Enumeration and Generation of Initially Connected Deterministic Finite Automata

Marco Almeida Nelma Moreira Rogério Reis

Technical Report Series: DCC-2006-07
Version 1.2 September 2010

Departamento de Ciência de Computadores
&
Laboratório de Inteligência Artificial e Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL
Tel: 220 402 900 Fax: 220 402 950
`http://www.dcc.fc.up.pt/Pubs/`

Enumeration and Generation of Initially Connected Deterministic Finite Automata *

Marco Almeida Nelma Moreira Rogério Reis
{mfa,nam,rvr}@ncc.up.pt
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 (ICDFA's) 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 ICDFA's. The exact generation algorithm can be used to partition the set of ICDFA's in order to parallelize the counting of minimal automata (and thus of regular languages). A uniform random generator for ICDFA's is presented that uses a table of pre-calculated values. Based on the same table, an optimal coding for ICDFA's is obtained. We also establish an explicit relationship between our method and the one used by Nicaud *et al.*.

Keywords 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 characterisations, 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 (ICDFA's) 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 of ICDFA's but, to some extent, the set of regular languages. This canonical form was first used in the **FAdo** project [MR05a, pro] to test if two minimal DFA's are *isomorphic*. However a precise characterisation of this representation as regular languages of $[0, n-1]^\star$ allows an exact and ordered generator of ICDFA's 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]

---

[*] This report extends the work represented in [AMR06, RMA05a, RMA05b]

considered the enumeration of non-isomorphic DFA's (and connected DFA's) up to permutation of alphabetic symbols. With the same criteria, Narushima [Nar77] enumerated minimal DFA's. Liskovets [Lis69] and Robinson [Rob85] counted strongly connected DFA's and also non-isomorphic ICDFA's. The work of Korshunov, surveyed in [Kor78], enumerates minimal automata and gives estimates of ICDFA's 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 DFA's and of finite languages. Nicaud [Nic00], Champarnaud and Paranthoën [CP05] presented a method for randomly generating complete ICDFA's. Bassino and Nicaud [BN07] showed that the number of complete ICDFA's is $\Theta(n2^nS(kn,n))$, where $S(kn,n)$ is a Stirling number of the second kind.

In this paper we obtain a new formula for the number of non-isomorphic complete ICDFA's and we precisely relate our methods to those used by Nicaud *et al.* in the cited works. The exact generation algorithm developed can be used to partition the set of ICDFA's in order to parallelize the process of counting minimal automata, and thus counting regular languages. We also designed a uniform random generator for ICDFA's 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 ICDFA's.

The work reported it this paper was already partially presented in [RMA05b, AMR06] and is organised 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 (*i.e* ICDFAs without final state information). 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 Section 6 the methods for parallelizing the counting of regular languages. Using a table of pre-calculated values, in Section 7 is designed a uniform random generator and in Section 8 an optimal coding for ICDFA$_\emptyset$'s. In Section 9 the results of previous sections are extended to incomplete ICDFA$_\emptyset$'s. Section 10 concludes and addresses some future work.

2 Preliminaries

Given two integers, m and n, let $[m,n]$ be the set $\{i \in \mathbb{Z} \mid m \leq i \ \wedge \ i \leq n\}$.

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 $|Q|$. If otherwise stated, we assume that the transition function is total, so we consider *complete* DFA's. 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] (ICDFA) if for each state $q \in Q$ there exists a sequence $(q'_i)_{i\in[0,j]}$ of states and a sequence $(\sigma_i)_{i\in[0,j-1]}$ of symbols, for some $j < |Q|$, such that $\delta(q'_m, \sigma_m) = q'_{m+1}$, $q'_0 = q_0$ and $q'_j = q$. 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 DFA's. We denote by ICDFA$_\emptyset$ the structure of an ICDFA.

Two DFA's $(Q, \Sigma, \delta, q_0, F)$ and $(Q', \Sigma', \delta', q'_0, F')$ are called *isomorphic* (by states) if $|\Sigma| = |\Sigma'| = k$, there exist bijections $\Pi_1 : \Sigma \to [0, k-1]$, $\Pi_2 : \Sigma' \to [0, k-1]$ and a bijection

---

[1] Also called *accessible.*

$\iota : Q \to Q'$ such that $\iota(q_0) = q'_0$, 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's are *equivalent* if they accept the same language. Obviously, two isomorphic automata (with the same alphabet) 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 DFA's are unique up to isomorphism. Domaratzki *et al.* [DKS02] give 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 DFA's and by $g_k(n)$ the number of distinct languages accepted by DFA's, where

$$g_k(n) = \sum_{i=1}^{n} f_k(i). \tag{1}$$

3 String representation for ICDFA's

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 characterisation and enumeration.

Let us disregard the set of *final states* of a DFA. A *naive* representation of a DFA$_\emptyset$ can be obtained by the enumeration of its states and for each state a list of its transitions for each symbol. For the DFA$_\emptyset$ in Fig.1 we have:

$$[[A\ (\mathtt{a}: A, \mathtt{b}: B)],\ [B\ (\mathtt{a}: A, \mathtt{b}: E)], [C\ (\mathtt{a}: B, \mathtt{b}: E)], \\ [D\ (\mathtt{a}: D, \mathtt{b}: C)], [E\ (\mathtt{a}: A, \mathtt{b}: E)]]. \tag{2}$$

Given a complete DFA$_\emptyset$ (Q, Σ, δ, q_0) with $|Q| = n$ and $|\Sigma| = k$ and considering a total order over Σ, the representation can be simplified by omitting the alphabetic symbols. For our example, we would have

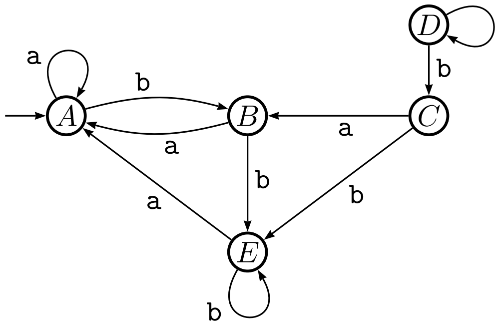

Figure 1: A DFA with no final states marked

$$[[A\ (A, B)], [B\ (A, E)], [C\ (B, E)], [D\ (D, C)], [E\ (A, E)]]. \tag{3}$$

The labels chosen for the states have a standard order (in the example, the alphabetic order). We can simplify the representation a bit if we use that order to identify the states,

