, $n-2$,...,1 states not accessible from the initial state. Then, they divide by $(n-1)!$, as the names of the states (except the initial) are irrelevant. On the other hand, the formula (2) we will derive is a direct positive summation.

First, let us consider the set of strings $(s_i)_{i\in[0,kn[}$ with $s_i \in [0, n[$ and satisfying only **G1** (*i.e.* **R0** and **R1**). The number of these strings gives an upper bound for $B_{k,n}$. We know that the last k symbols of any string can be chosen from $[0, n[$, so there are always n^k choices. For the others they belong to the language $A_n \cap [0, n[^{kn-k}$, where for $c > 0$,

$$A_c = L(0^\star \prod_{j\in[1,c[} j(0 + \cdots + j)^\star). \tag{1}$$

For each m, the words of length m of these languages are related with partitions of $[1, m]$ into $c \ge 1$ parts (see Moreira and Reis [MR05b]), and so they can be enumerated by Stirling numbers of the second kind, $\left\{ {m \atop c} \right\}$ [SF96]. In this case, we have $|A_n \cap [0, n[^{kn-k}| = \left\{ {k(n-1)+1 \atop n} \right\}$.

Theorem 7 *For all $n,\ k \ge 1$, $B_{k,n} \le \left\{ {k(n-1)+1 \atop n} \right\} n^k \le n \left\{ {kn \atop n} \right\}$.*

PROOF. The second inequality follows from the recursive definition of Stirling numbers of the second kind and the following property, $\left\{ {n-i \atop m} \right\} \le \frac{1}{n^i} \left\{ {n \atop m} \right\}$, for $i \in [0, n-m]$.

Our bound is slightly tighter than the one given by Bassino and Nicaud [BN], that is exactly the right member of the second inequality.

Now in order to simultaneously satisfy **R1** and **R2**, we must consider the sequences of flags. Given a sequence of flags $(f_j)_{j\in[1,n[}$ and considering $f_n = kn$, the correspondent set of canonical strings can be represented by the regular expression:

$$\left(0^{f_1} \prod_{j\in[1,n[} j(0+\cdots+j)^{f_{j+1}-f_j-1}\right),$$

which is a direct consequence of **G1**–**G5**.

Considering the set of sequences of flags (see Theorem 6) the set of canonical strings can be represented by the regular expression:

$$\sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \sum_{f_3=f_2+1}^{3k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} \left(0^{f_1} \prod_{j\in[1,n[} j(0+\cdots+j)^{f_{j+1}-f_j-1}\right).$$

From the above, we have that for each sequence of flags $(f_j)_{j\in[1,n]}$ the number of canonical strings is

$$\prod_{j\in[1,n]} j^{f_j-f_{j-1}-1}.$$

Theorem 8 *The number of strings $(s_i)_{i\in[0,kn[}$ representing* ICDFA$_\emptyset$*s with n states over an alphabet of k symbols is given by*

$$B_{k,n} = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \sum_{f_3=f_2+1}^{3k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} \prod_{j=1}^{n} j^{f_j-f_{j-1}-1}, \tag{2}$$

where $f_n = kn$ and $f_0 = -1$.

In Section 7 we give another recursive definition for $B_{k,n}$ more adequate for tabulation.

Corollary 9 *The number of non-isomorphic* ICDFA*s with n states over an alphabet of k symbols is $2^n B_{k,n}$.*

4.1 Analysis of the Nicaud et al. *Method*

Champarnaud and Paranthoën [CP05], generalizing work of Nicaud [Nic00] for $k = 2$, presented a method to generate and enumerate ICDFA$_\emptyset$s, although not giving an explicit and compact representation for them, as the string representation used here. The same method is used by Bassino and Nicaud [BN]. An order $<$ over $\Sigma^\star$ is a *prefix order* if $(\forall x \in \Sigma^\star)(\forall \sigma \in \Sigma) x < x\sigma$. Let $\mathcal{A}$ be an ICDFA$_\emptyset$ over Σ with k symbols and n states. Given a prefix order in $\Sigma^\star$, each automaton state is ordered according to the first word $x \in \Sigma^\star$ that

reaches it in a simple path from the initial state. The set of these words is a prefix set $\mathcal{P}$ and prefix sets are in bijection with k-ary trees with n internal nodes, and therefore to the set of sequences of flags, in our representation [2]. Then it is possible to obtain a valid ICDFA$_\emptyset$ by adding other transitions in a way that preserves the previous state labelling. For the generation of the sets $\mathcal{P}$ it is used another set of objects that are in bijection with k-ary trees with n internal nodes and are called *generalized tuples*. It is defined as

$$R_{k,n} = \{(x_1, \ldots, x_s) \in [1, n]^s \mid \forall i \in [2, s],\ (x_i \geq \lceil \tfrac{i}{k-1} \rceil \ \wedge\ x_i \geq x_{i-1})\}$$

with $s = (k-1)n$.

