Minimal Synthesis of String To String Functions From Examples

Jad Hamza and Viktor Kunčak

LARA, EPFL, Switzerland, firstname.lastname@epfl.ch

Abstract. We study the problem of synthesizing string to string transformations from a set of input/output examples. The transformations we consider are expressed using a particular class of transducers: functional non-deterministic Mealy machines (f-NDMM). These are machines that read input letters one at a time, and output one letter at each step. The *functionality* constraint ensures that, even though the machine is locally non-deterministic, each input string is mapped to exactly one output string by the transducer.
We suggest that, given a set of input/output examples, the smallest f-NDMM consistent with the examples is a good candidate for the transformation the user was expecting. We therefore study the problem of, given a set of examples, finding a minimal f-NDMM consistent with the examples and satisfying the functionality and totality constraints mentioned above.
We prove that, in general, the decision problem corresponding to that question is NP-complete, and we provide several NP-hardness proofs that show the hardness of multiple variants of the problem.
Finally, we propose an algorithm for finding the minimal f-NDMM consistent with input/output examples, that uses a reduction to SMT solvers. We implemented the algorithm, and used it to evaluate the likelihood that the minimal f-NDMM indeed corresponds to the transformation expected by the user.

1 Introduction

Programming by examples is a form of program synthesis that enables users to create programs by presenting input/output examples. In this paper, we analyze the problem of synthesizing string-to-string transformations from examples.

We consider string transformations that can be represented by finite-state automata, called *functional non-deterministic Mealy machines* (f-NDMM) [17]. f-NDMMs output one letter for each input letter which is read. Non-determinism refers to the fact that f-NDMMs are allowed to have two outgoing transitions from the same state labeled by the same input, while functionality ensures that overall, one input string is mapped to at most one output string. Moreover, if every input string has a corresponding output string, the automaton is called *total*.

Synthesizing an arbitrary total f-NDMM consistent with input/output examples can be solved in polynomial time, by having the f-NDMM return a default

string for the inputs which are not specified in the example. The issue with this basic approach is that the generated automaton might not be what the user had in mind when giving the input/output examples. In other words, input/output examples are not a complete specification, and are ambiguous.

As one of the simplest and robust criteria to rank possible solutions, we propose to synthesize a *minimal* automaton consistent with given input/output examples. For sufficiently long input/output descriptions, the requirement of minimality then forces the automaton to generalize from input/output examples. This rationale is analogous to motivation for Syntax-Guided Synthesis [2]. In our case we use automata minimality as a somewhat application-agnostic criterion. Furthermore, we can in principle leverage the insights from automata theory to improve the synthesis algorithm. Therefore, it is interesting to understand the precise computational complexity of such synthesis problems and to identify directions for promising synthesis approaches. This is the objective of our paper.

Complexity. We prove that the synthesis of minimal automata is in NP, by showing that for a given set of input-output examples E there always exist an f-NDMM consistent with E whose number of states is linear with respect to the size of E. Furthermore, we show how to check in deterministic polynomial time whether a given DFA is a total f-NDMM consitent with E. An NP procedure can iterate for i from 1 to the aforementioned bound, guess a DFA of size i, and check that it is a total f-NDMM consistent with the input/output examples.

We also consider the associated decision problem, which asks, given a set of input/output examples, and a *target* number of states k, whether there exists a total f-NDMM consistent with the examples and which has at most k states. We prove that this problem is NP-hard.

We give three distinct reductions, that apply for different variants of the problem. First, we show that the problem is NP-hard when the target number of states is fixed to 3 (but the input alphabet is part of the problem description). Second, we show that the decision problem is NP-hard when the input and output alphabets are fixed (but the target number of states is part of the problem description).

Third, we study a variant of the problem for *layered* automata for bitvectors, that recognize only words of some fixed length. The name *layered* comes from the fact that their states can be organized into layers that recognize only words of a certain length. We prove that the problem is still NP-hard in that setting, despite the fact that these automata have no cycles.

Algorithm. We provide a reduction to the satisfiability of a logical formula. We implement our reduction, and link it to the Z3 SMT solver. We evaluate our tool and show it can successfully recover simple relations on strings from not too many examples (but scales to many examples as well). We also evaluate the ability of our algorithm to recover a random automaton from a sample set of input-output examples. Our experiments suggest that it is better to give a large number of small examples, rather than a small number of large examples. Moreover,

to improve the chance that our algorithm finds a particular automaton, the examples given should generally be at least as long as the number of states.

Contributions of this paper are the following:

- NP-hardness proofs for the decision problem (Sections 5 and 6),
- Proof that the minimization problem can be solved in NP (Section 7),
- A reduction from the minimization problem to a logical formula that can be handled by SMT solvers (Section 8),
- An implementation of this reduction and experiments that evaluate the likelihood that minimization finds the automaton the user has in mind (Section 9).

