# On Generating a Random Deterministic Finite Automaton as well as its Failure Equivalent

Madoda Nxumalo[1], Derrick G. Kourie[2,3], Loek Cleophas[2,4], and Bruce W. Watson[2,3]

[1] Computer Science, Pretoria University, South Africa
[2] FASTAR Research, Information Science, Stellenbosch University, South Africa
[3] Centre for Artificial Intelligence Research, CSIR Meraka Institute, South Africa
[4] Foundations of Language Processing, Computer Science, Umeå University, Sweden
{madoda,derrick,loek,bruce}@fastar.org — http://www.fastar.org

**Abstract.** An algorithm is proposed that constructs a failure deterministic finite automaton in lockstep with the construction of a language-equivalent deterministic finite automaton. The states of both automata are assumed to be predefined and the failure deterministic finite automaton's symbol and failure transitions are randomised. It is guaranteed that the latter remains free of divergent failure cycles. The benefits are explained of using this algorithm to produce input data for testing algorithms that produce a language-equivalent FDFA from an arbitrary DFA.

**Key words:** Random Failure Deterministic Finite Automaton

## 1 Introduction

Failure deterministic finite automata (FDFAs) extend classical deterministic finite automata (DFAs) by allowing for at most one so-called failure transition per state. Such a transition is to be taken from a given state only in case no regular symbol transition for the current symbol leaves that state; and from the state reached by this failure transition, further processing of the current symbol is then to resume.

This text assumes familiarity with the formal definition and discussion about FDFAs in [1]. That source describes, in abstract terms, a formal concept lattice based algorithm called the DFA-Homomorphic Algorithm (DHA). The algorithm's purpose is to derive from an *arbitrary* DFA a language equivalent FDFA. Various concrete versions of DHA have recently been implemented. These concrete versions rely critically on data produced by the software tool called FCART. FCART takes in an FCA context and outputs the corresponding formal concept lattice. In this case, the context input into FCART is known as a state/out-transition formal context and the output is a state/out-transition concept lattice. (See [1] for details of how to construct that context from a given DFA)

The FCART tool was developed at the National Research University Higher School of Economics (NRU HSE), Moscow [2]. Its developers specifically provided the tool for the purposes of a wider study to test DHA's effectiveness.

Initial results in testing the algorithm's effectiveness on DFAs that have language-equivalent Aho-Corasick FDFAs have been reported in [3]. However, a means was subsequently sought to test the algorithm on DFA input data drawn from a less restricted set of DFAs.

In principle, one could adopt some known approach to generating random DFAs, and try to use these to evaluate the effectiveness of DHA in generating language equivalent FDFAs. However, it is known *a priori* that it is computationally hard to determine whether a given FDFA is optimal in the sense of having the fewest possible transitions while remaining language-equivalent to the original DFA. This was proved by Björklund et al. [4]. This means that, starting with some randomly generated complete DFA, there is no self-evident way of deciding how well or badly DHA has performed.

Moreover, generating DHA FDFAs from such randomly generated DFAs may not result in interesting transition reductions, simply because the generated DFAs do not offer much scope for introducing failure transitions in the first place. Our early experiments along these lines showed that the percentage reduction in transitions of DHA FDFAs from random DFAs was generally less than 10%. An alternative way of generating more interesting DFA test data was therefore sought.

These DFAs should be random in some broad sense, yet should nevertheless provide many opportunities for replacing symbol transitions with failure transitions. In addition, in order to have some notion about whether or not DHA has effectively exploited possibilities for replacing symbol transitions with failure transitions, it would be useful to have at least one independently derived exemplar of an FDFA that is language equivalent to each "random" DFA.

In order to meet these objectives, an algorithm was devised that constructs an FDFA in lockstep with the construction of a language-equivalent DFA. The DFA and FDFA states are assumed to be predefined. The FDFA's symbol and failure transitions are randomised, yet guaranteed to remain free of divergent failure cycles.

