Preprint

This is the submitted version of a paper published in *Theoretical Computer Science*.

Citation for the original published paper (version of record):

Compression of finite-state automata through failure transitions.
*Theoretical Computer Science*, 557: 87-100
http://dx.doi.org/10.1016/j.tcs.2014.09.007

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-93329
Accepted Manuscript

Compression of finite-state automata through failure transitions

Henrik Björklund, Johanna Björklund, Niklas Zechner

PII: S0304-3975(14)00672-0
DOI: 10.1016/j.tcs.2014.09.007
Reference: TCS 9857

To appear in: Theoretical Computer Science

Received date: 10 December 2013
Revised date: 14 August 2014
Accepted date: 8 September 2014

Please cite this article in press as: H. Björklund et al., Compression of finite-state automata through failure transitions, Theor. Comput. Sci. (2014), http://dx.doi.org/10.1016/j.tcs.2014.09.007

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
Compression of finite-state automata through failure transitions

Henrik Björklund*, Johanna Björklunda,*, Niklas Zechnera

a Computing Science Department, Umeå University, 901 87 Umeå, Sweden

Abstract
Several linear-time algorithms for automata-based pattern matching rely on failure transitions for efficient back-tracking. Like epsilon transitions, failure transition do not consume input symbols, but unlike them, they may only be taken when no other transition is applicable. At a semantic level, this conveniently models catch-all clauses and allows for compact language representation.

This work investigates the transition-reduction problem for deterministic finite-state automata (DFA). The input is a DFA \( A \) and an integer \( k \). The question is whether \( k \) or more transitions can be saved by replacing regular transitions with failure transitions. We show that while the problem is \( \text{NP} \)-complete, there are approximation techniques and heuristics that mitigate the computational complexity. We conclude by demonstrating the computational difficulty of two related minimisation problems, thereby cancelling the ongoing search for efficient algorithms.

Keywords: Failure automata, pattern matching, automata minimisation

1. Introduction

Deterministic finite-state automata (DFA) have applications in natural language processing (Roche and Shabes, 1997), medical data analysis (Lewis et al., 2010), network intrusion detection (Tuck et al., 2004), computational biology (Cameron et al., 2005), and other fields. Although DFA are less compact than their non-deterministic counterpart, they are easier to work with algorithmically, and their uniform membership problem, when also the language model is part of the input, can be decided in time \( O(|w| \log |Q|) \), where \( w \) is the input string and \( Q \) the state space. The corresponding figure for non-deterministic automata is \( O(|w||\delta|) \), where \( \delta \) is the transition relation.

A middle ground between compactness of representation and classification efficiency can be reached via failure transitions. Similar to epsilon transitions, these do not consume any input symbols, but unlike epsilon transitions, they can only be taken when there are no other applicable transitions. When states in an automaton share a set of outgoing

*Corresponding author

Email addresses: henrikb@cs.umu.se (Henrik Björklund), johanna.bjorklund@umu.se (Johanna Björklund), niklas.zechner@umu.se (Niklas Zechner)

Preprint submitted to Elsevier September 10, 2014
transitions, the automaton can be compressed by replacing these duplicates by a smaller number of failure transitions. The resulting automaton is called a failure finite-state automaton (FFA).

**Example 1.** Figure 1 shows a FFA for finding words in the dictionary \{ab, bb, babb\} as factors in the input string (Crochemore and Hancart, 1997). In the figure, failure transitions are drawn as dashed arrows. If it were not for these transitions, then each state would need one outgoing transition for each and every symbol in the alphabet, so as to be able to processes any input string in its entirety. This suggests that failure automata are particularly useful for pattern matching over large alphabets.

The addition of failure transitions does not preserve determinism in the classical sense, but when the input automaton is deterministic and each state is allowed at most one outgoing failure transition, then the result is a transition deterministic automaton. Such an automaton can go through multiple transitions when reading a single input symbol, but for a given state and a given input symbol, there is at most one such sequence of transitions. As a consequence, the complexity of the membership problem only increases by a factor \(|Q|\).

Empirical studies of the efficiency of failure minimisation are underway. Kumar et al. (2006) use failure transitions (under the name of default transitions) to reduce the size of automata for deep packet inspection, with the purpose of avoiding network intrusion. The authors report that the number of distinct transitions between states is reduced by more than 95%. Preliminary results for (heuristic, non-optimal) failure minimisation of randomly generated DFAs suggest size-reductions by 5–15% (Kourie et al., 2012b).

In this work, we look closer at the transition-reduction problem for FFA. The input is a DFA \(A\) and an integer \(k\). The question is whether a deterministic, language-equivalent, FFA \(B\) with \(k\) fewer transitions be constructed from \(A\) by removing regular transitions and adding failure transitions.

**Example 2.** Figure 2 (a) shows a state-minimal DFA over the alphabet of symbols \{a, b, c\} \(\cup\ A \cup B \cup C\), in which \(A\), \(B\), and \(C\) denote the sets \{a, b, c\}.
\( \{b_i \mid i \in \{1, \ldots, n\}\} \), and \( \{c_i \mid i \in \{1, \ldots, n\}\} \), respectively, for some natural number \( n \). A language-equivalent automaton in which regular transitions have been replaced by failure transitions is given in Figure 2 (b). In this case, the failure transitions help save \( 3n - 2 \) transitions. More precisely, \( 3n + 3 \) regular transitions are saved and 5 failure transitions are added. This family of instances is constructed to show the strengths of failure transitions, and will be illustrative when we discuss approximation techniques.

In addition to transition reduction, we study the related problems of transition minimisation and binary minimisation. The input to the transition-minimisation problem is the same as to the transition-reduction problem, but the question is now whether there is any deterministic FFA with \( k \) fewer transitions that recognises the same language as \( A \). The difference compared to the original formulation is that we are not required to preserve the structure of the input DFA. In particular, we are allowed more states.

The input to the binary-minimisation problem is a binary automaton \( A \) and an integer \( k \). A binary automaton is a failure automaton in which every state has at most two outgoing transitions; a regular transition and a failure transition (Kowaltowski et al., 1993). The question to be decided is whether there is a language-equivalent binary automaton with at most \( k \) transitions. In contrast to the two previous problems, \( k \) is now an upper bound on the number of transitions in the output automaton, and not a lower bound on the savings obtained. We chose this formulation because that is how the problem is presented in (Kowaltowski et al., 1993), and it does not affect the computational complexity since it is easy to translate from one way of looking at problem to the other.

**Contributions**

We prove that problems of transition reduction, transition minimisation, and binary minimisation are, in general, NP-complete. This cancels the search for efficient and optimal algorithms initiated by Kourie et al. (2012b) and answers a problem left open by Kowaltowski et al. (1993). It should be stressed that these results do not follow immediately from one another. In the case of transition reduction and transition minimisation, the freedom to add states could potentially make the problem easier, but on the other hand, failure reduction does not always produce a deterministic transition-minimal FFA, which if it were the case could make that problem easier.

In the second half of the paper, we look at alternative ways of making transition reduction tractable. Firstly, we give a polynomial-time approximation algorithm that saves at least two-thirds of the number of transitions that an optimal algorithm would. Secondly, we introduce simulation relations for failure automata, and combine simulation minimisation with an existing heuristic for transition reduction (Kourie et al., 2012c) to obtain an \( O(mn) \) reduction algorithm, where \( m \) is the size of the transition table of the input automaton, and \( n \) is the number of its states. There are no guarantees on the algorithms' performance; it may perform very well, or very poorly, depending on the input automaton. However, in contrast to the approximation algorithm, the heuristic algorithm can also compress nondeterministic automata.