However we can establish a direct bijection between this set and the set of sequences of flags. Let $X = (x_1, \ldots, x_s)$ be a generalized tuple. From it, we can build the sequence $(1^{p_1}, 2^{p_2}, \ldots, n^{p_n})$, where $p_j = |\{x_i \mid x_i = j\}|$ for $j \in [1, n]$. Let $f_1 = p_1$, $f_i = p_i + f_{i-1} + 1$, for $i \in [2, n[$ and $f_n = p_n + f_{n-1} + 2$. It is obvious that $(f_i)_{i \in [1,n[}$ satisfies **G1**. To prove that it satisfies **G2**, note that $f_i = (i-1) + \sum_{j=1}^{i} p_j$, for $i \in [1, n[$. By induction on i it can be proved that $\sum_{j=1}^{i} p_j \leq (k-1)i + 1$, for $1 \leq i \leq n$. But then we have, $f_i < ki$, as wanted. In a similar way, we can transform a sequence of flags in a generalized tuple.

Nicaud *et al.* compute the number of ICDFA$_\emptyset$s using recursive formulae associated with generalized tuples, akin the ones we present in Section 6. The upper bound refered above is obtained, disregarding the first condition in the definition of the generalized tuples.

5 Generation of Regular Languages

We briefly describe a method to generate all ICDFA$_\emptyset$s, given k and n. We start with an initial string, and then consecutively iterate over all allowed strings until the last one is reached. The main procedure is the one that given a string returns the *next* legal one. For each k and n, the first canonical string is $0^{k-1}10^{k-1}2\cdots0^{k-1}(n-1)0^k$ and the last is $12\cdots(n-1)(n-1)^{(k-1)n+1}$. We first generate a sequence of flags, according to the rules **G1**–**G2**, and then, for each one, the set of canonical strings in lexicographic order and according to **G3**–**G5**. When a new sequence of flags is generated, the first string has 0s in all other positions (*i.e.*, the lower bounds in rules **G3**–**G5**). The last string for each sequence of flags has the value $s_l = j$ for $l \in]f_j, f_{j+1}[$, with $j \in [1, n[$. The time complexity of the generator is linear in the number of automata. Finally, for the generation of ICDFAs we only need to add to the string representation of an ICDFA$_\emptyset$, a string of n 0's and 1's, correspondent

[2] This order over the set of states induces a prefix order in $\Sigma^\star$, namely a graded lexicographic order.

to one of the 2^n possible choices of final states.

To obtain the number of languages accepted by DFAs with n states over an alphabet of k symbols, we can generate all ICDFAs, determine which of them are minimal ($f_k(n)$) and calculate the value of $g_k(n)$. Obviously, this is in general an intractable procedure. But for small values of n and k some experiments can take place. We must have an efficient implementation of a minimization algorithm, not because of the size of each automaton but because the number of automata we need to cope with. For that we implemented Hopcroft's minimization algorithm [Hop71], using efficient set representations as described by Almeida and Reis [AR06].

The problem can be parallelized providing that the space search can be safely partitioned. Using the method presented in Section 5, we can easily generate *slices* of ICDFAs and feed them to the minimization algorithm. A *slice* is a sequence of ICDFAs. If we have a set of CPUs available, each one can receive a slice, generate all ICDFA$_\emptyset$s (in that slice), generate all the necessary ICDFAs and feed them to the minimization algorithm. In this way, we can safely divide the search space and distribute each slice to a different CPU. Note that this approach relies in the assumption that we have a much more efficient way to partition the search space than to actually perform the search (in this case a minimization algorithm). The task of creating the slices can be taken by a central process that successively generates the next slice and at the end assembles all the results. With this approach was possible to obtain some new exact values for the number of non-isomorphic minimal ICDFAs: $f_2(7) = 25493886852$, $f_4(4) = 7756763336$, $f_3(5) = 25184560134$ and $f_2(8) = 2567534031190$. For the last value, the process took about 8 days with a 48 CPU computer grid, that corresponds to more than an year of CPU time. More experimental results were reported in Almeida *et* al. [AMR06].

