Five Determinisation Algorithms

Rob van Glabbeek[1,2] Bas Ploeger[3*]
rvg@cs.stanford.edu s.c.w.ploeger@tue.nl

[1] National ICT Australia, Locked Bag 6016, Sydney, NSW1466, Australia

[2] School of Computer Science and Engineering, The University of New South Wales
Sydney, NSW 2052, Australia

[3] Department of Mathematics and Computer Science, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

Determinisation of nondeterministic finite automata is a well-studied problem that plays an important role in compiler theory and system verification. In the latter field, one often encounters automata consisting of millions or even billions of states. On such input, the memory usage of analysis tools becomes the major bottleneck. In this paper we present several determinisation algorithms, all variants of the well-known subset construction, that aim to reduce memory usage and produce smaller output automata. One of them produces automata that are already minimal. We apply our algorithms to determinise automata that describe the possible sequences appearing after a fixed-length run of cellular automaton 110, and obtain a significant improvement in both memory and time efficiency.

1 Introduction

Finite state automata (or finite state machines) are an established and well-studied model of computation. From a theoretical point of view, they are an interesting object of study because they are expressive yet conceptually easy to understand and intuitive. They find applications in compilers, natural language processing, system verification and testing, but also in fields outside of (theoretical) computer science like switching circuits and chip design. Over the years, many flavours and variants of finite state machines have been defined and studied for a large variety of purposes.

One of the most classic and elementary type of finite state machine is the *nondeterministic finite automaton* (NFA). Typical applications of finite state automata involve checking whether some sequence of symbols meets some syntactic criterion, such as displaying a prescribed pattern or being correct input for a given program, a problem that can often be recast as checking whether that sequence is accepted by a given NFA.

A more restrictive type of automaton is the *deterministic finite automaton* (DFA). DFAs are as expressive as NFAs, in the sense that for every NFA there exists a DFA that is *language equivalent* (*i.e.* accepts the same input sequences). Contrary to NFAs, for any DFA there is a trivial linear time, constant space, online algorithm to check whether

*This author is partially supported by the Netherlands Organisation for Scientific Research (NWO) under VoLTS grant number 612.065.410.

an input sequence is accepted or not. Consequently, lexical-analyser generators like LEX work on DFAs, and so do many implementations of GREP. For this reason, in many applications it pays to convert NFAs into DFAs, even though the worst-case time and space complexities of this conversion are exponential in the size of the input NFA.

Once a language equivalent DFA of an NFA has been found, it is usually minimised to obtain the smallest such DFA. This minimal DFA is unique and the problem of finding it for a given NFA is called the *canonisation problem*.

Another application of NFAs is in the realm of *process theory* and *system verification* where they are used to model the behaviour of distributed systems. Typically, both a specification and an implementation of a system are represented as NFAs, and the question arises whether the execution sequences of one NFA are a subset of those of another. This is the *trace inclusion problem*. Although PSPACE-hard in general, this problem is decidable in PTIME once the NFAs are converted into equivalent DFAs.

As we see, in both the canonisation problem and the trace inclusion problem, determinisation plays an essential role. The standard determinisation algorithm is called *subset construction* (see *e.g.* [11]). Although the determinisation problem is EXPTIME-hard, this algorithm is renowned for its good performance in practice. For DFA minimisation a lot of algorithms have been proposed, of which Watson presents a taxonomy and performance analyses [16]. The algorithm with the best time complexity is by Hopcroft [10]: $\mathcal{O}(n \log n)$ where n is the number of states in the input DFA.

Another algorithm for canonisation is by Brzozowski [2]. It generates the minimal DFA directly from an input NFA by repeating the process of "reversing" and determinising the automaton twice. Tabakov and Vardi compare both approaches to canonisation experimentally by running them on randomly generated automata [15]. They show that the "subset-Hopcroft" approach performs best overall and for smaller transition densities, but for larger transition densities Brzozowski's algorithm is faster.

On some NFAs, the exponential blow-up by subset construction is unavoidable. However, we have encountered NFAs for which subset construction consumes a lot of memory and generates a DFA that is much larger than the minimal DFA. Therefore, our main goal is to find algorithms that are more memory efficient and produce smaller DFAs than subset construction.

In this paper we present five determinisation algorithms based on subset construction. For all of them we prove correctness. One algorithm generates the minimal DFA directly and hence is a *canonisation algorithm*. However, it calculates language inclusion as a subroutine; as deciding language inclusion is PSPACE-complete, it is unattractive to use in an implementation. The other four produce a DFA that is not necessarily minimal but is usually smaller than the DFA produced by subset construction.

We have implemented subset construction and these four new algorithms. We have benchmarked these implementations by running them on NFAs that describe patterns on the lines of a cellular automaton's evolution and on randomly generated automata. We compare the implementations on the time and memory needed for the complete canonisation process (*i.e.* including minimisation) and the size of the DFA after determinisation.

2 Preliminaries

Finite automata. A *nondeterministic finite automaton* (NFA) $\mathcal{N}$ is a tuple $(S_\mathcal{N}, \Sigma_\mathcal{N}, \delta_\mathcal{N}, i_\mathcal{N}, F_\mathcal{N})$ where $S_\mathcal{N}$ is a finite set of states, $\Sigma_\mathcal{N}$ is a finite input alphabet, $\delta_\mathcal{N} \subseteq S_\mathcal{N} \times \Sigma_\mathcal{N} \times S_\mathcal{N}$ is a transition relation, $i_\mathcal{N} \in S_\mathcal{N}$ is the initial state and $F_\mathcal{N} \subseteq S_\mathcal{N}$

is a set of final (or accepting) states. A *deterministic finite automaton* (DFA) is an NFA $\mathcal{D}$ such that for all $p \in S_{\mathcal{D}}$ and $a \in \Sigma_{\mathcal{D}}$ there is precisely one $q \in S_{\mathcal{D}}$ such that $(p, a, q) \in \delta_{\mathcal{D}}$.

In graphical representations of DFAs we also allow states that have *at most* one outgoing a-transition for each alphabet symbol a. Formally speaking, these abbreviate the DFA obtained by adding a non-accepting *sink* state as the target of all missing transitions. Note that adding such a state preserves language equivalence (defined below).

For any alphabet Σ, Σ^* denotes the set of all finite strings over Σ and $\varepsilon \in \Sigma^*$ denotes the empty string. Any subset of Σ^* is called a *language over* Σ. For any states $p, q \in S_{\mathcal{N}}$ of an NFA $\mathcal{N}$ and string $\sigma \in \Sigma_{\mathcal{N}}^*$ with $\sigma = \sigma_1 \cdots \sigma_n$ and $\sigma_1, \ldots, \sigma_n \in \Sigma_{\mathcal{N}}$ for some $n \geq 0$, we write $p \xrightarrow{\sigma}_{\mathcal{N}} q$ to denote the fact that:

$$\exists p_0, \ldots, p_n \in S_{\mathcal{N}} \,.\, p_0 = p \,\wedge\, p_n = q \,\wedge\, (p_0, \sigma_1, p_1), \ldots, (p_{n-1}, \sigma_n, p_n) \in \delta_{\mathcal{N}}.$$

Language semantics. The *language of a state* $p \in S_{\mathcal{N}}$ of an NFA $\mathcal{N}$ is defined as: $\mathcal{L}_{\mathcal{N}}(p) = \{\sigma \in \Sigma_{\mathcal{N}}^* \mid \exists q \in F_{\mathcal{N}} \,.\, p \xrightarrow{\sigma}_{\mathcal{N}} q\}$. The *language of an NFA* $\mathcal{N}$ is defined as: $\mathcal{L}(\mathcal{N}) = \mathcal{L}_{\mathcal{N}}(i_{\mathcal{N}})$. For any NFAs $\mathcal{N}$ and $\mathcal{M}$ and states $p \in S_{\mathcal{N}}$ and $q \in S_{\mathcal{M}}$, p is *language included* in q, denoted $p \sqsubseteq_L q$, iff $\mathcal{L}_{\mathcal{N}}(p) \subseteq \mathcal{L}_{\mathcal{M}}(q)$. Moreover, p and q are *language equivalent*, denoted $p \equiv_L q$, iff $p \sqsubseteq_L q \wedge q \sqsubseteq_L p$. An NFA $\mathcal{N}$ is *language included* in an NFA $\mathcal{M}$ iff $i_{\mathcal{N}} \sqsubseteq_L i_{\mathcal{M}}$ and $\mathcal{N}$ and $\mathcal{M}$ are *language equivalent* iff $i_{\mathcal{N}} \equiv_L i_{\mathcal{M}}$.

Simulation semantics. Given NFAs $\mathcal{N}$ and $\mathcal{M}$, a relation $R \subseteq S_{\mathcal{N}} \times S_{\mathcal{M}}$ is a *simulation* iff for any $p \in S_{\mathcal{N}}$ and $q \in S_{\mathcal{M}}$, $p \; R \; q$ implies:

- $p \in F_{\mathcal{N}} \Rightarrow q \in F_{\mathcal{M}}$ and
- $\forall a \in \Sigma_{\mathcal{N}} \,.\, \forall p' \in S_{\mathcal{N}} \,.\, p \xrightarrow{a}_{\mathcal{N}} p' \;\Rightarrow\; \exists q' \in S_{\mathcal{M}} \,.\, q \xrightarrow{a}_{\mathcal{M}} q' \,\wedge\, p' \; R \; q'$.

Given NFAs $\mathcal{N}$ and $\mathcal{M}$, for any $p \in S_{\mathcal{N}}$ and $q \in S_{\mathcal{M}}$:

- p is *simulated by* q, denoted $p \subsetneq q$, iff there exists a simulation R such that $p \; R \; q$;
- p and q are *simulation equivalent*, denoted $p \rightleftarrows q$, iff $p \subsetneq q \wedge q \subsetneq p$;

Clearly $p \subsetneq q$ implies $p \sqsubseteq_L q$.

Subset construction. The subset construction (or powerset construction) is the standard way of determinising a given NFA. For reasons that will become apparent in the next sections, we slightly generalise the normal algorithm by augmenting it with a function f on sets of states of the input NFA, which is applied to every generated set. The algorithm is Algorithm 1 and shall be referred to as $\textsc{Subset}(f)$. It takes an NFA $\mathcal{N}$ and generates a DFA $\mathcal{D}$. Of course, it should be the case that $\mathcal{N} \equiv_L \mathcal{D}$, which depends strongly on the function f. For normal subset construction, $\textsc{Subset}(\mathcal{I})$, where $\mathcal{I}$ is the identity function, it is known that the language of $\mathcal{N}$ is indeed preserved. In Appendix A we show that the same holds for $\textsc{Subset}(f)$, for any function f that satisfies $f(P) \equiv_L P$ for every set of states $P \subseteq S_{\mathcal{N}}$. In the sequel, whenever we use the term "subset construction" we mean the normal algorithm, *i.e.* $\textsc{Subset}(\mathcal{I})$.

It is known that in the worst case, determinisation yields a DFA that is exponentially larger than the input NFA. An example of an NFA that gives rise to such an exponential blow-up is the NFA that accepts the language specified by the regular expression $\Sigma^* x \Sigma^n$ for some alphabet Σ, $x \in \Sigma$ and $n \geq 0$. Figure 1(a) shows the NFA for $\Sigma = \{a, b\}$ and $x = a$. This NFA has $n + 2$ states, whereas the corresponding DFA has 2^{n+1} states and is already minimal.

Algorithm 1 The $\text{SUBSET}(f)$ determinisation algorithm

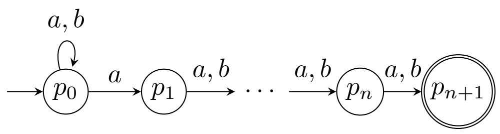
Pre: $\mathcal{N} = (S_\mathcal{N}, \Sigma_\mathcal{N}, \delta_\mathcal{N}, i_\mathcal{N}, F_\mathcal{N})$ is an NFA
Post: $\mathcal{D} = (S_\mathcal{D}, \Sigma_\mathcal{D}, \delta_\mathcal{D}, i_\mathcal{D}, F_\mathcal{D})$ is a DFA

$\Sigma_\mathcal{D} := \Sigma_\mathcal{N}$; $\delta_\mathcal{D} := \emptyset$; $i_\mathcal{D} := f(\{i_\mathcal{N}\})$; $F_\mathcal{D} := \emptyset$;
$S_\mathcal{D} := \{i_\mathcal{D}\}$; $\text{todo} := \{i_\mathcal{D}\}$; $\text{done} := \emptyset$;
while $\text{todo} \neq \emptyset$ **do**
  pick a $P \in \text{todo}$;
  for all $a \in \Sigma_\mathcal{N}$ **do**
    $P' := f(\{p' \in S_\mathcal{N} \mid \exists p \in P \,.\, p \xrightarrow{a}_\mathcal{N} p'\})$;
    $S_\mathcal{D} := S_\mathcal{D} \cup \{P'\}$;
    $\delta_\mathcal{D} := \delta_\mathcal{D} \cup \{(P, a, P')\}$;
    $\text{todo} := \text{todo} \cup (\{P'\} \setminus \text{done})$;
  end for
  if $\exists p \in P \,.\, p \in F_\mathcal{N}$ **then**
    $F_\mathcal{D} := F_\mathcal{D} \cup \{P\}$;
  end if
  $\text{todo} := \text{todo} \setminus \{P\}$;
  $\text{done} := \text{done} \cup \{P\}$;
end while

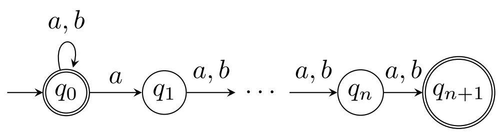
An interesting thing to note is that if the initial state were accepting (Figure 1(b)), the minimal DFA would consist of only one state with an a, b-loop: the accepted language has become Σ^*. However, subset construction still produces the exponentially larger DFA first, which should then be reduced to obtain the single-state, minimal DFA.

3 Determinisation using Transition Sets

In this section we show that subset construction can just as well be done on sets of transitions as on sets of states. We observe that the contribution of an NFA state p to the behaviour of a DFA state P consists entirely of p's outgoing transitions. We no longer think of a DFA state as being a set of NFA states, but rather a set of NFA *transitions*.



(a)



(b)

Figure 1: Two NFAs of size $\mathcal{O}(n)$ for which subset construction produces a DFA of size $\mathcal{O}(2^n)$. Here initial states are marked by unlabelled incoming arrows, and final states by double circles. In case (a) this DFA is already minimal; in case (b) the minimal DFA has size 1.

Algorithm 2 The TRANSSET(f) determinisation algorithm

Pre: $\mathcal{N} = (S_\mathcal{N}, \Sigma_\mathcal{N}, \delta_\mathcal{N}, i_\mathcal{N}, F_\mathcal{N})$ is an NFA
Post: $\mathcal{D} = (S_\mathcal{D}, \Sigma_\mathcal{D}, \delta_\mathcal{D}, i_\mathcal{D}, F_\mathcal{D})$ is a DFA

```
Σ_D := Σ_N; δ_D := ∅; i_D := f(tuple(i_N)); F_D := ∅;
S_D := {i_D}; todo := {i_D}; done := ∅;
while todo ≠ ∅ do
    pick a P ∈ todo;
    for all a ∈ Σ do
        P' := f( ⋃_{(a,p)∈set(P)} trans(p), ∃(a,p) ∈ set(P) . p ∈ F_N );
        S_D := S_D ∪ {P'};
        δ_D := δ_D ∪ {(P, a, P')};
        todo := todo ∪ ({P'} \ done);
    end for
    if fin(P) then
        F_D := F_D ∪ {P};
    end if
    todo := todo \ {P};
    done := done ∪ {P};
end while
```

Definition 1. Given an NFA $\mathcal{N}$, a *transition tuple* is a pair (T, b) where $T \in \mathcal{P}(\Sigma_\mathcal{N} \times S_\mathcal{N})$ is a set of transitions and $b \in \mathbb{B}$ is a boolean.

For every transition tuple (T, b) we define the projection functions set and fin as: $\mathsf{set}(T, b) = T$ and $\mathsf{fin}(T, b) = b$. For every state $p \in S_\mathcal{N}$ of NFA $\mathcal{N}$, $\mathsf{trans}(p)$ is the set of outgoing transitions of p and $\mathsf{tuple}(p)$ is the transition tuple belonging to p:

$$\begin{array}{ll} \mathsf{trans}(p) & = \{(a, q) \in \Sigma_\mathcal{N} \times S_\mathcal{N} \mid p \xrightarrow{a}_\mathcal{N} q\} \\ \mathsf{tuple}(p) & = (\mathsf{trans}(p), p \in F_\mathcal{N}). \end{array}$$

The DFA state $P \subseteq S_\mathcal{N}$ now corresponds to the transition tuple (T, b) where $T = \bigcup_{p \in P} \mathsf{trans}(p)$ and $b \equiv \exists p \in P \,.\, p \in F_\mathcal{N}$. We need the boolean b to indicate whether the DFA state is final as this can no longer be determined from the elements of the set. Only the labels and target states of the transitions are stored because the source states are irrelevant and would only make the sets unnecessarily large.

Given NFA $\mathcal{N}$, the *language of a transition* $(a, p) \in \Sigma_\mathcal{N} \times S_\mathcal{N}$ is defined as: $\mathcal{L}_\mathcal{N}(a, p) = \{a\sigma \in \Sigma_\mathcal{N}^* \mid \sigma \in \mathcal{L}_\mathcal{N}(p)\}$. The *language of a set of transitions* T is defined as $\mathcal{L}_\mathcal{N}(T) = \bigcup_{t \in T} \mathcal{L}_\mathcal{N}(t)$ and the *language of a transition tuple* (T, b) is defined as:

$$\mathcal{L}_\mathcal{N}(T, b) = \mathcal{L}_\mathcal{N}(T) \cup \begin{cases} \{\varepsilon\} & \text{if } b \\ \emptyset & \text{if } \neg b. \end{cases}$$

Language inclusion and equivalence for transitions and transition tuples can now be defined in the usual way by means of set inclusion and equality.

The determinisation algorithm that uses transition tuples is Algorithm 2. We shall refer to it as TRANSSET(f) where f is a function on transition tuples. Again, language preservation depends on the specific function f being used. For $f = \mathcal{I}$ — and more generally when f satisfies $f(P) \equiv_L P$ for each transition tuple P — this is indeed the case, which we prove in Appendix B. Using TRANSSET($\mathcal{I}$) for determinisation can give a smaller DFA than SUBSET($\mathcal{I}$) as is shown by the example in Figure 2. Here,

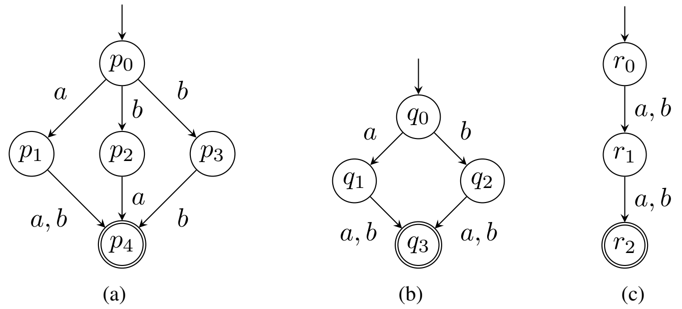


Figure 2: NFA (a) for which the DFA produced by $\text{SUBSET}(\mathcal{I})$ (b) is larger than the (minimal) DFA produced by $\text{TRANSSET}(\mathcal{I})$ (c).