Approximation techniques and heuristics for the transition-minimisation problem and the binary-minimisation problem are left for future work.

**Related work**

Failure transitions make their first appearance in an article on pattern matching by Knuth et al. (1974, 1977). The authors give a linear-time algorithm for finding
Figure 2: A pair of finite-state automata for the same language. The labels $A$, $B$, and $C$ denote the sets of symbols $\{a_i | i \in \{1, \ldots, n\}\}$, $\{b_i | i \in \{1, \ldots, n\}\}$, and $\{c_i | i \in \{1, \ldots, n\}\}$, respectively, for some natural number $n$. Failure transitions are drawn as dashed arrows.
all occurrences of a pattern string within a text string. The algorithm reads the text string from left-to-right, while moving a pointer back and forth in the pattern string to remember what prefix of it has been encountered so far. Whenever the text string diverges from the pattern string, the algorithm backtracks by shifting the pointer according to a pre-computed failure function.

Aho and Corasick (1975) build on this idea when they consider the problem of finding locations of dictionary entries in an input string. The dictionary consists of a finite set of words \( L \), and is represented as a prefix-tree acceptor \( A \). Recall that this is a partial DFA recognising \( L \), whose states are in one-to-one correspondence with the prefixes of \( L \). To allow \( A \) to process strings on the form \( \Sigma^* L \Sigma^* \), every state \( w \) is given an failure transition pointing to the longest suffix of \( w \) that is still a prefix of a string in \( L \Sigma^* \). Finally, a self-loop on the initial state \( \varepsilon \) is added, on those symbols that lack transitions from \( \varepsilon \). The advantage of failure transitions in this context is that they save space, simplify the automata construction, and allow for efficient classification of input strings.

Mohri (1997) in turn, continues the work of Aho and Corasick, but takes as its starting point a DFA \( A \) recognising a possibly infinite set of target patterns. By traversing the states of \( A \) breadth-first, while adding failure transitions and auxiliary states, his algorithm produces a deterministic FFA \( A' \) that recognises \( \Sigma^* L \). The time complexity is linear in the size of \( A' \), which in the worst case is exponential in the size of \( A \), but because of the failure transitions, the time complexity is not affected by the size of the alphabet.

A survey of automata for pattern matching has been compiled by Crochemore and Hancart (1997). In this context, failure transitions are sometimes treated under the name suffix links (Weiner, 1973).

Recently, Kourie et al. (2012a) considered the problem of using failure transitions to save as much space as possible, i.e., given an input DFA, try to find an equivalent automaton with failure transitions whose total number of transitions is minimal. They develop two heuristic algorithms that build on formal concept analysis to solve the problem, but leave the complexity of the problem open. The same team of researchers are also conducting experiments on failure minimisation, and initial results are described by Kourie et al. (2012b).

Outline

Section 2 and 3 recall central concepts and fixes notation. In Section 4, we prove that three minimisation problems related to the introduction of failure transitions are NP-complete. Section 5 investigates the extent to which solutions can be approximated. In Section 6, we discuss a heuristic approach to transition reduction that relies on simulation relations. Section 7 summarises our findings and concludes with suggestions for future work.

2. Preliminaries

This section covers the terminology and notations of failure automata. Since we want to allow nondeterminism in the discussion of simulation minimisation, we talk about transition and failure relations rather than functions.
Sets and numbers. We write \( \mathbb{N} \) for the natural numbers (including 0) and \( \mathbb{B} \) for the Booleans. For \( n \in \mathbb{N} \), if \( n = 0 \) then \( [n] = \emptyset \), and \( [n] = \{1, \ldots, n\} \) otherwise.

Let \( \delta \) and \( \gamma \) be binary relations on a set \( S \). The composition of \( \delta \) and \( \gamma \) is denoted \( \delta \circ \gamma \) and contains all pairs \( (s, s'') \) such that \( (s, s') \in \delta \) and \( (s', s'') \in \gamma \) for some \( s' \in S \). The domain of \( \delta \) is \( \text{dom}(\delta) = \{ s \in S : \exists s' \in S : (s, s') \in \delta \} \), and the reflexive and transitive closure of \( \delta \) is the smallest relation \( \delta^* \) such that

- \( \{(s, s') \in \delta \} \subseteq \delta^* \), and
- \((s, s') \in \delta^* \) and \((s', s'') \in \delta \) implies that \((s, s'') \in \delta^* \).

The transitive reduction of \( \delta \) is

\[
\delta^- = \{(s, s'') \in \delta : \exists s' : (s, s') \in \delta \text{ and } (s', s'') \in \delta \}. 
\]

If \( \delta \) is acyclic and finite, then \( \delta^- \) is well-defined. A preorder is a reflexive and transitive relation. A partial order is an anti-symmetric preorder.

Automata. A failure finite-state automaton (FFA) is a tuple \( B = (Q, \Sigma, \delta, \gamma, I, F) \) where:

1. \( Q \) is a finite set of states,
2. \( \Sigma \) is the input alphabet,
3. \( \delta = (\delta_a)_{a \in \Sigma} \) is a family of transition relations \( \delta_a : Q \times Q \rightarrow Q \),
4. \( \gamma : Q \times Q \rightarrow \text{bool} \) is a failure relation, and
5. \( I, F \subseteq Q \) are sets of initial and final states, respectively.

We derive from \( \delta \) and \( \gamma \) a family \( (\hat{\delta}_w)_{w \in \Sigma^*} \) of relations \( \hat{\delta}_w : Q \times Q \). For every \( P \subseteq Q \), \( a \in \Sigma \), and \( w \in \Sigma^* \), we have \( \hat{\delta}_w = P \times P \), and

\[
\hat{\delta}_w = \gamma_a \circ \delta_a \circ \hat{\delta}_w \text{ where } \gamma_a = \gamma \cap ((Q \setminus \text{dom}(\delta_a)) \times Q). 
\]

The intuition behind \( \hat{\delta} \) is that when the automaton encounters the symbol \( a \), then it explores the failure transitions given by \( \gamma \) until it reaches a state from which it can consume \( a \) with a transition in \( \delta_a \).

The language accepted by an FFA \( A \) is \( L(A) = \{ w \in \Sigma^* : (I, F) \cap \hat{\delta}_w \neq \emptyset \} \). From here on, we identify \( \delta \) and \( \gamma \), unless there is risk of confusion.

For \( q \in Q \), \( A^q \) is the automaton obtained from \( A \) by replacing its set of initial states by \( \{q\} \). Since we are concerned with reducing the number of transitions, we define the size of \( A \) as \( |A| = |\delta| + |\gamma| \).

A finite-state automaton (FA) is an FFA in which \( \gamma = \emptyset \). When we specify FAs, we may therefore omit the component \( \gamma \). A deterministic FFA (DFA) is an FFA in which \( |I| \leq 1 \), and \((\delta_a)_{a \in \Sigma}\) and \( \gamma \) are partial functions. A deterministic FA (DFA) is thus a deterministic FFA in which \( \gamma \), when viewed as a set, is empty.