Some proofs are deferred to the long version [14].

Note. A preliminary version of this paper, using a different encoding into SMT constraints for the synthesis algorithm, was presented at the SYNT 2018 workshop, without a proceedings entry. SYNT explicitly permits subsequent publication of such papers. Moreover, the present encoding into SMT constraints uses only quantifier-free linear integer arithmetic and is new to this submission.

2 Notation

An *alphabet* Σ is a non-empty finite set of *symbols*. Given a natural number $n \in \mathbb{N}$, we denote by Σ^n the set of sequences (or words) of n symbols of Σ. We denote by Σ^* the set of finite sequences $\bigcup_{n \geq 0} \Sigma^n$. For $u \in \Sigma^*$, $|u|$ denotes the length of the sequence u. A set of words is called a *language*.

A *non-deterministic finite automaton (NFA)* A is a tuple $(\Sigma, Q, q_{init}, \delta, F)$ where Σ is an alphabet, Q is the finite set of states, $q_{init} \in Q$ is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition function, and $F \subseteq Q$ is the set of *accepting* states. We denote by $\mathcal{L}(A)$ the language accepted by A, i.e. the set of words for which there exists an *accepting run* in A. By an abuse of notation, the set $\mathcal{L}(A)$ is sometimes denoted by A.

An NFA A is *unambiguous* (denoted *UFA*) if every word in Σ^* has at most one accepting run in A. An NFA is *deterministic* (denoted *DFA*) if for every $q_1 \in Q$, $a \in \Sigma$, there exists a unique $q_2 \in Q$ such that $(q_1, a, q_2) \in \delta$. The *size* of an NFA A is its number of states, and is denoted $|A|$.

Let Σ and Γ be two alphabets. For $u \in \Sigma^n$ and $v \in \Gamma^n$ where $u = u_1 \dots u_n$, $v = v_1 \dots v_n$, we denote by $u * v$ the sequence in $(\Sigma \times \Gamma)^n$ where $u * v = (u_1, v_1) \dots (u_n, v_n)$. Note that the operator $*$ is well defined only when $|u| = |v|$.

Given two words $u, v \in \Sigma^*$, we denote by $u \preceq_p v$ the fact that u is a prefix of v. Moreover, *Prefixes*(v) denotes the set of prefixes of v, that is *Prefixes*$(v) = \{u \mid u \preceq_p v\}$.

3 Functional Non-Deterministic Mealy Machines

We consider two alphabets, an *input alphabet* Σ and an *output alphabet* Γ. A *functional non-deterministic Mealy machine (f-NDMM)* is a DFA A over $\Sigma \times \Gamma$ satisfying: for all $u \in \Sigma^*$, $v_1, v_2 \in \Gamma^*$ where $|u| = |v_1| = |v_2|$, if $u * v_1 \in \mathcal{L}(A)$ and $u * v_2 \in \mathcal{L}(A)$, then $v_1 = v_2$.

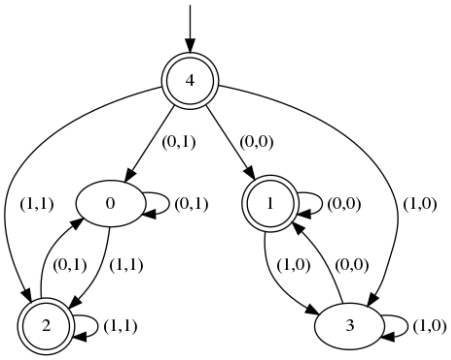

Fig. 1. An automaton that overwrites an input string with 0's or 1's depending on whether the last letter of the input is a 0 or 1.

Remark 1. Note here that we model f-NDMMs with *deterministic* finite automata. The determinism refers to the fact given a state, an input letter and an output letter, there is at most one outgoing transition labeled by those letters. On the other hand, the non-determinism in the f-NDMM refers to the fact that given one state and one input letter, there might be multiple outgoing transitions, each one labeled with a distinct output letter.

Example 1. Figure 1 shows a f-NDMM that outputs a sequence of 0's or a sequence of 1, depending on whether the last letter of the input is a 0 or a 1. Input letters are written on the left-hand-side of the pair, while output letters are on the right-hand-side.

Non-determinism is used in the initial state 4, to guess whether the last letter of the input is a 0 or a 1. In the states 0 and 2, the automaton expects the last letter to be a 1, while in the states 1 and 3, it expects the last letter to be a 0. The sink state is omitted for readability (e.g. reading a 1 and outputting a 1 in state 3 is not allowed).

