Automatic Sequences: The effect of local changes on complexity

This research was done as a master thesis for the MFoCS specialisation at the Radboud University in Nijmegen

Flip van Spaendonck
Master Student
S4343123

Hans Zantema
Supervisor

Wieb Bosma
Second Supervisor

Abstract

We take a look at k-DFAOs, which are Deterministic Finite Automata with Output with a special property: each k-DFAO represents a k-automatic sequence a, an infinite sequence in which the i-th element is the output the automata produces for the k-ary representation of i. Given any k-automatic sequence a, we define their complexity $||a||_k$ as the size of the smallest possible k-DFAO representing our sequence and similarly the reverse complexity $||a||_k^R$ for the right to left representation of k-ary numbers.

To be more specific, we look at local changes f to our sequences, that only change a finite amount of elements, and find an upper bound for the complexities $||f(a)||_k$ and $||f(a)||_k^R$, when applied to an arbitrary sequence a. We then use SAT/SMT solvers to prove that these upper bounds can not be further improved, thus establishing a lower bound as well. We also create an algorithm for minimizing any k-DFAO, which will give us a more efficient way of getting $||a||_k$ than using a SAT/SMT solver. We also look at translating k-DFAOs to a higher base k^p and their respective effects on complexity.

Contents

1 Introduction

One of the classic questions in automata theory is what the smallest equivalent automaton of any given automaton is. A very interesting set of automata is the set of k-DFAOs, DFA's that parse k-ary representations of natural numbers and give an output accordingly. An example of a k-DFAO is the automaton displayed in Figure 1.

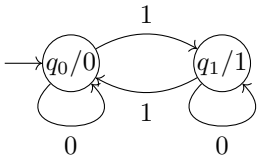

Figure 1: An example of a 2-DFAO mapping the natural numbers to either 0 or 1 depending on whether their binary representation contains an odd or even number of 1's.

For example, this automaton would map the number 5 (or 101 in binary) to state q_0 which gives 0 as an output, whereas 2 (or 10 in binary) would map to state q_1 giving 1 as an output. Using this we can make an infinite sequence called a k-automatic sequence where the i-th element is the output the automata would produce for the k-ary representation of i. The 2-automatic sequence of the 2-DFAO in Figure 1 would look as follows: $\mathsf{thue} = 0_0 1_1 1_2 0_3 1_4 ...$ (For clarity the corresponding natural number has been written in subscript).
In this paper we will be looking at the complexity $||a||_k$, which is the size of the smallest possible k-DFAO representing the k-automatic sequence a. For example $||\mathsf{thue}||_2 = 2$ as no smaller k-DFAO equivalent to the one in Figure 1 exists. We will also be looking at the reverse $||a||_k^R$, which is the size of the smallest possible k-DFAO representing the k-automatic sequence a, for which the DFAO parses the k-ary representation of natural numbers in reverse direction, from right to left instead of the usual left to right.
In the research done by Hans Zantema in [4] and its continuation in collaboration with Wieb Bosma in [5], removing the first element of any k-automatic sequence a or adding a new element at the start increases its respective complexity $||a||_k$ by a factor of k, and by a power of k for the reverse $||a||_k^R$. Using a combination of removing and adding new elements we can alter the first n elements, but this would mean that the new complexity will be much bigger than the old one.
In this paper, we will show that a local change f that alters the first n elements, can be done more efficiently than by combining removing and adding elements. We will even show that the complexity increase of $||f(a)||_k$ and $||f(a)||_k^R$ compared to the original complexity of $||a||_k$ and $||a||_k^R$ is actually independent of the original complexity of a.

First in Section 2 we give a brief introduction to DFAOs, k-automatic se-

quences and k-DFAOs. Then in Section 3 we discuss various local changes f on k-automatic sequence and their effects on the complexity $||f(a)||_k$ and $||f(a)||_k^R$ of the respective automata. The local changes that we will take a look at are: changing the first $n+1$ elements, changing only a single element, changing all elements in a range $[m, n]$ and replacing the first $n+1$ elements of a k-automatic sequence a with the elements of a k-automatic sequence b also known as fuse. In Section 3.5 we discuss a specific case in which we can further reduce the complexity given for the last local change. In Section 4 we take a look at algorithmically finding the smallest possible k-DFAO representing any k-automatic sequence using SMT/SATsolvers. In Section 5 we take a look at creating a minimization algorithm for DFAOs and k-DFAOs. And in section 6 we take a look at what happens to the complexity of our k-DFAOs by increasing the size of our input language. In Section 7 we summarize our findings and discuss questions that we were unable to answer in this paper.

2 Basic Definitions and Properties

The k-DFAOs and k-automatic sequences discussed in this paper will be according to the same definitions as in [5] and [1]. We summarize the definitions used in our paper, here:

Definition 1. *A DFAO D is defined as $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$, where:*

- $Q \neq \emptyset$ *is the finite set of states,*
- Σ *is the finite input alphabet,*
- $\delta : Q \times \Sigma \to Q$ *is the transition function,*
- $q_0 \in Q$ *is the initial state,*
- $\Gamma \neq \emptyset$ *is the finite output alphabet,*
- $\tau : Q \to \Gamma$ *is the output function.*

A k-DFAO is a DFAO with the input alphabet $\Sigma_k = \{0, 1, ..., k-1\}$.

Definition 2. *Given any $w = \sigma_1\sigma_2...\sigma_n \in \Sigma^*$, we write $\delta((\delta(\delta(q, \sigma_1), \sigma_2)...), \sigma_n)$ shorthand as $\delta(q, \sigma_1\sigma_2...\sigma_n)$.*
If $w = \epsilon$ then $\delta(q, \epsilon) = q$.

Definition 3. *Given any $n \in \mathbb{N}$, we have $(n)_k, (n)_k^R \in \Sigma_k^*$ where $(n)_k$ is the unique k-ary representation of n and $(n)_k^R$ is the reverse k-ary representation of n, e.g. : $(6)_2 = 110$ and $(6)_2^R = 011$*

Definition 4. *Conversely, given any $w \in \Sigma_k^*$, we have $[w]_k \in \mathbb{N}$ as the number that the k-ary representation w expresses, and $[w]_k^R \in \mathbb{N}$ as the number that the reverse k-ary representation w expresses, e.g. : $[110]_2 = 6$ and $[011]_2^R = 6$.*

Definition 5. *Given any infinite sequence $a \in \Gamma^{\mathbb{N}}$, we call this sequence k-automatic if and only if there exists a k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$ such that for all $n \in \mathbb{N}$, we have $\tau(\delta(q_0, (n)_k)) = a_n$.*
We refer to the k-DFAO D as the representation of a.

Definition 6. *Given any infinite sequence $a \in \Gamma^{\mathbb{N}}$, we call this sequence k-automatic if and only if there exists a k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$ such that for all $n \in \mathbb{N}$, we have $\tau(\delta(q_0, (n)_k^R)) = a_n$.*
We refer to this k-DFAO as the reverse representation of a.

Note: each k-DFAO is both the representation of a k-automatic sequence a, and the reverse representation of a k-automatic sequence a', with a and only a' possibly being the same k-automatic sequence.

Definition 7. *Given any k-automatic sequence a, we define $||a||_k$ as the number of states, also referred to as size or complexity of a, of the smallest k-DFAO representing a.*
Given any k-automatic sequence a, we define $||a||_k^R$ as the size of the smallest k-DFAO being the reverse representation of a.

Definition 8. *Given any word $w = \sigma_1\sigma_2...\sigma_n \in \Sigma^*$ and $i, j \in \mathbb{N}$ with $0 \leq i \leq j \leq n$; we define the subword $w[i, j] = \sigma_{i+1}\sigma_{i+2}...\sigma_{j-1}\sigma_j$.*
If $i = j$, we define the subword as $w[i, j] = \epsilon$.

2.1 Minimization of DFAOs

Definition 9. *Given any two DFAOs $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$ and $D' = (Q', \Sigma, \delta', q_0', \Gamma, \tau')$, D and D' are considered equivalent if and only if $\forall_{w \in \Sigma^*}[\tau(\delta(q_0, w)) = \tau'(\delta'(q_0', w_))]$.*

Definition 10. *A DFAO is considered minimal if and only if no smaller equivalent DFAO exists.*

Definition 11. *Given DFAO $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$ and $D = (Q', \Sigma, \delta, q_0, \Gamma, \tau)$ and language $L \subseteq \Sigma^*$, D and D' are considered L-equivalent iff for all $w \in L$ we have $\tau(\delta(q_0, w)) = \tau'(\delta'(q_0', w))$.*

Definition 12. *Given any DFAO $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$, and states $q, p \in Q$, q and p are considered equivalent if and only if for all words $w \in \Sigma^*$ we have $\tau(\delta(q, w)) = \tau(\delta(p, w))$.*

Two equivalent states p, q are denoted with $p \sim q$.
We refer to two states p, q that are not equivalent as 'distinct', and denote these as $p \nsim q$.

Theorem 1. *Any two states p, q are equivalent if and only if $\tau(p) = \tau(q)$ and for all $\sigma \in \Sigma$ we have $\delta(p, \sigma) \sim \delta(q, \sigma)$.*

Proof ($\Rightarrow$). As p and q are equivalent, we have $\tau(\delta(p,\epsilon)) = \tau(p) = \tau(q) = \tau(\delta(q,\epsilon))$. Thus the first half holds; we will prove the second half using contradiction. Given that a $\sigma \in \Sigma$ exists with $\delta(p,\sigma) \nsim \delta(q,\sigma)$, then a word $w \in \Sigma^*$ exists that differentiates $\delta(p,\sigma)$ and $\delta(q,\sigma)$. However this would also mean that σw would differentiate p and q, which contradicts the fact that they are equivalent.
Proof ($\Leftarrow$). We will prove that p and q are equivalent using case distinction on words $w \in \Sigma^*$:
Case 1: $w = \epsilon$. We have $\tau(\delta(p,\epsilon)) = \tau(p) = \tau(q) = \tau(\delta(q,\epsilon))$.
Case 2: $w = \sigma w'$ with $\sigma \in \Sigma$ and $w' \in \Sigma^*$. Which gives us: $\tau(\delta(p,\sigma w')) = \tau(\delta(\delta(p,\sigma),w')) = \tau(\delta(\delta(q,\sigma),w')) = \tau(\delta(q,\sigma w'))$.
With both sides proven we now also have an inductive definition of state equivalence. □

2.2 Smaller input languages

Because we will only be looking at the k-ary representation of natural numbers, the input language for k-DFAOs is a strict subset of Σ_k^* namely:

$$(\mathbb{N})_k = \{(n)_k | n \in \mathbb{N}\} = \Sigma_k^* \setminus \{0w | w \in \Sigma_k^*\}$$

And for the reverse k-ary representation of natural numbers, we have the following:

$$(\mathbb{N})_k^R = \{(n)_k^R | n \in \mathbb{N}\} = \Sigma_k^* \setminus \{w0 | w \in \Sigma_k^*\}$$

Because of this smaller input language, a k-DFAO might have an equivalent smaller automaton, whereas a normal DFAO would not have one and thus be minimal. An example of this is the following set of DFAOs:

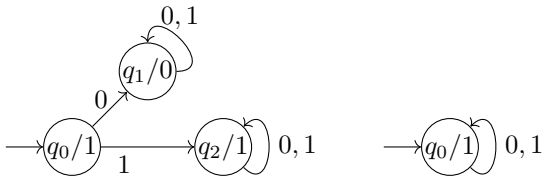

Figure 2: Two equivalent DFAOs unless the limited input language $(\mathbb{N})_2$ is used. Both automata represent the k-automatic sequence $a = 111....$

In Figure 2 the left DFAO has a different ouput than the right DFAO for words starting with 0. However since the input alphabet for k-ary representation of natural numbers does not contain words starting with a 0, these two k-DFAOs are considered equivalent in at least some way. Because of this we will now define a variation of earlier properties for when we are using a smaller input language L, such as $(\mathbb{N})_k$ and $(\mathbb{N})_k^R$.