For \( p \in Q \), we denote by \( \Sigma(p) \) the set of symbols \( \{ a \in \Sigma : \exists q \in Q : (p, q) \in \delta_a \} \). The abilities of \( p \in Q \) is the set \( \text{abil}(p) = \{(a, q) \in \Sigma \times Q : (p, q) \in \delta_a \} \), and the ability overlap of \( p \subseteq Q \) is \( \text{abil}(P) = \bigcap_{p \in P} \text{abil}(p) \).

3. Basic properties of FDFAs

Before we address the subject matter, we make some basic observations that will be helpful later. The first of these is that FDFAs can be efficiently rewritten as language-equivalent DFAs by computing the closure of the failure transitions. The technique is similar to epsilon-removal.
Observation 1. Given an FDFA, we can construct an equivalent DFA with the same number of states in polynomial time.

Proof. Given an FDFA $B = (Q, \Sigma, \delta, \gamma, F, q_0)$, let us show how to construct an equivalent DFA $A = (Q, \Sigma, \delta', F, q_0)$. Notice that every part of $A$ except for $\delta'$ is the same as the corresponding part of $B$. We change $\delta$ into $\delta'$ as follows. To begin with, we set $\delta' = \delta$. We then process the states in $Q$, possibly adding outgoing transitions. If $q_1 \in Q$ has no failure transition in $B$, the outgoing transitions from $q_1$ stay the same. If $q_1$ has a failure transition, let $q_1, q_2, \ldots, q_k$ be the path of states reached by starting from $q_1$ and following $\gamma$. In other words, $q_2 = \gamma(q_1)$, $q_3 = \gamma(\gamma(q_2))$, and so forth. Notice that since $\gamma$ is a function, this path is unique. If the path has a cycle, then let $q_k$ be the last state before the cycle closes. We look at the states on the path in order, starting with the state $q_2$. When we reach $q_i$, then for every $a$ such that the $q_1$ does not yet have an outgoing transition on $a$ in $\delta'$, and such that there is some $p$ with $\delta_a(q_i) = p$, we let $\delta'_a(q_1) = p$.

Observation 1 makes it clear that failure transitions may save on regular transitions, but never on states.

Observation 2. No FDFA for a language $L$ can have fewer states than the state-minimal DFA for $L$.

In fact, failure transitions are sometimes better leveraged by introducing more states. This situation is further discussed in the upcoming proof of Theorem 2.

Observation 3. For some languages $L$, every transition-minimal FDFA for $L$ has more states than the state-minimal DFA for $L$.

Example 3. Observation 3 is exemplified by the two automata in Figure 3. The DFA in Figure 3 (a) has four states and ten transitions. The FDFA in Figure 3 (b) recognizes the same language. It has five states, but only nine transitions. It is easy to verify that there is no language-equivalent FDFA that has four states and fewer than ten transitions.

By Observation 1, when given two FDFAs, we can construct equivalent DFAs, and then minimise and compare these, all in polynomial time.

Observation 4. Equivalence testing for FDFAs is polynomial.

However, unlike DFAs, FDFAs do not offer a canonical form of representation.

Observation 5. Given a language $L$, there is, in general, no unique (up to homomorphism) state-minimal or transition-minimal FDFA for $L$.

4. Three hard minimisation problems

In this section, we consider three minimisation problems relevant in the context of failure automata. As we shall see, they all turn out to be quite difficult.
Figure 3: A pair of finite-state automata for the same language. The FDFA to the right has more states but fewer transitions than the DFA to the left.
4.1. Transition reduction

We first prove that the transition-reduction problem, which is the focus of our attention, is computationally hard.

**Theorem 1.** The transition-reduction problem is NP-complete.

**Proof.** The problem is in NP since, by Observation 4, equivalence testing for FDFAs is polynomial. Given a DFA \( A \) and an integer \( k \), we can guess an F DFA with \( k \) fewer transitions than \( A \) and verify that it is equivalent to \( A \).

For NP-hardness, we reduce from [Hamiltonian Cycle](#). Given a graph \( G = (V, E) \) with \( |V| = n \) and \( |E| = m \) we construct a DFA \( A = (Q, \Sigma, \delta, I, F) \) such that there is an F DFA \( B \) for the language \( \mathcal{L}(A) \) with \( k = n(n - 2) \) fewer transitions if and only if \( G \) has a Hamiltonian cycle.

Let \( V = \{v_1, \ldots, v_n\} \) and \( E = \{e_{i,j} \mid (e_{i,j}) \in E \land i < j \} \). The alphabet \( \Sigma \) contains a letter for each vertex and for each edge of \( G \), i.e., \( \Sigma = V \cup E \). The state set of \( A \) is \( Q = \{q_I, q_F\} \cup \{p_1, \ldots, p_n\} \), with \( I = \{q_I\} \) and \( F = \{q_F\} \). We now describe the transition function of \( A \) in detail.

- For every vertex name \( v_i \), \( \delta_v(q_I) = p_i \).
- Every state \( p_i \in \{p_1, \ldots, p_n\} \) has the following outgoing transitions.
  - \( \delta_{v_i}(p_i) = p_i \),
  - \( \delta_{v_j}(p_i) = q_F \) for every \( v_j \neq v_i \),
  - \( \delta_{e_i,\ell}(p_i) = q_F \) for every edge name \( e_i,\ell \) such that \( i = j \) or \( i = \ell \),
  - \( \delta_{e_i,\ell}(p_i) = p_i \) for every edge name \( e_j,\ell \) such that \( i \neq j \) and \( i \neq \ell \).

This means that the language \( \mathcal{L}(A) \) of \( A \) consists of all words \( v_i \tau^* \sigma_i \), where \( \tau \) contains \( v_i \) and the names of all edges that are not adjacent to \( v_i \), while \( \sigma_i \) contains \( V \setminus \{v_i\} \) and the names of all edges that are adjacent to \( v_i \). Let \( \mathcal{L}_G = \mathcal{L}(A) \). It is straightforward to verify that \( A \) is the minimal DFA for \( \mathcal{L}_G \). Notice that \( q_I \) has \( n \) outgoing transitions, \( q_F \) has none and \( q_F \) has \( n + m \) for every \( i \in [n] \). In total, the automaton \( A \) has \( n + n(n + m) = n(n + m + 1) \) transitions.

First, we assume that \( G \) has a Hamiltonian cycle and show that there is an F DFA \( B \) with \( k = n(n - 2) \) fewer transitions than \( A \) such that \( \mathcal{L}(B) = \mathcal{L}_G \). By renaming vertices, we can assume that the cycle is \( v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n \rightarrow v_1 \). We construct \( B \) from \( A \) by adding a failure transition \( \gamma(p_i) = p_{i+1} \) for every \( i \in [n - 1] \) and the failure transition \( \gamma(p_n) = p_1 \). All transitions that have been made redundant are then removed. After this, \( q_I \) still has \( n \) outgoing transitions, while \( q_F \) has none. We argue that every \( p_i \) for \( i \in [n] \) has \( m + 2 \) outgoing transitions, i.e., \( n - 2 \) fewer than in \( A \). Indeed, looking at \( p_i \) and \( p_{i+1} \) (or \( p_1 \), if \( i = n \)), we see that in \( A \), they both have transitions to \( q_F \) for every \( v_j \) such that \( j \notin \{i, i + 1\} \). Thus \( n - 2 \) transitions can be removed from \( p_i \). Additionally they both have transitions to \( q_F \) on the edge name \( e_{i,i+1} \). Thus we can remove one additional outgoing transition from \( p_i \). On the other hand, we have added a failure transition from \( p_i \). This means that in total, \( p_i \) has \( n - 2 \) outgoing transitions fewer in \( B \) than in \( A \). This means that \( B \) has \( n(n - 2) \) fewer transitions than \( A \), as required.

