On the Representation of Finite Automata

Marco Almeida  Nelma Moreira  Rogério Reis

Technical Report Series: DCC-2005-04
Version 1.4

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto
Rua do Campo Alegre, 823 4150 Porto, Portugal
Tel: +351+2+6078830 – Fax: +351+2+6003654
http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html
Abstract

We give a unique string representation, up to isomorphism, for initially connected
deterministic finite automata (ICDFA’s) with \( n \) states over an alphabet of \( k \) symbols.
We show how to generate all these strings for each \( n \) and \( k \), and how its enumeration
provides an alternative way to obtain the exact number of ICDFA’s.

1 Motivation

In symbolic manipulation environments for finite automata, it is important to have an
adequate representation of automata and, dependent upon their use, several representations
may be available. For example, for testing if two finite automata are isomorphic objects
or for (random) generation of automata, the representation must be compact and somehow
canonical. In the FAdo project [MR05a, fad] a canonical form is used to test if two minimal
DFA’s are isomorphic (i.e are the same up to renaming of states). In this paper we prove
the correctness of that representation and show how it can be used for the exact enumeration
and generation of initially connected deterministic finite automata (ICDFA).

The problem of enumeration of finite automata was considered by several authors since
early 1960s, in particular see Harrison [Har65], Robinson [Rob85], Harary and Palmer [HP67]
and Liskovets [Lis69] amongst many others. A survey may be found in Domaratzki et
al. [DKS02]. More recently, several authors examined related problems. Domaratzki et
al. [DKS02] studied the enumeration of distinct languages accepted by finite automata
with \( n \) states; Nicaud [Nic99], Champarnaud and Paranthoen [CP05, Par04] and Bassino
and Nicaud [BN] analysed several aspects of the average behaviour of regular languages;
Liskovets [Lis03] and Domaratzki [Dom04] gave (exact and asymptotic) enumerations of
acyclic DFA’s and of finite languages.

The paper is organised as follows. In the next section, we review some basic notions
and introduce some notation. Section 3 describes a string representation for deterministic
finite automata that is unique up to isomorphism for initially connected deterministic finite
automata. Section 4 presents an efficient method to generate those strings. Section 5 shows
how their enumeration provides an upper bound and the exact value for the number of
ICDFA’s. Section 6 and Appendix A report some implementation issues and final remarks.