$\text{TRANSSET}(\mathcal{I})$ happens to produce the minimal DFA directly. This is generally not the case: on the NFA of Figure 1(b), $\text{TRANSSET}(\mathcal{I})$ generates a DFA of size 2^{n+1}, while the minimal DFA has size 1.

4 Determinisation using Closures

We introduce a *closure* operation that can be used in the SUBSET algorithm instead of the identity function $\mathcal{I}$. It aims to add NFA states to a given DFA state (*i.e.* a set of NFA states) without affecting its language. This results in an algorithm that generates smaller DFAs. In particular, we show that if the criterion to add a state is chosen suitably, SUBSET with closure is an algorithm that produces the minimal DFA directly.

Definition 2. For any set of states $P \subseteq S_{\mathcal{N}}$ of an NFA $\mathcal{N}$ and relation $\sqsubseteq\ \subseteq S_{\mathcal{N}} \times \mathcal{P}(S_{\mathcal{N}})$, the *closure* of P under $\sqsubseteq$, $\mathsf{close}_{\sqsubseteq}(P)$, is defined as:

$$\mathsf{close}_{\sqsubseteq}(P) = \{p \in S_{\mathcal{N}} \mid p \sqsubseteq P\}.$$

The language preorder $\sqsubseteq_L$ can be lifted to operate on states and sets of states in the following way. Define the *language of a set of states* P of an NFA $\mathcal{N}$ as: $\mathcal{L}_{\mathcal{N}}(P) = \bigcup_{p \in P} \mathcal{L}_{\mathcal{N}}(p)$. Language equivalence and inclusion can now be defined on any combination of states and sets of states, in terms of set equivalence and inclusion. For instance, for a state $p \in S_{\mathcal{N}}$ and a set of states $P \subseteq S_{\mathcal{N}}$, $p \sqsubseteq_L P$ holds if $\mathcal{L}_{\mathcal{N}}(p) \subseteq \mathcal{L}_{\mathcal{N}}(P)$.

Applying this, the algorithm $\text{SUBSET}(\mathsf{close}_{\sqsubseteq_L})$ generates the minimal DFA that is language equivalent to the input NFA. This statement is proven in Appendix A.1.

5 Simulation Preorder

Although it ensures that the output DFA of $\text{SUBSET}(\mathsf{close}_{\sqsubseteq_L})$ is minimal, language inclusion is an unattractive preorder to use. Deciding language inclusion is PSPACE-complete [13] which implies that known algorithms have an exponential time complexity. Moreover, most algorithms involve a determinisation step which would render our optimisation useless.

The simulation preorder $\subsetneq$ [12] is finer than language inclusion on NFAs, meaning it relates fewer NFAs. However, considering its PTIME complexity (see *e.g.* [1, 9]), it is

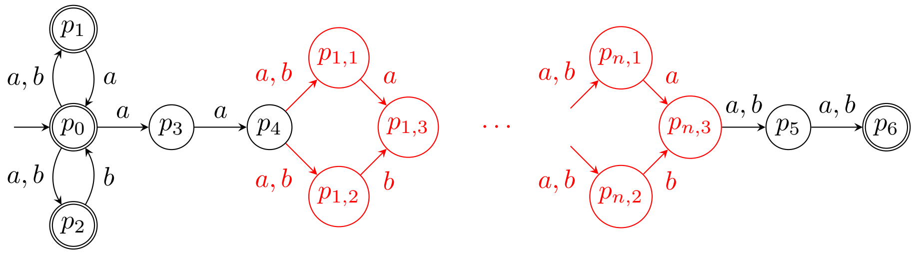


Figure 3: NFA of size $\mathcal{O}(n)$ for which $\text{SUBSET}(\mathsf{close}_{\subsetneq})$ generates a DFA of size $\mathcal{O}(2^n)$ for any $n \geq 1$. The minimal DFA has 1 state.

an attractive way to "approximate" language inclusion (see also [4]). Hence, as a more practical alternative to $\text{SUBSET}(\mathsf{close}_{\sqsubseteq_L})$ we define the algorithm $\text{SUBSET}(\mathsf{close}_{\subsetneq})$. The required lifting of $\subsetneq$ to states and sets of states is as follows. For any state $p \in S_\mathcal{N}$ and set of states $P \subseteq S_\mathcal{N}$ of an NFA $\mathcal{N}$, we have $p \subsetneq P$ iff:

- $p \in F_\mathcal{N} \Rightarrow \exists q \in P \,.\, q \in F_\mathcal{N}$ and
- there exists a simulation $R \subseteq S_\mathcal{N} \times S_\mathcal{N}$ such that:
  $\forall a \in \Sigma_\mathcal{N} \,.\, \forall p' \in S_\mathcal{N} \,.\, p \xrightarrow{a}_\mathcal{N} p' \;\Rightarrow\; \exists q, q' \in S_\mathcal{N} \,.\, q \in P \wedge q \xrightarrow{a}_\mathcal{N} q' \;\wedge\; p' \,R\, q'.$

The correctness of $\text{SUBSET}(\mathsf{close}_{\subsetneq})$ is established in Appendix A.2. The example in Figure 3 shows not only that the resulting DFA is no longer minimal, but moreover that it can be exponentially larger than the minimal DFA. This NFA contains a pattern that repeats itself n times for any $n \geq 1$. It is based on the NFA of Figure 1(b) interwoven with a pattern that prevents $\text{SUBSET}(\mathsf{close}_{\subsetneq})$ from merging states that will later turn out to be equivalent. The NFA accepts the language given by the regular expression $(a \mid b)^*$.

6 Determinisation using Compressions

Algorithm $\text{SUBSET}(\mathsf{close}_{\subsetneq})$ adds all simulated states to a generated set of states. Another option would be to remove all redundant states from such a set. More specifically, we remove every state that is simulated by another state in the set. For this operation to be well-defined, it is essential that no two different states in the set are simulation equivalent. This can be achieved by minimising the input NFA using simulation equivalence prior to determinisation. In turn, this amounts to computing the simulation preorder that was already necessary in the first place.

Definition 3. Given a set P such that $\neg\exists p, q \in P \,.\, p \neq q \wedge p \rightleftarrows q$. Then $\mathsf{compress}_{\subsetneq}(P)$ denotes the *compression* of P under $\subsetneq$ and is defined as:

$$\mathsf{compress}_{\subsetneq}(P) = \{p \in P \mid \forall q \in P \,.\, p \neq q \Rightarrow p \not\subsetneq q\}.$$

The function $\mathsf{compress}_{\subsetneq}$ can be used not only for sets of states but also for transition tuples. For that, we first define $\subsetneq$ on the transitions of an NFA $\mathcal{N}$ as follows. For any $(a, p), (b, q) \in \Sigma_\mathcal{N} \times S_\mathcal{N}$, we have $(a, p) \subsetneq (b, q)$ iff $a = b$ and $p \subsetneq q$. By Definition 3 $\mathsf{compress}_{\subsetneq}$ is now properly defined on sets of transitions and it can be extended to transition tuples in a straightforward manner: $\mathsf{compress}_{\subsetneq}(T, b) = (\mathsf{compress}_{\subsetneq}(T), b)$.

$$\begin{tikzcd}
 & \textsc{Subset}(\mathcal{I}) & \\
\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}}) \arrow[ur] & & \textsc{Transset}(\mathcal{I}) \arrow[ul] \\
 & \textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}}),\ \textsc{Transset}(\mathsf{compress}_{\underrightarrow{\subseteq}}) \arrow[ul] \arrow[ur] & \\
 & \textsc{Subset}(\mathsf{close}_{\sqsubseteq_L}) \arrow[u] &
\end{tikzcd}$$

Figure 4: The lattice of algorithms presented in the previous sections.

This way, we obtain two more determinisation algorithms: $\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ and $\textsc{Transset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$. Their correctness proofs are included in Appendices A.3 and B.2, respectively.

7 Lattice of Algorithms

We order the algorithms described in the previous sections in a lattice. The ordering $\preceq$ on the algorithms is as follows: $A \preceq B$ iff for every input NFA, A produces a DFA that is at most as large as the one produced by B. The lattice is depicted in Figure 4 where an arrow from A to B denotes that $A \preceq B$.

$\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$ and $\textsc{Transset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ are in the same class of the lattice, because these algorithms always yield isomorphic DFAs. This statement is substantiated in Appendix C, as well as the validity of the other $\preceq$-relations of Figure 4. The following shows that the lattice is complete, in the sense that there are no further $\preceq$-relations between our algorithms:

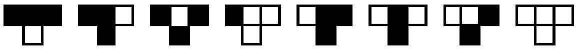
- $\textsc{Subset}(\mathsf{close}_{\sqsubseteq_L})$ is the unique $\preceq$-smallest algorithm because it is the only one that always generates the minimal DFA;
- $\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}}) \not\preceq \textsc{Transset}(\mathcal{I})$ by the example in Figure 2;
- $\textsc{Transset}(\mathcal{I}) \not\preceq \textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ by the example in Figure 1(b).

8 Implementation and Benchmarks

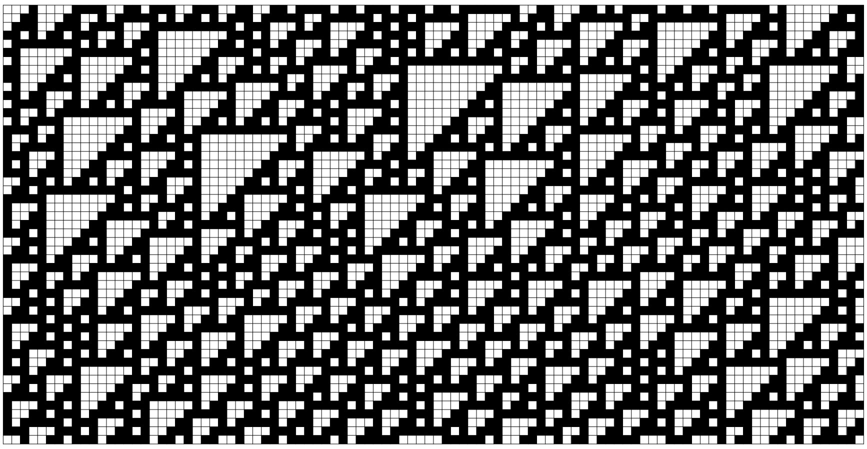
We have implemented the algorithms $\textsc{Subset}(\mathcal{I})$, $\textsc{Transset}(\mathcal{I})$, $\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$, $\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ and $\textsc{Transset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ in the C++ programming language. A set of states or transitions is stored as a tree with the elements in the leaves. All subtrees are shared among the sets to improve memory efficiency. A hash table provides fast and efficient lookup of existing subtrees.

The benchmarks are performed on a 32-bits architecture computer having two Intel Xeon 3.06 GHz CPUs and 4 GB of RAM. It runs Fedora Core 8 Linux, kernel 2.6.23. The code is compiled using the GNU C++ compiler (version 4.1.2).

Every benchmark starts off by minimising the NFA using simulation equivalence. For this we have implemented our partitioning algorithm [7] which is based on [6] and also computes the simulation preorder on the states of the resulting NFA. Every determinisation algorithm is applied to this minimised NFA, after which the resulting DFA is minimised by the tool ltsmin of the µCRL toolset [3, 8] (version 2.18.1).



(a) Rule 110



(b) Evolution

Figure 5: Example of a cellular automaton (rule 110).

8.1 Cellular Automaton 110

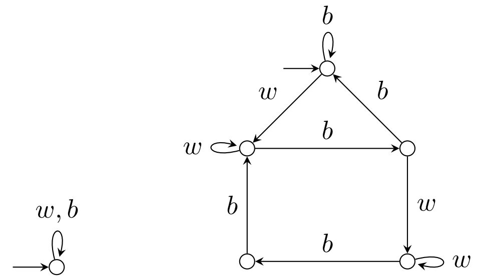
In his book [18], Wolfram studies cellular automata as a model of computation. A *cellular automaton* consists of a line of white or black cells[1] of which the colours are changed in every step of the automaton. The colour of a cell in the next step of the automaton's computation depends on its current colour and those of its left- and right-hand neighbours, as specified by a so-called *rule*. An example is given in Figure 5. The rule is depicted in Figure 5(a). For example, it specifies that if a cell is black and both of its neighbours are black, then that cell becomes white in the next step of the automaton's evolution (cf. the leftmost part of the rule).

From Figure 5(a), it is easy to see that there are 256 such rules. The rules can be numbered uniquely in a straightforward way by taking the bottom row and reading 0 for a white cell and 1 for a black cell. This gives the number for that rule in binary notation. For example, the rule in Figure 5(a) has number 110 in decimal notation (01101110 in binary).

Figure 5(b) shows the *evolution* of this automaton. The first line is the initial state, for which the colours of the cells have been chosen randomly. Every successive line shows the next step in the evolution and is computed by applying the rule to every subsequence of length 3 on the previous line. The line of white or black cells on which the rule operates can be chosen to be two-way infinite or cyclic. Figure 5(b) is an example of the latter: the lines are assumed to "wrap", meaning that the left-hand neighbour of a cell in the leftmost column is the cell in the rightmost column of the same line, and vice versa. Here, we have chosen a line width of 100 cells and the first 50 steps of the evolution are depicted.

In general, a one-dimensional cellular automaton can be formally represented by

[1] Actually, a wide variety of cellular automata can be defined. We consider a basic type here with two colours and a "neighbourhood" of 1.



(a) 0 steps (b) 1 step

Figure 6: Minimal DFAs describing the possible sequences of white and black cells that can occur after 0 steps (a) and 1 step (b) of cellular automaton 110.

a function $\rho : \Sigma^w \to \Sigma$, the rule, where Σ is an alphabet and $w \geq 1$ is the *width* of the automaton. Given an infinite sequence $\sigma \in \Sigma^\infty$, a *step* of a CA is an application of ρ to every w-length subsequence of σ, which produces a new infinite sequence. In the example of Figure 5(a) we have $\Sigma = \{\mathrm{white}, \mathrm{black}\}$, $w = 3$ and ρ is as depicted. In Figure 5(b) we assume that the infinite sequences $\sigma \in \Sigma^\infty$ are periodic with a period of 100 cells, and only one period is displayed. Periodicity of the input sequence guarantees that the successive sequences are also periodic, with a period of the same size.

Wolfram classifies cellular automata based on the complexity of the patterns that emerge in their evolutions. Four classes are distinguished of which class 1 contains the simplest automata and class 4 the most complex. An example of a class 1 automaton is the one with rule number 0, which simply colours every cell white in the first step and retains this state in subsequent steps. The complex pattern of Figure 5(b) identifies automaton 110 as a class 4 automaton. Moreover, Wolfram has shown that the 110 automaton is *universal* or *Turing complete*, which means it can perform exactly the same computations a Turing machine can. To be precise: given a (possibly universal) Turing machine M, there are finite sequences $\rho, \nu \in \{\mathrm{white}, \mathrm{black}\}^*$ as well as an encoding of any (finite) input sequence σ of M as a finite sequence $\sigma' \in \{\mathrm{white}, \mathrm{black}\}^*$, such that the behaviour of M on the input σ is in some sense mimicked (through a complicated encoding) by the evolution of cellular automaton 110 on the infinite input sequence composed of σ', flanked on the left by infinitely many repetitions of ρ and on the right by infinitely many repetitions of ν.

As described in [17], the possible finite sequences appearing as a continuous subsequence of the infinite sequence obtained after n steps of a given cellular automaton (starting from a random input sequence) constitute a language that can be described by a DFA. For example, the DFA that describes the possible sequences after 0 steps of cellular automaton 110 is depicted in Figure 6(a): any sequence of white or black cells is allowed. The DFA after 1 step is shown in Figure 6(b). Both DFAs are minimal. Here every state can be considered final, except for omitted sink states (see Section 2). It is known that for some rules, the size of these DFAs increases exponentially in n (*cf.* [14]). Rule 110 exhibits this phenomenon.

We have generated the minimal DFAs for steps 1 through 6 of this CA using the various algorithms presented here. We ran the algorithms $\textsc{Subset}(\mathcal{I})$ and $\textsc{TransSet}(\mathcal{I})$ with and without minimisation of the input NFA modulo simulation equivalence prior to determinisation. It turns out that the costs in time and memory of this prior minimisation step is very small compared to the costs of the subsequent determinisation

size of input NFA: 800
after applying SIM: 228
minimal DFA: 1 357

	STEP 4						
	SIM**T**	**DT**	**MT**	SIM**S**	**DS**	**MS**	$\lvert S_{\mathcal{D}} \rvert$
SUBSET($\mathcal{I}$)	–	2.55	1.08	–	15.4	5.3	152 804
TRANSSET($\mathcal{I}$)	–	1.87	0.60	–	18.0	3.3	94 473
SUBSET($\mathcal{I}$) after SIM	0.06	0.6	0.4	0.4	5.4	2.0	58 370
TRANSSET($\mathcal{I}$) after SIM	0.06	1.0	0.4	0.4	9.0	2.0	58 094
SUBSET($\mathsf{close}_{\underrightarrow{\subseteq}}$)	0.06	1.6	< 0.1	0.4	2.1	0.2	4 720
SUBSET($\mathsf{compress}_{\underrightarrow{\subseteq}}$)	0.06	< 0.1	< 0.1	0.4	0.6	0.2	4 745
TRANSSET($\mathsf{compress}_{\underrightarrow{\subseteq}}$)	0.06	< 0.1	< 0.1	0.4	0.7	0.2	4 720

size of input NFA: 5 224
after applying SIM: 1 421
minimal DFA: 18 824

	STEP 5						
	SIM**T**	**DT**	**MT**	SIM**S**	**DS**	**MS**	$\lvert S_{\mathcal{D}} \rvert$
SUBSET($\mathcal{I}$)	–	1250.3	179.3	–	1966.4	623.8	17 960 608
TRANSSET($\mathcal{I}$)	–	525.1	121.6	–	1376.5	418.3	12 083 653
SUBSET($\mathcal{I}$) after SIM	2.28	212.5	76.7	5.9	688.2	267.2	7 663 165
TRANSSET($\mathcal{I}$) after SIM	2.28	257.3	79.1	5.9	1 146.9	263.0	7 541 248
SUBSET($\mathsf{close}_{\underrightarrow{\subseteq}}$)	2.28	2 739.7	1.6	5.9	123.2	6.3	176 008
SUBSET($\mathsf{compress}_{\underrightarrow{\subseteq}}$)	2.28	4.3	1.4	5.9	16.7	6.4	179 146
TRANSSET($\mathsf{compress}_{\underrightarrow{\subseteq}}$)	2.28	4.1	1.6	5.9	22.9	6.3	176 008

size of input NFA: 73 905
after applying SIM: 18 934
minimal DFA: 136 401

	STEP 6						
	SIM**T**	**DT**	**MT**	SIM**S**	**DS**	**MS**	$\lvert S_{\mathcal{D}} \rvert$
SUBSET($\mathcal{I}$)	–	†	†	–	†	†	†
TRANSSET($\mathcal{I}$)	–	†	†	–	†	†	†
SUBSET($\mathcal{I}$) after SIM	1343.8	†	†	433.1	†	†	†
TRANSSET($\mathcal{I}$) after SIM	1343.8	†	†	433.1	†	†	†
SUBSET($\mathsf{close}_{\underrightarrow{\subseteq}}$)	1343.8	$> 160\,000$	?	433.1	?	?	?
SUBSET($\mathsf{compress}_{\underrightarrow{\subseteq}}$)	1343.8	1 467.2	90.0	433.1	770.9	244.9	7 100 549
TRANSSET($\mathsf{compress}_{\underrightarrow{\subseteq}}$)	1343.8	241.9	88.4	433.1	770.9	234.4	6 770 155

Table 1: Benchmark results for canonising NFAs of steps 4, 5 and 6 of CA 110.
Legend: SIM = Minimisation modulo simulation equivalence prior to determinisation, **D** = Determinisation, **M** = Minimisation, **T** = Time (sec), **S** = Space (peak memory use, MB), $\lvert S_{\mathcal{D}} \rvert$ = Size of DFA after determinisation, † = out of memory, ? = not measured.