Next, we assume that there is an F DFA \( B = (Q, \Sigma, \delta', \gamma, I, F) \) with \( k \) transitions fewer than \( A \) and such that \( \mathcal{L}(B) = \mathcal{L}_G \) and argue that \( G \) has a Hamiltonian cycle.
There have to be \( n \) transitions leaving \( q_I \), one for each vertex name \( v_i \). We can assume that these are the transitions \( \delta(v_i) = p_i \). On the other hand, no transitions need to leave \( q_F \). Thus we can focus on the transitions from the states \( p_1, \ldots, p_n \). Each failure transition will go from one such state to another such state. No pair of such states can share more than \( n - 1 \) abilities, which means that each such state will have at least \( m + n - (n - 1) + 1 = m + 2 \) outgoing transitions. This means that \( B \) will have at most \( k = n(n - 2) \) transitions fewer than \( A \) and that, for this number to be realised, each state in \( \{p_1, \ldots, p_n\} \) must have exactly \( m + 2 \) outgoing transitions.

In \( A \), each such state has one transition per edge name and one per state name, that is, \( n + m \) outgoing transitions. Therefore, every such state in \( B \) must have a failure transition. Assume that there is a failure transition from \( p_i \) to \( p_j \). Then we can remove the \( n - 2 \) outgoing transitions on the vertex names \( V \setminus \{v_i, v_j\} \) from \( p_i \). On the other hand, we have added a failure transition, leaving us with \( m + 3 \) transitions. This means that for \( p_i \) to have only \( m + 2 \) transitions, it has to share one more ability with \( p_j \). This is only possible if there is an edge between \( v_i \) and \( v_j \) in \( G \). In this case, both states have transitions to \( p_F \) on \( e_{i,j} \).

Next, we argue that the graph of the failure function \( \gamma \) must be connected and cyclic. Note that if there is a failure transition from \( p_i \) to \( p_j \), then \( p_i \) must have a transition to itself on \( v_1 \) and to \( q_F \) on \( v_j \). These are its only transitions on vertex names. This also means that for all transitions on vertex names to \( q_F \) to be represented somewhere, there can be no two states that fail to the same state. Since each such transition must be reachable via failure transitions from all but one state in \( \{p_1, \ldots, p_n\} \), the graph of \( \gamma \) is indeed connected and cyclic.

As shown above, each edge of the graph of \( \gamma \) also corresponds to an edge in \( G \). Thus \( \gamma \) induces a Hamiltonian cycle on \( G \).

4.2. Transition minimisation

Let us now turn to the transition-minimisation problem, that is, the case when we are allowed auxiliary states. The proof of Theorem 2 is inspired by a proof by Jiang and Ravikumar (1993), showing that the normal set basis problem is NP-hard. See also (Björklund and Martens, 2012).

**Theorem 2.** The transition-minimisation problem is NP-complete.

**Proof.** The transition-minimisation problem is in NP since, by Observation 4, we can guess an FDFA with at most \( s \) transitions and test it for equivalence with the input DFA (viz. an FDFA without failure transitions) in polynomial time.

To show NP-hardness, we reduce from Vertex Cover. Given a graph \( G = (V, E) \) with \( |V| = n \) and \( |E| = m \) and an integer \( k \), we construct a DFA \( A_G \) and an integer \( s \) such that there is a language-equivalent FDFA \( B_G \) that has at most \( s \) transitions if and only if \( G \) has a vertex cover of size at most \( k \).

We first define the language \( \mathcal{L}_G \) that \( A \) will accept. As in the proof of Theorem 1, let \( V = \{v_1, \ldots, v_n\} \) and \( E = \{e_{i,j} \mid (v_i, v_j) \in E \land i < j \} \). We define the alphabet that \( \mathcal{L}_G \) will use by \( \Sigma = V \cup \overline{E} \cup \{a_i, b_i, e_i \mid v_i \in V\} \). Thus \( \Sigma \) has one symbol per vertex, one symbol per edge, and three extra symbols per vertex, so the size of \( \Sigma \) is \( 4n + m \).

The language \( \mathcal{L}_G \) will only contain words of length two. The first symbol will be taken from \( V \cup \overline{E} \) and the second symbol will depend on the first. To this end, we define
the residual language of each member of \( V \cup E \) as follows.

\[
\text{res}(v_i) = \{a_i, b_i, c_i\} \quad \text{(for } v_i \in V) \\
\text{res}(e_{i,j}) = \{b_i, c_i, b_j, c_j\} \quad \text{(for } e_{i,j} \in E) 
\]

We now define \( L_G \) by

\[
L_G = \left( \bigcup_{v_i \in V} v_i \cdot \text{res}(v_i) \right) \cup \left( \bigcup_{e_{i,j} \in E} e_{i,j} \cdot \text{res}(e_{i,j}) \right).
\]

The automaton \( A_G \) is simply the minimal DFA for \( L_G \); see the illustration in Figure 4. We note that \( A_G \) has \( n + m + 2 \) states and \( 4n + 5m \) transitions. The integer \( s \) will be \( 4n + 4m + k \).

Let \( q_0 \) be the initial state of \( A_G \) and let \( q_f \) be the accepting state. For each \( v_i \in V \), let \( q_i \) be the state \( A_G \) takes after reading \( v_i \). Similarly, for each \( e_{i,j} \in E \), let \( p_{i,j} \) be the state \( A_G \) takes after reading \( e_{i,j} \).

Assume that \( G \) has a vertex cover of size \( k \). We show how to construct \( B_G \) with \( s \) transitions such that \( L(B_G) = L_G \). Let \( C \subseteq V \) be a vertex cover for \( G \) of size \( k \). For every \( v_i \in C \), do the following. Remove the transitions \( \delta_{b_i}(q_i) = q_f \) and \( \delta_{c_i}(q_i) = q_f \). Add a state \( r_i \) and the transitions \( \gamma(q_i) = r_i, \delta_{b_i}(r_i) = q_f, \) and \( \delta_{c_i}(r_i) = q_f \). See Figure 5 for an illustration. The automaton now has \( 4n + 5m + k \) transitions, but we can save \( m \) transitions as follows.

For every \( e_{i,j} \in E \), we know that at least one of \( v_i \) and \( v_j \) belongs to \( C \). Without loss of generality, assume that \( v_i \in C \). We then remove the transitions \( \delta_{b_i}(p_{i,j}) = q_f \) and \( \delta_{c_i}(p_{i,j}) = q_f \) and add the failure transition \( \gamma(p_{i,j}) = r_i \). This saves one transition. Since we can do this for every edge, we save \( m \) transitions and arrive at an automaton with \( s = 4n + 4m + k \) transitions.