6 Uniform Random Generation

The canonical strings for ICDFA$_\emptyset$s permit an easy random generation of ICDFA$_\emptyset$s, and thus of ICDFAs. To randomly generate an ICDFA for a given n and k, it is only necessary to: (i) randomly generate a valid sequence of flags $(f_i)_{i\in[1,n[}$ according to **G1** and **G2**; (ii) followed by the random generation of the rest of the kn elements of the string following **G3**–**G5** rules; (iii) and finally the random generation of the set of final states. The uniformity issue for steps (ii) and (iii) is quite straightforward. For step (iii) it is just necessary to use a uniform random integer generator for a value $i \in [0, 2^n]$. It is enough, for step (ii) the repeated use of the same number generator for values in the range $[0, i]$ for $0 \leq i < n$ according to **G3**–**G5**. Step (i) is the only step that needs special care. Consider the case $n = 5$ and $k = 2$. Because of **R1** flag f_1 can only be on positions 0 or 1. But there are 140450 ICDFA$_\emptyset$s with f_1 in

the first case and only 20225 in the second. Thus the random generation of flags, to be uniform, must take this into account by making the first case more probable than the second. We can generate a random $\mathrm{ICDFA}_\emptyset$ generating its representing string from left to right. Supposing that flag f_{m-1} is already placed at position i and all the symbols to its left are generated, *i.e.*, the prefix $s_0 s_1 \cdots s_i$ is already defined, then the process can be described by:

$r \leftarrow \mathbf{random}(1, \sum_{j \in]i, mk[} N_{m,j})$
for $j \in]i, mk[$:
  if $r \in \left[\sum_{l \in [i,j[} N_{m,l}, \sum_{l \in [i,j]} N_{m,l} \right]$ **then return** j

where `random(a,b)` is an uniform random generated integer between `a` and `b`, and $N_{m,j}$ is the number of $\mathrm{ICDFA}_\emptyset$s with prefix $s_0 s_1 \cdots s_i$ with the first occurrence of symbol m in position j, making $N_{m,i} = 0$ to simplify the expressions. The values for $N_{m,j}$ could be obtained from expressions similar to Equation (2), and used in a program. But the program would have a exponential time complexity. By expressing $N_{m,j}$ in a recursive form, we have

$$\begin{aligned} N_{n-1,j} &= n^{nk-1-j} && \text{with } j \in [n-2, (n-1)k[, \\ N_{m,j} &= \sum_{i=0}^{(m+1)k-j-2} (m+1)^i N_{m+1,j+i+1} && \text{with } m \in [1, n-2], \\ & && j \in [m-1, mk[. \end{aligned} \tag{3}$$

The second equation, can have an even simpler form:

$$\begin{aligned} N_{m,mk-1} &= \sum_{i=0}^{k-1} (m+1)^i N_{m+1,mk+i} && \text{with } m \in [1, n-2], \\ N_{m,i} &= (m+1) N_{m,i+1} + N_{m+1,i+1} && \text{with } m \in [1, n-2], \\ & && i \in [m-1, mk-2]. \end{aligned} \tag{4}$$

This evidences the fact that we keep repeating the same computations with very small variations, and thus, if we use some kind of tabulation of these values ($N_{m,j}$), with the obvious price of memory space, we can create a version of a uniform random generator, that apart of a constant overhead used for tabulation of the function refered, has a complexity of $\mathcal{O}(n^2 k)$.

The algorithm is described by the following:

```
g = −1, l ← 0
for i ∈ [1, n[ :
    f ← generateflag(i, g + 1)
    for j ∈ ]g, f[ :
        s_i ← random(0, i − 1)
        l ← l + 1
    s_l ← i , l ← l + 1
    g ← f
```

```
def generateflag(m, l) :
    r ← random( 0 , Σ_{i=l}^{km−1} m^{i−l} N_{m,i} )
    for i ∈ [l, mk[ :
        if r < m^{i−l} N_{m,i}
        then return i
        else r ← r − m^{i−l} N_{m,i}
```

This means that using a C implementation with `libgmp` the times reported in Table 1 were observed. It is possible, without unreasonable amounts of