## 2 Generating a DFA from a given FDFA

In Björklund et al. [4], [5], the following observation is made about building a DFA from an FDFA.

"Given an arbitrary FDFA, an equivalent DFA with the equal number of states can be built in polynomial time."

The proof of this observation, reproduced below, forms the basis for us to develop an algorithm that generates a random FDFA whilst concurrently building a language equivalent DFA.

*Proof.* Given an FDFA $\mathcal{F} = (Q, \Sigma, \delta, \mathfrak{f}, F, q_s)$, we construct an equivalent DFA $\mathcal{D} = (Q, \Sigma, \delta', F, q_s)$ as follows. Clearly, we see that every part of $\mathcal{D}$ is in $\mathcal{F}$ save for the differences in $\delta$ and $\delta'$. Firstly, we assign $\delta' = \delta$. Afterwards, we process the states in $Q$, possibly by adding outgoing transitions. If $q_1 \in Q$ has no failure transitions in $\mathfrak{f}$, the out-transitions from $q_1$ remain unchanged. If $q_1$ has a failure transition, then let $q_1, q_2, \ldots q_k$ be the failure path from $q_1$ — i.e. build the composite failure transition functions such that $q_2 = \mathfrak{f}(q_1), q_3 = \mathfrak{f}(\mathfrak{f}(q_1))$ etc. Note that the failure path is unique since $\mathfrak{f}$ is a function. If the failure path forms a failure cycle, then let $q_k$ be the last state before the cycle closes (to avoid a divergent failure cycle). We view the states on the path in some order, starting with state $q_2$. When some $q_i$ (from the failure path) is reached then for each $a \in \Sigma$ such that $q_1$ does not yet have an outgoing transition on $a$ in $\delta'$, and such that there exists some $q_j \in Q$ with $\delta(q_i, a) = q_j$, we set $\delta'(q_i, a) = q_j$.

We use some of the details from the above proof to generate a random FDFA $\mathcal{F} = (Q, \Sigma, \delta, \mathfrak{f}, F, q_s)$ and alongside it, a language equivalent DFA $\mathcal{D} = (Q, \Sigma, \delta', F, q_s)$. In our case we build transitions of the two language equivalent automata in some random manner as will be shown in the upcoming sections.

## 3 The Algorithm

Algorithm 1 below constructs an FDFA $\mathcal{F} = (Q, \Sigma, \delta, \mathfrak{f}, F, q_s)$ with various random features. It also constructs a language equivalent DFA $\mathcal{D} = (Q, \Sigma, \delta', F, q_s)$. The two automata have the same alphabet, $\Sigma$; the same sets of states, $Q$; a corresponding start state $q_s$; and equivalent sets of final states, $F$. These attributes for the pair of automata are assumed to have been defined before the algorithm is invoked. However the transitions $\delta, \delta'$ and $\mathfrak{f}$ are not yet fully defined when the algorithm starts executing. The algorithm is invoked with an integer parameter whose role is explained below.

### 3.1 The Transition Functions

The transition functions ($\delta'$ for the DFA, and $\delta$ and $\mathfrak{f}$ for the FDFA) are represented by global variables. We assume that the assignment operation establishes the function value for the given parameters. For example, $\delta(q_i, a) := q_j$