2 Preliminaries

We first recall some basic notions from automata theory and formal languages, that can be
found in standard books [HMU00]. An alphabet \( \Sigma \) is a nonempty set of symbols. A string
over \( \Sigma \) is a finite sequence of symbols of \( \Sigma \). The empty string is denoted by \( \epsilon \). The set \( \Sigma^* \)
is the set of all strings over \( \Sigma \). A language \( L \) is a subset of \( \Sigma^* \). The density of a language
\( L \) over \( \Sigma \), \( \rho_L(n) \), is the number of strings of length \( n \) that are in \( L \), i.e.,
\( \rho_L(n) = |L \cap \Sigma^n| \).
A regular expression (r.e.) \( \alpha \) over \( \Sigma \) represents a language \( L(\alpha) \subseteq \Sigma^* \) and is inductively
defined by: \( \emptyset \), \( \epsilon \) and \( \sigma \in \Sigma \) are a r.e., where \( L(\emptyset) = \emptyset \), \( L(\epsilon) = \{ \epsilon \} \) and \( L(\sigma) = \{ \sigma \} \); if \( \alpha_1 \) and
\( \alpha_2 \) are r.e., \( (\alpha_1 + \alpha_2) \), \( (\alpha_1 \alpha_2) \) and \( \alpha^*_1 \) are r.e., respectively with \( L((\alpha_1 + \alpha_2)) = L(\alpha_1) \cup L(\alpha_2) \),
\( L((\alpha_1 \alpha_2)) = L(\alpha_1) L(\alpha_2) \) and \( L(\alpha_1^*) = L(\alpha_1)^* \). In this paper, we will use regular expressions
to represent descriptions of finite automata. A deterministic finite automaton (DFA) \( A \) is a quintuple
\( \langle Q, \Sigma, \delta, q_0, F \rangle \) where \( Q \) is a finite set of states, \( \Sigma \) is the alphabet, \( \delta : Q \times \Sigma \rightarrow Q \)
is the transition function, \( q_0 \) the initial state and \( F \subseteq Q \) the set of final states. We assume
that the transition function is total, so we consider only complete DFA’s. The size of a DFA
is the number of its states, $|Q|$. Normally, we are not interested in the labels of the states and we can represent them by an integer $0 \leq i < |Q|$. The transition function $\delta$ extends naturally to $\Sigma^*$: for all $q \in Q$ if $x = \epsilon$ then $\delta(q, \epsilon) = q$; if $x = y\sigma$ then $\delta(q, x) = \delta(\delta(q, y), \sigma)$. A DFA is *initially connected* (IC DFA) if for each state $q \in Q$ there exists a string $x \in \Sigma^*$ such that $\delta(q_0, x) = q$. Two DFA’s $A = (Q, \Sigma, \delta, q_0, F)$ and $A' = (Q', \Sigma, \delta', q_0', F')$ are called isomorphic (by states) if there exists a bijection $f : Q \to Q'$ such that $f(q_0) = q_0'$ and for all $\sigma \in \Sigma$ and $q \in Q$, $f(\delta(q, \sigma)) = \delta'(f(q), \sigma)$. Furthermore, for all $q \in Q$, $q \in F$ if and only if $f(q) \in F'$. The language accepted by a DFA $A$ is $L(A) = \{ x \in \Sigma^* \mid \delta(q_0, x) \in F \}$. Two DFA are equivalent if they accept the same language. Obviously, two isomorphic automata are equivalent, but two non-isomorphic automata may be equivalent. A DFA $A$ is minimal if there is no DFA $A'$ with fewer states equivalent to $A$. Trivially a minimal DFA is an IC DFA. Minimal DFA’s are unique up to isomorphism. We are mainly concerned with the representation of the transition function of DFA’s of size $n$ over an alphabet of $k$ symbols, so we disregard the set of final states and we consider only a quadruple $(Q, \Sigma, \delta, q_0)$ called the *structure* of an automaton and referred as DFA$_\emptyset$. For each of our representations, there will be $2^n$ DFA’s. We denote by IC DFA$_\emptyset$ the structure of an IC DFA. We consider that any integer variable has always a nonnegative value (if not otherwise stated). Let $[n]_0 = \{0, 1, \ldots, n\}$ and $[n] = \{1, \ldots, n\}$.

## 3 Representations towards a normal form

The method used to represent a DFA has a significative 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:

$$[[A (a : A, b : B)], [B (a : A, b : E)], [C (a : B, b : E)],$$
$$[D (a : D, b : C)], [E (a : A, b : E)]] .$$  \hspace{1cm} (1)

![Figure 1: A DFA with no final states marked](image)

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

---

1 Also called accessible.
our example, we would have

\[
[[A (A, B)], [B (A, E)], [C (B, E)], [D (D, C)], [E (A, E)]].
\]  

(2)

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, and because we are representing complete DFA's we can drop the inner tuples as well. We obtain

\[
[0, 1, 0, 4, 1, 4, 3, 2, 0, 4].
\]  

(3)

Because this representation depends on the order we label the states, we have more than one representation for each DFA. Can we have a canonical order for the set of the states? Let the first state be the initial state \(q_0\) of the automaton, the second state the first one to be referred (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 in the example, 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:

\[
[0, 1, 0, 2, 0, 2, 1, 2, 4, 3] \quad \text{and} \quad [0, 1, 0, 2, 0, 2, 1, 2, 4, 3].
\]  

(4)

If we restrict this representation to ICDFA's, then this representation is unique and defines an order over the set of its states. In the example, the DFA restricted to the set of states \(\{A, B, E\}\) is represented by \([0, 1, 0, 2, 0, 2]\). Let \(\Sigma = \{\sigma_i \mid i < k\}\), with \(\sigma_0 < \sigma_1 < \cdots < \sigma_{k-1}\).

Given an ICDFA \((Q, \Sigma, \delta, q_0)\) with \(|Q| = n\), the representing string is of the form \([(S_i)_{i<n}]\) with \(S_i \in [n-1]_0\) and \(S_i = \delta([i/k], \sigma_i \mod k)\). In Figure 2, we present an algorithm for obtain these string representation.

```
uniqueStr {
  S = []
  Ord(q0) = 0
  i = j = 0
  while i ≤ j :
    for l in [k−1]0 :
      if Ord(\(\delta((\text{Ord}−1(i), \sigma_l))\)) not defined then
        j = j + 1
        Ord(\(\delta((\text{Ord}−1(i), \sigma_l))\)) = j
        S = S + [Ord(\(\delta((\text{Ord}−1(i), \sigma_l))\))]
    i = i + 1
  return S
}
```

Figure 2: Obtaining the string representation of an ICDFA.