Definition 13. *Given any two DFAOs* $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$ *and* $D' = (Q', \Sigma, \delta', q'_0, \Gamma, \tau')$ *and language* $L \subseteq \Sigma^*$*, we call* D *and* D' L*-equivalent if and only if* $\forall_{w \in L}[\tau(\delta(q_0, w)) = \tau'(\delta'(q'_0, w))]$.

Definition 14. *A* k*-DFAO* D *is considered minimal in* $(\mathbb{N})_k$ *if and only if there exists no smaller* $(\mathbb{N})_k$*-equivalent* k*-DFAO.*
And for the reverse, a k*-DFAO* D *is considered minimal in* $(\mathbb{N})_k^R$ *if and only if there exists no smaller* $(\mathbb{N})_k^R$*-equivalent* k*-DFAO.*

Definition 15. *Given any DFAO* $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$ *and language* $L \subseteq \Sigma^*$*, a state* $q \in Q$ *is considered reachable in* L *if and only if:*

$$\exists_{w \in \Sigma^*}[\delta(q_0, w) = q \wedge (w \notin L \Rightarrow \exists_{w' \in \Sigma^*}[ww' \in L])]$$

This tells us that, if q is reachable in L, there should either exist a word $w \in L$ that ends up in q or a word $ww' \in L$ that goes through q. If there are no words in L that end up in q, but q is still reachable, then we call it *hidden* as its output remains hidden. Languages such as $(\mathbb{N})_k^R$ can have hidden states, such as the state q_1 as seen in Figure 2.

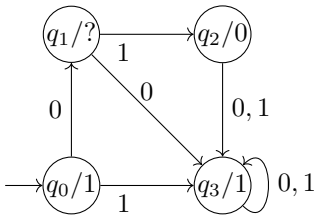

Figure 3: A $(\mathbb{N})_k^R$-minimal k-DFAO of k-automatic sequence $a = 110111...$ *(Note: As no input will ever end at state* q_1 *its output does not matter.)*

3 The effects of local changes in sequences and their effects

In this section we will discuss different local changes f on k-automatic sequences a, these are changes in which only a finite number of elements of the infinite sequence are changed. We will then construct k-DFAOs representing the new k-automatic sequences $f(a)$, thus providing an upper-bound for $||f(a)||_k$ and $||f(a)||_k^R$. We will look at the specific set of transformation on any k-automatic sequence a, in which the first n elements are possibly altered to different values of Γ. Throughout this section we will use summations in the shape of $\Sigma_{i=1}^{\max} k^i$ for calculating the size of the new constructed k-DFAOs. We opt to not write these as their closed formula, as $\Sigma_{i=\min}^{\max} k^i$ more closely represents all words $w \in \Sigma_k$ with a character length between min and max.

3.1 Changing the first $n+1$ elements of arbitrary k-automatic sequences

Definition 16. *Given k-automatic sequence a and $x_m, ..., x_n \in \Gamma$ and $m, n \in \mathbb{N}$ with $0 \leq m < n$, we define the sequence $a[m, ..., n \mapsto x_m, ..., x_n]$ as follows:*

$$a[m, ..., n \mapsto x_m, ..., x_n]_i = \begin{cases} x_i & \text{iff } m \leq i \leq n \\ a_i & \text{otherwise} \end{cases}$$

In [5] it was shown that removing the first element or adding an element to the start of a k-automatic sequence increases the size of the k-DFAO either by a factor k or to the power of k if we look at its reverse. Altering the first n elements of a k-automatic sequence using a combination of these two methods, would be very inefficient. Thus we will use a different method to create a way smaller resulting k-DFAO, namely:

Theorem 2. *Given any k-automatic sequence a we have*
$||a[0, ..., n \mapsto x_0, ..., x_n]||_k \leq ||a||_k + n + 1$.
And $||a[0, ..., n \mapsto x_0, ..., x_n]||_k^R \leq ||a||_k^R + \sum_{i=0}^{\lfloor \log_k(n) \rfloor} k^i \leq ||a||_k^R + 1 + \lceil \frac{nk}{k-1} \rceil$

Proof. For k-automatic sequence a with k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$, we define the following k-DFAO $D' = (Q \cup P, \Sigma_k, \delta', p_0, \Gamma, \tau')$:

- $P = (p_0, p_1, ..., p_n)$ and $Q \cap P = \emptyset$
- $\delta'(q, \sigma) = \delta(q, \sigma)$ for $q \in Q$
  $\delta'(p_i, \sigma) = p_{ki+\sigma}$ iff $ki + \sigma \leq n$
  $\delta'(p_i, \sigma) = \delta(q_0, (i)_k\sigma)$ otherwise
- $\tau'(q) = \tau(q)$
  $\tau'(p_i) = x_i$

The resulting k-DFAO D' has a size of $||a||_k + n + 1$. To prove that D' represents $a[0, .., n \mapsto x_0, .., x_n]$, we will first need to prove the following claim.

Given $w \in \Sigma_k^*$. If $[w]_k \leq n$ then $\delta'(p_0, w) = p_{[w]_k}$, otherwise $\delta'(p_0, w) = \delta(q_0, w)$.
Proof. By induction over $w \in \Sigma_k^*$ we have:
Base case: $w = \epsilon$. We have $[\epsilon]_k = 0$ and we can see that $\delta'(p_0, \epsilon) = p_0$.
Induction step: Given any $w \in \Sigma_k^*$ that satisfies our claim, there are two cases:
Case 1: $\delta'(p_0, w) = p_{[w]_k}$. $\delta'(p_0, w\sigma) = \delta'(p_{[w]_k}, \sigma)$.
Case 1a: $k[w]_k + \sigma \leq n$. $\delta'(p_0, w\sigma) = p_{k[w]_k+\sigma} = p_{[w\sigma]_k}$
Case 1b: $k[w]_k + \sigma > n$. $\delta'(p_0, w\sigma) = \delta(q_0, ([w]_k)_k\sigma) = \delta(q_0, w\sigma)$
Case 2: $\delta'(p_0, w) = \delta(q_0, w)$. $\delta'(p_0, w\sigma) = \delta'(\delta(q_0, w), \sigma) = \delta(q_0, w\sigma)$.

Now that we have proven our claim through case distinction and induction, we can continue proving our original theorem.
Given any $i \in \mathbb{N}$, we can make the following case distinction:
Case 1: $i \leq n$. $\tau'(\delta'(p_0, (i)_k)) = \tau'(p_i) = x_i$.
Case 2: $i > n$. $\tau'(\delta'(p_0, (i)_k)) = \tau'(\delta(q_0, (i)_k)) = \tau(\delta(q_0, (i)_k))$.
Conclusion: Through case distinction we have proven that D' represents $a[0, ..., n \mapsto x_0, ..., x_n]$. □
As an example, constructing the k-DFAO of $\mathsf{thue}[0, 1, 2, 3 \mapsto x_0, x_1, x_2, x_3]$ would look as follows:

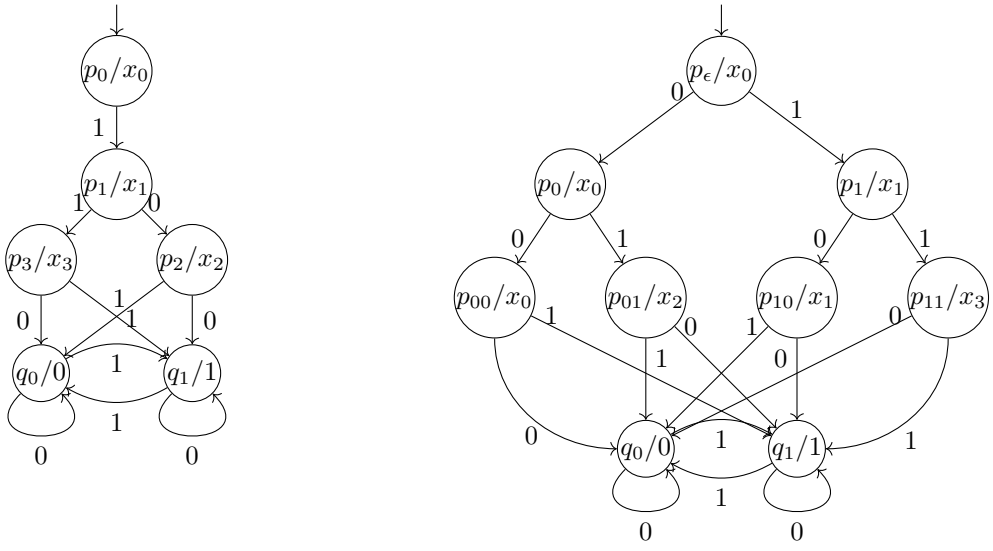

Figure 4: k-DFAO for $\mathsf{thue}[0, .., 3 \mapsto x_0, x_1, x_2, x_3]$ left, and its reverse representation on the right

Proof for reverse: Given any k-automatic sequence a with reverse k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$. We will construct the reverse k-DFAO $D' = (Q \cup P, \Sigma_k, \delta', p_\epsilon, \Gamma, \tau')$ as follows:

- $P = \{p_w | w \in \Sigma_k^*, [w]_k^R \leq n, w$ has a character length no longer than that of $(n)_k\}$ and $P \cap Q = \emptyset$.

- Given any $\sigma \in \Sigma_k$, we define δ' as follows:
  $\delta'(q, \sigma) = \delta(q, \sigma)$ for any $q \in Q$.
  $\delta'(p_w, \sigma) = p_{w\sigma}$ iff $[w\sigma]_k^R \leq n$ and $w\sigma$ is not longer than $(n)_k^R$.
  $\delta'(p_w, \sigma) = \delta(q_0, w\sigma)$ otherwise.

- $\tau'(q) = \tau(q)$ for any $q \in Q$.
  $\tau'(p_w) = x_{[w]_k^R}$.