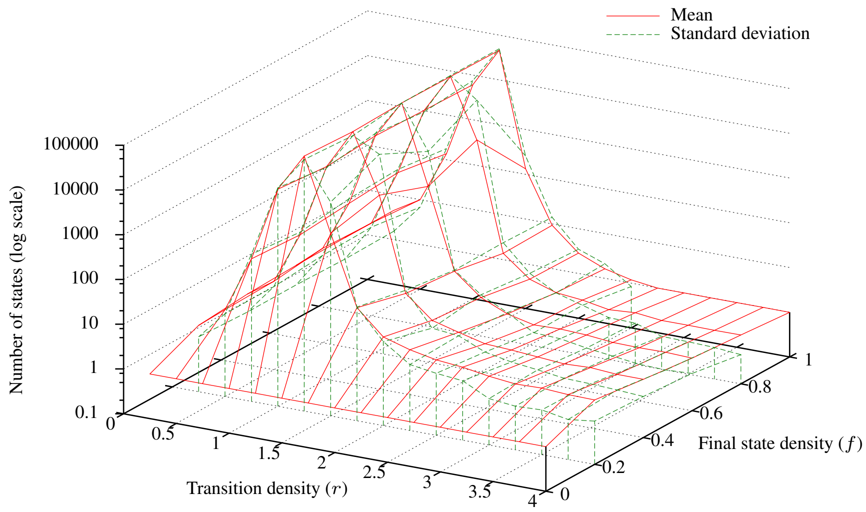


Figure 7: Mean and standard deviation of size of minimal DFA for $N = 100$

process, and that it makes makes the overall algorithm much more efficient. In order to run the algorithms $\textsc{Subset}(\mathsf{compress}_{\subseteq})$ or $\textsc{TransSet}(\mathsf{compress}_{\subseteq})$, prior minimisation modulo $\rightleftarrows$ is required. For $\textsc{Subset}(\mathsf{close}_{\subseteq})$ we always minimise modulo $\rightleftarrows$ as well, as we have to calculate the simulation relation between the states of the NFA anyway.

The most interesting results are those for steps 4 through 6, which are shown in Table 1. On step 6, algorithms $\textsc{Subset}(\mathcal{I})$ and $\textsc{TransSet}(\mathcal{I})$ ran out of memory on our 4 GB computer. We terminated $\textsc{Subset}(\mathsf{close}_{\subseteq})$ prematurely after roughly 44 hours of computation and did not measure its memory consumption. The algorithms that use $\mathsf{compress}_{\subseteq}$ clearly outperform the others, in both memory and time efficiency. Every algorithm that uses a function other than $\mathcal{I}$ generates a DFA that is an order of magnitude smaller than that of its $\mathcal{I}$-counterpart.

8.2 Random Automata

In [15], Tabakov and Vardi experimentally evaluate the performance of several automata algorithms by running them on randomly generated automata. In their model, the randomly generated NFAs have an alphabet $\Sigma = \{0, 1\}$. The parameters that can be set by the user are the number of states N, the *transition density* r, and the *final state density* f. The transition density indicates the ratio of the number of transitions to the number of states N for a given label $a \in \Sigma$. The final state density indicates the ratio of the number of final states $|F|$ to the total number of states N. For example, if we choose $N = 20$, $r = 2.0$, and $f = 0.4$, the resulting NFA will have 20 states, 40 0-labelled transitions, 40 1-labelled transitions, and 8 final states.

The generated automaton need not be connected: for every label $a \in \Sigma$ transitions are added by repeatedly choosing two states s, t at random and adding transition (s, a, t) only if it does not already exist, until the number of a-transitions equals (or exceeds) $r \cdot N$. So, if the transitions are chosen poorly or the transition density is not high enough, not all states are reachable from the initial state.

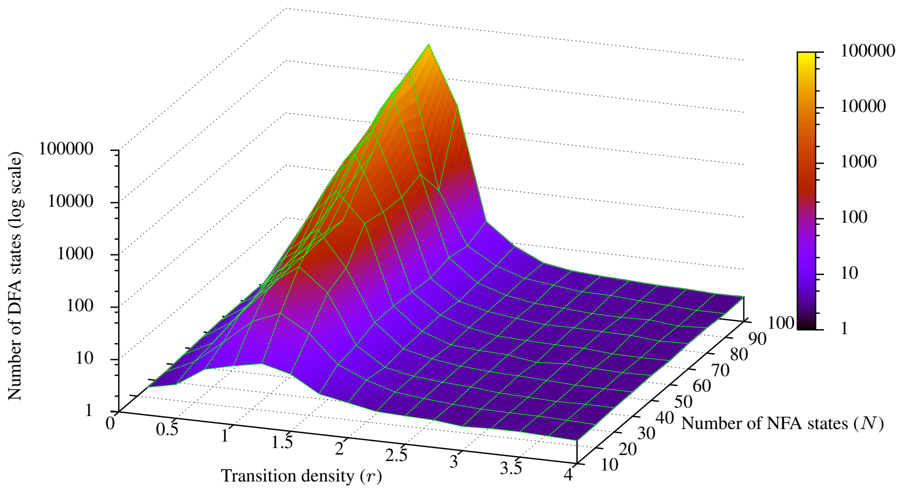


Figure 8: Mean size of minimal DFA for $f = 0.40$

For our experiments, we let N range from 10 to 100 with steps of 10, r ranges from 0.25 to 4.0 with steps of 0.25, and f ranges from 0.0 to 1.0 with steps of 0.2. For every combination of parameter values, we generate 100 random NFAs, giving a grand total of 96 000 automata. All these automata are first of all minimised modulo simulation equivalence; this step is the same for all algorithms and not included in our measurements. Each of the resulting automata is determinised in turn by each of the algorithms and subsequently minimised. The time, peak memory and size of the intermediate DFA of each run of an algorithm are measured. For each combination of parameter values and a determinisation algorithm we compute the mean of the 100 measurements.

For $N = 100$ we have plotted the size of the minimal DFA in Figure 7. Every data point is the mean (solid line) or standard deviation (dashed line) of the 100 minimal DFAs that were generated for that point. We see the same phenomena already noticed by Tabakov and Vardi: apart from the case $f = 0$, the final state density does not have a noteworthy effect on the size of the minimal DFA, but the transition density does. The mean size shows a clear peak at $r = 1.25$. For this value, the standard deviation is about as large as the mean, indicating that there's quite some variance in the minimal DFA sizes. For example, for $r = 1.25$ and $f = 0.40$ the mean is 44 213, the standard deviation is 36 182 and the sizes of the minimal DFAs range from 296 to 179 757. For other values of N the plot has the same shape as that of Figure 7.

Because the final state density does not seem to influence the results that much, we fix $f = 0.40$. Again we plot the mean size of the minimal DFA but this time with the size of the NFA (N) along the y-axis, see Figure 8. We see that the NFA size really matters for values of r that are in the range of 0.75 to 1.50. Outside of this range, the size of the minimal DFA remains almost constant as the size of the NFA increases. At $r = 1.25$ the effect of the NFA size is most dramatic: the mean size of the minimal DFA at $N = 100$ is more than 2 000 times as large as the mean size at $N = 10$.

As mentioned above, the value of f does not seem to influence the results that much, so we fix it at $f = 0.40$. In Appendix D we have plotted for various values of r the peak memory usage, total running time and size of the intermediate DFAs of canonisation using each of the four algorithms against the size of the input NFA N. Again, every data point is the mean of 100 experiments.

Just looking at the sizes of the intermediate DFAs (Section D.1), we see that, for all values of r and N, $\textsc{Subset}(\mathcal{I})$ and $\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ generate significantly larger DFAs than the other algorithms. For $r \leq 1.00$ these DFAs are very close (if not equal) to the minimal DFAs. We also see that for $r > 3$ the DFA generated by $\textsc{Transset}(\mathcal{I})$ is slightly larger than the ones generated by $\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$ and $\textsc{Transset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$. Note that the measurements are in accordance with our lattice.

On the time plots of Section D.2, it is clear that $\textsc{Subset}(\mathcal{I})$ is the fastest algorithm. Note that a mean value of $1 \cdot 10^{-4}$ indicates that the runs were too fast to allow proper measuring. $\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ is as fast as $\textsc{Subset}(\mathcal{I})$ for $0.25 \leq r \leq 1$, but for larger r-values it becomes slower, ending up in 4th place for $r > 2$. Overall, $\textsc{Transset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ can be considered the slowest algorithm.

Regarding memory consumption (Section D.3), the $\textsc{Subset}(f)$ algorithms overall perform better than the $\textsc{Transset}(f)$ algorithms, with $\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$ being the most memory efficient. The curves for $\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ closely follow the ones for $\textsc{Subset}(\mathcal{I})$, but for $r < 2$ $\textsc{Subset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$ performs somewhat better. Given the fact that $\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$ produces the same DFA as $\textsc{Transset}(\mathsf{compress}_{\underrightarrow{\subseteq}})$, the former is preferable as it has better time and space performance. Overall, $\textsc{Subset}(\mathcal{I})$ and $\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$ perform the best. $\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$ appears to use around 20% less memory overall, but is by far not as fast.

9 Conclusions

We have presented a schematic generalisation of the well-known subset construction algorithm that allows for a function to be applied to every generated set of states. We have given a similar scheme for a variant of subset construction that operates on sets of transitions rather than states. Next, we instantiated these schemes with several set-expanding or -reducing functions to obtain various determinisation algorithms. One of these algorithms even produces the minimal DFA directly, but its use of the PSPACE-hard language preorder renders it impractical. As our aim is to reduce the average-case workload in practice, we instead use the PTIME-decidable simulation preorder in the other algorithms. We have classified all presented algorithms in a lattice, based on the sizes of the DFAs they produce. This is a natural criterion, as the worst-case complexities are the same for all algorithms. To assess their performance, we have implemented and benchmarked them. The case study comprised NFAs describing patterns in the elementary cellular automaton with rule number 110 and randomly generated NFAs. On the cellular automaton examples, the algorithms that use a function to reduce the computed sets, convincingly outperformed the others. On the random automata, three of the algorithms generated smaller DFAs than subset construction, which led to less memory consumption in some cases. In particular, our algorithm $\textsc{Subset}(\mathsf{close}_{\underrightarrow{\subseteq}})$ systematically outperforms the standard subset construction in memory consumption. However, the gain is relatively small, and goes at the expense of the speed of the algorithm.

Based on our algorithm schemes, many more algorithms can be constructed by substituting various functions, depending on the specific needs and applications. Moreover, the functions we defined here could be equipped with any suitable preorder or partial order, *e.g.* from the linear time – branching time spectrum.

We believe that our optimisations to subset construction are particularly beneficial in cases where normal subset construction is known to leave a large gap between the generated DFA and the minimal one. Our first case study deals with such a situation and supports this theory.

Acknowledgements. We would like to thank Jan Friso Groote, Tim Willemse and Sebastian Maneth for valuable ideas, discussions and/or comments.

References

[1] B. Bloom & R. Paige (1995): *Transformational design and implementation of a new efficient solution to the ready simulation problem.* Science of Computer Programming 24(3), pp. 189–220.

[2] J.A. Brzozowski (1963): *Canonical regular expressions and minimal state graphs for definite events.* In Proceedings of the Symposium on Mathematical Theory of Automata, MRI Symposia Series, vol. 12, Polytechnic Press, Polytechnic Institute of Brooklyn, pp. 529–561.

[3] CWI: *μCRL Toolset Home Page.* `http://www.cwi.nl/~mcrl/`.

[4] D.L. Dill, A.J. Hu & H. Wong-Toi (1992): *Checking for language inclusion using simulation preorders.* In Proceedings of the Third International Workshop on Computer-Aided Verification (CAV'92), LNCS 575, Springer, pp. 255–265.

[5] R.W. Floyd & R. Beigel (1994): *The Language of Machines*. Freeman.

[6] R. Gentilini, C. Piazza & A. Policriti (2003): *From bisimulation to simulation: Coarsest partition problems.* Journal of Automated Reasoning 31(1), pp. 73–103.

[7] R.J. van Glabbeek & B. Ploeger (2008): *Correcting a space-efficient simulation algorithm.* CS-Report 08-06, Eindhoven University of Technology. To appear in Proceedings 20th International Conference on Computer Aided Verification (CAV'08), LNCS, Springer.

[8] J.F. Groote & M.A. Reniers (2001): *Algebraic process verification.* In J.A. Bergstra, A. Ponse & S.A. Smolka, editors: Handbook of Process Algebra, Elsevier, pp. 1151–1208.

[9] M.R. Henzinger, T.A. Henzinger & P.W. Kopke (1995): *Computing simulations on finite and infinite graphs.* In 36th Annual Symposium on Foundations of Computer Science (FOCS'95), IEEE Computer Society Press, pp. 453–462.

[10] J.E. Hopcroft (1971): *An $n \log n$ algorithm for minimizing states in a finite automaton.* In Z. Kohavi, editor: Theory of Machines and Computations, Academic Press, pp. 189–196.

[11] J.E. Hopcroft & J.D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

[12] D.M.R. Park (1981): *Concurrency and automata on infinite sequences.* In Proc. 5th GI-Conference on Theoretical Computer Science, LNCS 104, Springer, pp. 167–183.

[13] L.J. Stockmeyer & A.R. Meyer (1973): *Word problems requiring exponential time.* In Proc. 5th Annual ACM Symposium on *Theory of Computing* (STOC'73), ACM, pp. 1–9.

[14] K. Sutner (2003): *The size of power automata.* *Theoretical Computer Science* 295(1-3), pp. 371–386.

[15] D. Tabakov & M.Y. Vardi (2005): *Experimental evaluation of classical automata constructions.* In Proceedings of the 12th International Conference on *Logic for Programming, Artificial Intelligence, and Reasoning* (LPAR'05), LNCS 3835, Springer, pp. 396–411.

[16] B.W. Watson (1995): *Taxonomies and Toolkits of Regular Language Algorithms*. PhD thesis, Technische Universiteit Eindhoven.

[17] S. Wolfram (1984): *Computation theory of cellular automata.* *Communications in Mathematical Physics* 96(1), pp. 15–57.

[18] S. Wolfram (2002): *A New Kind of Science*. Wolfram Media, Inc.

A Correctness of SUBSET(f)

We need the following lemma, for which it is important to understand that a set of NFA states P denotes a set of states in $\mathcal{N}$ but must be regarded as only a single state in $\mathcal{D}$.

Lemma 1. *Let $\mathcal{D}$ be obtained by applying* SUBSET(f) *on the NFA $\mathcal{N}$, where the function $f : \mathcal{P}(S_\mathcal{N}) \to \mathcal{P}(S_\mathcal{N}))$ satisfies $\mathcal{L}_\mathcal{N}(f(Q)) = \mathcal{L}_\mathcal{N}(Q)$ for any $Q \subseteq S_\mathcal{N}$. Then for any $P \in S_\mathcal{D}$ it holds that $\mathcal{L}_\mathcal{N}(P) = \mathcal{L}_\mathcal{D}(P)$.*

Proof. We show set inclusion both ways.

- We prove that $\sigma \in \mathcal{L}_\mathcal{N}(P)$ implies $\sigma \in \mathcal{L}_\mathcal{D}(P)$ for any $\sigma \in \Sigma_\mathcal{N}$ and $P \in S_\mathcal{D}$ by induction on the length of σ.

  Base: $\sigma = \varepsilon$. Let $P \in S_\mathcal{D}$, and assume $\sigma \in \mathcal{L}_\mathcal{N}(P)$. Then there exists a $p \in P$ such that $\varepsilon \in \mathcal{L}_\mathcal{N}(p)$, and hence $p \in F_\mathcal{N}$. By line 12 of Algorithm 1 we have $P \in F_\mathcal{D}$ and thus $\varepsilon \in \mathcal{L}_\mathcal{D}(P)$.

  Step: $\sigma = a\rho$ for some $a \in \Sigma_\mathcal{N}$ and $\rho \in \Sigma_\mathcal{N}^*$. Let $P \in S_\mathcal{D}$ and assume $\sigma \in \mathcal{L}_\mathcal{N}(P)$. Then there exists a $p \in P$ such that $\sigma \in \mathcal{L}_\mathcal{N}(p)$. Hence there is a $q \in S_\mathcal{N}$ such that $p \xrightarrow{a}_\mathcal{N} q$ and $\rho \in \mathcal{L}_\mathcal{N}(q)$. Since $p \in P$ and $p \xrightarrow{a}_\mathcal{N} q$, SUBSET$(f)$ (lines 6–8) ensures that $P \xrightarrow{a}_\mathcal{D} f(Q)$ for some $Q \subseteq S_\mathcal{N}$ with $q \in Q$. We have $\rho \in \mathcal{L}_\mathcal{N}(q) \subseteq \mathcal{L}_\mathcal{N}(Q) = \mathcal{L}_\mathcal{N}(f(Q))$, and thus, by induction, $\rho \in \mathcal{L}_\mathcal{D}(f(Q))$. This implies $a\rho \in \mathcal{L}_\mathcal{D}(P)$.

- We prove that $\sigma \in \mathcal{L}_\mathcal{D}(P)$ implies $\sigma \in \mathcal{L}_\mathcal{N}(P)$ for any $\sigma \in \Sigma_\mathcal{N}$ and $P \in S_\mathcal{D}$ by induction on the length of σ.

  Base: $\sigma = \varepsilon$. Assume $\sigma \in \mathcal{L}_\mathcal{D}(P)$. Then $P \in F_\mathcal{D}$ so there exists a $p \in P$ such that $p \in F_\mathcal{N}$. For this p, $\varepsilon \in \mathcal{L}_\mathcal{N}(p)$. Then also $\varepsilon \in \bigcup_{q \in P} \mathcal{L}_\mathcal{N}(q)$ and thus $\varepsilon \in \mathcal{L}_\mathcal{N}(P)$.

  Step: $\sigma = a\rho$ for some $a \in \Sigma_\mathcal{D}$ and $\rho \in \Sigma_\mathcal{D}^*$. Let $P \in S_\mathcal{D}$ and assume $\sigma \in \mathcal{L}_\mathcal{D}(P)$. Then there exists a $P' \in S_\mathcal{D}$ such that $P \xrightarrow{a}_\mathcal{D} P'$ and $\rho \in \mathcal{L}_\mathcal{D}(P')$. By induction $\rho \in \mathcal{L}_\mathcal{N}(P')$. We know that $P' = f(Q)$ where $Q = \{q \in S_\mathcal{N} \mid \exists p \in P \,.\, p \xrightarrow{a}_\mathcal{N} q\}$. As $\mathcal{L}_\mathcal{N}(f(Q)) = \mathcal{L}_\mathcal{N}(Q)$, we have $\rho \in \mathcal{L}_\mathcal{N}(Q)$. Because $\mathcal{L}_\mathcal{N}(Q) = \bigcup_{q \in Q} \mathcal{L}_\mathcal{N}(q)$, there exists a $q \in Q$ such that $\rho \in \mathcal{L}_\mathcal{N}(q)$ and hence there is a $p \in P$ such that $p \xrightarrow{a}_\mathcal{N} q$. Now $a\rho \in \mathcal{L}_\mathcal{N}(p)$ for that p and thus we obtain $\sigma \in \mathcal{L}_\mathcal{N}(P)$. $\square$

By Lemma 1 we are now allowed to use language equivalence and inclusion without specifying whether we mean language equivalence (resp. inclusion) between sets of states in an NFA or single states in the generated DFA.