**Lemma 1.** Let \([(S_i)_{i<n}]\) be a representation of a complete ICDFA \((Q, \Sigma, \delta, q_0)\) with \(|Q| = n\) and \(|\Sigma| = k\), then:

\[
(\forall m > 1)(\forall i)(S_i = m \Rightarrow ((\exists j < i) S_j = m - 1)) \quad \text{(R1)}
\]

\[
(\forall m \in [n-1])(\exists j < km) S_j = m \quad \text{(R2)}
\]
Figure 3: \(R_1\) states that first references to each state occur sequentially.

\[
\begin{array}{cccc}
1 & 2 & n-2 & n-1 \\
\end{array}
\]

first occurences

Figure 4: \(R_2\) ensures that before the appearance of the set of transitions from a given state at least a reference to that state must appear in the string.

\[
\begin{array}{cccccc}
0 & 1 & \ldots & t-1 & t & m \\
\end{array}
\]

Proof. (of Lemma 1) The condition \(R_1\) establishes that a state label (greater than 1) can only occur after the occurrence of its predecessors. This is a direct consequence of the way we defined the representing string.

Suppose \(R_2\) does not verify, thus there exists a state \(m\) that does not occur in the first \(km\) symbols of the string (the \(m\) first state descriptions). Because the automaton is initially connected there must be a sequence of states \((m_i)_{i \leq l}\) and symbols \((\sigma_i)_{i \leq l}\) such that \(m_0 = 0, m_l = m\) and \(\delta(m_i, \sigma_i) = m_{i+1}\) for \(i < l\). We must have \(0 < m < m_{l-1}\) because \(m\) appears in the \(m_{l-1}\) description and we supposed no occurrences of \(m\) in the first \(m\) state descriptions. There must exist \(l' < l\) such that \(m_{l'-1} < m < m_{l'}\), implying that \(m_{l'} \in \{S_i \mid i < km\}\). This contradicts \(R_1\) because we are supposing that \(m \not\in \{S_i \mid i < km\}\) and \(m < m_{l'}\). Thus \(R_2\) is verified.

Lemma 2. Every string \([(S_i)_{i<kn}]\) with \(S_i \in [n-1]_0\) satisfying \(R_1\) and \(R_2\) represents a complete IC DFA with \(n\) states over an alphabet of \(k\) symbols.

Proof. Let \([(S_i)_{i<kn}]\) be a string in the referred conditions, and consider the associated automaton \(A\) using the string symbols as labels for the corresponding states. By its construction, \(A\) is a deterministic complete finite automaton structure. We only need to prove that it is initially connected. Let \(m\) be a state of the automaton.

A proof that \(m\) is reachable from the initial state 0 can be done by induction on \(m\).

If \(m = 0\) there is nothing to prove. If \(m = 1\) then, by \(R_2\), 1 must occur in the description of state 0, making state 1 reachable from state 0.

Let us suppose that every state \(m' < m\) is reachable from state 0 and prove that state \(m\) is reachable too. By \(R_2\), \(m\) occurs at least once before position \(km\), say in position \(km' + i\) with \(m' < m\) and \(i < k\). Then for some symbol \(\sigma\), \(\delta(m', \sigma) = m\). By induction hypothesis, state \(m'\) is reachable from state 0, thus state \(m\) is reachable too and the automaton is initially connected.

Now consider the string representation obtained for \(A\), \([(S'_i)_{i<kn}]\). By Lemma 1 it satisfies \(R_1\) and \(R_2\). It is easy to see that this representation is the same as \([(S_i)_{i<kn}]\). By \(R_1\),
$S_0 = S'_0$. Suppose that $(\forall i < j)(S_i = S'_i)$. Now we prove that $S'_j = S_j$. By R1, either $S_j \in \{S_i \mid i < j\}$ or $S_j = \max\{S_i \mid i < j\} + 1$. In the first case, there exists $l < j$ such that $S_j = S_l$ and, by induction hypothesis, $S_l = S'_l$, thus

$$S_j = \delta([j/k], \sigma_{j \mod k}) = S_l = \delta([l/k], \sigma_{l \mod k}) = S'_l = S'_j.$$

Analogously, by R1, in the second case we have that

$$S'_j = \max\{S_i \mid i < j\} + 1 = S_j.$$

\[\square\]

**Theorem 1.** There is a one-to-one mapping between strings $[(S_i)_{i<k}]$ with $S_i \in [n-1]_0$ satisfying R1 and R2, and the non-isomorphic ICDFAs with $n$ states, over an alphabet $\Sigma$ of size $k$.