	$k=2$	$k=3$	$k=5$	$k=10$	$k=15$
$n=10$	0.10s	0.16s	0.29s	0.61s	1.30s
$n=20$	0.31s	0.49s	1.26s	4.90s	12.24s
$n=30$	0.54s	1.37s	3.19s	19.91s	62.12s
$n=50$	1.61s	3.86s	17.58s	142.00s	947.71s
$n=75$	3.96s	12.98s	76.69s	700.20s	2459.34s
$n=100$	7.92s	36.33s	215.32s	2219.04s	8091.30s

Table 1
Times for the random generation of 10000 automata (AMD Athlon 64 at 2.5GHz)

RAM to generate random automata for unusually large values of n and k. For example, with $n = 1000$ and $k = 2$ the memory necessary is less than 450MB. The amount of memory used is so large not only because of the amount of tabulated values, but because the size of the values is enormous. To understand that, it is enough to note that the total number of ICDFA$_\emptyset$s for these values of n and k is greater than 10^{3350}, and the values tabulated are only bounded by this number.

7 Optimal coding of ICDFAs

Given a canonical string for ICDFAs of size n over an alphabet of k symbols, we can compute its number in the generation order (as described in Section 5) and vice-versa, *i.e.*, given a number less than $B_{k,n}$, we obtain the corresponding ICDFA$_\emptyset$. This provides an optimal encoding for ICDFAs, as defined by M. Lothaire [Lot05]. This bijection is accomplished by using the tables defined in Section 6 that correspond to partial sums of Equation (2). By expanding $N_{m,j}$ using Equations (3), we have

Theorem 10 $B_{k,n} = \sum_{l=0}^{k-1} N_{1,l}$.

From ICDFAs to Integers Let $(s_i)_{i\in[0,kn[}$ be the canonical string of an ICDFA$_\emptyset$, and let $(f_j)_{j\in[1,n[}$ be the corresponding sequence of flags. From the

sequence of flags we obtain the following number,

$$n_f \;=\; \sum_{j\in[1,n[} \left(\prod_{m\in[1,j[} (m+1)^{f_{m+1}-f_m-1} \right) \left(\sum_{l\in]f_j,jk[} (j^{l-f_j} N_{j,l}) \right), \qquad (5)$$

which is the number of the first $\mathrm{ICDFA}_\emptyset$ with flags $(f_j)_{j\in[1,n[}$. Then, we must add the information provided by the rest of the elements of the string $(s_i)_{i\in[0,kn[}$:

$$n_r \;=\; \sum_{j\in[1,n[} \left(\prod_{m\in]j,n[} (m+1)^{f_{m+1}-f_m-1} \right) \left(\sum_{l\in]f_j,f_{j+1}[} s_l (j+1)^{f_{j+1}-1-l} \right). \qquad (6)$$

The number of the canonical string is $n_s = n_f + n_r$.

From Integers to ICDFAs Given an integer $m \in [0, B_{k,n}[$ a canonical string for an $\mathrm{ICDFA}_\emptyset$ can be obtained using an inverse method. The flags $(f_j)_{j\in[1,n[}$ are generated from right-to-left, by successive subtractions. The rest of the string $(s_i)_{i\in[0,kn[}$ is generated considering the remainders of integer divisions. The algorithms are the following, where $f_0 = 0$:

```
s ← 1
for i ∈ [1, n[ :
    j ← ik − 1
    p ← i^{j−f_{i−1}−1}
    while j ≥ i − 1 and m ≥ ps × N_{i,j} :
        m ← m − ps × N_{i,j}
        j ← j − 1
        p ← p/i
    s ← s × i^{j−f_{i−1}−1}
    f_i ← j
```

```
i ← kn − 1
j ← n − 1
while m > 0 and j > 0 :
    while m > 0 and i > f_j :
        s_i ← m mod (j + 1)
        m ← m/(j + 1)
        i ← i − 1
    i ← i − 1
    j ← j − 1
```

8 Conclusion

The methods here presented were implemented and tested to obtain both exact and approximate values for the density of minimal automata. Our experimental results corroborate the ones of Champarnaud *et al.* [CP05], that lead to the conjecture that for $k > 2$ almost all ICDFAs are minimal. Of course, one challenge is to try to understand why this happens. Bassino and Nicaud [BN] presented a random generator of ICDFAs based on Boltzmann samplers, recently introduced by Duchon *et al.* [DFLS04]. However the sampler is uniform for partitions of a set and not for the universe of automata. These partitions correspond to string representations that verify **R1**. By considering **R2**, we plan to study the possibility to write Boltzmann Samplers for ICDFAs.