**Algorithm 1 (The Random (F)DFA Algorithm)**

```
{ pre $k \in [0, |Q|)$ }
proc $RFDFA(k)$
    for each $(q_i \in Q) \rightarrow$
        for each $(a \in \Sigma) \rightarrow$
            if $(\delta'(q_i, a) \neq \perp) \rightarrow$ skip
            ⫿ $(\delta'(q_i, a) = \perp) \rightarrow$
                $h := q_i$;
                $T, \ell := \{h\}, random([0, k])$;
                { Create a failure path starting from $q_i$ of length $\leq \ell$. }
                { $h$ is head of this failure path to date }
                { Path states are stored in $T$ }
                do $((|T| < \ell + 1) \wedge (\delta'(h, a) = \perp) \wedge \mathfrak{f}(h) \notin T) \rightarrow$
                    if $(\mathfrak{f}(h) = \perp) \rightarrow \mathfrak{f}(h) := random(Q \setminus T)$;
                    ⫿ $(\mathfrak{f}(h) \neq \perp) \rightarrow$ skip
                    fi;
                    $h := \mathfrak{f}(h)$;
                    $T := T \cup \{h\}$;
                od;
                { $(|T| \geq \ell + 1 \vee \delta'(h, a) \neq \perp \vee \mathfrak{f}(h) \in T)$ }
                { If $\delta'(h, a)$ is undefined, then assign random state as target }
                if $(\delta'(h, a) = \perp) \rightarrow \delta'(h, a) := random(Q)$;
                ⫿ $(\delta'(h, a) \neq \perp) \rightarrow$ skip
                fi;
                $\delta(h, a) := \delta'(h, a)$;
                for $(q_k \in T) \rightarrow$
                    $\delta'(q_k, a) := \delta(h, a)$;
                rof
            fi
        rof
    rof
corp
{ post Complete $\mathcal{D} \wedge \mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{F})$ }
```

means that the value $q_j$ is assigned to the function $\delta$ for the parameter values $(q_i, a)$. Similarly, for a failure transition we have the assignment operation; $\mathfrak{f}(q_i) := q_j$.

The algorithm does not change values of $\delta'$ or $\delta$ that have already been assigned prior to its invocation. Neither does it guarantee that all states will be reachable from the start state in the automata that it produces. A minimal number of $\delta'$ and matching $\delta$ values that establish this reachability should be assigned prior to invoking the algorithm.

This a priori connecting up of all states to the start state can be implemented as follows. For each $i < (|Q|-1)$, a randomly selected $a \in \Sigma$ is drawn and we ensure that $\delta(q_i, a) = \delta'(q_i, a) = q_{i+1}$. That is, the transition $\langle\langle q_i, a\rangle, q_{i+1}\rangle$ is added to both sets $\delta$ and $\delta'$. (Assume that $q_0$ is taken as the start state, $q_s$.)

The algorithm terminates with a complete DFA—i.e. when there is exactly one symbol transition on every symbol in the alphabet from every state. It ensures that upon termination $\delta$ and $\mathfrak{f}$ are such that the FDFA defines the same language as the DFA.

The symbol $\perp$ is used to denote a so-called *invalid or undefined* transition—e.g. $\delta(q_i, a) = \perp$ means that there is no symbol transition on $a$ in state $q_i$. Thus, if such a transition is not encountered at state $q_i$, then a failure transition should be taken. It is also possible for an FDFA to have undefined failure transitions at one or more states. If $\mathfrak{f}(q_i) = \perp$, this means that there is no failure transition exiting state $q_i$.

## 3.2 Other Variables

To influence the number of failure transitions in the final FDFA, the algorithm is invoked with an input parameter $k \in \mathbb{N} \wedge k < |Q|$, whose value is user-assigned. The integer $k$ serves as an upper bound on the length of any failure path that may appear in the FDFA. The role of $k$ is thus to constrain the number of failure transitions in the FDFA that need to be traversed before a symbol transition on any given symbol is encountered. Hence, when $k = 0$ the result will be a degenerate FDFA (i.e. $\mathcal{F} = \mathcal{D}$). Note that, because of various random features built into the algorithm, the extent to which the value of $k$ influences the number of failure transitions is not known or predictable *a priori*. All that can be said is that the larger the value of $k$, the fewer symbol transitions and the more failure transitions are likely to be present in the final FDFA, and *vice-versa*.

The algorithm loops over each state, $q_i$, taking certain actions for each possible alphabet symbol, $a$, at that state. If $\delta'(q_i, a)$ is already set, then no further action is taken; otherwise two variables are set: the integer $\ell$ and the set $T$.

$\ell$ is assigned a random value in the range $[0, \ldots, k]$. Its value determines the potential maximum failure transition path to traverse in order to consume the symbol $a$ if that $a$ is encountered at state $q_i$. The example below will further illustrate its role in the algorithm.