**Proof.** Let $(Q, \Sigma, \delta, q_0)$ and $(Q', \Sigma, \delta', q'_0)$ be two ICDFAs and $[(S_i)_{i<k}]$ and $[(S'_i)_{i<k}]$ their representing strings. By Lemma 1, these strings satisfy R1 and R2. Suppose that $f : Q \rightarrow Q'$ is an isomorphism between the ICDFAs. Then $0 = q_0$ and $f(q_0) = q'_0 = 0$. Either $S_0 = \delta(q_0, \sigma_0) = q_0 = 0$ or $S_0 = \delta(s_0, \sigma_0) = 1$ (by R1).

i) If $S_0 = 0$ then $f(q_0) = f(\delta(q_0, \sigma_0)) = \delta'(q'_0, \sigma_0) = S'_0 = 0$, because $\delta(q_0, \sigma_0) = q_0$ implies $\delta'(f(q_0), \sigma_0) = f(q_0)$.

ii) If $S_0 = 1$ then $f(1) = \delta'(q'_0, \sigma_0) = S'_0 \neq 0$, thus $S'_0 = 1$, again by R1.

Supposing that $(\forall i < j)(S_i = S'_i \wedge f(S_i) = S'_i)$ we need to prove that $S_j = S'_j \wedge f(S_j) = S'_j$. Trivially we have $\{S_i \mid i < j\} = \{S'_i \mid i < j\}$. We know that $S_j = \delta([j/k], \sigma_{j \mod k})$, and by R2 there exists $l < j$ such that $[j/k] = S_l$ thus $f([j/k]) = f(S_l) = S_l = S'_l = [j/k]$ by induction hypothesis. We have

$$S'_j = \delta'(f([j/k]), \sigma_{j \mod k}) = f(\delta([j/k], \sigma_{j \mod k})) = f(S_j).$$

By R1, either $S_j \notin \{S_i \mid i < j\}$ or $S_j = \max\{S_i \mid i < j\} + 1$.

i) If $S_j \in \{S_i \mid i < j\}$ then there exists $l < j$ such that $S_j = S_l$ and $S_l = S'_l$. Then

$$\delta([j/k], \sigma_{j \mod k}) = \delta([l/k], \sigma_{l \mod k}) \Rightarrow f(\delta([j/k], \sigma_{j \mod k})) = f(\delta([l/k], \sigma_{l \mod k})) \iff \delta'(f([j/k]), \sigma_{j \mod k}) = \delta'(f([l/k]), \sigma_{l \mod k})$$

Thus $S_j = S_l$ implies $S'_j = S'_l$, and so $S'_j = S_j$.

ii) If $S_j = \max\{S_i \mid i < j\} + 1$ then $S'_j \notin \{S_i \mid i < j\}$ because if there exists a $l < j$ such that $S'_l = S'_j$ by the same reason as before $S_j \in \{S_i \mid i < j\}$. Thus, by R1 $S'_j = \max\{S_i \mid i < j\} + 1 = S_j$.

Conversely, by Lemma 2, we have that each string represents a IC DFA up to a compatible renaming of states, i.e., if two ICDFAs are represented by the same string, that representation defines a isomorphism between them.

These string representations lead to a normal representation for IC DFAs. For each of them, if we add a sequence of final states, we obtain a normal form for ICDFAs.
4 Generating automata

Normal representations for ICDF A∅’s (as presented above) can be used as compact computer representations for that kind of objects, but even though rules R1 and R2 are quit simple, it is not evident how to write an enumerative algorithm in an efficient way. In a string representing an ICDF A∅ with n states over an alphabet of k symbols, [(S_i)_i<n], let (f_j)_0<j<n be the sequence of indexes of the first occurrence of each state label j. That those indexes exist is a direct consequence of the way the string is constructed. Now consider

\[ b_1 = f_1 + 1, \]
\[ b_j = f_j - f_{j-1}, \text{ for } 2 \leq j \leq n - 1 \]
\[ b_n = kn - f_{n-1}. \]