Because of the construction of P, we know the resulting k-DFAO will have a size of $||a||_k^R + \sum_{i=0}^{\lfloor \log_k(n) \rfloor} k^i$.
We will now prove that the constructed k-DFAO represents $a[0, .., n \mapsto x_0, .., x_n]$.
To do so we will first prove the following claim:
Given any $w \in \Sigma_k^*$, if $[w]_k^R \leq n$ and w is not longer than $(n)_k^R$, then $\delta'(p_\epsilon, w) = p_w$. Otherwise $\delta'(p_\epsilon, w) = \delta(q_0, w)$.
We will prove this claim via induction over $w \in \Sigma_k^*$:
Base case: $w = \epsilon$. This gives us $\delta'(p_\epsilon, \epsilon) = p_\epsilon$. We also have $[\epsilon]_k^R = 0$ which is always smaller than n, thus the condition is satisfied.
Induction step: For the induction step we will make a case distinction on w:
Case 1a: $\delta'(p_\epsilon, w) = p_w$ and $[w\sigma]_k^R \leq n$ and $w\sigma$ is not longer than $(n)_k^R$. This will give us $\delta'(p_\epsilon, w\sigma) = \delta'(p_w, \sigma) = p_{w\sigma}$. And the condition also holds as we have $[w\sigma]_k^R \leq n$.
Case 1b: $\delta'(p_\epsilon, w) = p_w$ but $[w\sigma]_k^R > n$ or $w\sigma$ is longer than $(n)_k^R$. This gives us $\delta'(p_\epsilon, w\sigma) = \delta'(p_w, \sigma) = \delta(q_0, w\sigma)$. And as by our case's definition the condition does not hold.
Case 2: $\delta'(p_\epsilon, w) = \delta(q_0, w)$ and $[w]_k^R > n$ or w is longer than $(n)_k^R$. This gives us $\delta'(p_\epsilon, w\sigma) = \delta'(\delta(q_0, w), \sigma) = \delta(\delta(q_0, w)\sigma) = \delta(q_0, w\sigma)$. And we have either $[w\sigma]_k^R \geq [w]_k^R > n$ or $w\sigma$ is longer than w which is longer than $(n)_k$, thus the condition does not hold.
Conclusion: By induction and case distinction we have proven our claim. We will now go on to prove that given any $i \in \mathbb{N}$, we have $\tau'(\delta'(p_\epsilon, (i)_k^R)) = x_i$ if $i \leq n$, and $\tau'(\delta'(p_\epsilon, (i)_k^R)) = \tau(\delta(q_0, w))$ otherwise. We will do this through case distinction on $i \in \mathbb{N}$:
Case 1: $(i)_k^R \leq n$. As $(i)_k^R \leq n$, we have $(i)_k^R$ is not longer than $(n)_k^R$. This gives us $\tau'(\delta'(p_\epsilon, (i)_k^R)) = \tau'(p_{(i)_k^R}) = x_{[(i)_k^R]_k^R} = x_i$.
Case 2: $(i)_k^R > n$. This gives us $\tau'(\delta'(p_\epsilon, (i)_k^R)) = \tau'(\delta(q_0, (i)_k^R)) = \tau(\delta(q_0, (i)_k^R))$.
Conclusion: The constructed k-DFAO represents the reverse sequence of $a[0, .., n \mapsto x_0, .., x_n]$.
Earlier in this proof we only showed that P has at most a size of $\sum_{i=0}^{\lfloor \log_k(n) \rfloor} k^i$. This bound can also be more precisely written as the following closed formula: $||a||_k^R + 1 + \lceil \frac{nk}{k-1} \rceil$.
Given any arbitrary length $m > 0$, there are $(k-1)k^{m-1}$ words in Σ_k^m that are also in $(\mathbb{N})_k^R$ as these are all the words not ending in a 0. This leaves us k^{m-1} words that are not in $(\mathbb{N})_k^R$. Thus given an arbitrary length $m > 0$, $\frac{1}{k-1}$ of all words in Σ_k^m are not in $(\mathbb{N})_k^R$. This means that for all the states in P, we have 1 state p_ϵ representing the emptyword, we have n states representing a word in $(\mathbb{N})_k^R$. And thus we have $\lceil \frac{n}{k-1} \rceil$ states representing a word not in $(\mathbb{N})_k^R$. The re-

sulting size of our automaton is thus $||a||_k^R+1+n+\lceil \frac{n}{k-1} \rceil = ||a||_k^R+1+\lceil \frac{nk}{k-1} \rceil$. $\square$

3.2 Changing only a single element a_n of an arbitrary k-automatic sequence

Definition 17. *Given a k-automatic sequence a, $n \in \mathbb{N}$ and $x \in \Gamma$; we define the sequence $a[n \mapsto x]$ as follows:*

$$a_i[n \mapsto x] = x \text{ iff } i = n$$
$$a_i \text{ otherwise}$$

The sequence $a[n \mapsto x]$ can also be expressed as $a[0, .., n-1, n \mapsto a_0, .., a_{n-1}, x]$, using the previously discussed method would yield a minimizable k-DFAO if $n \geq 2$. Thus we will use a closer approximation of the resulting k-DFAO:

Theorem 3. *For any k-automatic sequence a we have $||a[n \mapsto x]||_k \leq ||a||_k + \lfloor \log_k(n) \rfloor + 1$.*
And for the reverse we also have $||a[n \mapsto x]||_k^R \leq ||a||_k^R + \lfloor \log_k(n) \rfloor + 1$

Proof for part 1. For k-automatic sequence a with k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$, we define the following k-DFAO $D' = (Q \cup P, \Sigma_k, \delta', p_0, \Gamma, \tau')$:

- $P = p_0, p_1, ..., p_{\lfloor \log_k(n) \rfloor}$ and $Q \cap P = \emptyset$
- Given any $\sigma \in \Sigma_k$, we define δ' as follows:
  $\delta'(q, \sigma) = \delta(q, \sigma)$ for $q \in Q$
  $\delta'(p_i.\sigma) = p_{i+1}$ iff σ is the $i+1$th character in $(n)_k$
  $\delta'(p_i, \sigma) = \delta(q_0, (n)_k[0, i]\sigma)$ otherwise
- $\tau'(q) = \tau(q)$ for $q \in Q$
  $\tau'(p_i) = x$ iff $i = \lfloor \log_k(n) \rfloor$
  $\tau'(p_i) = \tau(\delta(q_0, (n)_k[0, i]))$ otherwise

The resulting k-DFAO D' has a size of $||a|| + \lfloor \log_k(n) \rfloor + 1$. To prove that D' represents $a[n \mapsto x]$, we will first need to prove the following claim.
We claim that, for any $w \in \Sigma_k^*$, if there exists an $i \in \mathbb{N}$ such that $w = (n)_k[0, i]$ then $\delta'(p_0, w) = p_i$, otherwise $\delta'(p_0, w) = \delta(q_0, w)$.
Proof. Through induction over $w \in \Sigma^*$, we have:
Base case: $w = \epsilon$. We find $i = 0$, we have $\epsilon = (n)_k[0, 0]$ and we have $\delta'(p_0, \epsilon) = p_0$.
Induction step: Given a $w \in \Sigma*$ we have the following two possible cases:
case 1a: $w = (n)_k[0, i]$, and σ is the $i+1$th character of $(n)_k$. We have $\delta'(p_i, \sigma) = p_{i+1}$.
case 1b: $w = (n)_k[0, i]$, and σ is not the $i+1$th character of $(n)_k$. We have $\delta'(p_i, \sigma) = \delta(q_0, (n)_k[0, i]\sigma) = \delta(q_0, w\sigma)$.
case 2: There exists no i such that $w = (n)_k[0, i]$. Thus $\delta'(p_0, w) = \delta(q_0, w)$. Thus $\delta'(p_0, w\sigma) = \delta'(\delta(q_0, w), \sigma) = \delta(q_0, w\sigma)$.
Conclusion: Through case distinction we have proven the induction step and

thus also through induction we will have proven our claim.
We can now prove that for all $w \in \Sigma_k^*$ we have $\tau'(\delta'(p_0, w)) = \tau(\delta(q_0, w))$, unless $w = (n)_k$, in which case $\tau'(\delta'(p_0, w)) = x$. We will do so through case distinction:
Case 1: $w = (n)_k$. Since $w = (n)_k = (n)_k[i, \lfloor \log_k(n) \rfloor]$ we have $\tau'(\delta'(p_0, w)) = \tau'(p_{\lfloor \log_k(n) \rfloor}) = x$.
Case 2: $w = (n)_k[0, i]$ with $i \neq \lfloor \log_k(n) \rfloor$. $\tau'(\delta'(p_0, w)) = \tau'(p_i) = \tau(\delta(q_0, w))$.
Case 3: There is no i such that $w = (n)_k[0, 1]$. With the previous claim we will see that $\tau'(\delta'(p_0, w)) = \tau'(\delta(q_0, w)) = \tau(\delta(q_0, w))$.
Conclusion: We have proven that the constructed k-DFAO D' has a size of $||a|| + \lfloor \log_k(n) \rfloor + 1$. And represents the k-automatic sequence of $a[n \mapsto x]$.

A similar proof exists for the reverse, in this we will work from right to left instead of left to right. Besides that the proof will follow the exact same structure of our original proof. □

Applying the theorem's method on $\mathsf{thue}[13 \mapsto x]$ will give us the following k-DFAO:

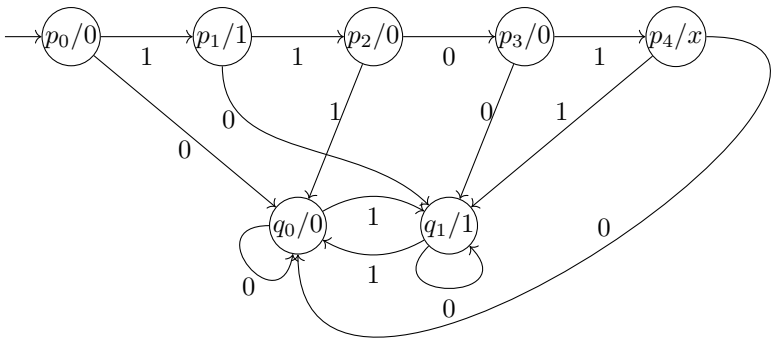

Figure 5: k-DFAO of $\mathsf{thue}[13 \mapsto x]$

The k-DFAOs size can be reduced for some examples. An easy example would be $(a[n \mapsto x])[n \mapsto a_n]$ as this actually reduces the number of states in $a[n \mapsto x]$, and brings us back to the original sequence a. Furthermore if we have $a[n \mapsto x][m \mapsto y]$ in which $(m)_k$ is a subword of $(n)_k$, the resulting size is the same as $a[n \mapsto x]$'s k-DFAO.
But for other k-DFAOs the resulting k-DFAO can not be reduced. For example the k-automatic sequence $\mathsf{thue}[n \mapsto x]$ with any $x \neq a_n$, will always have the

exact complexity as described by our theorem.

3.3 Changing a range $[m, n]$ of elements in an arbitrary k-automatic sequence

Another potential case exists, where we want to change only a part of the first n elements of a. If we want to change elements $a_m, .., a_n$ to different values and m and n both have a matching prefix in their k-ary representations $(m)_k$ and $(n)_k$ and have the exact same length; we are able to give an even closer approximation of the resulting k-DFAO, namely:

Theorem 4. *Given any k-automatic sequence a and $m, n \in \mathbb{N}$. If there exists a $\mu, \nu \in \mathbb{N}$ such that $\mu k^\nu \leq m \leq n < (\mu + 1)k^\nu$, or as we can also see it $\exists_{i \in \mathbb{N}}, (m)_k[0, i] = (\mu)_k = (n)_k[0, i]$, then we have:*

$$||a[m, .., n \mapsto x_m, .., x_n]||_k \leq ||a||_k + 1 + \lfloor log_k(\mu) \rfloor + \sum_{i=1}^{\nu} k^i = ||a||_k + \lfloor log_k(\mu) \rfloor + \frac{1 - k^\nu}{1 - k}$$

Proof. Given any k-automatic sequence a with k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$, we construct the following k-DFAO $D' = (Q \cup P \cup R, \Sigma_k, \delta', p_0, \Gamma, \tau')$:

- $P = \{p_0, ..., p_{\lceil log_k(\mu) \rceil}\}$ and $P \cap Q = \emptyset$.
- $R = \{r_w | 0 < [w]_k < k^\nu\}$ and $R \cap Q = \emptyset$.
- $\delta'(q, \sigma) = \delta(q, \sigma)$ for $q \in Q$
  $\delta'(p_{log \lfloor \mu \rfloor}, \sigma)) = r_\sigma$
  $\delta'(p_i, \sigma) = p_{i+1}$ if σ is the $i + 1$th character in $(\mu)_k$
  $\delta'(p_i, \sigma) = \delta(q_0, (\mu)_k \sigma))$ otherwise
  $\delta'(r_w, \sigma) = r_{w\sigma}$ iff $[w\sigma]_k < k^\nu$
  $\delta'(r_w, \sigma) = \delta(q_0, (\mu)_k w\sigma)$ otherwise.
- $\tau'(q) = \tau(q)$ for $q \in Q$
  $\tau'(p_i) = \tau(\delta(q_0, (\mu)_k[0, i]))$
  $\tau'(r_w) = x_{\mu k^\nu + [w]_k}$ iff $m \leq \mu k^\nu + [w]_k \leq n$ $\tau'(r_w) = \tau(\delta(q_0, (\mu)_k w))$

The resulting k-DFAO D' has a size of $||a||_k + 1 + log_k \lfloor \mu \rfloor + \sum_{i=1}^{\nu} k^i$. To make our proof easier, we will first prove the following claim:
Given any $w \in \Sigma_k^*$, only one of the following holds:

- $\delta'(p_0, w) = p_i$ if and only if there exists an i such that. $w = (\mu)_k[0, i]$
- $\delta'(p_0, w) = r_u$ if and only if there exist a $(u) < k^\nu$ with $w = (\mu)_k u$.
- $\delta'(p_0, w) = \delta(q_0, w)$ otherwise.