For the other direction, assume that there is an FDFA \( B_G = (Q, \delta, \gamma, F, q_0) \) for \( L_G \) with \( s \) transitions. We argue that \( G \) must have a vertex cover of size \( k \).
First, since all words in $L_G$ have length two, $Q$ contains three disjoint sets: those reachable after reading 0, 1, or 2 symbols, respectively. The first set is the singleton \{q_0\}. The third set can also be assumed to be a singleton \{q_f\} = F. As for the middle set, it has to have at least $n + m$ states, one for each possible first symbol. The reason for this is that all the symbols in $V \cup E$ have different residual languages. Let $Q_1 = \{q_i \mid v_i \in V\} \cup \{p_{i,j} \mid e_{i,j} \in E\}$ be the states reached by reading one symbol (before taking any failure transitions).

We also notice that no state in $Q_1$ can have a failure transition to another state in $Q_1$, since for every pair $t_i, t_j \in Q_1$, neither $\Sigma_{t_i} \subseteq \Sigma_{t_j}$ nor $\Sigma_{t_j} \subseteq \Sigma_{t_i}$. This means that every failure transition must lead to a state that is not in $Q_1$.

Creating new states and failure transitions can only save transitions when states in $Q_1$ have overlapping residual languages. The only case where this happens is when every “edge state” $q_{i,j}$ has overlapping residual languages with $q_i$ and $q_j$. In the case of $q_i$, the overlap is $\{b_i, c_i\}$ and in the case of $q_j$, it is $\{b_j, c_j\}$.

It follows that the only way failure edges can save transitions is to let states $q_i$ fail to a new state $r_i$ on $b_i$ and $c_i$, let $r_i$ lead to $q_f$ on $b_i$ and $c_i$, and let states $p_{i,j}$ or $p_{j,i}$ also fail to $r_i$ on $b_i$ and $c_i$. We can count the savings we achieve in the following way. For every $q_i$ we add a failure edge to, we get one extra transition. For every $p_{i,j}$, on the other hand, that can fail to an $r_i$ corresponding to an incident vertex, we save one transition.

If $B_G$ has $s = 4n + 4m + k$ transitions, this means that we have “saved” $m - k$ transitions. Assume that we have added failure edges to $k'$ “vertex states” $q_i$. How many “edge states” must then have received failure edges? Let this number be $\ell$. We get $\ell - k' = m - k$. Notice that we must have $k' \leq k$, since $\ell \leq m$. If $k' = k$, then $\ell = m$ and we immediately have that $G$ has a vertex cover of size $k$. If, on the other hand, $k' < k$, we note that $m - \ell = k - k'$. In other words, the number of edges that are not using failure transitions equals $k$ minus the number of vertices that are using failure transitions. We can now construct a vertex cover for $G$ as follows. Include the $k'$ vertices whose corresponding states in $B_G$ have failure transitions in the cover. This leaves $k - k'$ edges uncovered. For each such edge, we select one of its endpoints arbitrarily and include it in the cover. The result is a cover of size $k$ for all the edges. □

4.3. Minimisation of binary automata

Binary automata (BFDFAs) are a restricted form of FDFAs, introduced by Kowal-towski et al. (1993). An F DFA $B = (Q, \Sigma, \delta, \gamma, q_0, F)$ is a BF DFA if there is at most one
non-failure transition from each state, i.e., for every \( p \in Q \) there is at most one \( a \in \Sigma \) such that \( \delta_a(p) \) is defined. This means that the automaton can be represented as a set of four-tuples \( (p, a, q, q') \), with \( \delta_a(p) = q \) and \( \gamma(p) = q' \). To minimise a BF DFA means to minimise the number of such tuples. It was conjectured by Kowalowski et al. (1993) that this problem is NP-complete. We show that this is indeed the case.

**Theorem 3.** The minimisation problem for binary automata is NP-complete.

**Proof.** For membership, it is enough to notice that for every BF DFA \( B \), just as for every DF DFA, an equivalent DFA \( A_B \) can be constructed in polynomial time. Thus a nondeterministic algorithm can, given \( B \), guess a sufficiently small BF DFA \( B' \), construct \( A_B \) and \( A_{B'} \), minimise them, and check for equivalence.

For NP-hardness, we again reduce from Vertex Cover. Given a graph \( G = (V, E) \) with \( |V| = n \) and \( |E| = m \) and an integer \( k \), we will construct a BF DFA \( B_G \) and an integer \( s \) such that the minimal BF DFA for \( L(B_G) \) has \( s \) or fewer tuples if and only if \( G \) has a vertex cover of size \( k \).

We first define \( L(B_G) \). As in the proofs of Theorem 1 and Theorem 2 we will use names for the vertices and edges of \( G \) as letters in our alphabet. Let \( V = \{v_1, \ldots, v_n\} \) and \( E = \{(e_{i,j} \mid (v_i, v_j) \in E \land i < j)\} \). Let \( \Sigma = V \cup \overline{E} \). We now define our language by

\[
L(B_G) = \bigcup_{(v_i, v_j) \in E} (e_{i,j} \circ (v_i + v_j)).
\]

In other words, \( L(B_G) \) contains edge names followed by the name of one of the vertices incident to the edge. In particular, all strings in \( L(B_G) \) have length two and the language is thus finite.

Given \( L(B_G) \) we can trivially construct \( B_G \) with \( 3m \) tuples. What we will show is that there is an equivalent BF DFA \( B'_G \) with \( s = 2m + k + 1 \) tuples if and only if \( G \) has a vertex cover of size \( k \).

Assume that \( C \subseteq V \) is a vertex cover for \( G \) and that \( |C| = k \). We construct \( B'_G = (Q, \delta, \gamma, q_0) \) as follows: For every edge \( (v_i, v_j) \in E \), there are two states, \( p_{i,j} \) and \( q_{i,j} \) in \( Q \). Additionally, \( Q \) has one state \( r_i \) for every vertex \( v_i \) in the cover \( C \). Finally, \( Q \) has an accepting state \( \top \) and a rejecting state \( \bot \). In total,

\[
Q = \{p_{i,j}, q_{i,j} \mid i < j \land (v_i, v_j) \in E\} \cup \{r_i \mid v_i \in C\} \cup \{\top, \bot\}.
\]

Let \( \prec \) be the lexicographical ordering on the edge names \( e_{i,j} \), i.e., \( e_{i,j} \prec e_{i',j'} \) if \( i < i' \) or if \( i = i' \) and \( j < j' \). We will also use this ordering on the corresponding sets of states.