$T$ is used to accumulate states of a failure path starting at $q_i$ and ending at the last state of $h$. ($h$ a variable for the state which is the current head of the failure path.) An inner loop grows this failure path to not more than $\ell$ in length. At the head of this path, $h$, a symbol transition to a random state on symbol $a$ is provided, unless there is already a transition on symbol $a$ from $h$ to some pre-chosen state. This symbol transition is the same for both the emerging DFA and FDFA. In addition, all DFA states in $T$ are assigned transitions on symbol $a$ to the same state as the head's destination state, i.e. to $\delta(h, a)$.

Note that even though the finally obtained FDFA may contain failure cycles, it can be shown that no *divergent* failure cycles can be generated [6]. Recall that a failure cycle is not divergent if and only if for every $a \in \Sigma$, there is at least one state in the cycle, say $q_k$, such that $\delta(q_k, a) \neq \perp$.

## 4 An Example

Tables 1 (a)-(j) demonstrate how a random FDFA and a random DFA are constructed using the above stated algorithm. This example is for inserting transitions for two language equivalent automata defined by $Q = \{0, 1, 2, 3\}, \Sigma = \{a, b, c\}, q_s = 0, F = Q$. The assumed value for $k$ is 3. Some transition entries for the FDFA's $\delta$ and $\mathfrak{f}$ as well as the DFA's $\delta'$ are to be added by the algorithm.

The transition functions $\delta$, $\delta'$ and $\mathfrak{f}$ are represented as transition tables (see Tables 1 (a)-(j)) whose cell entries represent destination states. The transition tables of $\delta'$ and $\delta$ are presented as the algorithm reaches different stages of execution. The pair of automata are presented side by side with the DFA on the left hand side and the FDFA on the right hand side. They both contain three columns for the symbols of the alphabet and four rows for the states. The FDFA table has an additional column labelled $\mathfrak{f}$ for failure transitions.

1. Firstly, to connect all the states, for each $i \in \{0 \ldots 2\}$ the DFA and FDFA are provided with symbol transitions such that $\delta(i, g) = \delta'(i, g) = i+1$. Note that $g$ is randomly chosen from $\{a, b, c\}$. (See Tables 1 (a) and (b).) These transition values are recorded in the tables for $\delta$ and $\delta'$ before the algorithm is invoked. The remaining transition cells remain invalid. They are validated by the algorithm as described in the following paragraphs.

   The algorithm now iterates over all states (i.e. rows), in each iteration considering all alphabet symbols (columns).

2. Tables 1 (c)-(d) depicts the transition tables of the two automata after the algorithm has iterated over the initial state 0.

Table 1: An example: creating a ‘random’ FDFA and a ‘random’ DFA.

(a) Initialized DFA

| $Q/\Sigma$ | a | b | c |
|---|---|---|---|
| 0 | 1 | ⊥ | ⊥ |
| 1 | ⊥ | 2 | ⊥ |
| 2 | ⊥ | ⊥ | 3 |
| 3 | ⊥ | ⊥ | ⊥ |

(b) Initialized FDFA

| $Q/\Sigma$ | a | b | c | $\mathfrak{f}$ |
|---|---|---|---|---|
| 0 | 1 | ⊥ | ⊥ | ⊥ |
| 1 | ⊥ | 2 | ⊥ | ⊥ |
| 2 | ⊥ | ⊥ | 3 | ⊥ |
| 3 | ⊥ | ⊥ | ⊥ | ⊥ |

(c) DFA: After $q_i = 0$ entries

| $Q/\Sigma$ | a | b | c |
|---|---|---|---|
| 0 | 1 | 1 | 3 |
| 1 | ⊥ | 2 | ⊥ |
| 2 | ⊥ | 1 | 3 |
| 3 | ⊥ | 1 | 3 |