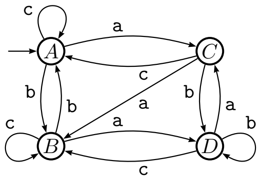

Figure 2: An ICDFA$_\emptyset$ for which the string representation is $[1, 2, 0, 2, 3, 0, 3, 0, 2, 1, 3, 2]$

and because we are representing complete DFA$_\emptyset$'s we can drop the inner tuples as well. We obtain

$$[0, 1, 0, 4, 1, 4, 3, 2, 0, 4]. \tag{4}$$

To obtain a canonical representation, given an order over the alphabet awe can consider an induced order in the states and transitions. A canonical order over the set of the states can be defined by exploring the automaton in a breadth-first way choosing at each node the outgoing edges in the order considered for Σ. The procedure is the following: let the first state 0 be the initial state q_0 of the automaton, 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... For the DFA$_\emptyset$ in Figure 1, this method induces an unique order for the first three states (A, B, E), but then we can arbitrate an order for the remaining states (C, D). Two different representations are thus admissible:

$$[0, 1, 0, 2, 0, 2, 3, 4, 1, 2] \text{ and } [0, 1, 0, 2, 0, 2, 1, 2, 4, 3]. \tag{5}$$

If we restrict this representation to ICDFA$_\emptyset$'s, then this representation is unique and defines an order over the set of its states. In the example, the DFA$_\emptyset$ restricted to the set of states $\{A, B, E\}$ is represented by $[0, 1, 0, 2, 0, 2]$.

For the ICDFA$_\emptyset$ represented in Figure 2, 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.