We thank the referees for their comments that helped improve this paper.

References

[AMR06] M. Almeida, N. Moreira, and R. Reis. Aspects of enumeration and generation with a string automata representation. In H. Leung and G. Pighizzini, editors, *Proc. of DCFS'06*, NMSU-CS-2006-001 in Comp. Sci. Tech. Report, pages 58–69, 2006. NMSU.

[AR06] M. Almeida and R. Reis. Efficient Representation of Integer Sets. Tech. Rep. DCC-2006-06, DCC-FC&LIACC, Univ. do Porto, 2006.

[BN] F. Bassino, C. Nicaud. Enumeration and random generation of accessible automata. *Theoret. Comput. Sci*, 381(1-3):86–104, 2007.

[CP05] J.-M. Champarnaud, T. Paranthoën. Random generation of DFAs. *Theoret. Comput. Sci*, 330(2):221–235, 2005.

[DFLS04] P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability & Computing*, 13(4-5):577–625, 2004.

[DKS02] M. Domaratzki, D. Kisman, and J. Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486, 2002.

[Dom04] M. Domaratzki. Combinatorial interpretations of a generalization of the Genocchi numbers. *Jour. of Int. Sequences*, 7(04.3.6), 2004.

[Dom06] M. Domaratzki. Enumeration of formal languages. *Bull. EATCS*, 89(113-133), June, 2006.

[Har65] M. A. Harrison. A census of finite automata. *Canad. J. Math.*, 17:100–113, 1965.

[Hop71] J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Proc. Inter. Symp. on the Theory of Machines and Computations*, pages 189–196, Haifa, Israel. AP. 1971.

[HP67] F. Harary and E. M. Palmer. Enumeration of finite automata. *Information and Control*, 10:499–508, 1967.

[HP73] F. Harary and E. M. Palmer. *Graphical Enumeration*. AP, 1973.

[Kor78] A. Korshunov. Enumeration of finite automata. *Problemy Kibernetiki*, 34:5–82, 1978.

[Lis69] V. A. Liskovets. The number of initially connected automata. *Kibernetika*, 3:16–19, 1969. (Engl.tr:*Cybernetics*, 4 (1969), 259-262).

[Lis06] V. A. Liskovets. Exact enumeration of acyclic deterministic automata. *Disc. Applied Math.*, 154(3):537–551, March 2006.

[Lot05] M. Lothaire. *Applied Combinatorics on Words*. Number 105 in Encyclopedia of Mathematics and its Applications. CUP, 2005.

[MR05a] N. Moreira and R. Reis. Interactive manipulation of regular objects with FAdo. In *Proc. of 2005 Inn. and Tech. in Computer Science Education (ITiCSE 2005)*, pages 335–339. ACM, 2005.

[MR05b] N. Moreira and R. Reis. On the density of languages representing finite set partitions. *Jour. of Int. Sequences*, 8(05.2.8), 2005.

[Nar77] H. Narushima. *Principles of inclusion-exclusion on semilattices and its applications.* PhD thesis, Waseda Univ., Tokyo, 1977.

[Nic00] C. Nicaud. *Étude du comportement en moyenne des automates finis et des langages rationnels.* PhD thesis, Université de Paris 7, 2000.

[FAdo] FAdo project. http://www.ncc.up.pt/fado.

[RMA05] R. Reis, N. Moreira, M. Almeida. On the representation of finite automata. In C. Mereghetti, B. Palano, G. Pighizzini, and D.Wotschke, editors, *Proc. of DCFS'05*, number 06-05 in Rap. Tec. DICO, Univ. di Studi Milano, pages 269–276. IFIP. 2005.

[Rob85] R. W. Robinson. Counting strongly connected finite automata. In *Graph Theory with Applications to Algorithms and Computer Science*, pages 671–685. Wiley, 1985.

[SF96] R. Sedgewick and P. Flajolet. *Analysis of Algorithms.* AW, 1996.

[Slo03] N. Sloane. The On-line Encyclopedia of Integer Sequences, 2003. http://www.research.att.com/~njas/sequences.