Theorem 1. *Let $\mathcal{D}$ be obtained by applying* SUBSET(f) *on the NFA $\mathcal{N}$, where the function $f : \mathcal{P}(S_\mathcal{N}) \to \mathcal{P}(S_\mathcal{N}))$ satisfies $\mathcal{L}_\mathcal{N}(f(Q)) = \mathcal{L}_\mathcal{N}(Q)$ for any $Q \subseteq S_\mathcal{N}$. Then $\mathcal{D}$ is deterministic and language equivalent to $\mathcal{N}$.*

Proof. It can be easily seen from the algorithm that for each combination of $P \in S_\mathcal{D}$ and $a \in \Sigma_\mathcal{D}$, precisely one tuple (P, a, P') for some $P' \in S_\mathcal{D}$ is added to $\delta_\mathcal{D}$. Hence $\mathcal{D}$ is deterministic. We derive $\mathcal{D} \equiv_L \mathcal{N}$ using Lemma 1:

$$\mathcal{L}(\mathcal{D}) = \mathcal{L}_\mathcal{D}(i_\mathcal{D}) = \mathcal{L}_\mathcal{D}(f(\{i_\mathcal{N}\})) \stackrel{\text{L1}}{=} \mathcal{L}_\mathcal{N}(f(\{i_\mathcal{N}\})) = \mathcal{L}_\mathcal{N}(\{i_\mathcal{N}\}) = \mathcal{L}_\mathcal{N}(i_\mathcal{N}) = \mathcal{L}(\mathcal{N}) \quad \square$$

A.1 Correctness of SUBSET$(\mathsf{close}_{\sqsubseteq_L})$

For any set of states P in an NFA $\mathcal{N}$, $\mathsf{close}_{\sqsubseteq_L}(P)$ contains all states that are language included in P. In particular, because $p \sqsubseteq_L P$ for all $p \in P$, we have that $P \subseteq \mathsf{close}_{\sqsubseteq_L}(P)$.

Proposition 1. *For any set of states $P \subseteq S_\mathcal{N}$, it holds that $P \equiv_L \mathsf{close}_{\sqsubseteq_L}(P)$.*

Proof. Let $Q := \mathsf{close}_{\sqsubseteq_L}(P)$. We show that $P \sqsubseteq_L Q$ and $Q \sqsubseteq_L P$:

- $P \sqsubseteq_L Q$: using the fact that $P \subseteq Q$, we derive: $\mathcal{L}_\mathcal{N}(P) = \bigcup_{p \in P} \mathcal{L}_\mathcal{N}(p) \subseteq \bigcup_{p \in Q} \mathcal{L}_\mathcal{N}(p) = \mathcal{L}_\mathcal{N}(Q)$, hence $P \sqsubseteq_L Q$;
- $Q \sqsubseteq_L P$: for all $p \in Q$ we know (by definition) that $p \sqsubseteq_L P$, *i.e.* $\mathcal{L}_\mathcal{N}(p) \subseteq \mathcal{L}_\mathcal{N}(P)$. Thus: $\mathcal{L}_\mathcal{N}(Q) = \bigcup_{p \in Q} \mathcal{L}_\mathcal{N}(p) \subseteq \mathcal{L}_\mathcal{N}(P)$, hence $Q \sqsubseteq_L P$. □

Theorem 2. *Given an NFA $\mathcal{N}$,* SUBSET$(\mathsf{close}_{\sqsubseteq_L})$ *constructs the minimal DFA that is language equivalent to $\mathcal{N}$.*

Proof. It follows immediately from Theorem 1 and Proposition 1 that the NFA $\mathcal{D}$ constructed by SUBSET$(\mathsf{close}_{\sqsubseteq_L})$ is a DFA that is language equivalent to $\mathcal{N}$.

$\mathcal{D}$ is minimal if there is no DFA that is language equivalent to $\mathcal{D}$ and has fewer states than $\mathcal{D}$. It is known that this follows directly if we prove that there is no pair of different states in $\mathcal{D}$ that are language equivalent (see for instance Corollary 4.24 in [5]). Suppose $\mathcal{D}$ contains states P and Q such that $P \equiv_L Q$ and for some $T, U \subseteq S_\mathcal{N}$, $P = \mathsf{close}_{\sqsubseteq_L}(T)$ and $Q = \mathsf{close}_{\sqsubseteq_L}(U)$. Then for all $p \in P$ we have $p \sqsubseteq_L P \equiv_L Q = \mathsf{close}_{\sqsubseteq_L}(U) \equiv_L U$, by Proposition 1. Because $Q = \{q \in S_\mathcal{N} \mid q \sqsubseteq_L U\}$ we see that $p \in Q$ and thus that $P \subseteq Q$. By symmetry we have that $Q \subseteq P$. Hence $P = Q$. □

A.2 Correctness of SUBSET$(\mathsf{close}_{\subsetneq\!\!\rightarrow})$

Proposition 2. *For any set of states $P \subseteq S_\mathcal{N}$, it holds that $P \equiv_L \mathsf{close}_{\subsetneq\!\!\rightarrow}(P)$.*

Proof. Let $Q := \mathsf{close}_{\subsetneq\!\!\rightarrow}(P) = \{q \in S_\mathcal{N} \mid q \subsetneq\!\!\rightarrow P\}$. We show that $P \equiv_L Q$:

- $P \sqsubseteq_L Q$: this follows immediately from $P \subseteq Q$;
- $Q \sqsubseteq_L P$: By induction on the length of $\sigma \in \Sigma_\mathcal{N}$ we show that $\sigma \in \mathcal{L}_\mathcal{N}(Q)$ implies $\sigma \in \mathcal{L}_\mathcal{N}(P)$. If $\varepsilon \in \mathcal{L}_\mathcal{N}(Q)$ then $\varepsilon \in \mathcal{L}_\mathcal{N}(q)$ for some $q \in Q$. Hence $q \in F_\mathcal{N}$. It must be that $q \subsetneq\!\!\rightarrow P$. Thus $\exists p \in P.\ p \in F_\mathcal{N}$ and therefore $\varepsilon \in \mathcal{L}_\mathcal{N}(p) \subseteq \mathcal{L}_\mathcal{N}(P)$.

  If $a\rho \in \mathcal{L}_\mathcal{N}(Q)$ then there is a $q \in Q$ and $q' \in S_\mathcal{N}$ such that $q \xrightarrow{a}_\mathcal{N} q'$ and $\rho \in \mathcal{L}_\mathcal{N}(q')$. It must be that $q \subsetneq\!\!\rightarrow P$, so there exists a simulation $R \subseteq S_\mathcal{N} \times S_\mathcal{N}$ such that $\exists p, p' \in S_\mathcal{N}.\ p \in P \wedge p \xrightarrow{a}_\mathcal{N} p' \wedge q' R p'$. Therefore, $q' \subsetneq\!\!\rightarrow p'$ and hence $q' \sqsubseteq_L p'$, so $\rho \in \mathcal{L}_\mathcal{N}(p')$. It follows that $a\rho \in \mathcal{L}_\mathcal{N}(P)$. □

Theorem 3. *Given an NFA $\mathcal{N}$,* SUBSET$(\mathsf{close}_{\subsetneq\!\!\rightarrow})$ *constructs a DFA that is language equivalent to $\mathcal{N}$.*

Proof. Immediate from Theorem 1 and Proposition 2. □

A.3 Correctness of SUBSET$(\mathsf{compress}_{\subsetneq\!\!\rightarrow})$

Note: In this section we fix an NFA $\mathcal{N}$ that is minimal under simulation equivalence.

Proposition 3. *For any set of states $P \subseteq S_\mathcal{N}$, it holds that $P \equiv_L \mathsf{compress}_{\subsetneq\!\!\rightarrow}(P)$.*

Proof. Let $Q := \mathsf{compress}_{\subsetneq\!\!\rightarrow}(P)$. We show that $P \sqsubseteq_L Q$ and $Q \sqsubseteq_L P$:

- $P \sqsubseteq_L Q$: for any $p \in P$ there is a $q \in Q$ with $p \subsetneq\!\!\rightarrow q$ and hence $p \sqsubseteq_L q$. Thus $\mathcal{L}_\mathcal{N}(P) = \bigcup_{p \in P} \mathcal{L}_\mathcal{N}(p) \subseteq \bigcup_{q \in Q} \mathcal{L}_\mathcal{N}(q) = \mathcal{L}_\mathcal{N}(Q)$, so $P \sqsubseteq_L Q$;
- $Q \sqsubseteq_L P$: this follows immediately from $Q \subseteq P$. □

Theorem 4. *When applied to $\mathcal{N}$,* SUBSET($\mathsf{compress}_{\subseteq}$) *constructs a DFA that is language equivalent to $\mathcal{N}$.*

Proof. Immediate from Theorem 1 and Proposition 3. □

B Correctness of TRANSSET(f)

Proposition 4. *Given NFA $\mathcal{N}$, for any $p \in S_\mathcal{N}$: $p \equiv_L \mathsf{tuple}(p)$.*

Proof. Let $p \in S_\mathcal{N}$ and $S = \{\varepsilon\}$ if $p \in F_\mathcal{N}$ and $S = \emptyset$ otherwise. We derive:

$$\begin{aligned}
\mathcal{L}_\mathcal{N}(p) &= \{\sigma \in \Sigma_\mathcal{N}^* \mid \exists q \in F_\mathcal{N} \,.\, p \xrightarrow{\sigma}_\mathcal{N} q\} \\
&= \{a\sigma \in \Sigma_\mathcal{N}^* \mid \exists q \in F_\mathcal{N} \,.\, p \xrightarrow{a\sigma}_\mathcal{N} q\} \cup S \\
&= \{a\sigma \in \Sigma_\mathcal{N}^* \mid \exists q \in S_\mathcal{N} \,.\, p \xrightarrow{a}_\mathcal{N} q \wedge \sigma \in \mathcal{L}_\mathcal{N}(q)\} \cup S \\
&= \{a\sigma \in \Sigma_\mathcal{N}^* \mid \exists (a,q) \in \mathsf{trans}(p) \,.\, \sigma \in \mathcal{L}_\mathcal{N}(q)\} \cup S \\
&= \textstyle\bigcup_{(a,q)\in\mathsf{trans}(p)} \{a\sigma \in \Sigma_\mathcal{N}^* \mid \sigma \in \mathcal{L}_\mathcal{N}(q)\} \cup S \\
&= \textstyle\bigcup_{t\in\mathsf{trans}(p)} \mathcal{L}_\mathcal{N}(t) \cup S \\
&= \mathcal{L}_\mathcal{N}(\mathsf{trans}(p)) \cup S \\
&= \mathcal{L}_\mathcal{N}(\mathsf{tuple}(p)).
\end{aligned}$$

□

We need the following lemma, in which a transition tuple (T, b) is regarded as such in the input NFA $\mathcal{N}$, but is regarded as a single state in the output NFA $\mathcal{D}$.

Lemma 2. *Let $\mathcal{N}$ be an NFA and $f : \mathcal{P}(\Sigma_\mathcal{N} \times S_\mathcal{N}) \times \mathbb{B} \to \mathcal{P}(\Sigma_\mathcal{N} \times S_\mathcal{N}) \times \mathbb{B}$ such that $\mathcal{L}_\mathcal{N}(f(T, b)) = \mathcal{L}_\mathcal{N}(T, b)$ for all transition tuples (T, b). Let $\mathcal{D}$ be obtained by applying* TRANSSET(f) *to $\mathcal{N}$. For any $(T, b) \in S_\mathcal{D}$, it holds that $\mathcal{L}_\mathcal{N}(T, b) = \mathcal{L}_\mathcal{D}(T, b)$.*

Proof. We show that $\sigma \in \mathcal{L}_\mathcal{N}(T, b) \Leftrightarrow \sigma \in \mathcal{L}_\mathcal{D}(T, b)$ for any $\sigma \in \Sigma_\mathcal{N}^*$ and $(T, b) \in S_\mathcal{D}$ by induction on the length of σ.

Base: $\sigma = \varepsilon$. Let $(T, b) \in S_\mathcal{D}$. Then $\varepsilon \in \mathcal{L}_\mathcal{N}(T, b) \Leftrightarrow b \overset{*}{\Leftrightarrow} (T, b) \in F_\mathcal{D} \Leftrightarrow \varepsilon \in \mathcal{L}_\mathcal{D}(T, b)$, where $*$ follows from lines 11–13 of TRANSSET(f).

Step: $\sigma = a\rho$ for some $a \in \Sigma_\mathcal{N}$ and $\rho \in \Sigma_\mathcal{N}^*$. We assume for any transition tuple $(T', b') \in S_\mathcal{D}$:

$$\text{(IH)} \qquad \rho \in \mathcal{L}_\mathcal{N}(T', b') \Leftrightarrow \rho \in \mathcal{L}_\mathcal{D}(T', b')$$

and derive, for any $(T, b) \in S_\mathcal{D}$:

$$\begin{aligned}
a\rho \in \mathcal{L}_\mathcal{N}(T, b) &\Leftrightarrow a\rho \in \mathcal{L}_\mathcal{N}(T) \;\Leftrightarrow\; a\rho \in \textstyle\bigcup_{t\in T} \mathcal{L}_\mathcal{N}(t) \\
&\Leftrightarrow a\rho \in \textstyle\bigcup_{(a,p)\in T} \mathcal{L}_\mathcal{N}(a, p) \;\Leftrightarrow\; \rho \in \textstyle\bigcup_{(a,p)\in T} \mathcal{L}_\mathcal{N}(p) \\
&\overset{*}{\Leftrightarrow} \rho \in \textstyle\bigcup_{(a,p)\in T} \mathcal{L}_\mathcal{N}(\mathsf{tuple}(p)) \\
&\Leftrightarrow \rho \in \textstyle\bigcup_{(a,p)\in T} \left(\mathcal{L}_\mathcal{N}(\mathsf{trans}(p)) \cup \begin{cases} \{\varepsilon\} & \text{if } p \in F_\mathcal{N} \\ \emptyset & \text{if } p \notin F_\mathcal{N} \end{cases}\right) \\
&\Leftrightarrow \rho \in \textstyle\bigcup_{(a,p)\in T} \mathcal{L}_\mathcal{N}(\mathsf{trans}(p)) \cup \begin{cases} \{\varepsilon\} & \text{if } \exists (a,p) \in T \,.\, p \in F_\mathcal{N} \\ \emptyset & \text{if } \neg\exists (a,p) \in T \,.\, p \in F_\mathcal{N} \end{cases} \\
&\Leftrightarrow \rho \in \mathcal{L}_\mathcal{N}(\textstyle\bigcup_{(a,p)\in T} \mathsf{trans}(p), \exists (a,p) \in T \,.\, p \in F_\mathcal{N}) \\
&\Leftrightarrow \rho \in \mathcal{L}_\mathcal{N}(f(\textstyle\bigcup_{(a,p)\in T} \mathsf{trans}(p), \exists (a,p) \in T \,.\, p \in F_\mathcal{N})) \\
&\overset{\dagger}{\Leftrightarrow} \rho \in \mathcal{L}_\mathcal{D}(f(\textstyle\bigcup_{(a,p)\in T} \mathsf{trans}(p), \exists (a,p) \in T \,.\, p \in F_\mathcal{N})) \\
&\overset{\ddagger}{\Leftrightarrow} a\rho \in \mathcal{L}_\mathcal{D}(T, b)
\end{aligned}$$

where at $*$ we used Proposition 4; at † we used (IH) and the fact that lines 6–7 of $\textsc{TransSet}(f)$ ensure that $f(\ldots) \in S_{\mathcal{D}}$; and at ‡ we used that line 8 of $\textsc{TransSet}(f)$ ensures that $(T,b) \xrightarrow{a}_{\mathcal{D}} f(\ldots)$. □

Theorem 5. *Let $\mathcal{N}$ be an NFA and $f : \mathcal{P}(\Sigma_{\mathcal{N}} \times S_{\mathcal{N}}) \times \mathbb{B} \to \mathcal{P}(\Sigma_{\mathcal{N}} \times S_{\mathcal{N}}) \times \mathbb{B}$ such that $\mathcal{L}_{\mathcal{N}}(f(T,b)) = \mathcal{L}_{\mathcal{N}}(T,b)$ for all transition tuples (T,b). Let $\mathcal{D}$ be obtained by applying $\textsc{TransSet}(f)$ to $\mathcal{N}$. Then $\mathcal{D}$ is deterministic and language equivalent to $\mathcal{N}$.*

Proof. It can be easily seen from the algorithm that for each combination of $P \in S_{\mathcal{D}}$ and $a \in \Sigma_{\mathcal{D}}$, precisely one tuple (P,a,P') for some $P' \in S_{\mathcal{D}}$ is added to $\delta_{\mathcal{D}}$. Hence $\mathcal{D}$ is deterministic. We derive $\mathcal{D} \equiv_L \mathcal{N}$ as follows:

$$\begin{aligned}\mathcal{L}(\mathcal{D}) &= \mathcal{L}_{\mathcal{D}}(i_{\mathcal{D}}) = \mathcal{L}_{\mathcal{D}}(f(\mathsf{tuple}(i_{\mathcal{N}}))) \stackrel{*}{=} \mathcal{L}_{\mathcal{N}}(f(\mathsf{tuple}(i_{\mathcal{N}})))\\ &= \mathcal{L}_{\mathcal{N}}(\mathsf{tuple}(i_{\mathcal{N}})) \stackrel{\dagger}{=} \mathcal{L}_{\mathcal{N}}(i_{\mathcal{N}}) = \mathcal{L}(\mathcal{N})\end{aligned}$$

where at $*$ we used Lemma 2 and at † we used Proposition 4. □

B.1 Correctness of $\textsc{TransSet}(\mathcal{I})$

As the identity function $\mathcal{I}$ trivially satisfies $\mathcal{L}_{\mathcal{N}}(T,b) = \mathcal{L}_{\mathcal{N}}(\mathcal{I}(T,b))$, the following result follows immediately from Theorem 5.

Corollary 1. *Let $\mathcal{D}$ be obtained by applying $\textsc{TransSet}(\mathcal{I})$ to an NFA $\mathcal{N}$. Then $\mathcal{D}$ is deterministic and language equivalent to $\mathcal{N}$.* □

B.2 Correctness of $\textsc{TransSet}(\mathsf{compress}_{\subsetneq\!\!\!\to})$

Note: In this section we fix an NFA $\mathcal{N}$ that is minimal under simulation equivalence.