Remark 2. This example illustrates the higher expressive power of f-NDMMs compared to deterministic Mealy machines, which cannot express this transformation. On the other hand, this transformation can be expressed using more

expressive deterministic transducers, such as transducers with look-ahead (that are able to take decisions based by seeking ahead in the input word) or two-way transducers (which are allowed to read the input word multiple times, back and forth).

Due to the functionality restriction described above, an f-NDMM A defines a partial function $\bar{A} \subseteq \Sigma^* \times \Gamma^*$, which is defined for $u \in \Sigma^*$ only when there exists $v \in \Gamma^*$ such that $u * v \in \mathcal{L}(A)$. This unique word v is denoted by $A(u)$. An f-NDMM A is called *total* if the partial function $\bar{A}$ is total. For a set $E \subseteq \Sigma^* \times \Gamma^*$ we say that an f-NDMM A is consistent with E if $E \subseteq \bar{A}$.

Problem 1. Let $E \subseteq (\Sigma \times \Gamma)^*$ be finite a set of input/output examples.
Find a total f-NDMM, consistent with E (if it exists), whose size is minimal (among all total f-NDMMs consistent with E).

We also investigate the following corresponding decision problem.

Problem 2. Let $E \subseteq (\Sigma \times \Gamma)^*$ be a set of input/output examples, and let $n \in \mathbb{N}$.
Does there exist a total f-NDMM, consistent with E, with size at most n?

When stating complexity results, we consider that the size of the problem is the sum of the sizes of each word in E, plus the size of n. Our hardness result hold even when n is represented in unary, while our proofs that Problems 1 and 2 belong to NP hold even when n is represented in binary.

3.1 Summary of the Complexity Results

Table 1 summarizes the complexity results proved in this paper. As far as we know, the problem is open when the input alphabet has size one, i.e. $|\Sigma| = 1$. On the other hand, when $|\Gamma| = 1$, the problem becomes trivial as the minimal total f-NDMM consistent with given input/output examples always has a single state with a self-loop.

Layered f-NDMMs are defined in Section 6.2, and are f-NDMMs that only recognize words of some particular length. Even in that setting, the problem is NP-complete.

Problem	Layered f-NDMMs	f-NDMMs
Problem 2	NP-complete	NP-complete
With $\|\Gamma\| = 2$, $n = 3$, $\|E\| = 1$	$O(1)$ (Remark 4)	NP-complete (Sect. 5)
With $\|\Sigma\| = 3$, $\|\Gamma\| = 2$	NP-complete (Sect. 6.2)	NP-complete (Sect. 6.1)
With $\|\Sigma\| = 3$, $\|\Gamma\| = 2$, $\|E\| = 1$	$O(1)$ (Remark 4)	NP-complete (Sect. 6.1)
When Σ, Γ and n are fixed	in P (Remark 3)	in P (Remark 3)

Table 1. Summary of the complexity results

4 Preliminaries for the NP-hardness proofs

In Sections 5, 6.1, and 6.2, we prove NP hardness results for Problem 2 and variants. These hardness results carry directly over to Problem 1. Indeed, any algorithm for solving Problem 1 can be adapted to solve Problem 2.

Our proofs rely on reductions from a variant of the boolean satisfiability problem (SAT), called One-In-Three Positive SAT.

Problem 3 (One-In-Three Positive SAT). Given a set of variables V and a set of clauses $C \subseteq V^3$, does there exist an assignment $f : V \to \{\bot, \top\}$ such that for each $(x, y, z) \in C$, exactly one variable out of x, y, z, evaluates to $\top$ through f.

In all reductions, our goal is to build from an instance φ of One-In-Three Positive SAT a set of input/output examples such that φ is satisfiable if and only if there exists a total f-NDMM consistent with the examples (and satisfying the constraints of the minimization problem at hand).

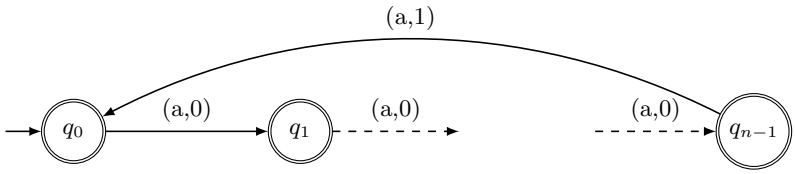

Fig. 2. The form of automata that have an $(a, 0, 1)$-loop.

Our strategy for these reductions is to give input/output examples that constrain the shape of *any* total f-NDMM consistent with these examples. We give input/output examples that ensure that any total f-NDMM consistent with the examples must have certain transitions, and cannot have certain other transitions.

For example, in Sections 5 and 6.1, we provide input/output examples that restrict the shape of any solution to be of the form given in Figure 2. Then, knowing that any solution must have this shape, we give additional examples that correspond to our encoding of φ.