Note that
\[ \sum_{l=1}^{j} b_l = f_j + 1, \text{ for } j \in [n-1]. \]

It is easy to see that
1. Rule R1 simply states that
   \[ (\forall 2 \leq j \leq n - 1)(b_j > 0). \]
   \[ (G1) \]
2. Rule R2 establishes that
   \[ (\forall m \in [n - 1])(f_m < km). \]
   \[ (G2) \]

To generate all the automata, for each allowed sequence of (b_j)_0<j<n we can generate all the remaining symbols S_i (those with i ∉ {f_j | 0 < j < n}) according to the following rules:

\[ i < b_1 \Rightarrow S_i = 0; \]
\[ (\forall j \in [n - 2])(f_j < i < f_{j+1} \Rightarrow S_i = [j]_0); \]
\[ i > f_{n-1} \Rightarrow S_i = [n-1]_0. \]

5 Enumeration of ICDFA’s

In this section we obtain a formula \( B_k(n) \) for the number of strings [(S_i)_i<n] representing ICDFA∅’s with n states over an alphabet of k symbols. Although it is already known a formula for the number of non-isomorphic ICDFA∅’s, we think that our method is new. Liskovets [Lis69] and, independently, Robinson [Rob85] gave for that number the formula

\[ H_k(n) = h_k(n) \]
\[ = \frac{h_k(n)}{(n-1)!}, \]
\[ \text{where } h_k(1) = 1 \text{ and for } n > 1 \]
\[ h_k(n) = n^{kn} - \sum_{1 \leq j < n} \binom{n-1}{j-1} n^{k(n-j)} h_k(j) \]
Note that $n^{kn}$ is the number of transition functions, from which we subtract the number of them that have $n - 1$, $n - 2, \ldots, 1$ states not accessible from the initial state. And then, we may divide by $(n - 1)!$, as the names of the remaining states (except the initial) are irrelevant. Reciprocally, the formula we will derive (of them that have $n$ $B$ irrelevant. Reciprocally, the formula we will derive.

First, let us consider the set of strings $[(S_i)_{i<n}]$ with $S_i \in [n-1]_0$ and satisfying only rule $R_1$. The number of these strings gives an upper bound for $B_k(n)$. This set can be given by $A_n \cap [n-1]_0^{kn}$, where for $c > 0$,

$$A_c = L(0^* + \sum_{i=1}^{c-1} 0^* \prod_{j=1}^{i} (0 + \cdots + j)^*). \quad (10)$$

These languages belong to a family of languages $L_c$ presented by Moreira and Reis [MR05b] and that represent partitions of $[n]$ with no more than $c \geq 1$ parts, i.e.,

$$L_c = L(\sum_{i=1}^{c} \prod_{j=1}^{i} (1 + \cdots + j)^*). \quad (11)$$

We have that $\rho_{A_c}(n) = \rho_{L_c}(n + 1)$ and that $\rho_{L_c}(n) = \sum_{i=1}^{c} S(n, i)$, where $S(n, i)$ are Stirling numbers of second kind. So we get that the number of strings of length $kn$ that are in $A_n$, is $\rho_{A_n}(kn) = \sum_{i=1}^{n} S(kn + 1, i)$. We have the proposition,

**Proposition 1.** For all $n$, $k \geq 1$, $B_k(n) \leq \sum_{i=1}^{n} S(kn + 1, i)$.

For $n = 3$ and $k = 2$, $B_2(3) \leq 365$. For $k = 2$, Bassino and Nicaud [BN] presented a better upper bound, namely that $B_2(n) \leq nS(2n, n)$.

Now let us consider only the rule $R_2$. This rule can be formulated as

$$\bigwedge_{m=1}^{n-1} \bigvee_{j=0}^{km-1} S_j = m. \quad (12)$$

From this formula it is easy to see that the strings $[(S_i)_{i<n}]$ with $S_i \in [n-1]_0$ and satisfying only rule $R_2$ can be represented by the regular expression

$$\bigcap_{m=1}^{n-1} \sum_{j=0}^{km-2} (0 + \cdots + (m - 1)j) m (0 + \cdots + (n - 1))^{kn-j-1}, \quad (13)$$

where we extended the operators of regular expressions to intersection.