(d) FDFA After $q_i = 0$ entries

| $Q/\Sigma$ | a | b | c | $\mathfrak{f}$ |
|---|---|---|---|---|
| 0 | 1 | ⊥ | ⊥ | 3 |
| 1 | ⊥ | 2 | ⊥ | ⊥ |
| 2 | ⊥ | 1 | 3 | ⊥ |
| 3 | ⊥ | ⊥ | ⊥ | 2 |

(e) DFA: After $q_i =$ 1 entries

| $Q/\Sigma$ | a | b | c |
|---|---|---|---|
| 0 | 1 | 1 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | ⊥ | 1 | 3 |
| 3 | ⊥ | 1 | 3 |

(f) FDFA: After $q_i = 1$ entries

| $Q/\Sigma$ | a | b | c | $\mathfrak{f}$ |
|---|---|---|---|---|
| 0 | 1 | ⊥ | 3 | 3 |
| 1 | ⊥ | 2 | ⊥ | 0 |
| 2 | ⊥ | 1 | 3 | ⊥ |
| 3 | ⊥ | ⊥ | ⊥ | 2 |

(g) DFA: After $q_i =$ 2 entries

| $Q/\Sigma$ | a | b | c |
|---|---|---|---|
| 0 | 1 | 1 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 2 | 1 | 3 |
| 3 | ⊥ | 1 | 3 |

(h) FDFA: After $q_i = 2$ entries

| $Q/\Sigma$ | a | b | c | $\mathfrak{f}$ |
|---|---|---|---|---|
| 0 | 1 | ⊥ | 3 | 3 |
| 1 | ⊥ | 2 | ⊥ | 0 |
| 2 | 2 | 1 | 3 | ⊥ |
| 3 | ⊥ | ⊥ | ⊥ | 2 |

(i) Finally; DFA, $q_i = 3$

| $Q/\Sigma$ | a | b | c |
|---|---|---|---|
| 0 | 1 | 1 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 2 | 1 | 3 |
| 3 | 2 | 1 | 3 |

(j) Finally; FDFA, $q_i = 3$

| $Q/\Sigma$ | a | b | c | $\mathfrak{f}$ |
|---|---|---|---|---|
| 0 | 1 | ⊥ | 3 | 3 |
| 1 | ⊥ | 2 | ⊥ | 0 |
| 2 | 2 | 1 | 3 | ⊥ |
| 3 | ⊥ | ⊥ | ⊥ | 2 |

Starting with the column for symbol $a$, it is seen that transition $\delta'(0, a)$ already have a value (namely 1), so there is nothing more to do.

Thus, the next symbol, $b$, is considered. Since $\delta'(0, b)$ is not yet determined, a new value for $\ell$ is determined randomly. Suppose this value is $\ell = 2$. This means that the algorithm needs to generate a path of failure transitions of maximum length 2. The algorithm adds the current state 0 to the list $T$ of states on this failure path and it assigns the head value, $h$, to be 0. State 0 is thus the origin of a failure transition that terminates at a randomly selected state, say 3 in this case. Thus 3 is entered into the $\mathfrak{f}$ column. Additionally, state 3 is added to list $T$ and is set as the head $h$. To complete the creation of a failure path of length 2, the algorithm now move to the row for state 3 and generate a new random destination for this failure transition. Suppose it is 2. It indicates this in column $\mathfrak{f}$ of row 3. It then adds 2 to the list $T$ and $h$ becomes 3. By now $|T| = (\ell + 1)$ so there are no additional transitions to be added. Finally, at the head of this failure path, in state 2, the algorithm create a new random destination for a transition on symbol $b$. Suppose this is 1. It then sets both $\delta'(2, b)$ and $\delta(2, b)$ to 1. And then, based on $\delta(2, b) = 1$, it also sets $\delta'(j, b)$ to 1 for each state $j \in T$, namely for $\delta'(1, b)$ and $\delta'(3, b)$.