For a state \( p_{i,j} \) we write \( \text{Next}(p_{i,j}) \) for the state that comes next in this ordering. The initial state of \( B'_G \) is \( q_0 = \min_{\prec} \{p_{i,j}\} \). For every edge name \( e_{i,j} \), we set \( \delta_{e_{i,j}}(p_{i,j}) = q_{i,j} \). For every edge name \( e_{i,j} \) except \( e_{i',j'} = \max_{\prec} \{e_{i,j}\} \) we also set \( \gamma(p_{i,j}) = \text{Next}(p_{i,j}) \). For \( e_{i',j'} \) we set \( \delta(e_{i',j'}) = \bot \). Next, we describe the transitions leaving the states \( q_{i,j} \). By assumption, either \( v_i \) or \( v_j \) (or both) belongs to \( C \). Assume, without loss of generality, that \( v_i \in C \). Then we set \( \delta_{v_i}(q_{i,j}) = \top \) and \( \gamma(q_{i,j}) = r_j \). For the states \( r_i \), we set \( \delta_{v_i}(r_i) = \top \) and \( \gamma(r_i) = \bot \). Finally, we set \( \gamma(\top) = \bot \). This completes the description of \( B'_G \). If we represent it as four-tuples, it will have one tuple per state, except for \( \bot \). Thus it has \( 2m + k + 1 \) tuples. It should be clear that \( B'_G \) accepts \( L(B_G) \).
We now need to show that if $G$ has no vertex cover of size $k$, then there is no BFDFA for $L(B_G)$ with $s$ or fewer tuples. Since each state can have only one transition that reads a letter, there must be $m$ four-tuples where the letter is an edge name. We can now ask how many different states we can be in after having just read one letter and not taken any failure transitions after that. Notice that for each edge name, the residual language is unique. In other words, there are no two edge names $e_{i,j}$ and $e_{i',j'}$ such that the sets of suffixes we can read after them to complete a string in $L(B_G)$ are identical. Thus there must be $m$ different states that we can be in directly after reading an edge name. Each such state contributes another tuple. These cannot, however, be the only states from which we can read a vertex name. Indeed, from each such state, we should be able to read two distinct vertex names. Thus there must be some extra states, which these states can fail to, and from where we can read exactly one vertex name. If two edge names represent edges that share an incident vertex, then the corresponding states could share an extra state. Therefore the smallest number of extra states is equal to the size of the smallest set of vertices such that each edge has at least one incident vertex in the set, or, in other words, the size of the smallest vertex cover for $G$. Additionally, we will need an accepting state and its corresponding tuple. Thus, if $G$ has no vertex cover of size $k$, then there can be no BFDFA for $L(B_G)$ of size smaller than $2m + k + 1$. □

5. Approximate transition reduction

Section 4 underlines the difficulty of finding optimal solutions. We therefore investigate the feasibility of approximations, focusing on the transition-reduction problem. As we shall see, there is a fast and easily implemented algorithm that saves at least two-thirds as many transitions as an optimal algorithm.

**Lemma 1.** Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA and $B = (Q, \Sigma, \delta_B, \gamma_B, q_0, F)$ a transition-minimal language-equivalent FDFA that can be constructed from $A$ by adding failure transitions and removing redundant regular transitions. Let $k = |A| - |B|$. There is a language-equivalent FDFA $C = (Q, \Sigma, \delta_C, \gamma_C, q_0, F)$ such that $k' = |A| - |C| \geq 2k/3$ and such that $\gamma_C$ is acyclic.

**Proof.** We first show that every cycle in $\gamma_B$ is of length 3 or more. Suppose that $B$ has a failure cycle of length two through states $p$ and $q$. This implies that $\Sigma(p) = \Sigma(q)$, since the states can fail to each other. By removing the failure transition from $p$ to $q$, and moving all transitions on tuples in $abil(q) \cap abil(p)$ from $q$ to $p$, we obtain a smaller automaton. Since the operation preserves the residual languages of $p$ and $q$, the new automaton is language-equivalent to the original one, contrary to the minimality assumption.

By repeatedly removing from each cycle of $\gamma_B$ the failure transition that saves the least regular transitions, the failure function can be made acyclic. Since $\gamma_B$ has out-degree at most 1, no edge can belong to more than one cycle. It therefore suffices to drop at most one third of the edges to clear all cycles. For each failure edge that is removed, at least two will remain, and each of them will save at least as many regular transitions as the removed edge. This means that when all cycles have been eliminated, we are left with a failure function $\gamma_C$ that saves at least $2/3$ as many transitions as $\gamma_B$.

Each failure function $\gamma$ on $Q$ describes a function graph $(Q, \gamma)$, i.e., a graph where each node has out-degree at most one.
Figure 6: The prospect graph for the DFA in Figure 2 (a)

**Observation 6.** Let $G = (V, E, w)$ be a directed graph with positive edge weights. Let $\gamma \subseteq E$ be such that $(V, \gamma)$ is an acyclic function graph. Then $(V, \gamma^{-1})$ is a forest, that is, an acyclic directed graph such that no vertex has in-degree larger than 1. Further more, if $(V, \gamma^{-1}, w)$ is a maximum-weight forest on $(V, E^{-1}, w)$, then $(V, \gamma, w)$ is a maximum-weight acyclic function graph on $G = (V, E, w)$.

In preparation for Theorem 4, we introduce the notion of a *prospect graph* for an automaton $A$. Intuitively, the graph tells us between what states failure transitions are useful and allowable: It is only meaningful to add failure transitions if they save regular transitions, and of course, they should not change the accepted language.

**Definition 1 (Prospect graph).** The *prospect graph* for $A$ is the weighted directed graph $P(A) = (Q, E, w)$, with

$$E = \{ (p, q) \mid \text{abil}(p) \cap \text{abil}(q) \neq \emptyset \text{ and } \Sigma(p) \subseteq \Sigma(q) \},$$

and

$$w((p, q)) = |\text{abil}(p) \cap \text{abil}(q)| - 1, \text{ for every } (p, q) \in E.$$

The prospect graph for the DFA of Figure 2 is shown in Figure 6. By adding a failure transition between any two of the states $q_1$, $q_2$, and $q_3$, we can save $n$ regular transitions at the cost of one failure transition. We may also add a failure transition from $q_4$ to $q_5$ or $q_6$, thereby saving 0 or 1 transitions, but the opposite direction is not allowed: if a failure transition were added from $q_6$ to $q_4$ it would be possible to read the symbol $a$ from $q_6$, and this would increase the language.

Theorem 4 below now follows immediately from the fact that it is possible to find a maximum forest on the prospect graph in polynomial time. An algorithm for this problem was discovered by Chu and Liu (1965) and, independently, by Edmonds (1967). A version with time complexity $O(|E| \log |V|)$ was provided by Tarjan (1977).

**Theorem 4.** The transition-reduction problem can be approximated within a factor $2/3$ in polynomial time.

The automaton in Figure 2 (b) is a transition-minimal state-minimal FDFA for $L(A)$ and saves $3n - 2$ transitions. Since its failure function contains a cycle, the above approximation technique will not find it, but it will find the FDFA in Figure 7 which saves $2n - 1$ transitions.
6. Heuristics for transition reduction

An alternative way of mitigating the computation complexity is to combine the heuristic minimisation algorithm by Kourie et al. (2012a) with simulation minimisation (Milner, 1982; Abdulla et al., 2009). The resulting algorithm is not an approximation, i.e. its performance is not guaranteed, but it has the upside of being applicable to nondeterministic input automata.

In the original algorithm, failure transitions are added between states with similar abilities to save on regular transitions. The simulation relation provides an additional layer of abstraction that lets us discover and do away with more redundancies.

6.1. Simulation relations

Given a preorder \( \preceq \) on \( Q \), we define the partition \( (Q/\preceq) \) by

\[
[p] = [q] \text{ if and only if } p \preceq q \text{ and } q \preceq p.
\]

We note that \( \preceq \) can be lifted to a preorder on \( (Q/\preceq) \) by letting \([p] \preceq [q]\) if and only if \( p \preceq q \). In fact, \( \preceq \) is a partial order on the new domain, because all equivalence classes are now singletons.