Formally, let Σ be an alphabet with $|\Sigma| = k$, and $\Pi : \Sigma \to [0, k-1]$ a bijection. Given an ICDFA$_\emptyset$ (Q, Σ, δ, q_0) with $|Q| = n$, let $\varphi : Q \to [0, n-1]$ be defined by the following algorithm:

```
define $\varphi(q_0)$ = 0
i = 0
s = 0
do
  for $\sigma \in \Sigma$ (according to the order induced by $\Pi$):
      if $\delta(\varphi^{-1}(s), \sigma) \notin \varphi^{-1}([0, i])$ then
          define $\varphi(\delta(\varphi^{-1}(s), \sigma))$ = i + 1
          i = i + 1
  s = s + 1
while s \leq 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 $\mathrm{ICDFA}_\emptyset$ there exist sequences $(q'_i)_{i \in [0,i]}$ and $(\sigma_i)_{i \in [0,j-1]}$ with $j < n$ such that $\delta(q'_m, \sigma_m) = q'_{m+1}$, for $m \in [0, j-1]$, $q'_0 = q_0$ and $q'_j = q$. We have $\varphi(q'_0) = 0$. For $m \in [0, j-1]$, if $q'_m \in \varphi^{-1}([0, n-1])$ then $q'_{m+1} \in \varphi^{-1}([0, n-1])$. Then $\varphi^{-1}([0, n-1]) = Q$, 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 $([0, n-1], \Sigma, \delta', 0)$ with $\delta'(i, \sigma) = \varphi(\delta(\varphi^{-1}(i), \sigma))$. Moreover the **canonical string** that represents this automaton, as described before, is: $(s_i)_{i \in [0,kn-1]}$ with $s_i \in [0, n-1]$ and $s_i = \delta'(\lfloor i/k \rfloor, \Pi^{-1}(i \bmod k))$, for $i \in [0, kn-1]$.*

Lemma 3 *Let $(s_i)_{i \in [0,kn-1]}$ be the canonical string of a complete* $\mathrm{ICDFA}_\emptyset$ *$\mathcal{A} = (Q, \Sigma, \delta, q_0)$ with $|Q| = n$ and $|\Sigma| = k$, then:*

$$(\exists j \in [0, n-1])\; s_j = n-1, \tag{\textbf{R0}}$$

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

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

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 exists a state $r \in Q$, that for $m = \varphi(r)$, m that does not occur in the first km symbols of the string (the m first state descriptions). But $m \notin \{s_i \mid i \in [0, km-1]\} = \{\delta'(i, \sigma) \mid i \in [0, m-1], \sigma \in \Sigma\}$ means that m is not accessible from state 0 in $([0, n-1], \Sigma, \delta', 0)$, 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 $(s_i)_{i \in [0,kn-1]}$ with $s_i \in [0, n-1]$ satisfying **R1** and **R2** represents a complete* $\mathrm{ICDFA}_\emptyset$ *with n states over an alphabet of k symbols.*

Proof Let $S = \{s_i \mid i \in [0, kn-1]\}$. Because of **R2**, $(n-1) \in S$, and using **R1**, we have $S = [0, n-1]$. Thus let us consider the automaton $([0, n-1], [0, k-1], \delta, 0)$ where $\delta(r, \sigma) = s_{kr+\sigma}$. Trivially this defines a $\mathrm{DFA}_\emptyset$, so it only remains to show that it is initially connected. Let m be a state of the automaton. Because of **R2** there must exist $j < km$ such that $s_j = m$. This means that $\delta(\lfloor j/k \rfloor, j \mod k) = m$. 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 1 *There is a one-to-one mapping between $(s_i)_{i \in [0,kn-1]}$ with $s_i \in [0, n-1]$ satisfying rules **R1** and **R2**, and the non-isomorphic* $\mathrm{ICDFA}_\emptyset$*'s with n states, over an alphabet of size k.*

This canonical representation can be extended to incomplete $\mathrm{ICDFA}_\emptyset$'s ($\mathrm{IDFA}_\emptyset$), 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 (normally called *dead-state*) are into itself. It is also easy to verify that the representation is unique for non-isomorphic IDFA$_\emptyset$'s.

For each canonical string representing an ICDFA$_\emptyset$, if we add a sequence of *final states*, we obtain a **canonical form** for ICDFA's. The same applies to IDFA$_\emptyset$, with the proviso that the *dead-state* cannot be final.

4 Enumeration of ICDFA's

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 of an ICDFA$_\emptyset$, let $(f_j)_{j\in[1,n-1]}$ 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**):

$$(\forall j \in [2, n-1])(f_j > f_{j-1}), \tag{\textbf{G1}}$$

$$(\forall m \in [1, n-1])\ (f_m < km). \tag{\textbf{G2}}$$

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

Theorem 2 *Given k and n, the number of sequences $(f_j)_{j\in[1,n-1]}$, $F_{k,n}$, 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 (generalised) Fuss-Catalan numbers.

Proof The first equality follows from the definition of the $(f_j)_{j\in[1,n-1]}$. 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 **OEIS** [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.

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 $j \in [0, n-1]$. For example, the bitmaps of the trees in Figure 3 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, $(f_i)_{i\in[1,n-1]}$, *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 $(f_i)_{i\in[1,n-1]}$ verifies **G1**. For **G2**, note that for the each internal node the outdegree of the previous internal nodes

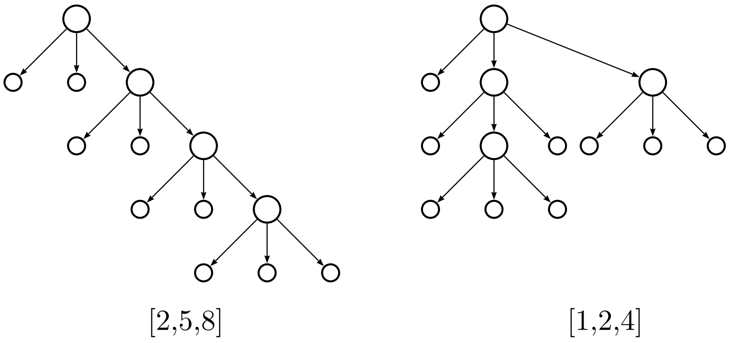

Figure 3: Two 3-ary trees with 4 internal nodes and the correspondent sequence of flags.

is k. Conversely, given a sequence of flags $(f_i)_{i\in[1,n-1]}$, we construct the bitmap such that b_{f_i}=1 for $i \in [1, n-1]$ and $b_j = 0$ for the remaining values, for $j \in [0, kn-1]$. 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 $(f_j)_{j\in[1,n-1]}$, all the remaining symbols, s_i, can be generated according to the following rules:

$$i < f_1 \;\Rightarrow\; s_i = 0, \tag{\textbf{G3}}$$

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

$$i > f_{n-1} \;\Rightarrow\; s_i \in [0, n-1]. \tag{\textbf{G5}}$$

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 complete ICDFA$_\emptyset$'s, $B_{k,n}$, the formula $B_{k,n} = \frac{b_{k,n}}{(n-1)!}$ where $b_{k,1} = 1$ and for $n > 1$ $b_{k,n} = n^{kn} - \sum_{1\le j<n} \binom{n-1}{j-1} n^{k(n-j)} b_{k,j}$. 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 (8) we will derive is a direct positive summation.

First, let us consider the set of strings $(s_i)_{i\in[0,kn-1]}$ with $s_i \in [0, n-1]$ 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 $[0, n-1]$, so there always n^k choices. For the others they belong to the language $A_n \cap [0, n-1]^{kn-k}$, where for $c > 0$,

$$A_c = L(0^\star \prod_{j=1}^{c-1} j(0 + \cdots + j)^\star). \tag{6}$$

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

Theorem 3 *For all* $n,\ k \geq 1$, $B_{k,n} \leq S(k(n-1)+1, n)n^k \leq nS(kn, n)$.

Proof The second inequality follows from the recursive definition of Stirling numbers of the second kind and the following propriety, $S(n-i,m) \leq \frac{1}{n^i}S(n,m)$, for $i \in [0, n-m]$.

Our bound is slightly more tight 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 $(f_j)_{j\in[1,n-1]}$ and considering $f_n = kn$, the correspondent set of canonical strings can be represented by the regular expression:

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

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

Considering the set of sequences of flags (see Theorem 2) the set of canonical strings can be represented by the regular expression:

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

For $n = 3$ and $k = 2$ we have

$$(01+1(0+1))((0+1)2+2(0+1+2))(0+1+2)^2+12(0+1+2)^4,$$

and the number of these strings is $(1+2)((2+3)3^2)+3^4 = 216$.

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

$$\prod_{j=1}^{n} j^{f_j-f_{j-1}-1}, \tag{7}$$

Theorem 4 *The number of canonical strings $(s_i)_{i\in[0,kn-1]}$ representing* ICDFA$_\emptyset$*'s with n states over an alphabet of k symbols is given by*

$$B_{k,n} = \sum_{f_1=0}^{k-1}\sum_{f_2=f_1+1}^{2k-1}\sum_{f_3=f_2+1}^{3k-1}\cdots\sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1}\prod_{j=1}^{n} j^{f_j-f_{j-1}-1}, \tag{8}$$

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

In Section 8 we give another recursive definition for $B_{k,n}$ more adequate for tabulation.

Corollary 1 *The number of non-isomorphic* ICDFA*'s with n states over an alphabet of k symbols is $2^n B_{k,n}$.*

4.1 Enumeration of incomplete ICDFA$_\emptyset$'s

Theorem 4 can be easily extended to obtain a formula for incomplete ICDFA$_\emptyset$'s, i.e., IDFA$_\emptyset$'s. We assume that the number of states of an IDFA$_\emptyset$ does not count the dead-state.

Theorem 5 *The number of canonical strings* $(s_i)_{i\in[0,kn-1]}$ *and* $-1 \leq s_i \leq n-1$ *representing incomplete* IDFA$_\emptyset$ *with* n *states over an alphabet of* k *symbols is given by*

$$B^1_{k,n} = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \sum_{f_3=f_2+1}^{3k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} (\prod_{j=1}^{n} (j+1)^{f_j-f_{j-1}-1}), \tag{9}$$

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

Corollary 2 *The number of non-isomorphic* IDFA*'s with* n *states over an alphabet of* k *symbols is* $2^n B^1_{k,n}$*.*

4.2 Analysis of the Nicaud *et al.* Method

Champarnaud and Paranthoën [CP05], generalising 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 sets of this words $\mathcal{P}$ 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 generalised 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 $i \in [2, n-1]$ and $f_n = p_n + f_{n-1} + 2$. It is obvious that $(f_i)_{i\in[1,n-1]}$ satisfies **G1**. To prove that it satisfies **G2**, note that $f_i = (i-1) + \sum_{j=1}^{i} p_j$, for $i \in [1, n-1]$. 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 generalised tuples, akin the ones we present in Section 7. The upper bound refered above is obtained, disregarding the first condition in the definition of the generalized tuples.

5 Generating ICDFA$_\emptyset$'s

In this section, we present 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 ICDFA$_\emptyset$ is represented by the string $0^{k-1}10^{k-1}\ldots(n-1)0^k$

---

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

and the last is represented by $12\ldots(n-1)(n-1)^{(k-1)n+1}$. According to the rules **G1- G5**, we first generate a sequence of flags, and then, for each one, the set of strings representing the ICDFA$_\emptyset$'s in lexicographic order. The initial sequence of flags is $(ki-1)_{i\in[1,n-1]}$. The algorithm to generate the next sequence of flags is the following:

```
def nextflags($i$):
  if $i$==1 then $f_i = f_i - 1$
  else
      if ($f_i$-1 == $f_{i-1}$) then
          $f_i = k*i-1$
          nextflags($i-1$)
      else $f_i = f_i - 1$
```

To generate a new sequence, we must call **nextflags(n-1)**. Given the rules **G1** and **G2** the correctness of the algorithm is easily proved. When a new sequence of flags is generated, the first ICDFA$_\emptyset$ is represented by a string with 0s in all positions different from the flags (i.e., the lower bounds in rules **G3**–**G5**). The following strings, with the same sequence of flags, are computed lexicographically using the procedure **nexticdfa**, called with $a = n-1$ and $b = k-1$:

```
def nexticdfa($a, b$):
    $i = a*k+b$
    if $a < n-1$ then
        while $i \in (f_j)_{j\in[1,n-1]}$:
            $b = b - 1$
            $i = i - 1$
        $f_j$ = the nearest flag not exceeding $i$
        if $s_i == s_{f_j}$ then
            $s_i = 0$
            if $b == 0$ then nexticdfa($a-1, k-1$)
            else nexticdfa($a, b-1$)
        else $s_i = s_i + 1$
```

The generator can then be implemented by the following procedure:

```
def generator():
    if islast($(s_i)_{i\in[0,kn-1]}$) then
        return None
    if isfull($(s_i)_{i\in[0,kn-1]}$) then
        nextflags($n-1$)
        reset()
    nexticdfa($n-1, k-1$)
    return $(s_i)_{i\in[0,kn-1]}$
```

where **islast**() tests if the current string represents the last automaton; **isfull**() tests if the current string is the last automaton for a given sequence of flags,namely $s_l = j$ for $l \in [f_j+1, f_{j+1}-1]$, with $j \in [1, n-1]$ and $i \in [0, kn-1]$; and **reset**() computes the first automaton for a new sequence of flags (with 0s in every position different from the flags).

$k:n$	ICDFA$_\emptyset$	Time		
		h	m	s
2:2	12			0.000
3:2	56			0.000
2:3	216			0.000
4:2	240			0.000
5:2	992			0.000
6:2	4032			0.000
2:4	5248			0.000
3:3	7965			0.000
7:2	16256			0.000
8:2	65280			0.000
2:5	160675			0.008
4:3	243000			0.013
9:2	261632			0.021
10:2	1047552			0.065
3:4	2128064			0.100
2:6	5931540			0.304
5:3	6903873			0.384
6:3	190505196			9.881
2:7	256182290			13.879
4:4	642959360			31.766
3:5	914929500			43.400
7:3	5192233245		7	9.193
2:8	12665445248		12	48.542
8:3	140764942800	3	21	34.260
5:4	175483321344	3	39	49.899
3:6	576689214816	11	49	32.790
2:9	705068085303	12	10	51.000
4:5	3508208993750	71	52	28.92

Table 1: Times for the generation of all ICDFA$_\emptyset$'s for small values of k and n, ordered by magnitude.

The time complexity of the generator is linear in the number of automata. As an example, for $k = 2$ and $n = 9$ it took about 12 hours to generate all the 705068085303 ICDFA$_\emptyset$'s, using a AMD Athlon at 2.5GHz. In Table 1 we present the time for the generation of all ICDFA$_\emptyset$'s for some values of k and n.

Finally, for the generation of ICDFA's we only need to add to the string representation of an ICDFA$_\emptyset$, a string of n 0's and 1's, correspondent to one of the 2^n possible choices of final states.

6 Counting Regular Languages (in Slices)

To obtain the number of languages accepted by DFA's with n states over an alphabet of k symbols, we can generate all ICDFA's, determine which of them are minimal ($f_k(n)$) and calculate the value of $g_k(n)$, by Equation (1). However, even for small values of n and k the total number of ICDFA's can be considerable. As an example, for $n = 3$ and $k = 2$ there are 1728 ICDFA's that we can generate and minimize in less than a second using an AMD Athlon 64 3800+. But if we take $n = 7$ and $k = 2$, the same Athlon 64 3800+ would require about 344 hours to generate and minimize all the 32791333120 ICDFA's. For even greater values of n or k this is an intractable problem.

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. For very small values of n and k ($n + k < 16$) we represented sets as bitmaps and for set partitions AVL trees [avl] were used [AR06].

6.1 Make it parallel

If we manage to partition the search space in a safe way, we can parallelize the problem and execute several instances of the minimization algorithm simultaneously. Because our method generates ICDFA$_\emptyset$ in an ordered way, we can very easily consider intervals of arbitrary size from any family of ICDFA$_\emptyset$'s with n states over an alphabet of k symbols. We call these intervals *slices*. They are independent and can be simultaneously given to the minimization algorithm. A slice is represented by a tuple (A_1, F, A_2) where A_1 is the first ICDFA$_\emptyset$, F is the sequence of flags of A_1 and A_2 is the last ICDFA$_\emptyset$. Based on this procedure, the following method can be used to enumerate all the regular languages recognized by a given family of ICDFA$_\emptyset$'s taking advantage of an environment with m CPUs available:

```
Let $S$ be an array of $s <= m$ slices, each corresponding to size ICDFA$_\emptyset$'s
while $i < s$
    spawn_minimize_slice($S[i]$)
    $i = i + 1$
```

The **spawn_minimize_slice()** procedure starts a new process, on an available CPU. For each one of the *size* ICDFA$_\emptyset$'s, and for each possible set of final states, this process will test the minimality of the ICDFA. Because there are m simultaneous processes, the actual time needed to enumerate the regular languages is roughly t/m, where t is the time that would be required if a single CPU were enumerating the same family of ICDFA's. For the generation of ICDFA's, we used the observation by Domaratzki *et al.* [DKS02], that is enough to test 2^{n-1} sets of final states, using the fact that a DFA is minimal *iff* its complementary automaton is minimal too.

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).

6.2 Creating the slices

The task of creating the slices, can be achieved as follows. Given an initial automaton, A_1, and an integer, *size*, the `nexticdfa()` procedure can be called *size* times or until the final automaton is reached. The algorithm returns the automaton A_2 and, if exists, the *next* one. The *next*, if exists, is memorised and will be used as the first automaton (A_1) for the next slice.

However using the bijections described in Section 8, a much efficient method can be used. Given a canonical string for an ICDFA$_\emptyset$ of size n over an alphabet of k symbols, we can compute its number in the generation order and vice-versa, *i.e.*, given a number less than $B_{k,n}$, we can obtain the corresponding ICDFA$_\emptyset$. So, instead of generating all the ICDFA$_\emptyset$'s so that then we can save the strings that represent slices, each slice can be given by an tuple of integers and only for those numbers is necessary to obtain the correspondent ICDFA$_\emptyset$.

For $n = 3$ and $k = 2$, for example, taking slices of 100 ICDFA$_\emptyset$'s we get the following sequence:

$$\{(0, 99), (100, 199), (200, 215)\}$$

Now, instead of generating 216 ICDFA$_\emptyset$'s, we can compute the string representation of only 6. The sequence of flags is also obtained from this conversion.

6.3 Experimental results

For this experiment we used two approaches. We developed a simple slave management system – called `Hydra` — based on `Python` threads, that was composed by a server and a variable set of *slaves*. In this case, the slaves can be any computer[3]. For each slice a process was executed via `ssh`, and the result was returned to the server. Another approach was to use a computer grid, in particular 24 AMD Opteron 250 2.4GHz (dual core).

In Table 2, we summarise some experimental results. Most of the values for $k = 2$ and $k = 3$, were already given by Domaratzki *et al.* in [DKS02] and the new results are in bold in the table. For $k = 2, n = 8$ we have divided the universe of ICDFA$_\emptyset$'s in 254 slices and the estimated CPU time for each one to be processed is 11 days.

Moreover, the slicing process can give new insights about the distribution of minimal automata. Figure 4 presents two examples of the values obtained for the rate of minimal DFA's. For $n = 7$ and $k = 2$ we give the percentage of minimal automata for each of the 257 slices we had used to divide the search space (256182290 ICDFA$_\emptyset$'s, 32791333120 ICDFA's). Each slice had about 100000 ICDFA$_\emptyset$'s, and so 128000000 ICDFA's, and it took about 2 minutes to conclude the process. The whole set of automata was processed in less than 3 hours of real time of a 24 CPUs grid, that corresponds to 70 hours of CPU time.

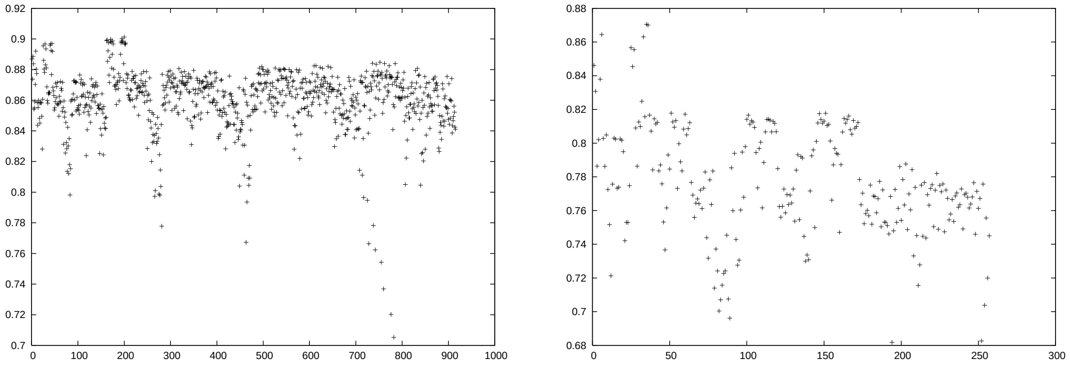

Figure 4: Rate of minimal DFA's with ($k = 3$,$n = 5$) for 915 slices and with ($k = 2$,$n = 7$) for 257 slices.

---

[3] We used all the normal desktop computers of our colleagues in the CS Department.

	n	ICDFA$_\emptyset$	ICDFA	Minimal ($f_k(n)$)	Minimal %	Time (s)
$k = 2$	1	1	2	2	100%	0
	2	12	48	24	50%	0
	3	216	1728	1028	59%	0.018
	4	5248	83968	56014	66%	0.99
	5	160675	5141600	3705306	72%	79.12
	6	5931540	379618560	286717796	75%	8700
	7	256182290	32791333120	**25493886852**	77%	1237313
	8	12665445248	3242353983488	**2567534031190**	79%	
$k = 3$	1	1	2	2	100%	0
	2	56	224	112	50%	0.002
	3	7965	63720	41928	65%	0.7
	4	2128064	34049024	26617614	78%	494.72
	5	914929500	29277744000	**25184560134**	86%	652703
$k = 4$	1	1	2	2	100%	0
	2	240	960	480	50%	0.01
	3	243000	1944000	1352732	69%	23.5
	4	642959360	10287349760	**7756763336**	75%	184808
$k = 5$	1	1	2	2	100%	0
	2	992	3968	1984	50%	0.041
	3	6903873	55230984	36818904	66%	756.2

Table 2: Performance and number of minimal automata.

7 Uniform Random Generation

The canonical strings for ICDFA$_\emptyset$'s (Section 3) 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 $(f_i)_{i\in[1,n-1]}$ 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 $0 \le i < n$ 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 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 $s_0s_1\cdots s_i$ is already defined, then the process can be described by:

```
$r = \mathbf{random}(1, \sum_{j=i+1}^{km-1} N_{m,j})$
for $j = i + 1$ to $km - 1$:
    if $r \in \left[\sum_{l=i}^{j-1} N_{m,l}, \sum_{l=i}^{j} N_{m,l}\right]$ 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 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 (8), 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, given k and n:

$$\begin{aligned} N_{n-1,j} &= n^{kn-1-j} && \text{with } j \in [n-2, k(n-1)-1], \\ N_{m,j} &= \sum_{i=0}^{(m+1)k-j-2} (m+1)^i N_{m+1,j+i+1} && \text{with } m \in [1, n-2], \\ & && j \in [m-1, km-1]. \end{aligned} \tag{10}$$

The second equation, can have an even simpler form:

$$\begin{aligned} N_{m,km-1} &= \sum_{i=0}^{k-1} (m+1)^i N_{m+1,km+i} && \text{with } m \in [1, n-2], \\ N_{m,i} &= (m+1) N_{m,i+1} + N_{m+1,i+1} && \text{with } m \in [1, n-2], \\ & && i \in [m-1, km-2]. \end{aligned} \tag{11}$$

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 this 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 $\mathcal{O}(n^3 k)$.

The algorithm is described by the following:

```
$g = -1$
for $i = 1$ to $n - 1$:
    $f =$ generateflag($i, g + 1$)
    for $j = g + 1$ to $f - 1$:
        print random($0, i - 1$)
    print $i$
    $g = f$

def generateflag($m, l$):
    $r =$ random( $0$ , $\sum_{i=l}^{km-1} m^{i-l} N_{m,i}$ )
    for $i = l$ to $km - 1$:
        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 3 were observed. It is possible, without unreasonable amounts of 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.

	$k = 2$	$k = 3$	$k = 5$	$k = 10$	$k = 15$
$n = 10$	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 3: Times for the random generation of 10000 automata (AMD Athlon 64 at 2.5GHz)

7.1 Statistical test of the random generator

Although the method used to generate random automata is, by its own construction, uniform, we used χ^2 test to evaluate the random generation quality. The universe of ICDFA$_\emptyset$'s with 6 states and 2 symbols has a total size of 5931540. This size is large enough for a test with some significance and it is still reasonable, both in time and space, to perform the test. We generated three different sets of 3000000 ICDFA$_\emptyset$'s and perform the test in each one. Because of the size of the data, we could not find any tabulated values for acceptance, and thus the following formula was used with $v = 30000000 - 1$ and x_p being the significance level (1% in this case):

$$v + 2\sqrt{vx_p} + \frac{3}{4}x_p^2 - \frac{2}{3}.$$

The size of the data sets and the repetition of the test for three times, is the recommended procedure by Knuth ([Knu81], pages 35–39). For the three experiments the values obtained were, respectively, 5933268.92456, 5925676.75108 and 5935733.28172, that are all smaller than the acceptance limit, that for this case was 5938980.75468.

8 Optimal coding of ICDFA$_\emptyset$'s

Given a canonical string for an ICDFA$_\emptyset$ 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 can obtain the corresponding ICDFA$_\emptyset$. This provides an optimal encoding for ICDFA$_\emptyset$'s, as defined by M. Lothaire [Lot05]. This bijection is accomplished by using the tables defined in Section 7 that correspond to partial sums of Equation (8). By expanding $N_{m,j}$ using Equations (10), we have:

Theorem 6 $B_{k,n} = \sum_{l=0}^{k-1} N_{1,l}$.

8.1 From ICDFA$_\emptyset$'s to Integers

Let $(s_i)_{i\in[0,kn-1]}$ be the canonical string of an ICDFA$_\emptyset$, and let $(f_j)_{j\in[1,n-1]}$ be the corresponding sequence of flags. From the sequence of flags we obtain the following number,

$$n_f = \sum_{j=1}^{n-1}\left(\prod_{m=1}^{j-1}(m+1)^{f_{m+1}-f_m-1}\right)\left(\sum_{l=f_j+1}^{kj-1}(j^{l-f_j}N_{j,l})\right), \tag{12}$$

which is the number of the first ICDFA$_\emptyset$ with flags $(f_j)_{j\in[1,n-1]}$. For each $j \in [1, n-1]$, the product $\prod_{m=1}^{j-1}(m^{f_{m+1}-f_m-1})$ corresponds to the number of strings $s'_0s'_1\dots s'_{f_j-1}$ that have flags $f_1, \ldots f_{j-1}$. The parameter l ranges over the possible values of flag j before f_j and the factor $j^{l-f_j} N_{i,l}$ counts the number of the correspondent strings $(s'_{f_j+1} \ldots s'_{kn-1})$.

Then, we must add the information provided by the rest of the elements of the string $(s_i)_{i\in[0,kn-1]}$:

$$n_r = \sum_{j=1}^{n-1} \left(\prod_{m=j+1}^{n-1} (m+1)^{f_{m+1}-f_m-1} \right) \left(\sum_{l=f_j+1}^{f_{j+1}-1} s_l (j+1)^{f_{j+1}-1-l}. \right) \tag{13}$$

The number of the canonical string $(s_i)_{i\in[0,kn-1]}$ is $n_s = n_f + n_r$.

8.2 From Integers to ICDFA$_\emptyset$'s

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