We will prove this claim via induction over $w \in \Sigma_k^*$:
Base case: $w = \epsilon$. We then have $\delta'(p_0, \epsilon) = p_0$ and $\epsilon = \mu[0, 0]$
Induction step: For the induction step we will make a case distinction on w:
Case 1aa: $\delta'(p_0, w) = p_i$ with $w = (\mu)_k[0, i]$ with $i < log\lfloor\mu\rfloor$ and σ is the $i+1$th character . This gives us $\delta'(p_0, w\sigma) = \delta'(p_i, \sigma) = p_{i+1}$, and $w\sigma = (\mu)_k[0, i + 1]$.
Case 1ab: $\delta'(p_0, w) = p_i$ with $w = (\mu)_k[0, i]$ with $i < log\lfloor\mu\rfloor$ and σ is **not** the $i + 1$th character. This gives us $\delta'(p_0, w\sigma) = \delta'(p_i, \sigma) = \delta(q_0, w\sigma)$.
Case 1b: $\delta'(p_0, w) = p_{\lfloor log_k(\mu)\rfloor}$ with $w = (\mu)_k$. This gives us $\delta'(p_0, w\sigma) = \delta'(p_{log\lfloor\mu\rfloor}, \sigma) = r_\sigma$ and $w\sigma = (\mu)_k\sigma$.
Case 2a: $\delta'(p_0, w) = r_u$ with $w = (\mu)_k u$ and $[w\sigma]_k < k^\nu$. This gives us $\delta'(p_0, w\sigma) = \delta'(r_u, \sigma) = r_{u\sigma}$ and we have $w\sigma = ((\mu)_k u)\sigma$.
Case 2b: $\delta'(p_0, w) = r_u$ with $w = (\mu)_k u$ and $[w\sigma]_k \geq k^\nu$. This gives us $\delta'(p_0, w\sigma) = \delta'(r_u, \sigma) = \delta(q_0, w\sigma)$. Since $w = (\mu)_k u$ there exists no i such that $w = (\mu)_k[0, i]$.
Case 3: $\delta'(p_0, w) = \delta(q_0, w)$. This will give us $\delta'(p_0, w\sigma) = \delta'(\delta(q_0, w), \sigma) = \delta(q_0, w\sigma)$.
Conclusion: we have now proven our claim through induction and are now able to finish up the original proof. We will do so using case distinction on $w \in \Sigma_k^*$ corresponding to the three cases in our claim:
Case 1: $w = (\mu)_k[0, i]$. This gives us $\tau'(\delta'(p_0, w)) = \tau'(p_i) = \tau(\delta(q_0, (\mu)_k[0, i])) = \tau(\delta(q_0, w))$.
Case 2a: $w = (\mu)_k u$ where $m \leq \mu k^\nu + [u]_k \leq n$. This gives us $\tau'(\delta'(p_0, w)) = \tau'(r_u) = x_{\mu k^\nu + [u]_k}$. And $\mu + k^\nu + [u]_k = [\mu u]_k = [w]_k$.
Case 2b: $w = (\mu)_k u$ where not $m \leq \mu k^\nu + [u]_k \leq n$. $\tau'(\delta'(p_0, w)) = \tau'(r_u) = \tau(\delta(q_0, (\mu)_k w$.
Case 3: w is anything but the possibilities listed above. this means that $m \leq [w]_k \leq n$ is not true. Which gives us $\tau'(\delta'(p_0, w)) = \tau(\delta(q_0, w))$.
Conclusion: Through case distinction and the usage of our claim we have shown that D' represents $||a[m, .., n \mapsto x_m, .., x_n]||_k$. □
If we create the k-DFAO of $\mathsf{thue}[28, 29, 30, 31 \mapsto a, b, c, d]$as described in our theorem:

We can construct a version of this for the reverse as well by using the k-DFAO constructed in Theorem 2 and then merging all untouched branches into the original k-DFAO. However if $m \leq k^{\lfloor log_k(n)\rfloor}$, there will be no minimizable branches.

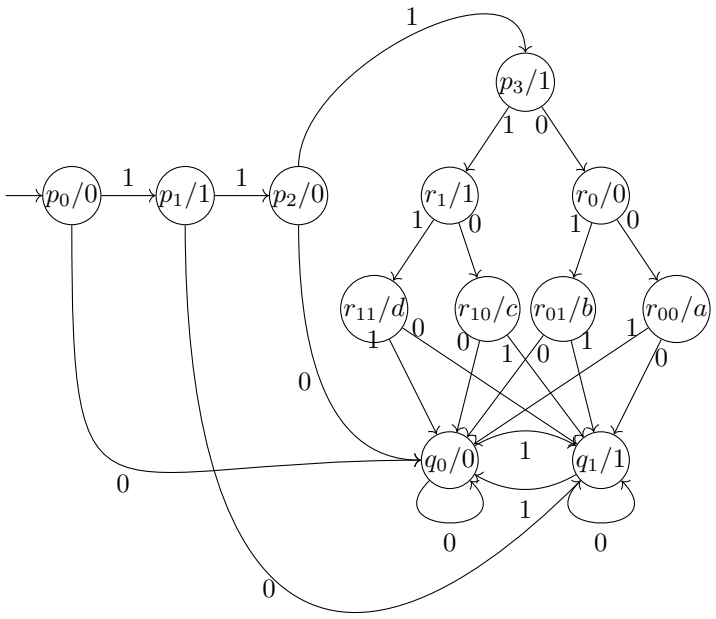

Figure 6: The k-DFAO of $\mathsf{thue}[28, 29, 30, 31 \mapsto a, b, c, d]$

3.4 Combining two arbitrary k-automatic sequences

The last local change to k-automatic sequences that we will take a look at is a variant of $a[0, .., n \mapsto x_0, .., x_n]$ in which $x_0, .., x_n$ can be seen as the first $n+1$ elements of a k-automatic sequence b, we define this transformation as follows:

Definition 18. *Given k-automatic sequences a, b, we define the sequence* $\mathsf{fuse}(b, n, a)$ *as follows*

$$\begin{aligned}\mathsf{fuse}(b, n, a)_i &= b_i \text{ iff } i \leq n \\ &= a_i \text{ otherwise}\end{aligned}$$

Theorem 5. *Given any two k-automatic sequences a, b and $n = k^p - 1$ for any $p \in \mathbb{N}$, we have $||\mathsf{fuse}(b, n, a)||_k \leq ||a||_k + (p+1)||a||_k||b||_k$ and $||\mathsf{fuse}(b, n, a)||_k^R \leq ||a||_k^R + (p+1)||a||_k^R||b||_k^R$.*

Proof. For k-automatic sequences a with k-DFAO $D_a = (Q_a, \Sigma_k, \delta_a, q_{a0}, \Gamma_a, \tau_a)$ and b with k-DFAO $D_b = (Q_b, \Sigma_k, \delta_b, q_{b0}, \Gamma_b, \tau_b)$, we can construct the k-DFAO $D = (Q_a \cup P, \Sigma_k, \delta, p_{q_{b0},0,q_{a0}}, \Gamma_a \cup \Gamma_b, \tau)$ as follows:

- $P = \{p_{q_b,i,q_a} | i \in [0,p], q_a \in Q_a, q_b \in Q_b\}$ and $Q_a \cap P = \emptyset$. We will abbreviate $p_{q_{b0},0,q_{a0}}$ as p_0.
- $\delta(q,\sigma) = \delta_a(q,\sigma)$ for $q \in Q_a$, $\sigma \in \Sigma_k$
  $\delta(p_{q_b,i,q_a},\sigma) = p_{\delta_b(q_b,\sigma),i+1,\delta_a(q_a,\sigma)}$ iff $i < p-1$
  $\delta(p_{q_b,i,q_a},\sigma) = \delta_a(q_a,\sigma)$ otherwise
- $\tau'(q) = \tau_a(q)$ for $q \in Q_b$.
  $\tau'(p_{q_b,i,q_a}) = \tau_b(q_b)$.

The constructed k-DFAO D consists of the original k-DFAO D_a, and a $(p+1) \times ||a|| \times ||b||$ cube of states, which helps us keep track of which state we would be at in D_a and D_b after i steps, respectively.
The constructed k-DFAO D has a size of $||a||_k + (p+1)||a||_k||b||_k$. And we will now prove that it does indeed represent $\mathsf{fuse}(b,n,a)$.
For any $w \in \Sigma_k^*$ if $[w]_k \leq n$ then $\delta(p_{q_{b0},0,q_{a0}}, w) = p_{q_b,i,q_a}$ with i being the length of w, $q_a = \delta_a(q_{a0}, w)$ and $\delta_b(q_{b0}, w)$. Otherwise $\delta(q_0, w) = \delta_a(q_{a0}, w)$.
We will prove this claim through induction over $w \in \Sigma_k^*$.
Base case: $w = \epsilon$. $[\epsilon]_k = 0 \leq n$ and $\delta(q_0, \epsilon) = q_{0,q_{a0},q_{b0}}$.
Induction step: Given a $\sigma \in \Sigma_k$ and any w for which the claim holds. We will make a case distinction on w.
Case 1a: $\delta(p_0, w) = p_{q_b,i,q_a}$ and $i < k^{p-1}$. This would give us $\delta(p_0.w\sigma) = \delta(p_{q_b,i,q_a}, \sigma) = p_{i+1,\delta(q_a,\sigma),\delta(q_b,\sigma)}$. Since we have $q_a = \delta(q_{a0}, w)$ and $q_b = \delta(q_{b0}, w)$ we also have $\delta_a(q_a,\sigma) = \delta_a(q_{a0}, w\sigma)$ and $\delta_b(q_b,\sigma) = \delta_b(q_{b0}, w\sigma)$.
Case 1b: $\delta(p_{q_b,i,q_a}, \sigma) = p_{i+1,\delta_a(q_a,\sigma),\delta_b(q_b,\sigma)}$ and $i \geq k^{p-1}$. This would give us $\delta(p_0, w\sigma) = \delta_a(q_a,\sigma) = \delta_a(q_{a0}, w\sigma)$ since $q_a = \delta_a(q_{a0}, w)$.
Case 2: $\delta(q_0, w) = \delta_a(q_{a0}, w)$. This would give us $\delta(q_0, w\sigma) = \delta(\delta_a(q_{a0}, w), \sigma) = \delta_a(q_{a0}, w\sigma)$.
Conclusion: Through case distinction and induction we have proven our claim.
Now with the claim proven we can easily finish up our proof.
For any $w \in \Sigma_k^*$, we have two possibilities:
Case 1: $[w]_k \leq n$. $\tau(\delta(q_0, w)) = \tau(q_{q_b,i,q_a}) = \tau_a(q_a)$ where $q_a = \delta_a(q_{a0}.w)$.
Case 2: $[w]_k > n$. $\tau(\delta(q_0, w)) = \tau(\delta_b(q_{b0}, w))$.
Conclusion: The constructed k-DFAO D represents $\mathsf{fuse}(a,n,b)$. □