Proposition 5. *For any transition tuple (T,b): $\mathcal{L}_{\mathcal{N}}(T,b) = \mathcal{L}_{\mathcal{N}}(\mathsf{compress}_{\subsetneq\!\!\!\to}(T,b))$.*

Proof. Let $U := \mathsf{compress}_{\subsetneq\!\!\!\to}(T)$. We have to show that:

$$\mathcal{L}_{\mathcal{N}}(T) \cup \begin{cases} \{\varepsilon\} & \text{if } b \\ \emptyset & \text{if } \neg b \end{cases} \quad = \quad \mathcal{L}_{\mathcal{N}}(U) \cup \begin{cases} \{\varepsilon\} & \text{if } b \\ \emptyset & \text{if } \neg b \end{cases}$$

which follows naturally if $\mathcal{L}_{\mathcal{N}}(T) = \mathcal{L}_{\mathcal{N}}(U)$. For any $t \in T$ there is a $u \in U$ with $t \subsetneq\!\!\!\to u$ and hence $t \sqsubseteq_L u$. Thus: $\mathcal{L}_{\mathcal{N}}(T) = \bigcup_{t\in T} \mathcal{L}_{\mathcal{N}}(t) \subseteq \bigcup_{t\in U} \mathcal{L}_{\mathcal{N}}(t) = \mathcal{L}_{\mathcal{N}}(U)$. Because $U \subseteq T$, we also have that $\mathcal{L}_{\mathcal{N}}(U) \subseteq \mathcal{L}_{\mathcal{N}}(T)$. Hence $\mathcal{L}_{\mathcal{N}}(T) = \mathcal{L}_{\mathcal{N}}(U)$. □

Theorem 6. *Let $\mathcal{D}$ be obtained by applying $\textsc{TransSet}(\mathsf{compress}_{\subsetneq\!\!\!\to})$ to $\mathcal{N}$. Then $\mathcal{D}$ is deterministic and language equivalent to $\mathcal{N}$.*

Proof. This follows immediately from Proposition 5 and Theorem 5. □

C Correctness of the Lattice of Algorithms

Given an NFA $\mathcal{N}$, let $\mathcal{D}(\textsc{Subset}(f))$ denote the DFA generated from $\mathcal{N}$ by algorithm $\textsc{Subset}(f)$, and likewise for the algorithms $\textsc{TransSet}(f)$. For $P \subseteq S_{\mathcal{N}}$ let

$$P/a := \{q \in S_{\mathcal{N}} \mid \exists p \in P.\ p \xrightarrow{a}_{\mathcal{N}} q\},$$

so that line 6 of Algorithm 1 can be rewritten as $P' := f(P/a)$. It follows that $S_{\mathcal{D}(\textsc{Subset}(f))}$ is the smallest set $S \subseteq \mathcal{P}(S_\mathcal{N})$ that contains $f(\{i_\mathcal{N}\})$ and satisfies

(1) $$P \in S \Rightarrow f(P/a) \in S\,.$$

Likewise, for P a transition tuple, let

$$P /\!\!/ a := \Big(\bigcup_{(a,p)\in\mathsf{set}(P)} \mathsf{trans}(p),\ \exists (a,p) \in \mathsf{set}(P)\,.\,p \in F_\mathcal{N}\Big)\,,$$

so that line 6 of Algorithm 2 can be rewritten as $P' := f(P/\!\!/a)$. Moreover, for $P \subseteq S_\mathcal{N}$ we define

$$\mathsf{tuple}(P) := \Big(\bigcup_{p\in P} \mathsf{trans}(p), \exists p \in P.\, p \in F_\mathcal{N}\Big)\,.$$

It follows that $S_{\mathcal{D}(\textsc{TransSet}(f))}$ is the smallest $S \subseteq \mathcal{P}(S_\mathcal{N})$ that contains $f(\mathsf{tuple}(i_\mathcal{N}))$, which equals $f(\mathsf{tuple}(\{i_\mathcal{N}\}))$, and satisfies $P \in S \Rightarrow f(P/\!\!/a) \in S$.

The following definition extends the operator $\mathsf{close}_{\underset{\to}{\subseteq}}$, originally defined on sets of states, to transition tuples. The result of applying the operator remains a set of states.

Definition 4. For any state $p \in S_\mathcal{N}$ and transition tuple (T, b) we write $p \underset{\to}{\subseteq} (T, b)$ iff

- $p \in F_\mathcal{N} \Rightarrow b$ and
- $\forall a \in \Sigma_\mathcal{N}\,.\,\forall p' \in S_\mathcal{N}\,.\; p \xrightarrow{a}_\mathcal{N} p' \;\Rightarrow\; \exists (a, q') \in T\,.\; p' \underset{\to}{\subseteq} q'$.

For any transition tuple P of an NFA $\mathcal{N}$ we define $\mathsf{close}_{\underset{\to}{\subseteq}}(P) := \{p \in S_\mathcal{N} \mid p \underset{\to}{\subseteq} P\}$.

Lemma 3. *Let $\mathcal{N}$ be an NFA, $a \in \Sigma_\mathcal{N}$ and $P \subseteq S_\mathcal{N}$. Then*

1. $\mathsf{tuple}(P)/\!\!/a = \mathsf{tuple}(P/a)$,
2. $\mathsf{compress}_{\underset{\to}{\subseteq}}(P/a) \subseteq \mathsf{compress}_{\underset{\to}{\subseteq}}(P)/a$,
3. $\mathsf{compress}_{\underset{\to}{\subseteq}}(\mathsf{compress}_{\underset{\to}{\subseteq}}(P)/a) = \mathsf{compress}_{\underset{\to}{\subseteq}}(P/a)$,
4. $\mathsf{close}_{\underset{\to}{\subseteq}}(P/a) \supseteq \mathsf{close}_{\underset{\to}{\subseteq}}(P)/a$,
5. $\mathsf{close}_{\underset{\to}{\subseteq}}(\mathsf{close}_{\underset{\to}{\subseteq}}(P)/a) = \mathsf{close}_{\underset{\to}{\subseteq}}(P/a)$,
6. $\mathsf{close}_{\underset{\to}{\subseteq}}(P) = \mathsf{close}_{\underset{\to}{\subseteq}}(\mathsf{compress}_{\underset{\to}{\subseteq}}(P))$,
7. $\mathsf{close}_{\underset{\to}{\subseteq}}(P) = \mathsf{close}_{\underset{\to}{\subseteq}}(\mathsf{tuple}(P))$,
8. $\mathsf{compress}_{\underset{\to}{\subseteq}}(\mathsf{tuple}(P)) = \mathsf{compress}_{\underset{\to}{\subseteq}}(\mathsf{tuple}(\mathsf{close}_{\underset{\to}{\subseteq}}(P)))$.

Proof. Ad 1. Note that $(a, p) \in \mathsf{set}(\mathsf{tuple}(P)) \Leftrightarrow p \in (P/a)$. Hence

$$\begin{aligned}\mathsf{tuple}(P)/\!\!/a &= \big(\textstyle\bigcup_{(a,p)\in\mathsf{set}(\mathsf{tuple}(P))} \mathsf{trans}(p),\ \exists (a,p) \in \mathsf{set}(\mathsf{tuple}(P))\,.\,p \in F_\mathcal{N}\big)\\ &= \big(\textstyle\bigcup_{p\in(P/a)} \mathsf{trans}(p),\ \exists p \in (P/a)\,.\,p \in F_\mathcal{N}\big)\\ &= \mathsf{tuple}(P/a).\end{aligned}$$

Ad 2. Let $p' \in \mathsf{compress}_{\underset{\to}{\subseteq}}(P/a)$. Then $p' \in P/a$, so $\exists p \in P.\ p \xrightarrow{a}_\mathcal{N} p'$. In fact, the set of all $p \in P$ with $p \xrightarrow{a}_\mathcal{N} p'$ is ordered by $\underset{\to}{\subseteq}$, and we take a $\underset{\to}{\subseteq}$-maximal p within that set. Suppose, towards a contradiction, that $\exists q \in P.\ q \neq p \wedge p \underset{\to}{\subseteq} q$. Take that q. Then, by definition of simulation, $\exists q' \in S_\mathcal{N}.\ q \xrightarrow{a}_\mathcal{N} q' \wedge p' \underset{\to}{\subseteq} q'$, so $q' \in P/a$. By construction, $q' \neq p'$. Hence $p' \notin \mathsf{compress}_{\underset{\to}{\subseteq}}(P/a)$. So $\nexists q \in P.\ q \neq p \wedge p \underset{\to}{\subseteq} q$, and thus $p \in \mathsf{compress}_{\underset{\to}{\subseteq}}(P)$. Hence $p' \in \mathsf{compress}_{\underset{\to}{\subseteq}}(P)/a$.

Ad 3. By Lemma 3.2 we have

$$\mathsf{compress}_{\underset{\to}{\subseteq}}(P/a) = \mathsf{compress}_{\underset{\to}{\subseteq}}(\mathsf{compress}_{\underset{\to}{\subseteq}}(P/a)) \subseteq \mathsf{compress}_{\underset{\to}{\subseteq}}(\mathsf{compress}_{\underset{\to}{\subseteq}}(P)/a).$$

On the other hand, $\mathsf{compress}_{\underset{\to}{\subseteq}}(P) \subseteq P$ (by Definition 3) so $\mathsf{compress}_{\underset{\to}{\subseteq}}(P)/a \subseteq P/a$ and thus $\mathsf{compress}_{\underset{\to}{\subseteq}}(\mathsf{compress}_{\underset{\to}{\subseteq}}(P)/a) \subseteq \mathsf{compress}_{\underset{\to}{\subseteq}}(P/a)$.

Ad 4. Let $p' \in \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)/a$. Then $\exists p \in \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P).\ p \xrightarrow{a}_{\mathcal{N}} p'$. Take that p. So $p \underset{\rightarrow}{\subsetneq} P$. Hence, $\exists q, q' {\in} S_{\mathcal{N}}\ .\ q {\in} P \wedge q \xrightarrow{a}_{\mathcal{N}} q' \wedge p' \underset{\rightarrow}{\subsetneq} q'$. Therefore, $q' {\in} P/a$ and $p' {\in} \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P/a)$. (In the latter step we use that if $p \underset{\rightarrow}{\subsetneq} q$ and $q \in Q$ then surely $p \underset{\rightarrow}{\subsetneq} Q$.)

Ad 5. By Lemma 3.4 we have

$$\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P/a) = \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P/a)) \supseteq \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)/a).$$

Moreover, $\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P) \supseteq P$ so $\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)/a \supseteq P/a$ and thus $\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)/a) \supseteq \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P/a)$.

Ad 6. As $\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(P) \subseteq P$ we have $\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(P)) \subseteq \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)$.
For the other direction, note that $p \in P$ implies $\exists q \in \mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(P)\ .\ p \underset{\rightarrow}{\subsetneq} q$. From this it easily follows that $p \underset{\rightarrow}{\subsetneq} P$ implies $p \underset{\rightarrow}{\subsetneq} \mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(P)$, which in turn yields $\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P) \subseteq \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(P))$.

Ad 7. This can be restated as $p \underset{\rightarrow}{\subsetneq} P \Leftrightarrow p \underset{\rightarrow}{\subsetneq} \mathsf{tuple}(P)$, which follows directly from the definitions.

Ad 8. $\mathsf{fin}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{tuple}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)))) = \mathsf{fin}(\mathsf{tuple}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P))) = \exists p \underset{\rightarrow}{\subsetneq} P.\ p {\in} F_{\mathcal{N}}$ $= \exists q \in P.\ q \in F_{\mathcal{N}} = \mathsf{fin}(\mathsf{tuple}(P)) = \mathsf{fin}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{tuple}(P)))$.

Furthermore, $\mathsf{set}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{tuple}(P))) \subseteq \mathsf{set}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{tuple}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P))))$, because $P \subseteq \mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)$. Finally, take $(a,p') {\in} \mathsf{set}(\mathsf{tuple}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P))) \setminus \mathsf{set}(\mathsf{tuple}(P))$. Then $p \xrightarrow{a}_{\mathcal{N}} p'$ for some $p \underset{\rightarrow}{\subsetneq} P$. Hence $\exists q, q' \in S_{\mathcal{N}}.\ q \in P \wedge q \xrightarrow{a}_{\mathcal{N}} q' \wedge p' \underset{\rightarrow}{\subsetneq} q'$. Therefore $(a,q') \in \mathsf{set}(\mathsf{tuple}(P))$, so $(a,p') \neq (a,q')$, and $(a,p') \underset{\rightarrow}{\subsetneq} (a,q')$, from which we obtain $(a,p') \notin \mathsf{set}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{tuple}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P))))$ by Definition 3. This implies that $\mathsf{set}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{tuple}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P)))) \subseteq \mathsf{set}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(\mathsf{tuple}(P)))$. $\square$

Theorem 7. $\textsc{Transset}(\mathcal{I}) \preceq \textsc{Subset}(\mathcal{I})$.

Proof. A straightforward induction on (1), using Lemma 3.1 at the induction step, yields

$$S_{\mathcal{D}(\textsc{Transset}(\mathcal{I}))} = \{\mathsf{tuple}(P) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\}. \tag{2}$$

This immediately implies that $|S_{\mathcal{D}(\textsc{Transset}(\mathcal{I}))}| \leq |S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}|$. (Note that we do need not have an equality here, because if might be that $\mathsf{tuple}(P) = \mathsf{tuple}(Q)$ for different $P, Q \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}$.) Hence $\textsc{Transset}(\mathcal{I}) \preceq \textsc{Subset}(\mathcal{I})$. $\square$

Theorem 8. $\textsc{Subset}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}) \preceq \textsc{Subset}(\mathcal{I})$.

Proof. A straightforward induction on (1), using Lemma 3.3 at the induction step, yields

$$S_{\mathcal{D}(\textsc{Subset}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}))} = \{\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\}. \tag{3}$$

Hence $|S_{\mathcal{D}(\textsc{Subset}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}}))}| \leq |S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}|$, from which the theorem follows. $\square$

Theorem 9. $\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}) \preceq \textsc{Subset}(\mathsf{compress}_{\underset{\rightarrow}{\subsetneq}})$.

Proof. By a similar induction as in the previous proof, this time using Lemma 3.5, we obtain

$$S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}))} = \{\mathsf{close}_{\underset{\rightarrow}{\subsetneq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\}. \tag{4}$$

In combination with (3) and Lemma 3.6 this yields

$$\begin{aligned} S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}}))} &\overset{(4)}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\} \\ &\overset{3.6}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}((P)) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\} \\ &\overset{(3)}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))}\}\,. \end{aligned}$$

Hence $|S_{\mathcal{D}(\textsc{Subset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))}| \leq |S_{\mathcal{D}(\textsc{Subset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))}|$. □

In the following lemma, whose proof is similar to the previous one, we employ a relation $\subseteq$ between transition tuples defined by $(T, b) \subseteq (T', b')$ iff $T \subseteq T'$ and $b \Rightarrow b'$.

Lemma 4. *Let $\mathcal{N}$ be an NFA, $a \in \Sigma_{\mathcal{N}}$ and P a transition tuple. Then*

1. $\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P /\!\!/ a) \subseteq \mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P) /\!\!/ a)$,
2. $\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P) /\!\!/ a) = \mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P /\!\!/ a)$,
3. $\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(P) = \mathsf{close}_{\underset{\rightarrow}{\subseteq}}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P))$. □

Theorem 10. $\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}) \preceq \textsc{Transset}(\mathcal{I})$.

Proof. A straightforward induction using Lemma 4.2, yields

$$S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))} = \{\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Transset}(\mathcal{I}))}\}. \tag{5}$$

Hence $|S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))}| \leq |S_{\mathcal{D}(\textsc{Transset}(\mathcal{I}))}|$. □

Theorem 11. $\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}) \preceq \textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}})$.

Proof. From (2), (4), (5) and Lemma 3.8 we obtain

$$\begin{aligned} S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))} &\overset{(5)}{=} \{\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Transset}(\mathcal{I}))}\} \\ &\overset{(2)}{=} \{\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(\mathsf{tuple}(P)) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\} \\ &\overset{3.8}{=} \{\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(\mathsf{tuple}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(P))) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\} \\ &\overset{(4)}{=} \{\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(\mathsf{tuple}(P)) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}}))}\}\,. \end{aligned}$$

Hence $|S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))}| \leq |S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}}))}|$. □

Theorem 12. $\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}}) \preceq \textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}})$.

Proof. From (2), (4), (5) and Lemmas 3.7 and 4.3 we obtain

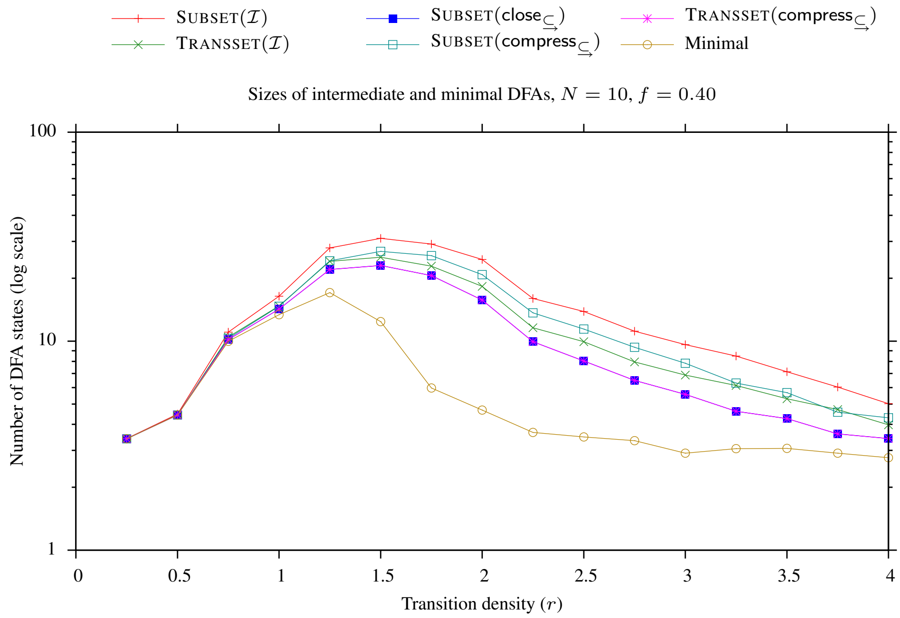
$$\begin{aligned} S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}}))} &\overset{(4)}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\} \\ &\overset{3.7}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(\mathsf{tuple}(P)) \mid P \in S_{\mathcal{D}(\textsc{Subset}(\mathcal{I}))}\} \\ &\overset{(2)}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Transset}(\mathcal{I}))}\} \\ &\overset{4.3}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}(P)) \mid P \in S_{\mathcal{D}(\textsc{Transset}(\mathcal{I}))}\} \\ &\overset{(5)}{=} \{\mathsf{close}_{\underset{\rightarrow}{\subseteq}}(P) \mid P \in S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))}\}\,. \end{aligned}$$