To complete the iteration for row 0, the algorithm now has to provide an entry for $\delta'(0, c)$. Again, the algorithm randomly generates a value for $\ell$. Suppose it is $\ell = 2$ as before. $T$ is reinitialised by storing state 0. $h$ becomes 0. From state 0 a failure transition to the target state 3 has already been established. Since $\ell = 2$, we again have to follow a failure path of length 2 when encountering a symbol $c$ in state 0. The algorithm thus moves one failure transition ahead to state 3. It also adds 3 to $T$ and assigns $h$ to be 3. Once more the algorithm fails to the same destination state previously generated, namely 2 and inserts state 2 into list $T$. Then 2 becomes $h$. The failure path of length 2 has now been reached in state 2 and an entry for $\delta(2, c)$ is required. Again it randomly generates a destination state, say 3 and sets $\delta(2, c)$ and $\delta'(2, c)$ to 3. Following the same logic as for symbol $b$, the algorithm sets $\delta'(2, c)$, $\delta'(0, c)$ and $\delta'(3, c)$ to 3.

3. Tables 1 (e)-(f) illustrate the two automata after the algorithm has inserted all transitions that arise when dealing with state 1.

   The algorithm is considering to insert a transition at $\delta'(1, a)$. Because this entry $\delta'(1, a)$ has an invalid value, a valid entry must be made. Suppose that, in considering column $a$, the random failure path length $\ell$ turns out to be 1. As before, state 1 is stored in $T$ and is assigned head $h$. Suppose a new failure transition destination 0 is generated randomly. The state 0 is added to $T$ and the head $h$ becomes 0. The failure path has reached the limit $\ell = 1$ and since a valid transition destination already exists for $\delta'(0, a)$, the addition of failure transitions is aborted. Since there is an established symbol transition $\delta'(0, a)$ namely 1 at the current control state $h$, the algorithm simply leaves it in place—i.e. it does not randomly generate a new FDFA symbol transition destination at 0 on symbol $a$. Then $\delta'(0,a)$ is copied into $\delta(0,a)$. Now for each state $j \in T$ the algorithm sets $\delta'(j,a)$ to 1. Thus, $\delta'(0,a)$ remains at 1 and $\delta'(1,a)$ is set to 1.

The algorithm now considers transitions on symbol $b$ in state 1. Since there is already a valid transition for $\delta'(1,b)$ (namely 2), the algorithm simply continues to execute the next transition entry $\delta(1,c)$.

Now column $c$ for state 1 has to be considered, and this entry $\delta'(1,c)$ has an invalid transition value. Suppose that at this stage, the algorithm randomly generates $\ell = 1$. Hence, 0 is inserted to the entry at row 1, column $\mathfrak{f}$. The algorithm adds state 1 to $T$ and sets $h$ to be 1. Recall that when dealing with column $a$ for state 1, a failure transition to state 0 was created, so there is already a failure transition from state 1. The algorithm therefore adds state 0 to $T$ and sets the head $h$ to be state 0. Because $|T| = (\ell + 1)$, there no further failure transition destinations to consider. The algorithm generates a random symbol transition at $\delta'(0,c)$. Suppose that the new transition is 3. The value 3 is copied to $\delta(0,a)$. Thus as usual, all values of $T$ are updated with 3—i.e. $\delta'(0,c)$ and $\delta'(1,c)$ are all set to 3.

4. Tables 1 (g)-(h) shows the status of the automata after all transitions from state 2 have been generated by the algorithm.

   We now look at the algorithm inserting a transition at $\delta'(2,a)$. This entry is invalid so transitions must be added to the pair of automata. Suppose a random value set for $\ell$ is 0. The head $h$ is initialized by the current state value 2 and the list $T$ is also initialized by the same value 2. Since $\ell = 0$, there are no failure transitions to be inserted. A new symbol transition value is then randomly generated for $\delta(2,a)$. Let the generated value to be 2. As usual the newly generated DFA transition is copied into the corresponding FDFA transition. Lastly, using the lone value in $T$, the $\delta'$ is updated such that $\delta'(2,a)$ becomes 2.