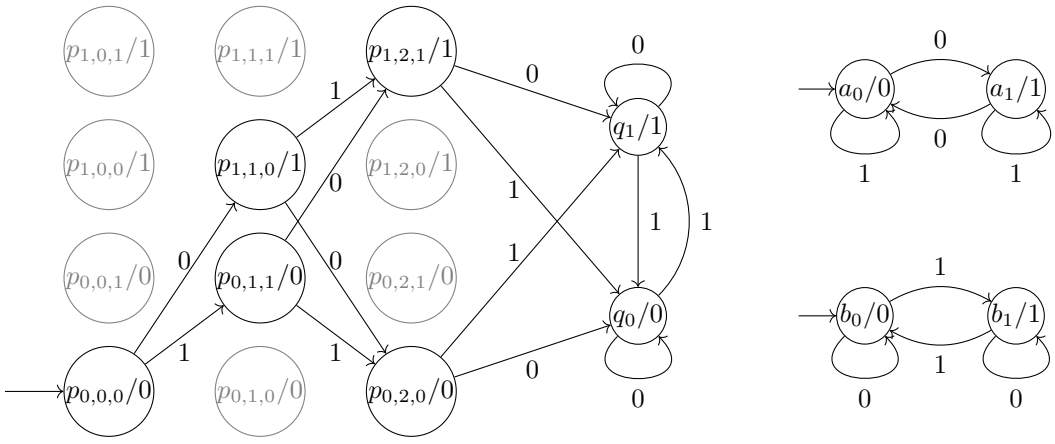

Figure 7: The k-DFAO of $\mathsf{fuse}(\mathsf{odd0}, 3, \mathsf{thue})$, unreachable states have been colored gray, and transitions from unreachable states are not drawn. On the right are the k-DFAO of thue (bottom) and $\mathsf{odd0}$ (top)

The proof for the reverse follows the exact same structure, except that it is right to left.
The constructed k-DFAO will very likely be minimizable. For example, if $Q_a = \{q_{a0}, q_{a1}\}$ then $P_{0,q_{a1},q_{b0}}$ will never be reachable. There are numerous other circumstances in which our k-DFAO can be further reduced, which mostly boils down to states being unreachable, however we will not discuss those here.
It is clear that given any $n = k^p - 1$, instead of using $a[0, ..., n \mapsto x_0, .., n]$, we can create an identical k-automatic sequence using $\mathsf{fuse}(b, n, a)$ in which b is a k-automatic sequence that for any $i \leq n$ has $b_i = x_i$. Depending on n, k and $||b||$, the upper bound given in Theorem 5 might thus be smaller than the one given in Theorem 2. Theorem 5 unfortunately only works if $n = k^p - 1$, of course we would like to be able to make a more generalised solution for any n, if we combine Theorem 5 and Theorem 4 we can construct a k-DFAO satisfying cases in which $n \neq k^p - 1$.

3.5 Optimizing fuse

The method used in Theorem 5 can be further improved if at least one of the k-DFAOs has the structure shown in Figure 8. We will refer to these k-automatic sequences as cyclic sequences.

Theorem 6. *Given any two k-automatic sequences a, b and $n = k^p - 1$. If a is cyclic, we have $||\mathsf{fuse}(b, n, a)||_k \leq ||a||_k + \lfloor \log_k(n) \rfloor ||b||_k$.*

Proof. Because of the specific structure of the k-DFAO of a, we don't have to separately keep track of what state in a we would be in after i transitions, instead we know that we would be in $q_{i \mod ||a||}$.

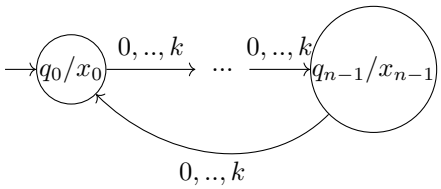

Figure 8: k-DFAO for sequence a with $a_0 = x_0$ and $a_i = x_{\lfloor log_k(i) \rfloor \mod n}$.

Given the k-DFAO $D = (Q \cap P, \Sigma_k, \delta, q_0, \Gamma_a \cap \Gamma_b, \tau)$ of $\mathsf{fuse}(b, n, a)$ as is defined in Theorem 5. We will show that given any $q_{i,q_j,q_b} \in P$ if $j \neq i \mod ||a||$ then q_{i,q_j,q_b} is unreachable; we wil do so using induction over i:
Base case: $i = 0$. We know from construction that the only state reachable with $i = 0$ is $q_{0,q_0,q_{b0}}$.
Induction Step: Given i at which only elements of $Dom = \{q_{i,q_{i \mod ||a||_k},q_b} | q_b \in Q_b\}$ are reachable. We have the image $\delta[Dom, \Sigma_k] =$
$\{q_{i+1,\delta_a(q_{i \mod ||a||_k},\sigma),\delta_b(q_b,\sigma)} | q_b \in Q_b, \sigma \in \Sigma_k\}$ in which $\delta(q_{(i \mod ||a||_k)}, \sigma) = q_{(i \mod ||a||_k)+1} = q_{(i+1 \mod ||a||_k)}$. Thus the hypothesis holds.
Conclusion: Through induction we have proven that for each $i \leq \lfloor \log_k(n) \rfloor$ only states of the form $q_{(i,q_{(i \mod ||a||_k)},q_b)}$ are reachable thus we can reduce the original k-DFAO D to a size of $||a||_k + \lfloor \log_k(n) \rfloor ||b||_k$. □
Note that a similar proof can be given for the reverse and cases in which b is cyclic.

4 Expressing DFAOs in SMT formulae

To show that our maximum bounds can not be further optimized, we will want to find examples of $(\mathbb{N})_k$- or $(\mathbb{N})_k^R$-minimal k-DFAOs of the exact size our theorems predict their maximum size to be.

We will try to find these examples with the use of a SAT solver. By representing k-DFAOs and their k-automatic sequences as SAT/SMT formulae, we can optimize over the variable n, which represents the smallest possible size of the k-DFAO. The SAT/SMT formula represents the question of whether a k-DFAO of size n exists with $\tau(\delta(q_0, (i)_k)) = a_i$.

Theorem 7. *If a SAT/SMT formula representing k-automatic sequence a with $n = x$ is unsatisfiable, then no k-DFAO exists of size x or smaller representing a.*
If the SAT/SMT formula representing k-automatic sequence a with $n = x$ is satisfiable, but the the SAT/SMT formula representing k-automatic sequence a with $n = x - 1$ is unsatisfiable, than the k-DFAO found with $n = x$ is minimal and we have $||a||_k = x$.

We will define our k-DFAO as follows:

- We can opt to either represent our states and input alphabet as finite sets, or as natural numbers.
- The transition function δ will be defined either as:
  $d : (Q \times \Sigma_k) \to Q$ or
  $d : (\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$
  If we opt to represent our states as natural numbers we will introduce the following constraint:

$$\bigwedge_{q=0}^{A} \bigwedge_{s \in \Sigma_k} (0 \leq d(q, s) < A)$$

- The output function τ will be defines either as:
  $t : Q \to \mathbb{N}$ or
  $t : \mathbb{N} \to \mathbb{N}$

We can then represent a specific element a_i in the k-automatic sequence a by stringing together i transition functions and one output formula. If we do this for a large portion of our sequence, we can say with quite some certainty whether our automata is correct or not. When using natural numbers for representation,a size of $n = 12$, and checking only the first 127 elements, our SMTLib formula would look something like this:

```
(benchmark smp.txt
:logic QF_UFLIA

:extrafuns (
(d Int Int Int)
(t Int Int)
)

:formula (and
    (>= (d 0 0) 0)
    (< (d 0 0) 11)
    (>= (d 0 1) 0)
    (< (d 0 1) 11)
      ...
    (>= (d 10 0) 0)
    (< (d 10 0) 11)
    (>= (d 10 1) 0)
    (< (d 10 1) 11)

    (= (t 0) 0)
    (= (t (d 0 1)) 1)
    (= (t (d (d 0 0) 1)) 1)
    (= (t (d (d 0 1) 1)) 0)
     ...
```

```
    (= (t (d (d (d (d (d (d 0 1) 0) 1) 1) 1) 1) 1)) 0)
))
```

4.1 Examples of minimal representations of k-automatic sequences for Theorem 2

One example of a minimal k-DFAO for sequences of the form $a[0,..,n \mapsto x_0,..,x_n]$ is the k-DFAO of k-automatic sequence $\mathsf{thue}[0,..,3 \mapsto 1,0,1,0]$:

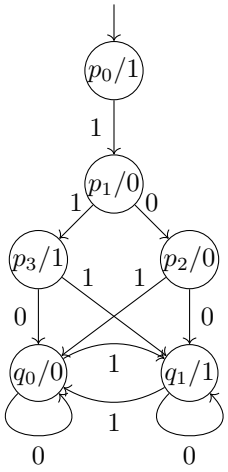

Figure 9: k-DFAO for $\mathsf{thue}[0,..,3 \mapsto 1,0,0,1]$

However the moment we use $n > 3$ we will discover that we can not in fact discover any configuration of $x_0,..,x_n$ that yields a minimal k-DFAO. This is because the states in P of which all of the transitions lead to states in Q, mimic the behaviour of a state in Q except for output. This means that we can only have $|a|_k(|\Gamma|-1)$ unique configurations, so for $k^{\lfloor log_k(n) \rfloor} > |a|_k(|\Gamma|-1)$ our method as described in Theorem 2 will always create a minimizable k-DFAO.

When looking at k-DFAOs, representing the reverse of k-automatic sequences, through our SAT-solver, we will see that for $n \geq 2$, we will always get a k-DFAO of a size smaller than what we would get with our construction technique. This is because we have states such as state p_{01} in Figure 4 that end on a 0. This means that there will be no $n \in \mathbb{N}$ that will end in this state, at most go through it. Thus the states representing a k-ary word ending on a 0 that does not connect to a state in P can be removed. This means that previous set bound of $||a[0,..,n \mapsto x_0,..,x_n]||_k^R \leq ||a||_k^R + \lceil \frac{kn}{k-1} \rceil$ can be further reduced.

Besides these states, the same problem exists as we had with the non-reverse, in which only $||a||_k^R(\#\Gamma - 1)$ unique configurations exist. Thus if $k^{\lfloor \log_k(n) \rfloor} > ||a||_k^R(\#\Gamma - 1)$ we can also reduce our k-DFAO by merging states.