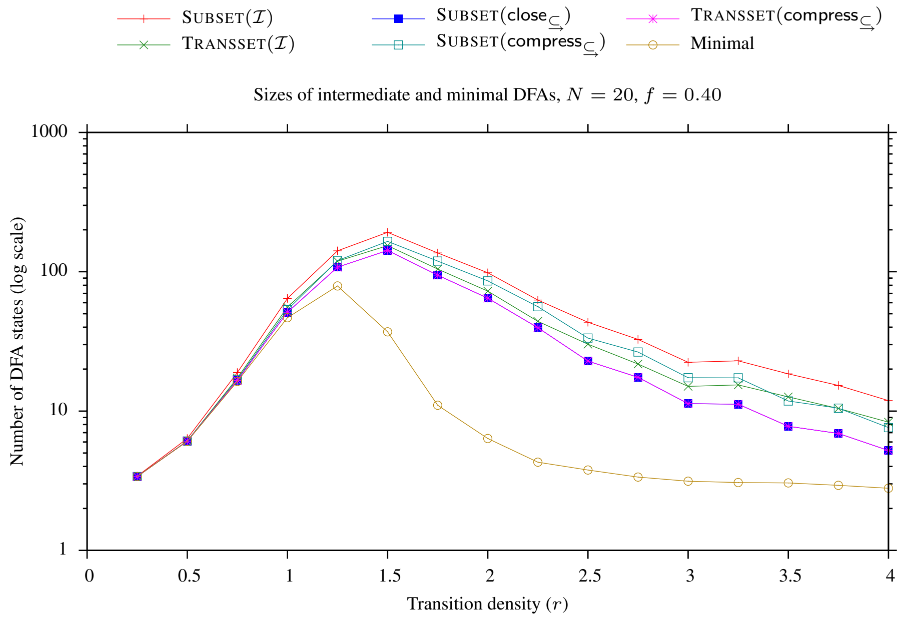
Hence $|S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\underset{\rightarrow}{\subseteq}}))}| \leq |S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\underset{\rightarrow}{\subseteq}}))}|$. □

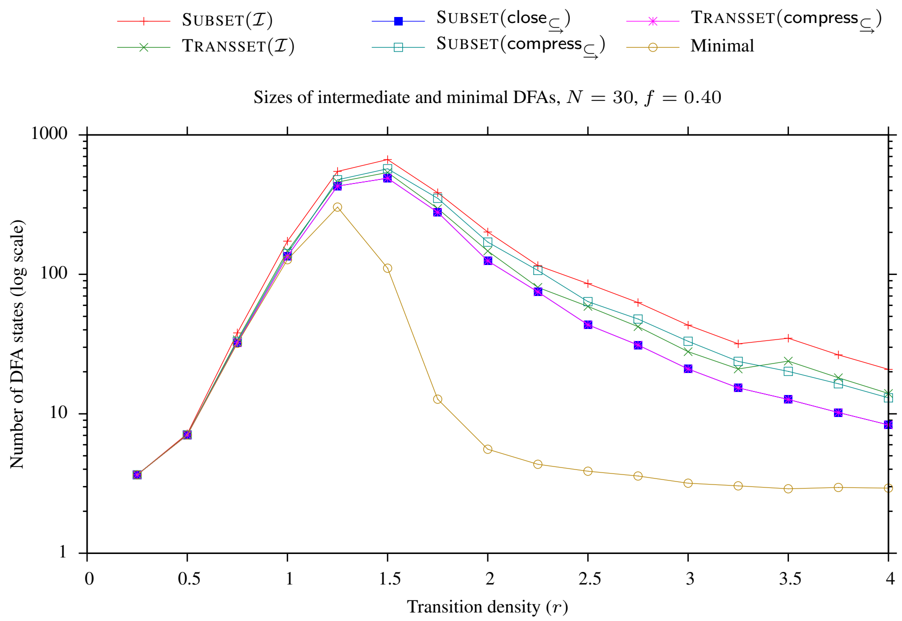
By the last two theorems we have $|S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\subsetneq}))}| = |S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\subsetneq}))}|$ for any input NFA $\mathcal{N}$. Thus, the surjective function $\mathsf{close}_{\subsetneq}$ from $S_{\mathcal{D}(\textsc{Transset}(\mathsf{compress}_{\subsetneq}))}$ to $S_{\mathcal{D}(\textsc{Subset}(\mathsf{close}_{\subsetneq}))}$ constructed in the last proof must be a bijection. It is not hard to see that it therefore is an isomorphism.

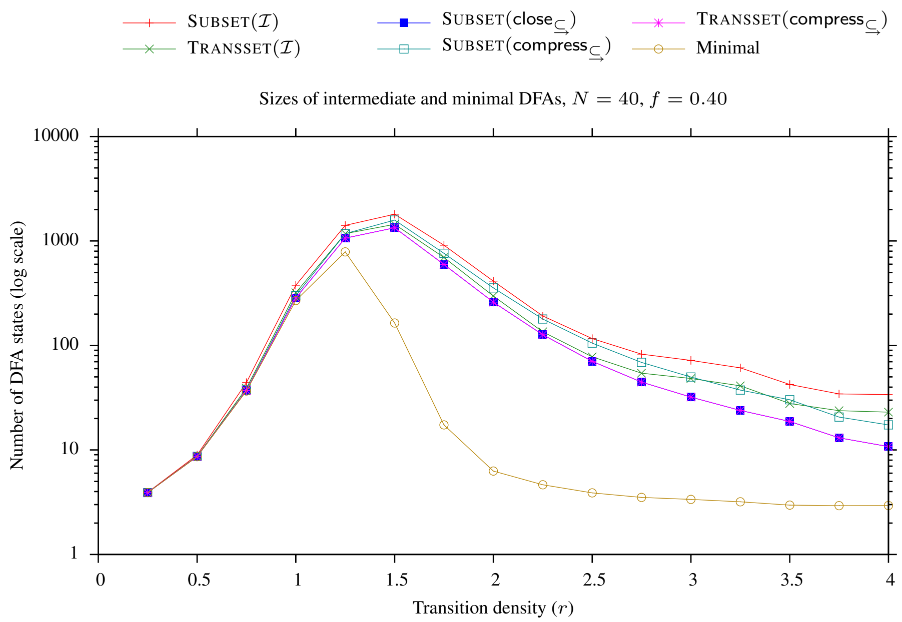
D Plots for random automata

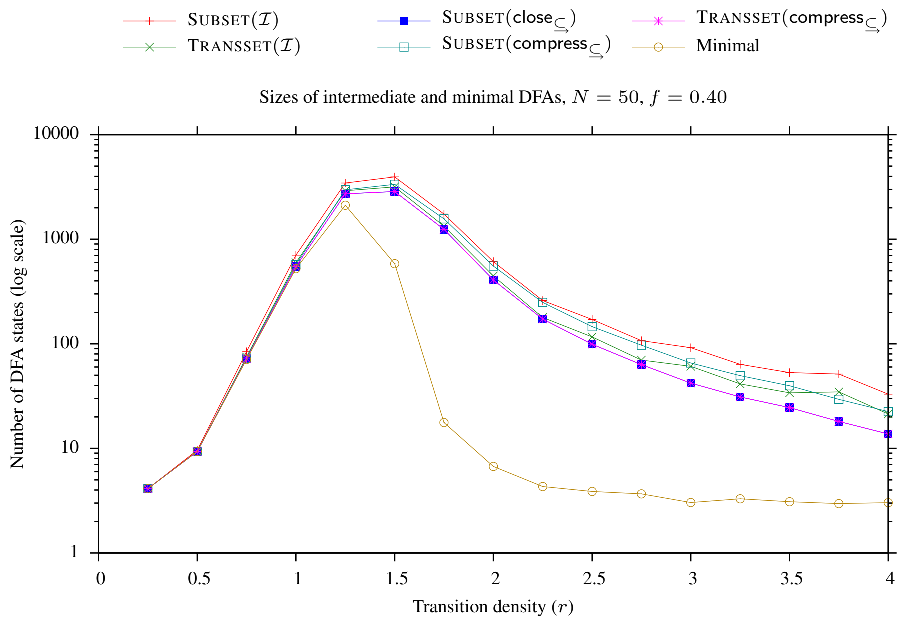
D.1 DFA Size

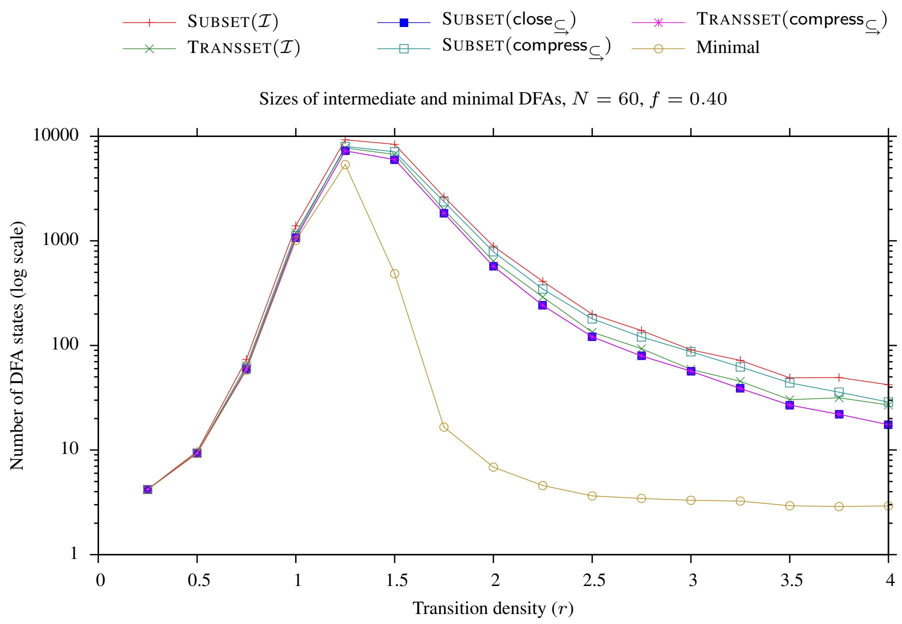


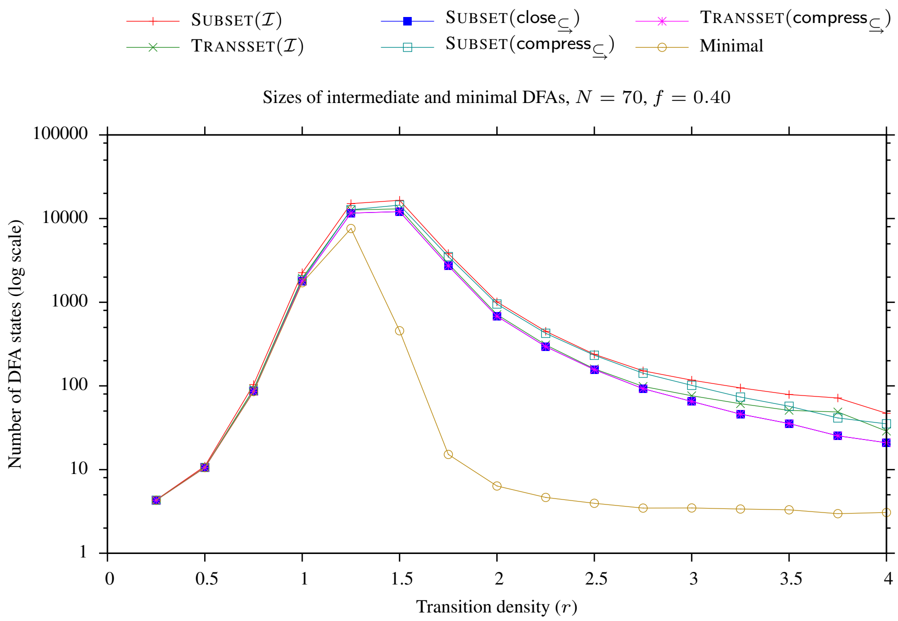


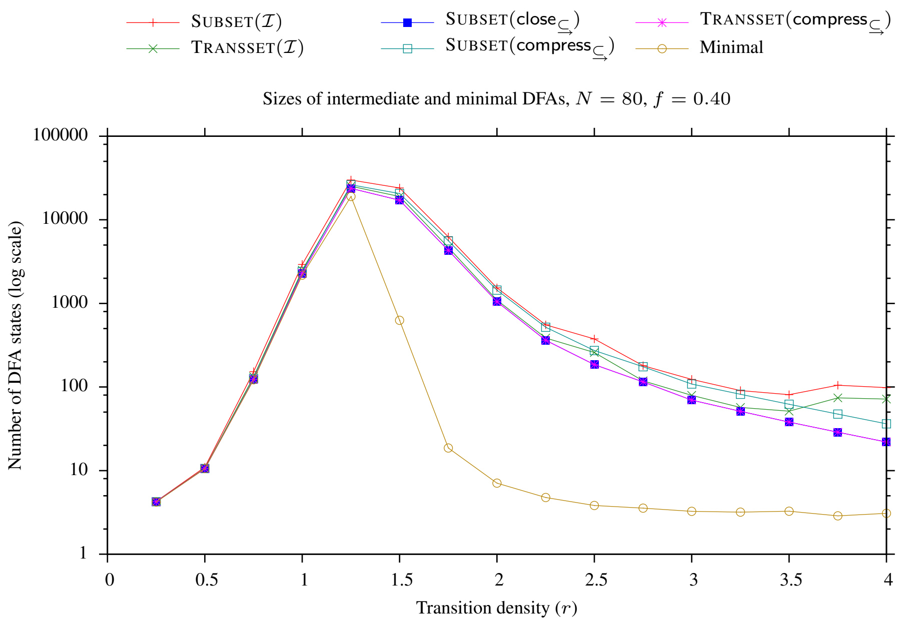

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 30$, $f = 0.40$
Number of DFA states (log scale)
Transition density (r)
1
10
100
1000
0
0.5
1
1.5
2
2.5
3
3.5
4

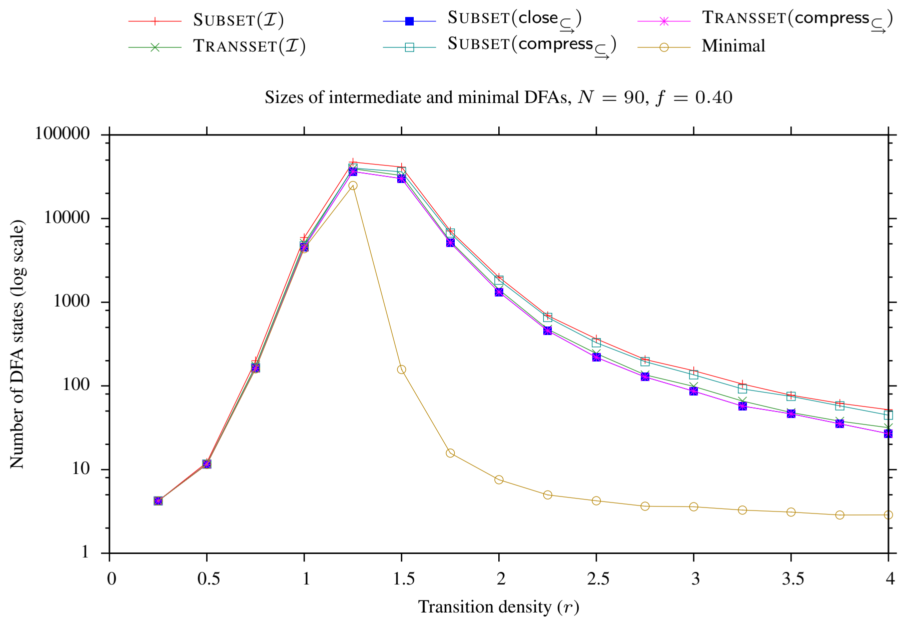

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 40$, $f = 0.40$
Number of DFA states (log scale)
Transition density (r)
1
10
100
1000
10000
0
0.5
1
1.5
2
2.5
3
3.5
4


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subsetneq}$)
SUBSET(compress$_{\subsetneq}$)
TRANSSET(compress$_{\subsetneq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 50$, $f = 0.40$
Number of DFA states (log scale)
Transition density (r)

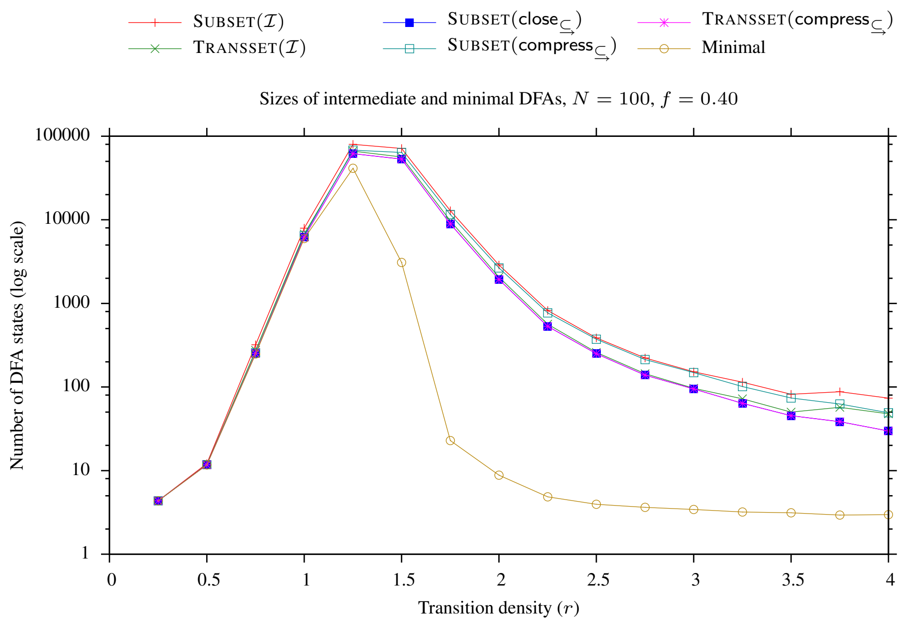

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subsetneq}$)
SUBSET(compress$_{\subsetneq}$)
TRANSSET(compress$_{\subsetneq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 60$, $f = 0.40$
Number of DFA states (log scale)
Transition density (r)


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 70$, $f = 0.40$
Number of DFA states (log scale)
Transition density (r)


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 80$, $f = 0.40$
Number of DFA states (log scale)
Transition density (r)