A simulation relation on an FFA \( A \) is a particular kind of preorder on its state set. Intuitively, a state \( q \) simulates a state \( p \) if \( A \) has a greater degree of freedom in terms of what symbols it can read when starting from \( q \) as compared to \( p \).

**Definition 2 (Simulation).** Let \( A = (Q, \Sigma, \delta, \gamma, I, F) \) be a FFA, and let \( \preceq \) be a pre-order on \( Q \). The relation \( \preceq \) is a simulation on \( A \) if for every \( p, q \in Q \) with \( p \preceq q \),

(i) \( p \in F \) implies \( q \in F \), and

(ii) if \( (p, p') \in \gamma_a \circ \delta_a \) for some \( a \in \Sigma \) and \( p' \in Q \), then there is a \( q' \in Q \) such that \( (q, q') \in \gamma_a \circ \delta_a \) and \( p' \preceq q' \). See Figure 6.1 for an illustration.

If \( p \) and \( q \) are such that \( p \preceq q \), then \( q \) is said to simulate \( p \). Recall that \( p \preceq q \) implies \( L(A^p) \subseteq L(A^q) \), but that the opposite direction is not necessarily true (Milner, 1982).
Figure 8: The preorder \( \preceq \) is a simulation, if it always follows from \( p \preceq q, a \in \Sigma, \) and \((p,p') \in \gamma^*_a \circ \delta_a\) that there is a \( q' \) such that \((q,q') \in \gamma^*_a \circ \delta_a\) and \( p' \preceq q' \).

From here on, let \( A = (Q, \Sigma, \delta, I, F) \) be an FA, and let \( \preceq \) be a simulation on \( A \). We can minimise \( A \) with respect to \( \preceq \) as follows:

Definition 3 (cf. (Buchholz, 2008, Definition 3.3)). The minimisation of \( A \) with respect to the simulation relation \( \preceq \) is the FA \( (A/\preceq) = ((Q/\preceq), \Sigma, \delta', I', F') \), where \( I' = \{ [q] \mid [q] \cap I \neq \emptyset \} \), \( F' = \{ [q] \mid q \in F \} \), and for every \( p \in Q \),

\[
\delta'_a([p]) = \max_{\preceq} \{ [q] \mid (p,q) \in \delta_a \}.
\]

The FA \( (A/\preceq) \) is language-equivalent with \( A \). There is a unique coarsest simulation \( \preceq_A \) on \( A \) (Paige and Tarjan, 1987), among all simulations on \( A \), the simulation \( \preceq_A \) yields the smallest output automaton, and \( \preceq_A \) is the coarsest simulation on \((A/\preceq_A)\) as well (Buchholz, 2008).

6.2. A heuristic algorithm

Algorithm 1 uses simulation relations to minimise a finite-state automaton \( A \) by adding failure transitions. Since the technique is effective even for nondeterministic automata, we present the algorithm at this more general level and then discuss the deterministic case separately. In particular, we now allow states to have more than one outgoing failure transition. We choose a ‘local’ interpretation of the semantics; if one computation branch of the automaton reaches a state \( q \) and cannot continue along a regular transition on the input symbol \( a \), then the computation may branch and follow each failure transition leaving \( q \). An alternative would be to use a ‘global’ condition, and require that every computation branch must be stuck on \( a \) before the failure transitions are explored. This second type of semantics is not treated here.

The first step is to minimise the input FA \( A \) with respect to \( \preceq_A \) to obtain the language-equivalent FA \((A/\preceq_A)\). When \( A \) is deterministic, this has the same effect as regular DFA minimisation. The FA \((A/\preceq_A)\) is then turned into an FFA \( B \) by using the transitive reduction \( \preceq_A \) of \( \preceq_A \) as failure relation. This means that a state \( p \) fails to a state \( p'' \), if \( p'' \preceq_A p \) and there is no state \( p' \) such that \( p'' \preceq_A p' \preceq_A p \). Finally, superfluous transitions are removed through a bottom-up traversal of \( \preceq_A \): If a state \( p \) can move on \( a \) to \( p' \), and \( p \preceq_A q \), then there is no sense in \( q \) also moving on \( a \) to \( p' \) since the failure edges will vouch for this behaviour. A formal presentation is given in Algorithm 1.

Before we turn to correctness and complexity, let us illustrate Algorithm 1 with an application from natural language processing.
Figure 9: Algorithm 1 minimises the input NFA (top) by adding failure transitions pointing downwards in the simulation hierarchy, and removing regular transitions made redundant, thereby turning it into a potentially smaller FNFA (below).
Algorithm 1. Reduce. Replace transitions with failure edges to save space.

Require: $A$ is a finite-state automaton
Ensure: $B$ is a language-equivalent FFA

1: compute $\preceq_A$ from $A$
2: compute $(A/\preceq_A) = (Q, \Sigma, (\delta_a)_{q\in\Sigma}, I, F)$
3: traverse $Q$ by following $\preceq_A$ bottom-up
4: for every $p \in Q$ do
5:    for every $q \in Q$ such that $p \preceq_A q$ do
6:       for every $a \in \Sigma$ do
7:          $\delta_a \leftarrow \delta_a \setminus \{(q, p') \mid (p, p') \in \delta_a\}$
8: return $B \leftarrow (Q, \Sigma, (\delta_a)_{q\in\Sigma}, \preceq_A, I, F)$

Example 4. One application of automata-based pattern matching is the extraction of chemical names in medical texts. A named-entity recogniser constructed for this purpose might identify the following names: antimony trisulphide, antimony tribromide, antimony trioxide, antimony disulphide, antimony dibromide, antimony dichloride, arsenic trisulphide, arsenic tribromide, arsenic trioxide, arsenic disulphide, arsenic dibromide, carbon disulphide, and carbon dibromide and store them as the FA in Figure 9 (top). It is easy to verify that the FA is minimal with respect to simulation, but since the states’ abilities overlap, there is room for improvement.

Algorithm 1 reduces the automaton further by replacing regular transitions by failure transitions pointing downwards in the hierarchy induced by the coarsest simulation. In this example, this means adding failure transitions from $q_4$ to $q_5$, from $q_5$ to $q_6$, and from $q_7$ and $q_8$ to $q_9$. The resulting FFA is shown in Figure 9 (bottom). Compared to the original automaton which had 18 regular transitions, the failure automaton has 11 regular and 4 failure transitions.

The output automaton $B$ of Algorithm 1 can be used with the regular FFA semantics, but we can, if we choose, shorten the time it takes to process an input string by exploiting the fact that $\preceq_A$ is also a simulation relation for $B$.

Lemma 2. Let $A$, $\preceq_A$, and $B$ be as in Alg. 1. The relation $\preceq_A$ is a simulation on $B$.

Proof. Recall that $\preceq_A$ is the coarsest simulation on $(A/\preceq_A)$. Since $(A/\preceq_A)$ and $B$ have the same sets of states and accepting states, Condition (i) of Definition 2 holds for all pairs in $\preceq_A$.