Now in order to simultaneously satisfy rules $R_1$ and $R_2$, in formula (13), the first occurrence of $m$ must precede the one of $m - 1$, for $2 \leq m \leq n - 1$. These positions are exactly the sequence $(f_j)_{0<j<n}$ defined in Section 4. Given these positions and considering the correspondent sequence $(b_j)_{0<j<n}$ we obtain the regular expression:

$$\left( \prod_{j=1}^{n-1} (0 + \ldots + (j - 1))^{b_j - 1}j \right) (0 + \ldots + (n - 1))^{b_n - 1},$$

and we must consider the possible values of $(b_j)_{0<j<n}$, constrained to $G_1$ and $G_2$:

$$\sum_{b_1=1}^{k} \sum_{b_2=1}^{2k-b_1} \cdots \sum_{b_{n-1}=1}^{k(n-1) - \sum_{i=1}^{n-2} b_i} \left( \prod_{j=1}^{n-1} (0 + \ldots + (j - 1))^{b_j - 1}j \right) (0 + \ldots + (n - 1))^{b_n - 1}$$
For \( n = 3 \) and \( k = 2 \) we have
\[
(01 + 1(0 + 1))(0 + 1 + 2)(0 + 1 + 2)^2 + 12(0 + 1 + 2)^4,
\]
and the number of these strings is \((1 + 2)(2 + 3)^2 + 3^4 = 216\).

For each sequence \((b_j)_{0 < j < n}\) the number of strings \([(S_i)_{i < kn}]\) with \( S_i \in [n - 1][0] \) and satisfying \( R1 \) and \( R2 \) is
\[
\prod_{j=1}^{n} j^{b_j-1},
\]
a direct consequence of rules \( G3, G4 \) and \( G5 \). And then we must take the sums over all \( b_j \) constrained to rules \( G1 \) and \( G2 \).

**Theorem 2.** We have
\[
B_k(n) = k \sum_{b_1=1}^{k-1} \sum_{b_2=1}^{3k-b_1} \sum_{b_3=1}^{3k-b_1-b_2} \cdots \sum_{b_{n-1}=1}^{k(n-1)-\sum_{i=1}^{n-2} b_i} \prod_{j=1}^{n} j^{b_j-1}.
\]

**Proof.** It is an immediate consequence of rules \( G1 \) to \( G5 \). \( \square \)

The above formula can also be rewritten using the sequence \((f_i)_{0 < i < n}\), and considering \( f_n = kn \):
\[
B_k(n) = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \sum_{f_3=f_2+1}^{3k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} \prod_{i=2}^{n} i^{f_i-f_{i-1}-1}.
\]

**Corollary 1.** The number of non-isomorphic IC DFA’s with \( n \) states over an alphabet of \( k \) symbols is
\[
2^n B_k(n).
\]

**Proof.** By Theorems 1 and 2 and considering the possible sets of final states. \( \square \)

6 Conclusion

The method described in Section 4 was implemented in Python [pyt] and used to generate all IC DFA’s for \( k = 2 \) and \( n < 10 \), and \( k = 3 \) and \( n < 7 \). The time complexity of the program is linear in the number of automata and took about a week to generate all the referred IC DFA’s, in a PPC G4 1.5MHz.

One of the advantage of this method is that only the allowed strings are computed so it is not a generate-and-test algorithm and because automata are generated in lexicographic order it is easy to generate them as needed for consumption by another algorithm.

The formula \( B_k(n) \) was also implemented and its values where computed for \( k = 1..10 \) and \( n = 1..9 \). Those values are presented in Appendix A. The sequences \( B_2(n) \) and \( B_3(n) \) appear in Sloane [Slo03] as \( \textbf{A082165} \) and \( \textbf{A065756} \), respectively. If an IC DFA with \( n \) states accepts a finite language then there exists a topological order of its states such that \( \delta(i, \sigma) > i \), for all \( i < n - 1 \) and \( \sigma \in \Sigma \). But the order we used for string representations is not a topological order. So we can not determine directly from the string if the accepted language is finite, as was done by Domaratzki [Dom04] only for finite languages. Although the formula \( B_k(n) \) is quite similar to the one obtained in [Dom04] for an upper bound of the number of finite languages, the meaning of the parameters \( (b_j) \) are not directly related.
References

[BN] Frédérique Bassino and Cyril Nicaud. Enumeration of complete accessible deterministic automata over a 2-letter alphabet. Submitted.