4.2 Examples of $(\mathbb{N})_k$- and $(\mathbb{N})_k^R$-minimal k-DFAOs for Theorem 3

When we look at examples of $(\mathbb{N})_k$- and $(\mathbb{N})_k^R$-minimal k-DFAOs representing k-automatic sequences of the form $a[n \mapsto x]$, we can find plenty of examples. We will list a couple of them here:

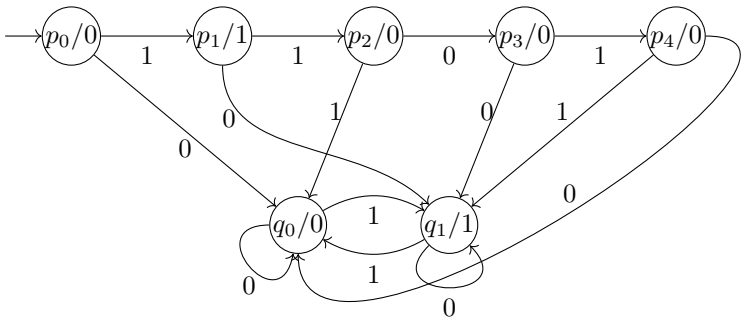

Figure 10: k-DFAO of $\mathsf{thue}[13 \mapsto 0]$

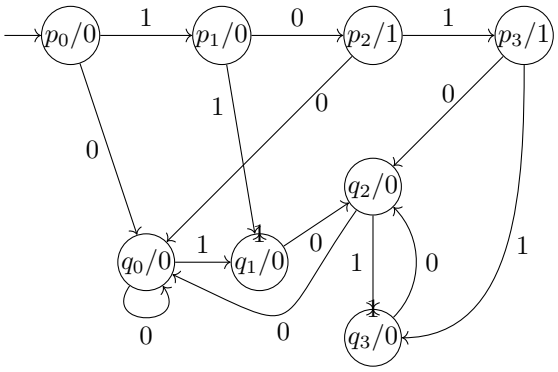

Figure 11: k-DFAO of $\mathsf{paper}[5 \mapsto 0]$

It is very easy to find examples of both $(\mathbb{N})_k$- and $(\mathbb{N})_k^R$-minimal k-DFAOs representing k-automatic sequences of the form $a[n \mapsto x]$ for varying values of k and n, with a size complexity we give as maximum in Theorem 3. It is thus safe to assume, that for nonspecific k-automatic sequences, the given upper bound cannot be further reduced.

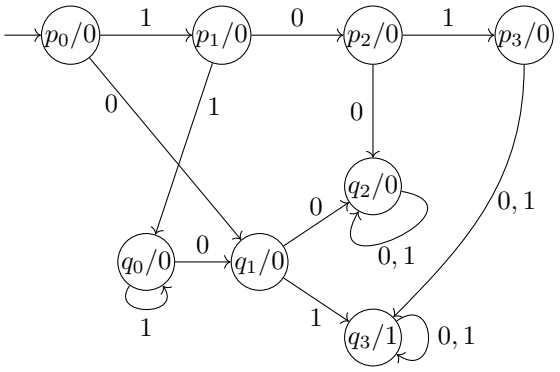

Figure 12: k-DFAO of the reverse $\mathsf{paper}[5 \mapsto 0]$ sequence

4.3 Exponential Time Complexity

Since basically every non-trivial k-DFAO as constructed by Theorem 5 is not minimal in $(\mathbb{N})_k$ or $(\mathbb{N})_k^R$, due to the fact that not every state is instantly reached, we would like to discuss examples of k-DFAOs that while minimizable are still closer to the smallest possible k-DFAO than the one we would construct using Theorem 2. Unfortunately a standard SATsolver such as Yices will not be able to find k-DFAOs of sizes of 12 or more in a timely fashion. This is because given a k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$ there exist a total of $|Q|^2 \times |\Sigma_k|$ different configurations for δ alone. This combined with the exponential nature of SAT solving, leaves us with a lot to be desired. One possibility for this would be using the construction techniques described in this paper to construct a k-DFAO and than use a minimization algorithm on the k-DFAO.

5 Minimization Algorithms for DFAOs

Our first step to creating a minimization algorithm for k-DFAOs is to adjust the standard minimization algorithm for DFAs, to a version for DFAOs. The algorithm that we will use, will require a DFA to have no unreachable states; for finding all unreachable states in a DFA, DFAO or k-DFAO representing a reverse k-automatic sequence we will use the following algorithm:

Algorithm 1: An algorithm that finds all reachable states in $\mathcal{O}(ns)$ time, with n being the amount of states and s the size of our input alphabet.

```
C ← {q_0};
F ← {q_0};
while F ≠ ∅ do
    Remove q from F;
    for σ ∈ Σ do
        if δ(q, σ) ∉ C then
            C ← C ∪ {δ(q, σ)};
            F ← F ∪ {δ(q, σ)};
        end
    end
end
```

When this algorithm terminates, C will contain all reachable states in Q, from this we can easily construct a version without unreachable states. The minimization algorithm we will use is the Hopcroft algorithm as described in [2], which does minimization on standard DFAs in $\mathcal{O}(n \log n)$ time complexity. The original Hopcroft minimization algorithm looks as follows:

Algorithm 2: The original Hopcroft minimization algorithm

```
Data: Any DFA D = (Q, Σ, δ, q_0, F) of which all states are reachable.
1  P ← {Q, Q \ F} ;
2  W ← P ;
3  while W ≠ ∅ do
4      Remove A from W;
5      for σ ∈ Σ do
6          X ← {q ∈ Q | δ(q, σ) ∈ A};
7          for Y ∈ P with Y ∩ X ≠ ∅ and Y \ X ≠ ∅ do
8              Replace Y in P with Y ∩ X and Y \ X;
9              if Y ∈ W then
10                 Replace Y in W with Y ∩ X and Y \ X ;
11             else
12                 Add Y ∩ X and Y \ X to W;
13             end
14         end
15     end
16 end
```

This algorithm sorts the states in Q into classes that might be equivalent. Or to be more precise, two states $p, q \in Q$ are only in different classes/partitions $P_i, P_j \in P$ if we already know that they are not equivalent. The algorithm then splits partitions into a smaller partition by finding a character σ, for which their transition would lead them to states that are known to be not equivalent. As the algorithm terminates, each element in P will be an equivalence class, from

which we can construct a new DFA of size $|P|$ by creating the transitions as follows: for states $P_i, P_j \in P$ we have $\delta(P_i, \sigma) = P_j$ if and only if there are states $p_i \in P_i, p_j \in P_j$ with $\delta(p_i, \sigma) = p_j$. The equivalence relation $\sim \subseteq Q \times Q$ on states in DFAs is defined as follows:

$$p \sim q \iff (p \in F \Leftrightarrow q \in F) \wedge \forall_{\sigma \in \Sigma}[\delta(p, \sigma) \sim \delta(q, \sigma)]$$

As per Theorem 1, the equivalence on states in DFAOs can also be inductively written as follows:

$$p \sim q \iff (\tau(p) = \tau(q)) \wedge \forall_{\sigma \in \Sigma}[\delta(p, \sigma) \sim \delta(q, \sigma)]$$

We will now alter the Hopcroft algorithm to reflect this new equivalence by only changing the initialization of P (and thus W) at line 1 to:

$P \leftarrow \{Q_\gamma = \{q \in Q | \tau(q) = \gamma\} | \gamma \in \Gamma\}$;

To prove that this small change, makes the algorithm work on DFAOs instead of DFAs, we will first prove the following:

Theorem 8. *As algorithm 2 with the alterations to line (1) terminates with DFAO $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$ as input, the following properties hold:*

- *Any state $q \in Q$ is an element of exactly one partition $Y \in P$.*
- $\forall_{p,p' \in P}[p \neq p' \Rightarrow \forall_{q \in p, q' \in p'}[q \nsim q']]$
- *Each set $Y \in P$ is an equivalence class, with the equivalence relation for states of a DFAO.*

Proof. We will provide the proof of this one item at a time. However first we will prove a couple of intermediate claims that will help us:
Claim 1: At line 15, the following property holds: $\forall_{Y \in P, \sigma \in \Sigma}[\exists_{q \in Y}[\delta(q, \sigma) \in A] \Rightarrow \forall_{q \in Y}[\delta(q, \sigma) \in A]]$. This property holds as in each iteration of the for loop at line 7, if a class Y contains both states that lead to a state in A on σ, namely $Y \cap X$, and states that don't $Y \setminus X$, then Y gets split up in those two halves.
Claim 2: $W \subseteq P$ at each iteration of the while loop on line 3. On initialisation this clearly holds, as $W = P$. On line 4, content is removed from W, however this does not affect our claim. On line 8, an element $Y \in P$ gets split into two halves, however these two halves are added to W, thus our claim holds.
Item 1: At the start of our algorithm we know that P each state $q \in Q$ exactly once, as each state has exactly one output $\tau(q)$. After that P only gets altered on line 8, in which a set Y is divided into two halves $Y \cap X$ and $Y \setminus X$. Which means that an element previously contained in Y will be in one of these smaller subsets, and only once a state is either in X or not in X.
Item 2: $\forall_{p,p' \in P}[p \neq p' \Rightarrow \forall_{q \in p, q' \in p'}[q \nsim q']]$. At the start of our algorithm we know that two elements contained in different subsets in P, will not be similar, as they will have different output. As P gets only altered on line 8, we need to show that for each element $q \in Y/capX$, and $q' \in Y \setminus X$, we have $q \nsim q'$: Since

$\delta(q', \sigma) \notin A$ and $\delta(q, \sigma) \in A$, they can not be equivalent, thus we have $q \nsim q'$ and our condition does not only hold on termination, it holds for each iteration for every loop in our algorithm, both for P, and as per claim 2 also for W.
Item 3: We will prove the last item using inversion and the previous claims. Since we know that the sets in P are separated by their output upon initialization, if the sets $Y \in P$ are not equivalence classes then the following claim should hold: $\exists_{Y,Y' \in P}[\exists_{q,q' \in Y, \sigma \in \Sigma}[\delta(q, \sigma) \in Y' \Rightarrow \delta(q', \sigma) \notin Y']]$. However as per claim 1, and the fact that each set $Y \in P$ has been in W at some point, we know that this property can not hold. Thus as per inversion P is a set of equivalence classes with the equivalence relation on states in a DFAO. □

With these properties proven, we will claim that we can generate an equivalent DFAO to the original DFAO, using the algorithm:

Theorem 9. *Given any DFAO $D = (Q, \Sigma, \delta, q_0, \Gamma, \tau)$and DFAO $D' = (P, \Sigma, \delta', p_0, \Gamma, \tau')$ with $\delta'(Y, \sigma) = Y'$ iff $\exists_{q \in Y}[\delta(q, \sigma) \in Y']$, $p_0 = Y \in Q$ iff $q_0 \in Y$, and $\tau'(Y) = \gamma$ iff $\exists_{q \in Y}[\tau(q) = \gamma]$ where P is the set of equivalence classes of DFAO D generated by algorithm 2. D and D' are equivalent DFAOs.*

Note that, because P is the set of equivalence classes, the properties for δ' and τ' hold for each state $q \in Y$ if it holds for at least one $q' \in Y$.
Proof. To prove this theorem, we will first prove this intermediate claim:
For all $w \in \Sigma^*$ we have $\delta(q_0, w) \in \delta'(p_0, w)$.
We will prove this claim using induction over $w \in \Sigma^*$:
Base Case: $w = \epsilon$. This gives us $\delta(q_0, \epsilon) = q_0 \in p_0 = \delta'(p_0, \epsilon)$.
Induction Step: $w' = w\sigma$ with our claim holding for w. This gives us $\delta'(p_0, w\sigma) = Y$ with $\delta(q, \sigma) \in Y$ and $q \in \delta'(p_0, w)$. As we know $\delta(q_0, w) \in \delta'(p_0, w)$, and thus $\delta(q_0, w\sigma) \in Y = \delta''(p_0, w\sigma)$.
Conclusion: With our claim proven, we can show that $\forall_{w \in \Sigma^*}[\tau(\delta(q_0, w)) = \tau'(\delta'(p_0, w))]$, as $\delta(q_0, w) = q \in \delta'(p_0, w)$ and thus $\tau(q) = \tau'(\delta'(p_0, w))$. □
Now that we know that our algorithm returns an equivalent DFAO with each state being distinct, we can prove that the DFAO constructed is actually minimal.