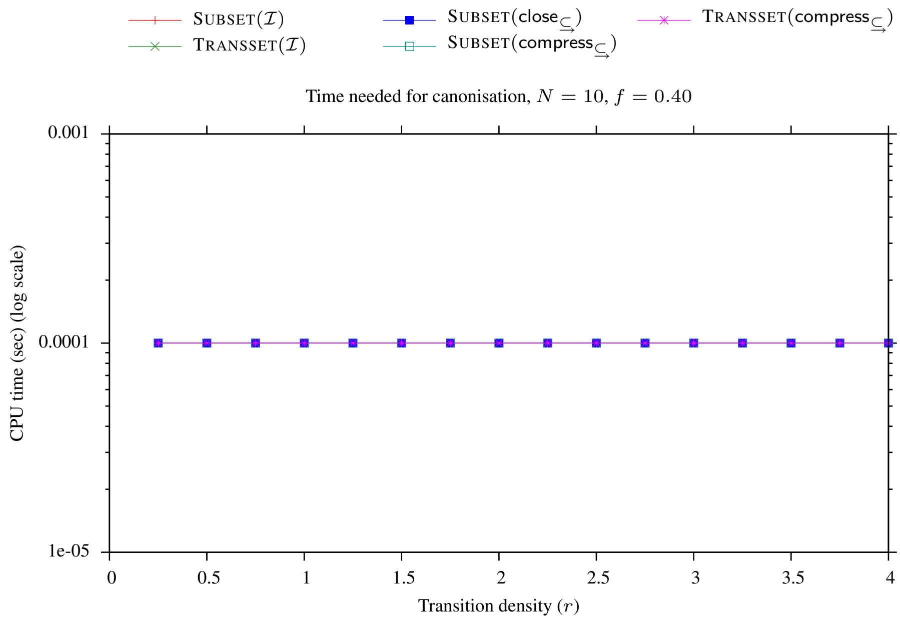


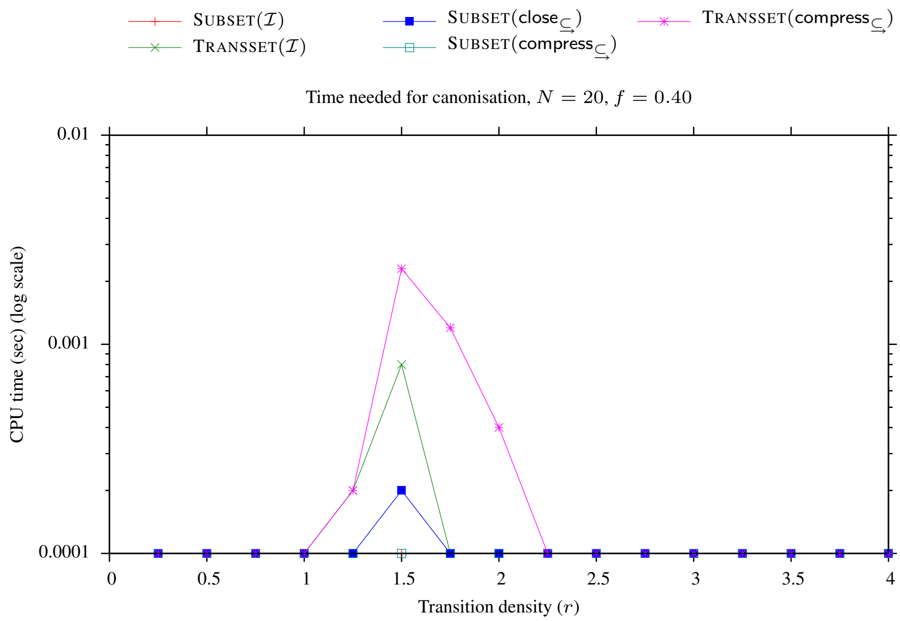
SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 90$, $f = 0.40$
Number of DFA states (log scale)
100000
10000
1000
100
10
1
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

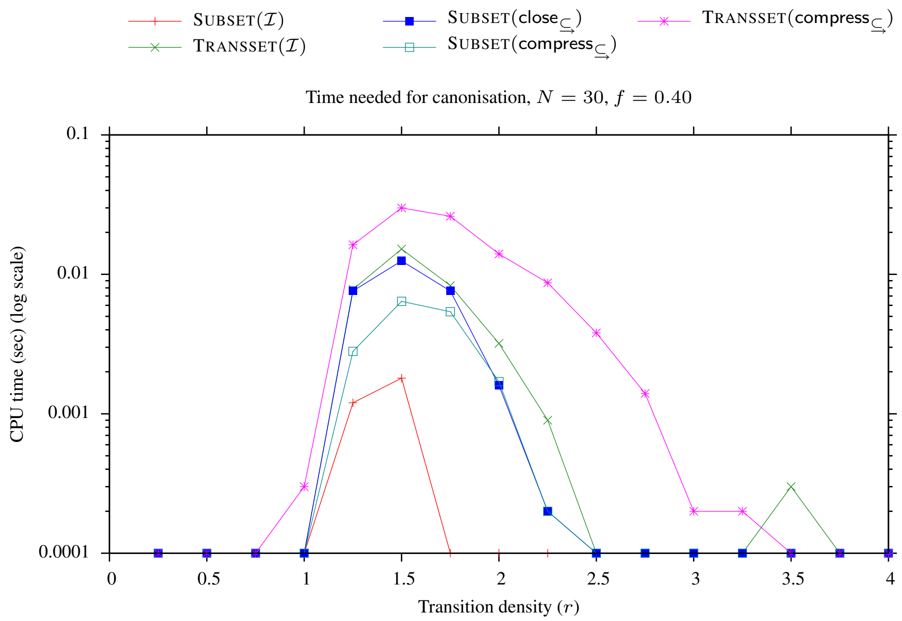


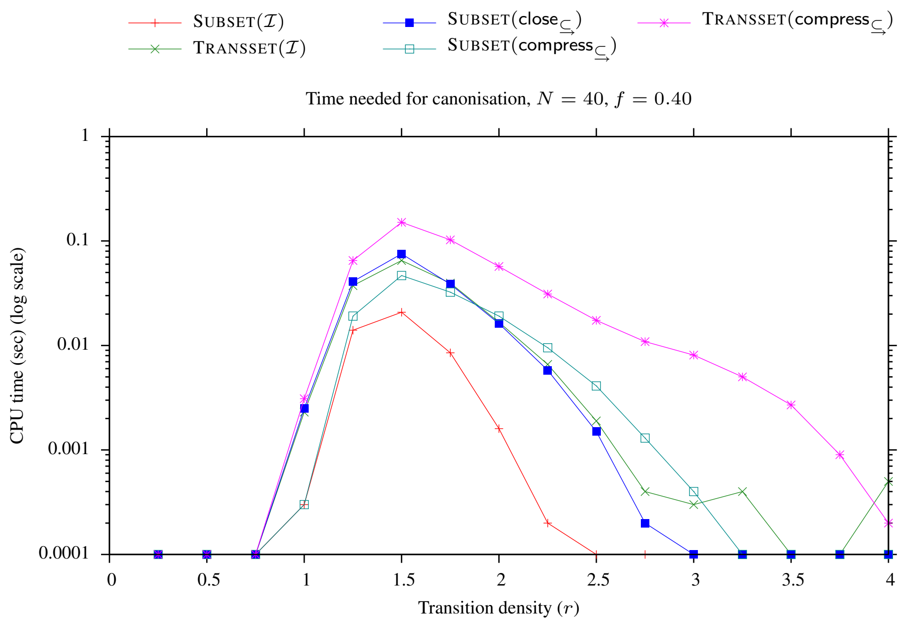
SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Minimal
Sizes of intermediate and minimal DFAs, $N = 100$, $f = 0.40$
Number of DFA states (log scale)
100000
10000
1000
100
10
1
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

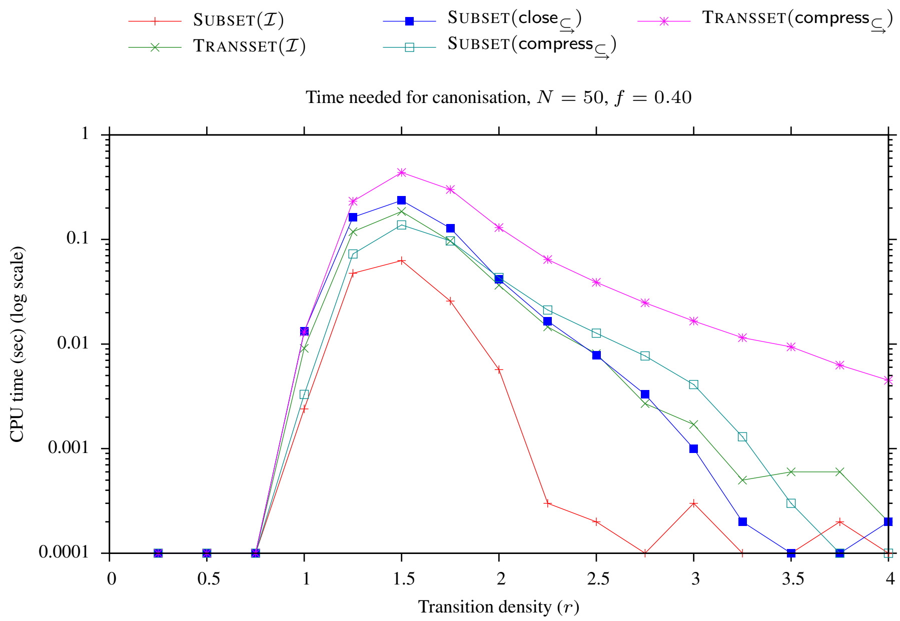
D.2 Time

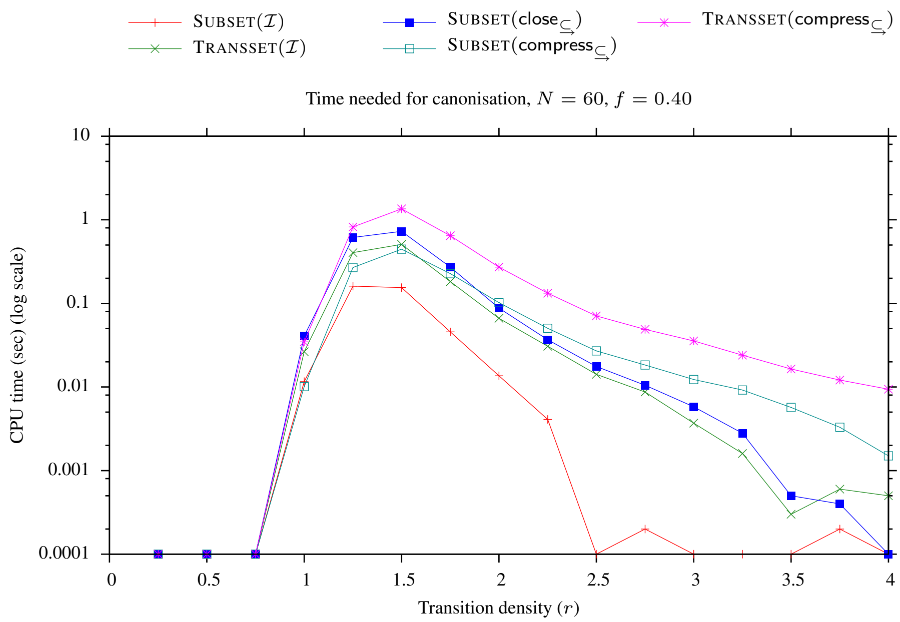

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Time needed for canonisation, $N = 10$, $f = 0.40$
CPU time (sec) (log scale)
0.001
0.0001
1e-05
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

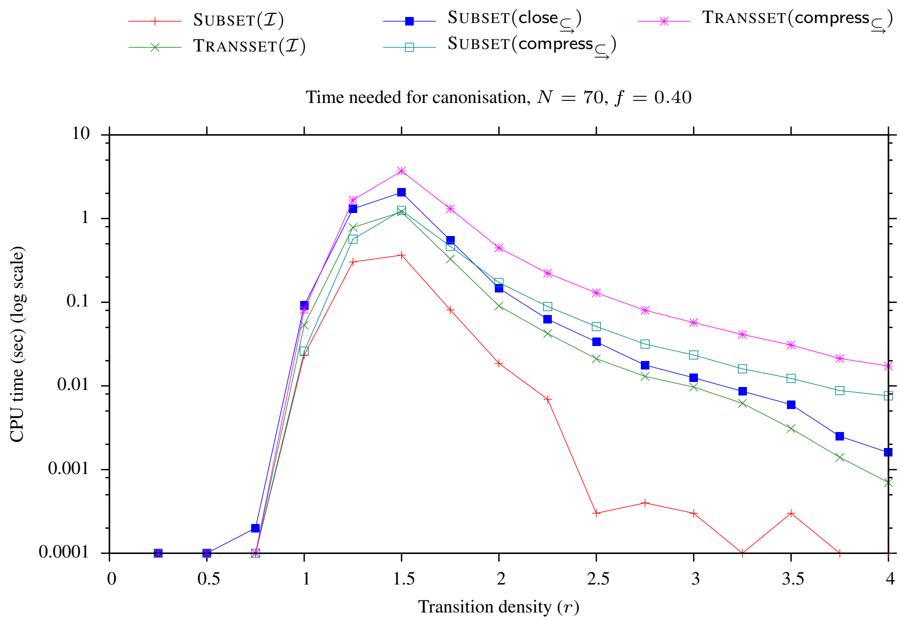

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Time needed for canonisation, $N = 20$, $f = 0.40$
CPU time (sec) (log scale)
0.01
0.001
0.0001
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

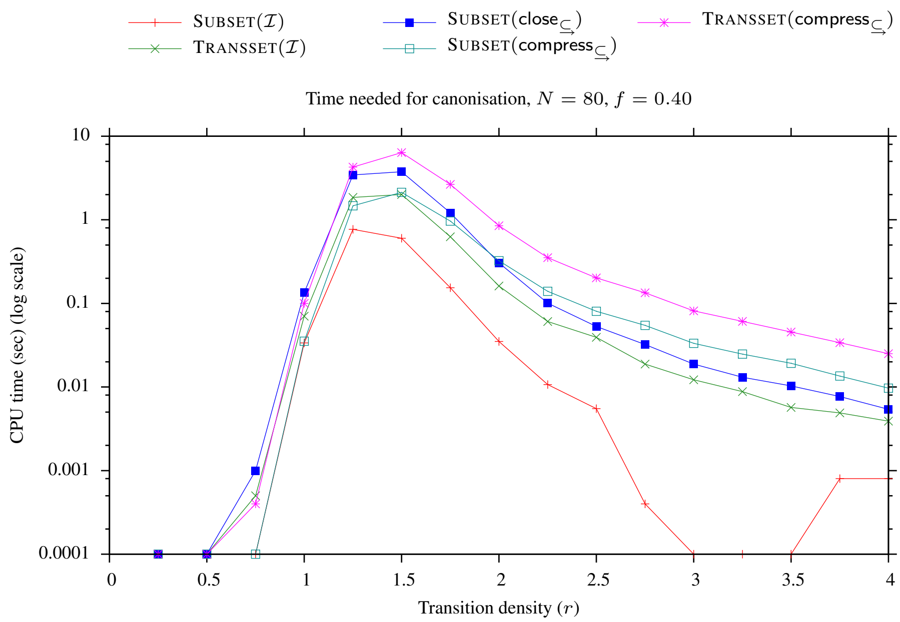

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Time needed for canonisation, $N = 30$, $f = 0.40$
CPU time (sec) (log scale)
0.1
0.01
0.001
0.0001
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

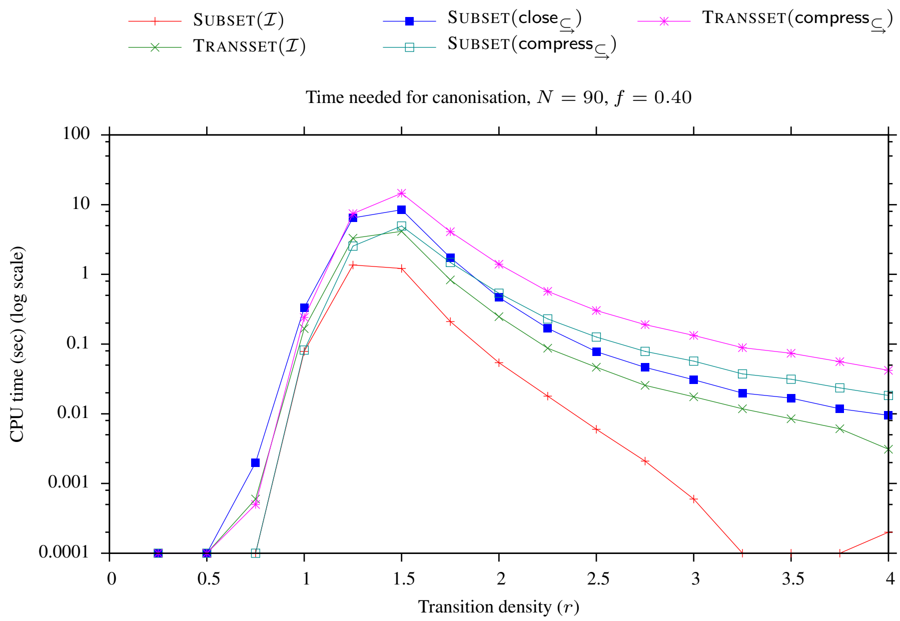

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Time needed for canonisation, $N = 40$, $f = 0.40$
CPU time (sec) (log scale)
1
0.1
0.01
0.001
0.0001
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

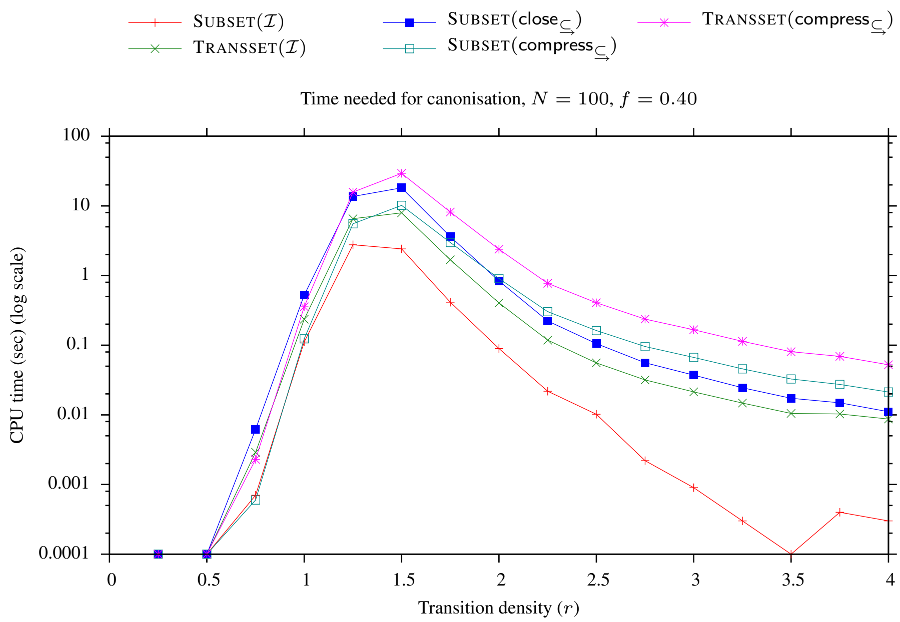

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Time needed for canonisation, $N = 50$, $f = 0.40$
CPU time (sec) (log scale)
1
0.1
0.01
0.001
0.0001
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Time needed for canonisation, $N = 60$, $f = 0.40$
CPU time (sec) (log scale)
10
1
0.1
0.01
0.001
0.0001
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

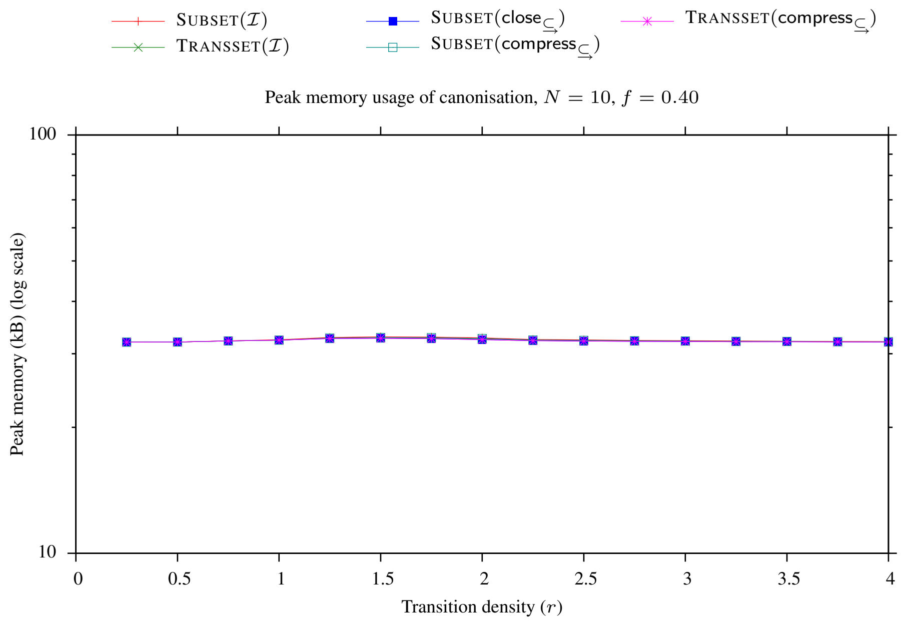

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Time needed for canonisation, $N = 70$, $f = 0.40$
CPU time (sec) (log scale)
10
1
0.1
0.01
0.001
0.0001
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