[CP05] J.-M. Champarnaud and T. Paranthoën. Random generation of DFAs. Theoretical Computer Science, 330(2):221–235, 2005.

[DKS02] Michael Domaratzki, Derek Kisman, and Jeffrey 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] Michael Domaratzki. Combinatorial interpretations of a generalization of the Genocchi numbers. Journal of Integer Sequences, 7(04.3.6), 2004.

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

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

[HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to Automata Theory, Languages and Computation. Addison Wesley, 2000.

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

[Lis69] Valery Liskovets. The number of initially connected automata. Kibernetika, 3:16–19, 1969. (in Russian; Engl. transl: Cybernetics, 4 (1969), 259-262).

[Lis03] Valery Liskovets. Exact enumeration of acyclic automata. In Proc. 15th Conf. "Formal Power Series and Algebr. Combin. (FPSAC’03), 2003.

[MR05a] Nelma Moreira and Rogério Reis. Interactive manipulation of regular objects with FAdo. In Proceedings of 2005 Innovation and Technology in Computer Science Education (ITiCSE 2005). ACM, 2005.

[MR05b] Nelma Moreira and Rogério Reis. On the density of languages representing finite set partitions. Journal of Integer Sequences, 8(05.2.8), 2005.

[Nic99] Cyril Nicaud. Average state complexity of operations on unary automata. In M. Kurylowski, L. Pacholski, and T. Wierzbicki, editors, Proc. 24th Symposium, Mathematical Foundations of Computer Science, volume 1672 of Lecture Notes on Computer Science, pages 231–240. Springer-Verlag, 1999.

[Par04] T. Paranthoën. Génération aléatoire et structure des automates à états finis. PhD thesis, Université de Rouen, 2004.

[pyt] Python language website. http://python.org.

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

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

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

| $k=2$ on | $n$ | $k=3$ on | $n$ | $k=4$ on | $n$ | $k=5$ on | $n$ | $k=6$ on | $n$ | $k=7$ on | $n$ | $k=8$ on | $n$ | $k=9$ on | $n$ | $k=10$ on | $n$ |
|---------|-----|---------|-----|---------|-----|---------|-----|---------|-----|---------|-----|---------|-----|---------|-----|---------|-----|
|          |     |          |     |          |     |          |     |          |     |          |     |          |     |          |     |          |     |
| 1       | 1   |          | 1   |          | 1   |          | 1   |          | 1   |          | 1   |          | 1   |          | 1   |          | 1   |
| 2       | 12  |          | 2   | 56     | 2   | 992     | 2   | 4092   | 2   | 16256   | 2   | 65280   | 2   | 261632  | 2   | 8074532593 | 2   |
| 3       | 216 |          | 3   | 576   | 3   | 99289   | 3   | 409289 | 3   | 1625689 | 3   | 6528089 | 3   | 26163289 | 3   | 807453259389 | 3   |
| 4       | 5248|          | 4   | 212864| 4   | 914929500 | 4   | 36904929500 | 4   | 14769429500 | 4   | 5881369429500 | 4   | 22563469429500 | 4   | 88074532593899429500 | 4   |
| 5       | 160675|         | 5   | 57668921481  | 5   | 20075017237121 | 5   | 75287912629178688 | 5   | 2646911445028656000024832 | 5   |
| 6       | 5931540 |        | 6   | 52186085383 | 6   | 130567474759038787655 | 6   | 10240579484115206012643051897871090 | 6   |
| 7       | 75056805833 |      | 7   | 45677874292647947600097994111 | 7   |          | 7   |          | 7   |          | 7   |          | 7   |          | 7   |
| 8       | 13147753660096919923732736 |        | 8   | 8350774774759038787655 | 8   |          | 8   |          | 8   |          | 8   |          | 8   |          | 8   |
| 9       | 458677874292647947600097994111 |     | 9   | 200977948415228061026430518977871090 | 9   |          | 9   |          | 9   |          | 9   |          | 9   |          | 9   |

| k=10 on | $n$ |
|---------|-----|
| 1      | 1   |
| 2      | 12  |
| 3      | 4092 |
| 4      | 16256 |
| 5      | 65280 |
| 6      | 261632 |
| 7      | 8074532593 |
| 8      | 22563469429500 |
| 9      | 88074532593899429500 |
| 10     | 33550774774759038787655 |