   The algorithm then continues with symbol transition iterations at state 2, first to $\delta'(2,b)$, and lastly to $\delta'(2,c)$. Since there are already established valid symbol transitions at $\delta'(2,b)$ and $\delta'(2,c)$, there is no need to replace them. Then the transitions from the next state will be considered.

   At the end of iterating all transitions at state 2 there is no failure transition provided from this state. Thus, $\mathfrak{f}(2)$ remains $\perp$. This is a case whereby a "useless" failure transition is not generated.

5. Transitions at state 3 are considered. The resulting transitions are shown in Tables 1 (i)-(j).

   The algorithm will now insert transitions starting at an entry for state 3 and symbol $a$. The transition $\delta'(3,a)$, currently contains an invalid entry, therefore the algorithm must insert some transitions. The maximum failure path length $\ell$ is generated randomly. Let $\ell$ to be 1, set $h$ to be the current state 3 and $T$ is instantiated by adding 3 into itself. A previously defined failure transition from state 3 to 2 is taken. The state 2 is added into $T$ and it is assigned as $h$. Since a single failure transition has been taken and because $\delta'(2, a)$ is a valid transition, namely 2, then no further attempts to generate failure path generation are to be made. The transition value $\delta'(2, a) = 2$ is copied to the value at $\delta(2, a)$. As usual, for each state in $T$, the algorithm inserts a couple of DFA symbol transitions, i.e. $\delta'(2, a)$ and $\delta'(3, a)$ become 2.

The remaining two entries to look at for state 3 are the columns $b$ and $c$, i.e. $\delta'(3, b)$ and $\delta'(3, c)$. They both have existing valid DFA symbol transitions namely: $\delta'(3, b) = 1$ and $\delta'(3, c) = 3$. Therefore, the algorithm will simply skip these last two transition iterations and terminate executing.

At this stage, the two automata have been created: a complete DFA and a language equivalent FDFA. As expected, the complete DFA has 12 symbol transitions. The FDFA has 9 transitions, 3 being failure transitions and 6 being symbol transitions. Thus, the FDFA shows a reduction of 3 transitions from the DFA transitions size—i.e. it has 25% fewer transitions than the DFA.

## 5 Conclusion

This algorithm described here has successfully been used to generate a variety of DFAs for input to DHA with the $k$ parameter setting varying between 10 and 100. The simultaneously produced FDFA then served as a basis against which to compare the FDFAs produced by the various DHA variants. The comparison is with respect to the extent to which the respective FDFAs reduce the number of symbol transitions in the language-equivalent DFA. Detailed analysis of these results will be reported elsewhere.

## References

1. D. G. Kourie, B. W. Watson, L. Cleophas, and F. Venter, “Failure deterministic finite automata,” in *Proceedings of the Prague Stringology Conference 2012* (J. Holub and J. Zdárek, eds.), pp. 28–41, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2012.
2. A. Neznanov, D. Ilvovsky, and A. Parinov, “Advancing FCA workflow in FCART system for knowledge discovery in quantitative data,” in *Proceedings of the Second International Conference on Information Technology and Quantitative Management, ITQM 2014, National Research University Higher School of Economics (HSE), Moscow, Russia, June 3-5, 2014*, pp. 201–210, 2014.
3. M. Nxumalo, D. Kourie, L. Cleophas, and B. Watson, “An aho-corasick based assessment of algorithms generating failure deterministic finite automata,” in *Proceedings of the 12th International Conference on Concept Lattices and their Applications (CLA 2015)*, (Clermont-Ferrand, France), 2015.

4. H. Björklund, J. Björklund, and N. Zechner, “Compression of finite-state automata through failure transitions,” *Theor. Comput. Sci.*, vol. 557, pp. 87–100, 2014.
5. H. Björklund, J. Björklund, and N. Zechner, “Compact representation of finite automata with failure transitions,” technical report, Umeä University, 2013.
6. M. Nxumalo, “An assessment of selected algorithms for generating failure deterministic finite automata,” masters thesis, University of Pretoria, 2016. Under examination.