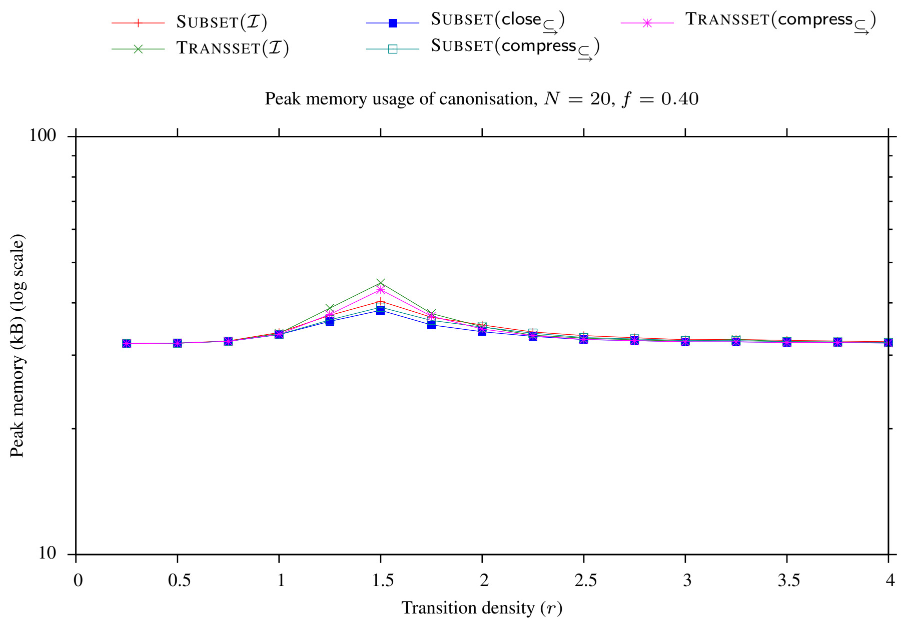

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Time needed for canonisation, $N = 80$, $f = 0.40$
CPU time (sec) (log scale)
10
1
0.1
0.01
0.001
0.0001
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

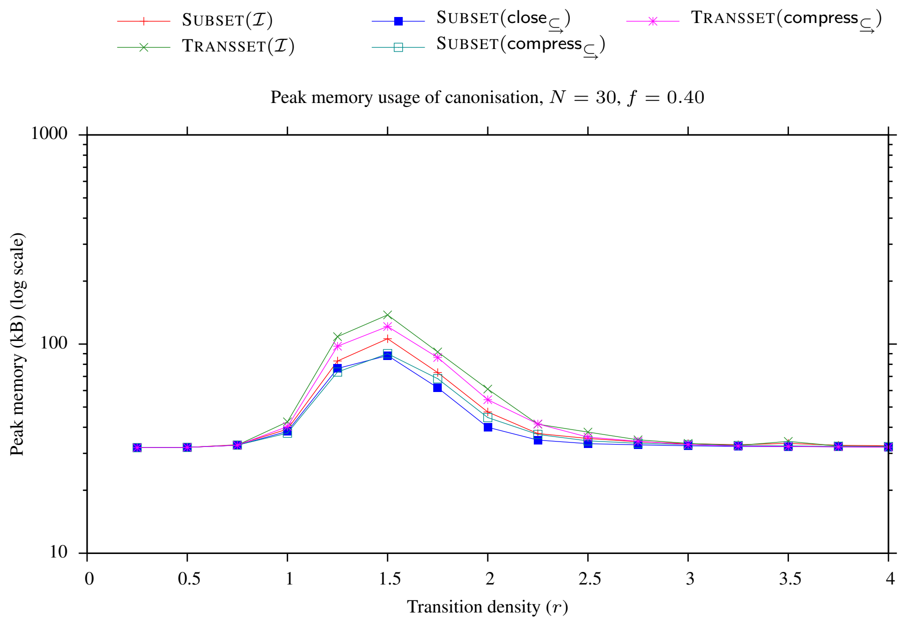

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Time needed for canonisation, $N = 90$, $f = 0.40$
CPU time (sec) (log scale)
Transition density (r)

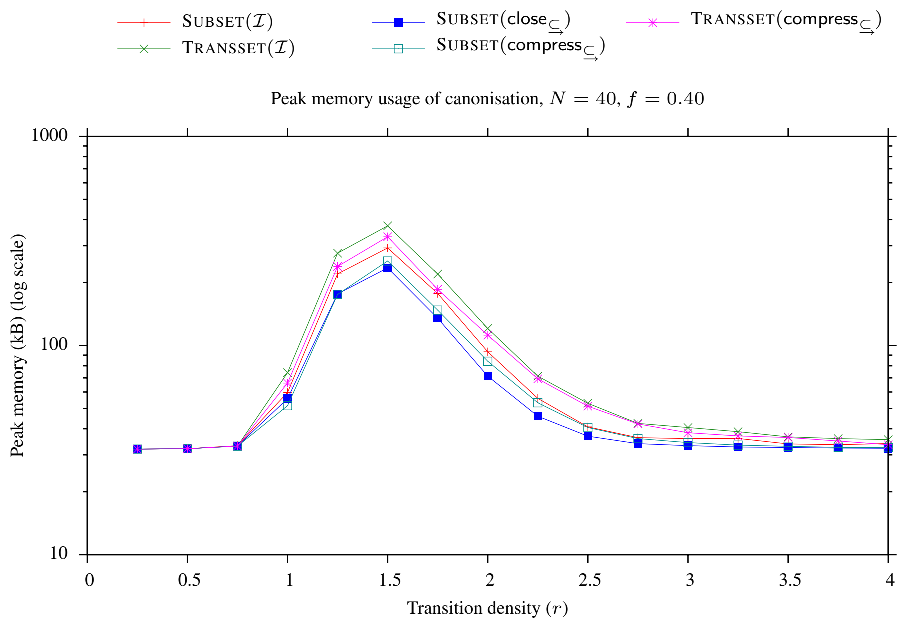

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Time needed for canonisation, $N = 100$, $f = 0.40$
CPU time (sec) (log scale)
Transition density (r)

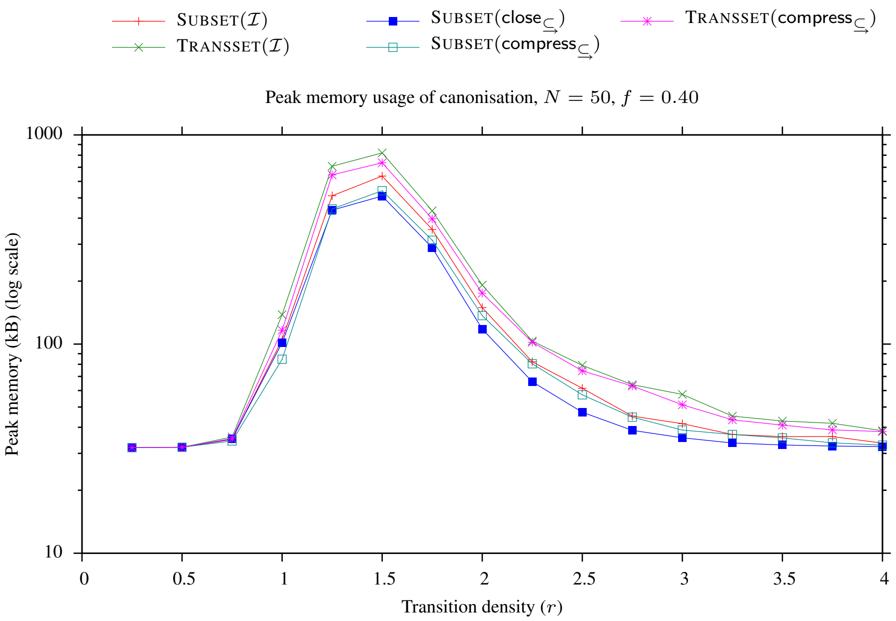
D.3 Memory

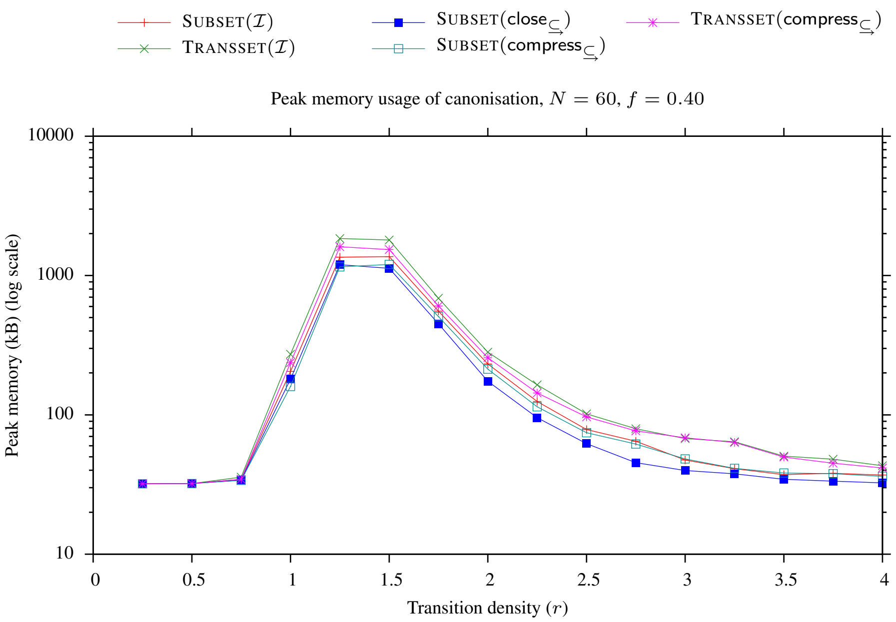

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Peak memory usage of canonisation, $N = 10$, $f = 0.40$
Peak memory (kB) (log scale)
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

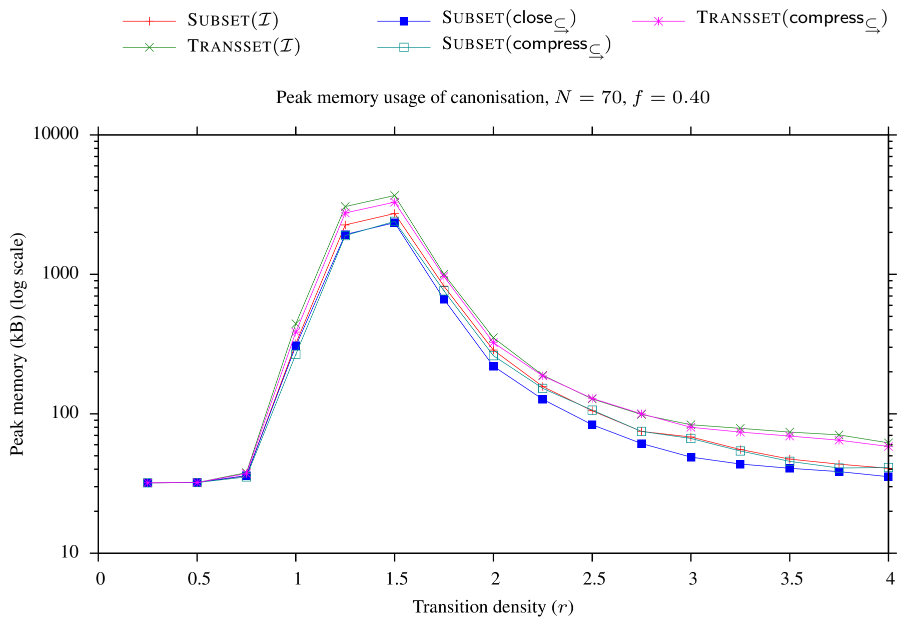

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Peak memory usage of canonisation, $N = 20$, $f = 0.40$
Peak memory (kB) (log scale)
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Peak memory usage of canonisation, $N = 30$, $f = 0.40$
Peak memory (kB) (log scale)
1000
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)

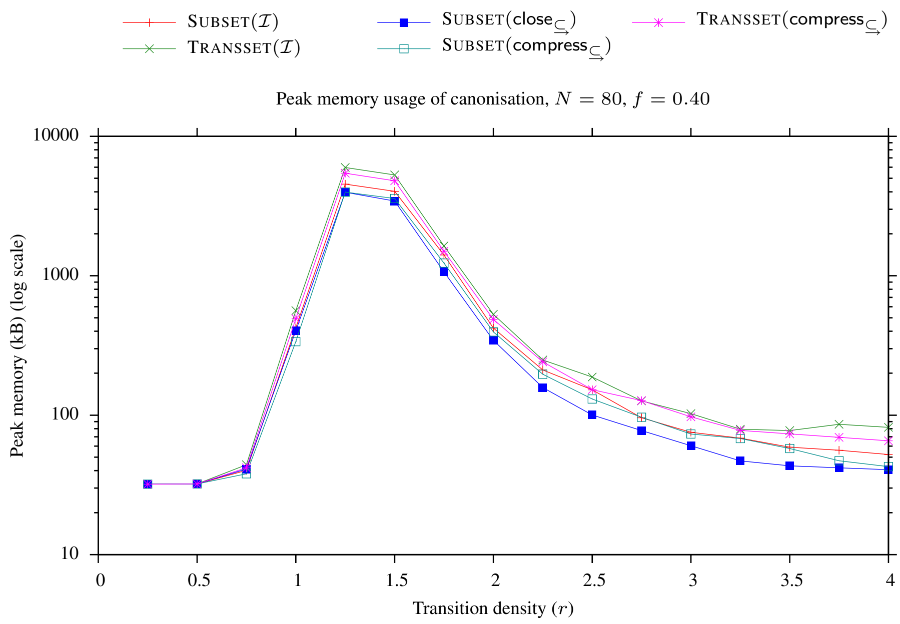

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Peak memory usage of canonisation, $N = 40$, $f = 0.40$
Peak memory (kB) (log scale)
1000
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4
Transition density (r)


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Peak memory usage of canonisation, $N = 50$, $f = 0.40$
Peak memory (kB) (log scale)
Transition density (r)
1000
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4

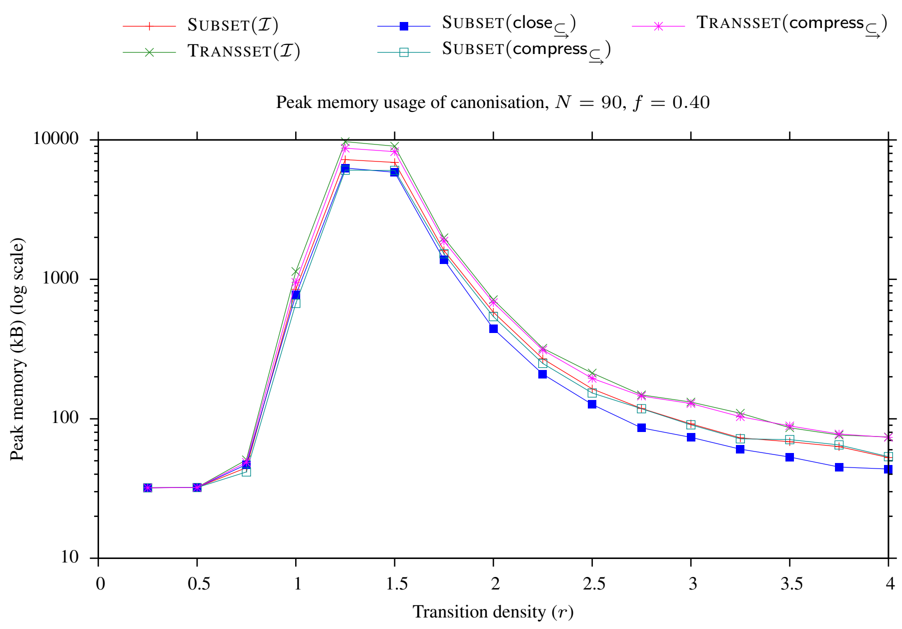

SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Peak memory usage of canonisation, $N = 60$, $f = 0.40$
Peak memory (kB) (log scale)
Transition density (r)
10000
1000
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4

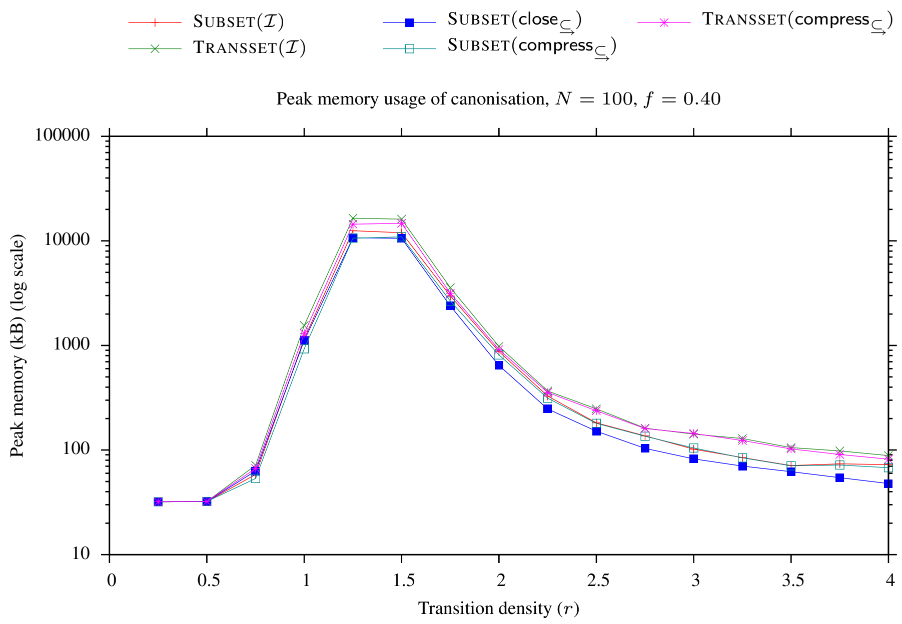


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Peak memory usage of canonisation, $N = 70$, $f = 0.40$
Peak memory (kB) (log scale)
Transition density (r)
10000
1000
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4



SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_\subseteq$)
SUBSET(compress$_\subseteq$)
TRANSSET(compress$_\subseteq$)
Peak memory usage of canonisation, $N = 80$, $f = 0.40$
Peak memory (kB) (log scale)
Transition density (r)
10000
1000
100
10
0
0.5
1
1.5
2
2.5
3
3.5
4


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Peak memory usage of canonisation, $N = 90$, $f = 0.40$
Peak memory (kB) (log scale)
Transition density (r)


SUBSET($\mathcal{I}$)
TRANSSET($\mathcal{I}$)
SUBSET(close$_{\subseteq}$)
SUBSET(compress$_{\subseteq}$)
TRANSSET(compress$_{\subseteq}$)
Peak memory usage of canonisation, $N = 100$, $f = 0.40$
Peak memory (kB) (log scale)
Transition density (r)