For Condition (ii), we note that whenever a state $q$ is reachable from a state $p$ on the symbol $a$ in $B$, then there is a state $q' \in Q$ that is reachable from $p$ on $a$ in $(A/\preceq_A)$ and such that $q \preceq_A q'$. This means that the set of maximal states reachable from $p$ on $a$ does not change. It follows that Condition (ii) of Definition 2 holds for all pairs of states in $\preceq_A$ under the transition relation of $B$, since it holds for them under the transition relation of $(A/\preceq_A)$.\[\square\]

Since we have access to a simulation $\preceq_A$ on $B$, the execution of $B$ can be made more efficient by repeatedly trimming the set of next states down to its maximal elements with
respect to $\preceq_A$. In other words, rather than moving from states $P$ to states $S = P \times Q \cap \delta_a$ on the symbol $a$, $B$ moves to $\max_{\preceq_A}(S)$. This limits the number of parallel computation branches, but does not change the accepted language.

6.3. Correctness and complexity

Let us now verify that Algorithm 1 behaves as expected.

**Theorem 5.** $\mathcal{L}(A) = \mathcal{L}(B)$.

**Proof.** Let $A = (Q, \Sigma, \delta, I, F)$ and $B = (Q, \Sigma, \delta', \gamma, I, F)$. We show that for every $q \in Q$, $\mathcal{L}(A^q) = \mathcal{L}(B^q)$. Since the automata have the same initial states, this proves the theorem.

Since every state that was reachable from $q \in Q$ in $A$ is still reachable on the same symbols in $B$, though possibly through a number of failure transitions, it is clear that $\mathcal{L}(A^q) \subseteq \mathcal{L}(B^q)$.

It remains to check that $\mathcal{L}(B^q) \subseteq \mathcal{L}(A^q)$. The proof is by induction on the length of $w \in \Sigma^*$. For $w = \varepsilon$, the claim is true because the final states of the two automata are the same.

Suppose then that $w = au \in \Sigma^*$, and that $(q, p) \in \delta'_a$ in $B$. If $(q, p) \in \delta_a$, then the induction hypothesis applied to $p$ and $u$ takes care of the rest, so suppose that $(q, p) \notin \delta_a$. There must then be a state $q'$ such that $(q, q') \in \gamma \circ \gamma^*$, and $(q', p) \in \delta'_a$. Since $q'$ can be reached from $q$ by following failure transitions, $q$ simulates $q'$. By definition of simulation, there is a state $p'$ that simulates $p$ such that $(q, p') \in \delta_a$. By the induction hypothesis, $u \in \mathcal{L}(B^p)$ implies that $u \in \mathcal{L}(A^p)$, and the simulation relation gives us $u \in \mathcal{L}(A^p')$, which completes the proof. $\square$

**Theorem 6.** Given an FA $A$ with $n$ states and a transition table of size $m$, Algorithm Reduce runs in time $O(mn)$.

**Proof.** The unique coarsest simulation on $A$ can be computed in time $O(mn)$ (Abdulla et al., 2009), and so can the transitive reduction of $\mathcal{R}$. The traversal of the transitive reduction needs $O(m)$ steps. During this traversal, we have to check the reachability of at most $n$ states. The overall time complexity is $O(mn + mn + mn) = O(mn)$. $\square$

We omit the proof of Theorem 7, since it follows directly from Lemma 2 and the definition of simulation.

**Theorem 7.** Let $\mathcal{L}'(B)$ be the language accepted by $B$ when $\preceq_A$ is used to trim the set of the next states. $\mathcal{L}'(B) = \mathcal{L}(B)$.

Intuitively, it makes sense to abandon a computational branch that reaches a state $p$, if there is a parallel branch that has reached $q$ and $q$ simulates $p$. In the worst case, however, we end up doing the additional work of computing maximal elements, but are not able to quit any branches. This makes the operation theoretically costly.

**Lemma 3.** Let $\preceq$ be a partial order on the set $Q$, and let $P \subseteq Q$. Computing the maximal elements of $P$ with respect to $\preceq$ is in $O(|P|^2)$. 

Proof. Under the assumption that \( \preceq \) is represented by a graph \( G \), we find the relevant subgraph \( G' \) of \( G \) in time \( O(|P|^2) \) by intersecting its vertex set with \( P \) and its edge set with \( P \times P \). We then identify the maximal elements of \( P \) by traversing \( G' \). This is feasible in linear time in the size of \( G' \), which in the worst case has \( O(|P|^2) \) edges (Turan, 1941).

Corollary 1. Trimming the set of next states down to its maximal elements has a quadratic worst-case time complexity.

6.4. The deterministic case

Even if the input automaton \( A \) is deterministic, there can still be a nontrivial simulation relation between the states in \((A/\preceq_A)\) that saves transitions. Furthermore, by trimming the set of next states, \( B \) becomes transition deterministic.

Theorem 8. Let \( A \) be a DFA, and let \( \delta \) be the transition relation of \( B = \text{Reduce}(A) \). For every \( w \in \Sigma^* \), \( |\max_{\preceq_A}(\delta(w))| \leq 1 \).

Proof. The proof is by induction on the length of the input string \( w \). If \( w = \varepsilon \), then the statement is true because \( A \) and \( B \) have the same initial states, and \( A \) is deterministic.

Suppose that \( w = ua \) for some string \( u \in \Sigma^* \) and symbol \( a \in \Sigma \). By the induction hypothesis, \( |\max_{\preceq_A}(\delta(u))| \leq 1 \). If \( |\max_{\preceq_A}(\delta(u))| = 0 \), then we are done. Suppose therefore that \( \max_{\preceq_A}(\delta(u)) = \{p\} \) for some \( p \in Q \), and let \( (p,q),(p,q') \in \hat{\delta}_a \). This means that there are states \( r,r' \in Q \) that are reachable from \( p \) along failure transitions, and such that \( (r,q),(r',q') \in \delta_a \). Since they are reachable along failure transitions, \( p \) simulates both \( r \) and \( r' \). It follows that there is a state \( q'' \) that simulates both \( q \) and \( q' \), and \( (p,q'') \in \delta_a \). By construction, \( q'' \) is reachable on \( a \) from \( p \) in \( B \), and since it simulates both \( q \) and \( q' \), the only way these states can be in \( \max_{\preceq_A}(\delta(wa)) \) is if \( q = q' = q'' \).

7. Conclusion

We have shown that the transition-reduction problem, and the related problems of transition minimisation and binary minimisation, are NP-complete. On a more positive note, transition-reduction can be approximated in polynomial time within two-thirds of the optimal solution, and redundancies in the input automaton’s structure can be detected by computing a simulation relation on its states. It remains to be seen whether transition-minimisation and binary minimisation also allow efficient approximation. In the case of transition-minimisation we conjecture that this is not the case, since the addition of auxiliary states appears to complicate matters.

The investigation of automata minimisation with failure transitions is practically motivated, and should also be practically verified. We therefore follow the line of work by Koutik et al. (2012c,a) with great interest. As mentioned earlier, the authors conduct experiments with failure minimisations of real-world and randomly generated automata. In the future, a comparative evaluation of our simulation-based algorithm is called for.

On the theoretical side, we look for a simple, necessary, and sufficient criterion to decide when additional states improve minimisation. There are also open questions related to the combination of failure transitions with other kinds of memory-saving techniques, and the interplay of failure semantics and nondeterminism is basically untouched.
**Acknowledgements.** We would like to thank anonymous reviewers for many useful comments. We gratefully acknowledge the financial support from the Swedish Research Council grant 621-2011-6080.

**References**


URL http://doi.acm.org/10.1145/360825.360855


URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1354682