Theorem 10. *Given any DFAO $D' = (P, \Sigma, \delta', p_0, \Gamma, \tau')$ as constructed in theorem 9, that DFAO is minimal.*

Proof. This proof is identical to the proof for DFAs in [3]. However we will give a short summary of the proof. Given that a smaller equivalent DFAO D would exist, then there would be an equivalence mapping between the states of D and D'. However as D' has more states than D, at least two states in D' would be equivalent to the same state in D, and thus each other. However as each state in D' is distinct, this is impossible. □

5.1 Finding the minimal k-DFAO in $(\mathbb{N})_k$

Now that we have a minimization algorithm for DFAOs, we can start looking at using it to minimize k-DFAOs in $(\mathbb{N})_k$.
If we look at DFAOs as a mapping from each word $w \in \Sigma^*$ to a value in Γ, then k-DFAOs are mappings from either $(\mathbb{N})_k$ or $(\mathbb{N})_k^R$ to Γ. Since each word in $(\mathbb{N})_k$ is either the empty word ϵ or can be constructed as σw with any non-zero $\sigma \in \Sigma_k$ and $w \in \Sigma_k^*$, each k-DFAO also contains $k-1$ DFAOs mapping $w \in \Sigma_k^*$ to Γ. If we can find these minimal 'subDFAOs', then we can at least find a lower bound for $||a||_k$.

Theorem 11. *Given any k-automatic sequence a, a minimal DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$ and non-zero $\sigma \in \Sigma_k$ with $\tau(\delta(q_0, \sigma w)) = a_{[\sigma w]_k}$ for all $w \in \Sigma_k^*$. Then we have $||a||_k \geq |Q|$.*

Proof. We can prove this easily using contradiction. Given a k-DFAO $D' = (Q', \Sigma_k,$
$\delta', q_0', \Gamma, \tau')$ with $|Q'| < |Q|$, we can construct a DFAO equivalent to D by taking $\delta'(q_0', \sigma)$ as our initial state instead. However this contradicts that D is minimal, and thus it is impossible for a smaller k-DFAO to exist. □

With this theorem we can construct an algorithm for finding the smallest $(\mathbb{N})_k$-equivalent k-DFAO of a k-DFAO. One possibility would be to, for every non-zero $\sigma \in \Sigma_k$, find the smallest minimal DFAO with $\tau(\delta(q_0, \sigma w)) = a_{[\sigma w]_k}$ for all $w \in \Sigma_k^*$. And then put all of these minimal DFAOs together to get a DFAO equivalent to our k-DFAO, however this is very inefficient. Luckily for us, for the Hopcroft minimization algorithm, the initial state does not matter, as all it does is group states together into equivalence classes. First we will use the following algorithm to remove any unreachable states, and Figure out if one of the DFAOs contained in our k-DFAO contains the initial state q_0:

Algorithm 3: An algorithm that finds all reachable states in $\mathcal{O}(ns)$ time, and detects wether we can revisit q_0

```
C ← {δ(q0, σ) | σ ∈ Σk ∧ σ ≠ 0};
F ← {δ(q0, σ) | σ ∈ Σk ∧ σ ≠ 0};
while F ≠ ∅ do
    Remove q from F;
    for σ ∈ Σ do
        if δ(q, σ) ∉ C then
            C ← C ∪ {δ(q, σ)};
            F ← F ∪ {δ(q, σ)};
        end
    end
end
```

When this algorithm terminatesk, $\{q_0\} \cup C$ will be the collection of all reachable states in the k-DFAO. If $q_0 \in C$ then one of the subDFAOs contains q_0 thus the resulting minimal DFAO will also q_0. In this case we will just pass C

to the Hopcroft algorithm, from which we will generate our k-DFAO minimal in $(\mathbb{N})_k$.
If $q_0 \notin C$ then q_0 is not revisited and thus q_0 is not contained in one of the subDFAOs. We can then pass C to the Hopcroft algorithm, from which we can generate the minimal subDFAO D. We will now see if there is a state $Y_0 \in P$ that would work as our initial state, or if we have to add an initial state to D. We do this by checking whether a state $Y_0 \in P$ exists with $\tau'(Y_0) = \tau(q_0)$ and for all non-zero $\sigma \in \Sigma_k$ $\delta'(Y_0, \sigma) = Y_\sigma$ with $\delta(q_0, \sigma) \in Y_\sigma$. If no such state can be found we will add q_0 to our minimal DFAO D with $\tau'(q_0) = a_0$ and for all non-zero $\sigma \in \Sigma_k$ we have $\delta'(q_0, \sigma) = P$ with $\delta(q_0, \sigma) \in P$.

Theorem 12. *Given any k-DFAO $D_{initial} = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$, and k-DFAO D_{final} being the k-DFAO generated as described above. D_{final} is $(\mathbb{N})_k$-equivalent to $D_{initial}$, and is $(\mathbb{N})_k$-minimal.*

Proof. Given D_{initial}, we know that $D' = (\{q_0\} \cup C, ...)$ is equivalent, as only states $q \in Q$ with no word $w \in (\mathbb{N})_k$ with $\delta(q_0, w) = q$ are removed. We also know that for any non-zero $\sigma \in \Sigma_k$, $D_\sigma = (C, \Sigma_k, \delta, \delta(q_0, \sigma), \Gamma, \tau)$ is a subDFAO of D_{initial} with the property as described in theorem 11. Algorithm 2 for DFAOs, will then generate the minimal DFAO $D'' = (P, \Sigma_k, \delta'', \_, \Gamma, \tau'')$ equivalent to D_σ with $Y_\sigma \in P$ as initial state, where $\delta(q_0, \sigma) \in Y_\sigma$. This means that the k-DFAO $(\mathbb{N})_k$-equivalent to D_{initial} is at least as large as D''. We will now make a case distinction on whether q_0 was revisited in the initial k-DFAO.
Case 1. $q_0 \in C$. As q_0 is revisited, q_0 is part of one of the subDFAOs D_σ and thus also part of one of an equivalence class $Y_0 \in P$. This means that D'' is $(\mathbb{N})_k$-equivalent to D_{initial} with Y_0 as initial state. And as per Theorem 11 we know that no smaller $(\mathbb{N})_k$-equivalent k-DFAO can exist, thus D'' is minimal in $(\mathbb{N})_k$.
Case 2. $q_0 \notin C$. Now, we want to know whether a $Y \in P$ exists, with the following property: $\tau''(Y) = \tau(q_0)$ and for each non-zero $\sigma \in \Sigma_k$ we have $\delta(Y, \sigma) = Y_\sigma$. If it does, D'' with Y as our initial state, will be $(\mathbb{N})_k$-equivalent to D_{initial}. Since D'' is our minimal subDFAO, this k-DFAO is also minimal.
If no such Y exists however, we will add a state q_0'' with $\delta''(q_0'', \sigma) = Y_\sigma$ for each non-zero $\sigma \in \Sigma$ and $\tau''(q_0'') = \tau(q_0)$, to D'' as our initial state. It is clear that this k-DFAO is $(\mathbb{N})_k$-equivalent to D_{initial}, however we will need to show that it is also minimal in $(\mathbb{N})_k$. As we know that D'' is already minimal, it will be sufficient that none of its states would be a suitable initial state, and thus at least 1 state has to be added to make it $(\mathbb{N})_k$-equivalent to D_{initial}. Each state Y_σ is unique, as if another state Y'_σ with the same property woulkd exist, they would be equivalent, which would mean that D'' is not minimal. Thus our initial state needs to have the property that Y should have; as no such state exists, we will have to add an initial state. □

5.2 Minimization of large construct k-DFAOs

Now that we have an algorithm to find a $(\mathbb{N})_k$- or $(\mathbb{N})_k^R$-minimal k-DFAO with a smaller time complexity than our SATsolver, we can try to construct large, possibly not minimal, k-DFAOs and try to find their respective $(\mathbb{N})_k$- or $(\mathbb{N})_k^R$-minimal equivalent. Namely we will take a look at a large k-DFAO that we would construct using our fuse approach as described in theorem 5, and then run these through our algorithm. When constructing a k-DFAO representing $\mathsf{fuse}(\mathsf{thue}, 511, \mathsf{paper})$, we end up with a k-DFAO with 74 states. We already know this can be reduced, due to the fact that we will always have unreachable states using the prescribed construction method. Using an implementation of the algorithm as described in 5.1 in python, we get the minimized k-DFAO within seconds. The $(\mathbb{N})_k$-minimal k-DFAO as found by our algorithm, has a size of 39 states, something that would have taken forever using a SAT/SMT-solver approach.

In Figure 13 we have plotted out two different fusecombinations. In the upper graph, we see that the complexity of our k-automatic sequence scales linearly with p for $p \geq 7$. When the complexity starts scaling linearly it also becomes parallel to upper bound given by out theorem, implying that for at least some combinations the upper bound can only be improved by a constant value. In the bottom graph, the complexity and upper bound are not parallel, which means that for some combinations the upper bound can be improved.

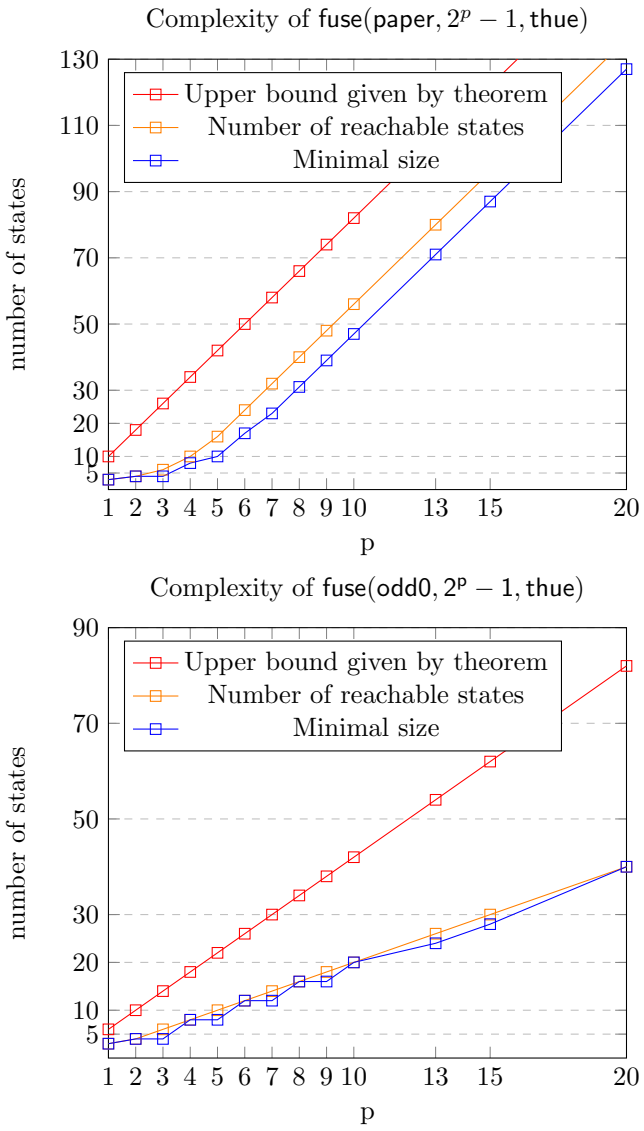

Figure 13: Graph containing various complexities of different fuse configurations

6 Translating k-DFAOs to higher bases

Definition 19. *Given any $w \in \Sigma_{k^d}^*$, with $d \in \mathbb{N}_{>0}$, we define w/d as the shorthand style of writing $([w]_{k^d})_k$.*
We define $\lceil w/d \rceil$ as w/d but with the minimum amount of 0's added to the left, to make its length divisible by d, e.g. $21 \in \Sigma_4/2 = 110$, while $\lceil 21/2 \rceil = 0110$.
As the difference between w/d and $\lceil w/d \rceil$ is only dependent on the left-most character, they have the following properties:

- $\forall_{w,w' \in \Sigma_{k^d}^*}, \lceil w/d \rceil \lceil w'/d \rceil = \lceil ww'/d \rceil$.
- $\forall_{w,w' \in \Sigma_{k^d}^*}, w \neq \epsilon \implies (w/d)\lceil w'/d \rceil = (ww')/d$.
- $\forall_{\sigma \in \Sigma_{k^d}}, \sigma \neq 0 \implies \lceil \sigma/d \rceil = 0^{d-|\sigma/d|}(\sigma/d)$

Theorem 13. *Given a k-automatic sequence a with k-DFAOs in k, and $d \in \mathbb{N}_{>0}$. We have $||a||_{k^d} \leq 1 + ||a||_k$ and we have $||a||_{k^d}^R \geq d||a||_k^R$.*

Informal proof. We can rewrite every word in Σ_{k^d} as a word in Σ_k, that is d times as large. This means that our new transition function just has to mimic the d transitions it would normally take. However when we have a word such as 12 in $k = 4$, and translate it to $|1|10|$ (with vertical lines added for clarity,) in $k = 2$, we can see that the leftmost letter does not always become d times as large. When we parse a word from left to right, we only have to keep this in mind for our first transition (see Figure 6). However, when parsing in reverse direction, we will have to add intermediate states, so we can keep track of how many 0's we will have to add, if this is not the last letter.
Proof. Given k-automatic sequence a with k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$ with size $||a||_k$, we cconstruct the k^d-DFAO $D' = (s \cup Q, \Sigma_{k^d}, \delta', s, \Gamma, \tau')$ as follows:

- $s \notin Q$.
- $\delta'(s, \sigma) = \delta(q_0, \sigma/d)$.
  $\delta'(q, \sigma) = \delta(q, \lceil \sigma/d \rceil)$.
- $\tau'(s) = \tau(q_0)$.
  $\tau'(q) = \tau(q)$.