```
//obtainning the flags
$s = 1$
for $i$ = 1 to $n-1$:
    $j = k * i - 1$
    $p = i^{j-f_{i-1}-1}$
    while $j >= i-1$ and $m \ge p * s * N_{i,j}$:
        $m$ = $m$ - $N_{i,j} * p * s$
        $j = j - 1$
        $p = p/i$
    $s = s * i^{j-f_{i-1}-1}$
    $f_i = j$
//the rest
$i = k * n - 1$
$j = n - 1$
while $m > 0$ and $j > 0$:
    while $m > 0$ and $i > f_j$:
        $s_i = m \mod (j+1)$
        $m = m \div (j+1)$
        $i = i - 1$
    $i = i - 1$
    $j = j - 1$
```

9 Randon generation and optimal coding for IDFA$_\emptyset$'s

The recursive formulas $N_{i,j}$ can be extended to deal with incomplete ICDFA$_\emptyset$'s:

$$\begin{array}{rcll} N^1_{n-1,j} & = & (n+1)^{kn-1-j} & \text{with } j \in [n-2, (n-1)k-1], \\ N^1_{m,km-1} & = & \sum\limits_{i=0}^{k-1} (m+2)^i N^1_{m+1,km+i} & \text{with } m \in [1, n-2], \\ N^1_{m,i} & = & (m+2) N^1_{m,i+1} + N^1_{m+1,i+1} & \text{with } m \in [1, n-2], \\ & & & i \in [m-1, km-2]. \end{array}$$

And, we have:

Theorem 7

$$B^1_{k,n} = \sum_{l=0}^{k-1} 2^l * N^1_{1,l}. \tag{14}$$

For k=2, $B^1_{2,n}$ is sequence **A107668** in Sloane **OEIS** [Slo03].

9.1 Uniform Random Generation

The algorithm for a uniform random generator can be trivially modified for the generation of IDFA$_\emptyset$'s. Letting the parameter t be 0 for the generation of ICDFA$_\emptyset$'s (and N be renamed as N^0) and 1 for IDFA$_\emptyset$'s, we have the following general algorithm:

```
$g = -1$
for $i = 1$ to $n - 1$:
    $f = \mathbf{generateflag}(i, g + 1, t)$
    for $j = g + 1$ to $f - 1$:
        print $\mathbf{random}(0, i - 1)$
    print $i$
    $g = f$

def $\mathbf{generateflag}(m, l, t)$:
    $r = \mathbf{random}(0, \sum_{i=l}^{km-1} (m + t)^{i-l} N^t_{m,i})$
    for $i = l$ to $km - 1$:
        if $r < (m + t)^{i-l} N^t_{m,i}$
        then return $i$
        else $r = r - (m + t)^{i-l} N^t_{m,i}$
```

9.2 Optimal coding for IDFA$_\emptyset$'s

In the same way we can obtain formulae for the number of an IDFA$_\emptyset$, and, reciprocally, given an integer $0 \leq m < B^1_{k,n}$, we can obtain a canonical string for an IDFA$_\emptyset$. In the conversion from IDFA$_\emptyset$'s to integers, besides the use of $N^1_{i,j}$ we must add one to the base of the powers ($m + 1$ and j in (12) and $m + 1$ and $j + 1$ in (13)).

The general code (for both ICDFA$_\emptyset$'s and IDFA$_\emptyset$'s) is as follows:

$$n_f = \sum_{j=1}^{n-1} \left(\prod_{m=1}^{j-1} (m + 1 + t)^{f_{m+1} - f_m - 1} \right) \left(\sum_{l=f_j+1}^{kj-1} (j + t)^{l - f_j} N^t_{j,l} \right), \tag{15}$$

$$n_r = \sum_{j=1}^{n-1} \left(\prod_{m=j+1}^{n-1} (m + 1 + t)^{f_{m+1} - f_m - 1} \right) \left(\sum_{l=f_j+1}^{f_{j+1}-1} s_l (j + 1 + t)^{f_{j+1} - 1 - l} \right). \tag{16}$$

Likewise, for the conversion from integers to IDFA$_\emptyset$s, we must: take $f_0 = -1$, add one to the base of the powers in line 4 and line 9, and to the divisor in line 8 and line 16; and line 15 becomes $s_i = m \mod (j + 2) - 1$.

The general code (for both ICDFA$_\emptyset$'s ($t = 0$) and IDFA$_\emptyset$'s, ($t = 1$)) is as follows:

```
//obtainning the flags
$s = 1$
for $i$ = 1 to $n-1$:
    $j = k*i-1$
    $p = (i+t)^{j-f_{i-1}-1}$
    while $j >= i-1$ and $m \geq p*s*N_{i,j}$:
        $m = m - N_{i,j}*p*s$
        $j = j-1$
        $p = p/(i+t)$
    $s = s*(i+t)^{j-f_{i-1}-1}$
    $f_i = j$
//the rest
$i = k*n-1$
$j = n-1$
while $m > 0$ and $j+t > 0$:
    while $m > 0$ and $i > f_j$:
        $s_i = m \mod (j+1+t) - t$
        $m = m \div (j+1+t)$
        $i = i-1$
    $i = i-1$
    $j = j-1$
```

10 Conclusion

The methods here presented were implemented and tested to obtain both exact and approximate values for the density of minimal automata. A web interface to the random generator can be found in the **FAdo** project web page [pro].

Champarnaud *et al.* in [CP05], checked a conjecture of Nicaud that for $k = 2$ the number of minimal ICDFA's is about 80% of the total, by sampling automata with 100 states (for all possible number of final states). Our results also corroborate that conjecture, being the exact values for some small values of n and samples for greater values. In particular, for $k = 2$ and $n = 100$ we obtained the same results as Champarnaud *et al.*. It seems that for $k > 2$ almost all ICDFA's are minimal. For $k = 3, 5$ and $n = 100$ that was also checked by Champarnaud *et al.*. For a confidence interval of 99% and significance level of 1% the following table presents the percentages of minimal ICDFA's for several values of k and n, and each possible number of final states.

$k \backslash n$	5	6	7	8	9	10	20	40	80	160
3	85.8%	90.8%	93.3%	95.0%	96.1%	96.7%	98.7%	99.4%	99.7%	99.8%
5	93.0%	96.5%	98.2%	99.1%	99.5%	99.8%	100.0%	100.0%	100.0%	100.0%
7	93.7%	96.8%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	100.0%	–
9	93.7%	96.9%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	–	–
11	93.8%	96.9%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	–	–
13	93.7%	96.9%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	–	–

Of course, one challenge is try to understand why this happens. Bassino and Nicaud [BN] presented a random generator of ICDFA's based on Boltzmann Samplers, recently introduced by Duchon *et al.* [DFLS04]. However the sampler is uniform for partitions of a set with kn elements into n nonempty subsets 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 ICDFA's.

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, *Proceedings of the 8th Int. Workshop on Descriptional Complexity of Formal Systems (DCFS06)*, number NMSU-CS-2006-001 in Computer Science Technical Report, pages 58–69, Las Cruces, New Mexico, June 2006. NMSU.

[AR06] M. Almeida and R. Reis. Efficient Representation of Integer Sets. Technical Report DCC-2006-06, DCC - FC & LIACC, Universidade do Porto, December 2006.

[avl] Gnu libavl, binary search trees library. http://www.stanford.edu/ blp/avl/.

[BN] F. Bassino and C. Nicaud. Enumeration and random generation of accessible automata. Submitted.

[BN07] F. Bassino and C. Nicaud. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104, 2007.

[CP05] J.-M. Champarnaud and T. Paranthoën. Random generation of DFAs. *Theoretical Computer Science*, 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. *J. of Automata, Languages and Combinatorics*, 7(4):469–486, 2002.

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

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

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

[Hop71] John 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, 1971. Academic Press.

[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.* Academic Press, 1973.

[Knu81] D. E. Knuth. *The Art of Computer Programming. Seminumerical Algorithms.*, volume 2. Addison Wesley, 2nd edition, 1981.

[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. (in Russian; Engl. transl: *Cybernetics*, 4 (1969), 259-262).

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

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

[MR05a] N. Moreira and R. Reis. Interactive manipulation of regular objects with FAdo. In *Proceedings of 2005 Innovation and Technology 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. *Journal of Integer 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.

[pro] FAdo project. FAdo: tools for formal languages manipulation. http://www.ncc.up.pt/fado.

[RMA05a] R. Reis, N. Moreira, and M. Almeida. On the representation of finite automata. Technical Report DCC-2005-04, DCC - FC & LIACC, Universidade do Porto, April 2005.

[RMA05b] R. Reis, N. Moreira, and M. Almeida. On the representation of finite automata. In C. Mereghetti, B. Palano, G. Pighizzini, and D.Wotschke, editors, *7th International Workshop on Descriptional Complexity of Formal Systems*, number 06-05 in Rapporto Tecnico delo Departimento de Informatica e Comunicazione dela Università degli Studi di Milano, pages 269–276, Como, Italy, June 2005. International Federation for Information Processing.

[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.

A Enumeration of ICDFA$_\emptyset$'s

In this appendix we present the number of ICDFA$_\emptyset$'s non-isomorphic without final states for $n = 1..9$ states and $k = 2..10$ alphabetic symbols.

k= 2	n	
	1	1
	2	12
	3	216
	4	5248
	5	160675
	6	5931540
	7	256182290
	8	12665445248
	9	705068085303

k= 3	n	
	1	1
	2	56
	3	7965
	4	2128064
	5	914929500
	6	576689214816
	7	500750172337212
	8	572879126392178688
	9	835007874759393878655

k=4	n	
	1	1
	2	240
	3	243000
	4	642959360
	5	3508208993750
	6	34253071111894176
	7	544271118689873008532
	8	13147735690099619023732736
	9	458677874292647947600097994111

k=5	n	
	1	1
	2	992
	3	6903873
	4	175483321344
	5	11826519415721875
	6	1744085190146957291232
	7	494949686355427145872161111
	8	246491144450280856073240885624832
	9	200977948941552280610264305518977871090

k=6	n	
	1	1
	2	4032
	3	190505196
	4	46086910722048
	5	38056697263376203125
	6	841219431860064457132248896
	7	4231177947498521895020064109054 62
	8	43107988409138813783150335301212915630 08
	9	815107805311143262786462289568559768017449599 08

k= 7	n	
	1	1
	2	16256
	3	5192233245
	4	11921614605697024
	5	120315894541852283281250
	6	3976063029034767886935933510912
	7	35352134880615199574345580083298157131 4
	8	734846387070056298279788113670019663567320514 56
	9	321349870998846096288347260235824118088229800021316975 74

k=8	n	
	1	1
	2	65280
	3	140764942800
	4	3065045074098257920
	5	37774648436758551936 7187500
	6	186463110898012043254861617993372672
	7	29279032751153335518638081828541936916513450 4
	8	12405178593678541407417860030685556146529447406647377 92
	9	12533845162122187320986901745839566315023480777415952875118142242

k=9	n	
	1	1
	2	261632
	3	3807455329593
	4	786050986901533097984
	5	11826944437401392213967597 65625
	6	871747741776552611066960692066106 1954048
	7	2416632098931660293112357094492968484890071 50038885
	8	20862781312540752296309668431262192459252081308963680368459776
	9	486856205478210115424000890496937433528904062936219271916063746838 4235331

k=10	n	
	1	1
	2	1047552
	3	102881965757076
	4	2013789889909260529 17248
	5	369877137637580907432377565 4296875
	6	4070566200314093649826901757963106408770 07872
	7	199195425299637859859159104431333727959687905790340860554
	8	3503507735895374166049344715275101368355116712542005486766647022 71488
	9	18880963360320663330992680074514720259464695005177220879245815882004727092412 34833