This DFAO has a size of $1 + ||a||_k$.
We have $\tau'(\delta'(s, \epsilon)) = \tau(\delta(q_0, \epsilon))$. We will now prove that for all $w \in \Sigma_{k^d}/\{\epsilon\}$ we have $\delta'(s, w) = \delta(q_0, w/d)$ by induction over w:
Base case: $w = \sigma \in \Sigma_{k^d}$. $\delta'(s, \sigma) = \delta(q_0, \sigma/d)$.
Induction step: Given any $w \, in \Sigma_{k^d}^+$, we have: $\delta'(s, w\sigma) = \delta'(\delta(q_0, w/d), \sigma)$**Co** $= \delta(\delta(q_0, w/d), \lceil \sigma/d \rceil) = \delta(q_0, w\sigma/d)$
Conclusion: With our proof for the empty word, and our induction proof for any $w \in \Sigma_{k^d}^+$, we can conclude that the constructed k-DFAO D' represents a.
Proof for Reverse. Given k-automatic sequence a with k-DFAO $D = (Q, \Sigma_k, \delta, q_0, \Gamma, \tau)$ with size $||a||_k^R$. We construct the k^d-DFAO $D' = (P, \Sigma_{k^d}, \delta', p_{0,q_0}, \Gamma, \tau')$ as follows:

- $P = \{p_{i,q} | i \in [0, d-1], q \in Q\}$.
- $\delta'(p_{i,q}, 0) = p_{0,\delta(q,0^{d+i})}$.
  $\delta'(p_{i,q}, \sigma) = p_{j,q'}$ where $j = d - |\sigma/d|$ and $q' = \delta(q, \sigma/d0^i)$. (Note that $0^0 = \epsilon$.)
- $\tau'(p_{i,q}) = \tau(q)$.

The resulting k-DFAO has $d \times |Q|$ states. We will now show that for any $w \in \Sigma^*_{k^d}$ we have $\tau'(\delta'(p_{0,q_0}, w) = \tau(\delta(q_0, ([w]_{k^d})_k))$, by induction over w:
Case ϵ: $w = \epsilon$.

$$\begin{aligned}
\tau'(\delta'(p_{0,q_0}, \epsilon) &= \tau(\delta(q_0, ([\epsilon]_{k^d})_k)) \\
\tau'(\delta'(p_{0,q_0}, \epsilon) &= \tau(\delta(q_0, \epsilon)) \\
\tau'(p_{0,q_0}) &= \tau(q_0) \\
\tau(q_0) &= \tau(q_0)
\end{aligned}$$

We will claim that: Given any word $\sigma w \in \Sigma^+_{k^d}$ we have $\delta'(p_{0,q_0}, \sigma w) = p_{j,q'}$ for which we have $j = d - |\sigma/d|$, and $q' = \delta(\sigma w/d)$. And given any word $0w \in \Sigma_{k^d}$, we have $\delta'(p_{0,q_0}, 0w) = p_{0,q'}$ for which we have $q' = \delta(q_0, 0^d\lceil w/d \rceil)$. We will prove this claim through induction over $\sigma w \in \Sigma^+_{k^d}$:
Base case 0: $\sigma w = 0$. This gives us $\delta'(q_{0,q_0}, 0) = p_{0,q'}$ with $q' = \delta(q_0, 0^d) = \delta(q_0, 0^d\lceil \epsilon/d \rceil)$.
Base case σ: $\sigma w = \sigma$, with $\sigma \neq 0$. This gives us $\delta'(p_{0,q_0}, \sigma) = p_{j,q'}$ where $q' = \delta(q_0, \sigma/d)$.
Induction step 0: We have $0w$ with $\delta'(p_{0,q_0}, 0w) = p_{0,q}$ with $q = \delta(q_0, 0^d\lceil w/d \rceil)$. We have $\delta'(p_{0.q_0}, 00w) = p_{0,q'}$ with $q' = \delta(\delta(q_0, 0^d\lceil w/d \rceil), 0^d) = \delta(q_0, 0^d0^d\lceil w/d \rceil)$. For any other $\sigma \neq 0$, we have $\delta'(p_{0,q_0}, \sigma 0w) = \delta(p_{0,q}.\sigma) = p_{i,q'}$, with $i = d - |\sigma/d|$ and $q' = \delta(q, \sigma/d) = \delta(q_0, (\sigma/d)0^d\lceil w/d \rceil) = \delta(q_0, (\sigma 0w)/d)$.
Induction step σ: We have σw with $\delta'(p_{0,q_0}, \sigma w) = p_{i,q}$ with $i = d - |\sigma/d|$, and $q = \delta(q_0, (\sigma w)/d)$. This gives us:
$\delta'(p_{0,q_0}, 0\sigma w) = p_{0,q'}$ and $q' = \delta(q, 0^{d+i}) = \delta(q_0, 0^{d+i}(\sigma w/d)) = \delta(q_0, 0^d\lceil 0w/d \rceil)$. And for any other $\sigma' \neq 0$, we have $\delta'(p_{0,q_0}, \sigma'\sigma w) = p_{j,q'}$ with $j = d - |\sigma'/d|$ and $q' = \delta(q, (\sigma/d)0^i) = \delta(q_0, (sigma'/d)0^{d-|\sigma/d|}(\sigma w)/d) = \delta(q_0, (\sigma'\sigma w)/d)$.
Conclusion: Since we have $\tau'(p_{i,q}) = \tau(q)$. The constructed DFAO D' represents the k-automatic sequence a. □

7 Conclusion and Future Work

We have shown that smaller upper bounds exist for finite changes to k-automatic sequences than the upper bounds we would have by using a combination of removing and adding the left-most element. To be more precise we have found the following upper bound for changing the first $n+1$ elements: $||a[0, .., n \mapsto x_0, .., x_n]||_k \leq ||a||_k + 1 + n$ which is smaller than the bound obtained by repeatedly removing and adding elements as per [5]: $||a[0, .., n \mapsto x_0, .., x_n]||_k \leq$

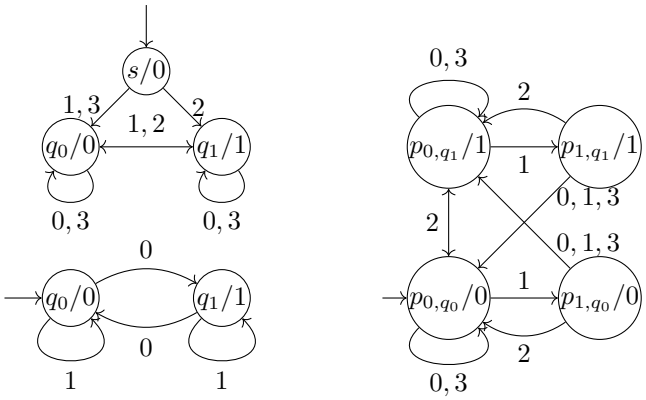

Figure 14: A list of different minimal DFAOs; the original 2-DFAO in the bottom-left, its 4-DFAO counterpart in the top-left, and its reverse 4-DFAO on the right

$k^{2(n+1)}||a||_k$.
Similarly for the reverse we have shown that changing the first $n+1$ elements of any k-automatic sequence gives us the following upper bound: $||a[0,..,n \mapsto x_0,..,x_n]||_k^R \leq ||a||_k^R + 1 + \lceil \frac{nk}{k-1} \rceil$ which is also smaller than the exponential bound obtained by repeatedly removing and adding elements as per [5].
We have also given further optimized bounds for the instance in which only the element a_n of an arbitrary k-automatic sequence a is changed: $||a[n \mapsto x]||_k \leq ||a||_k + 1 + \lfloor \log_k(n) \rfloor$ and $||a[n \mapsto x]||_k^R \leq ||a||_k^R + 1 + \lfloor \log_k(n) \rfloor$ for the reverse. Both for which we have shown examples of thue and paper in which the given upper bound is the exact complexity of the k-automatic sequence using a SAT/SMT solver. Which leads us to believe that this upper bound can not be reduced.
Optimized bounds where also obtained for the instance in which only element a_m up to and including a_n are changed, and for the fuse instance. In which the first $n+1$ elements of a are not changed to arbitrary values, but to the first $n+1$ elements of a different k-automatic sequence b. However we have also shown that this bound can still be improved upon as was shown with the examples in Figure 13.
Throughout the paper we have used a SAT/SMT solver to show that automata can not be further improved upon, however the exponential nature of SAT/SMT solving has made it unfeasible to do this for large automata. Thus we have given an algorithm for the minimization of DFAOs and an algorithm that uses this to minimize k-DFAOs in $(\mathbb{N})_k$ in polynomial time. However no such algorithm was discovered for the minimization of k-DFAOs in $(\mathbb{N})_k^R$. And the question remains whether a polynomial algorithm exists for DFAOS in $(\mathbb{N})_k^R$ or any other arbitrary sub-language $L \subseteq \Sigma_k^*$.
In the final section we have given an upper bound for translating known k-

DFAOs to k^d-DFAOs, showing that using a larger language can in fact cause the complexity to increase as is shown by the minimal k-DFAOs in Figure 14.

A Appendix

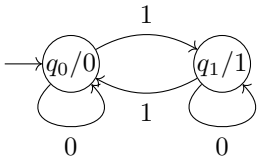

Figure 15: k-DFAO for sequence thue

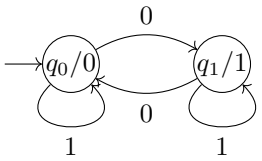

Figure 16: k-DFAO for sequence odd0

A python implementation of the pseudocode in this paper can be found at: `https://github.com/FlipvanSpaendonck/k-DFAORedux`

References

[1] Jean-Paul Allouche and Jeffrey Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, 2003.

[2] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189 – 196. Academic Press, 1971.

[3] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[4] Hans Zantema. Complexity of automatic sequences. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications*, pages 260–271, Cham, 2020. Springer International Publishing.

[5] Hans Zantema and Wieb Bosma. Complexity of automatic sequences, awaiting publishing. *Elsevier*, 2020.