We first give a formal definition for automata that are of the shape of the automaton given in Figure 2.

Definition 1. *Let $A = (\Sigma \times \Gamma, Q, q_{init}, \delta, F)$ be an f-NDMM with $n \in \mathbb{N}$ states, $n \geq 1$. We say that A* has an $(a, 0, 1)$-loop *if $a \in \Sigma$, and $0, 1 \in \Gamma$, $0 \neq 1$, and the states Q of A can be ordered in a sequence $q_0, \ldots, q_{n-1}$ such that:*

- $q_{init} = q_0$,
- *for every* $0 \leq i < n - 1$, $(q_i, (a, 0), q_{i+1}) \in \delta$,
- $(q_{n-1}, (a, 1), q_0) \in \delta$,
- $F = Q$,

– *there are no transitions in δ labeled with letter a other than the ones mentioned above.*

The following lemma, used in Theorems 1 and 2, shows that we can give an input/output example that forces automata to have an $(a, 0, 1)$-loop. The idea is to give a long example that can only be recognized if the total f-NDMM has an $(a, 0, 1)$-loop.

Lemma 1. *Let $A = (\Sigma \times \Gamma, Q, q_{init}, \delta, F)$ be a total f-NDMM with n states, $n \geq 1$. Let u and v be two words such that:*

$$A(a^{2n} \cdot u) = 0^{n-1}10^{n-1}1 \cdot v.$$

Then A has an $(a, 0, 1)$-loop.

Proof. Consider the run of $a^{2n} * 0^{n-1}10^{n-1}1$ in A, of the form:

$$q_{init} = q_0 \xrightarrow{(a,0)} q_1 \xrightarrow{(a,0)} \dots \xrightarrow{(a,0)} q_{n-1} \xrightarrow{(a,1)} q_n \xrightarrow{(a,0)} q_{n+1} \dots \xrightarrow{(a,0)} q_{2n-1} \xrightarrow{(a,1)} q_{2n}$$

where for all $0 \leq i \leq 2n$, $q_i \in Q$. By assumption, we know that from state q_{2n}, A accepts $u * v$.

We want to prove that:

1. the states q_0 to q_{n-1} are all distinct, and
2. $q_n = q_0$, and
3. there are no transitions labeled by a except the ones from the run above, and
4. $F = Q$.

Note that this entails that $q_i = q_{n+i}$ for all $0 \leq i \leq n$.

(1) Assume by contradiction that there exist $0 \leq i < j \leq n-1$ such that $q_i = q_j$. Since A only has n states, we know that there exist $n \leq k < l \leq 2n$ such that $q_k = q_l$. We consider two cases, either $l < 2n$, or $l = 2n$. If $l < 2n$, then the following words are accepted by A, leading to a contradiction to the output-uniqueness property of f-NDMMs.

– $a^{2n-j+i-l+k+(j-i)(l-k)} \cdot u * 0^{n-1-j+i}10^{n-1-l+k+(j-i)(l-k)}1 \cdot v$, by going through $q_0 \dots q_i q_{j+1} \dots q_{k-1}(q_k \dots q_{l-1})^{j-i} q_l \dots q_{2n} \dots$,
– $a^{2n-j+i-l+k+(j-i)(l-k)} \cdot u * 0^{n-1-j+i+(j-i)(l-k)}10^{n-1-l+k}1 \cdot v$, by going through $q_0 \dots q_{i-1}(q_i \dots q_{j-1})^{l-k} q_j \dots q_k q_{l+1} \dots q_{2n} \dots$.

Similarly, if $l = 2n$, the following words are accepted by A, again leading to a contradiction.

– $a^{2n-j+i-l+k+(j-i)(l-k)} \cdot u * 0^{n-1-j+i}10^{n-l+k}(0^{l-k-1}1)^{(j-i)} \cdot v$,
– $a^{2n-j+i-l+k+(j-i)(l-k)} \cdot u * 0^{n-1-j+i+(j-i)(l-k)}10^{n-l+k} \cdot v$.

We conclude that the states q_0 to q_{n-1} are all distinct.

(2) Since the states q_0 to q_{n-1} are all distinct, we know that $q_n = q_i$ for some $0 \leq i \leq n-1$. Assume by contradiction that $0 < i$. By doing the same case analysis as above (either $l < 2n$, or $l = 2n$), we again find contradictions to the output-uniqueness property of A.

(3) Assume by contradiction that there exist $i \neq j$ with $0 \leq i, j \leq n-1$ and $b \in \Gamma$ such that $\delta(q_i, (a, b)) = q_j$ and this transition is different than the transitions from the run above.

If $i < j$, then there is an alternative loop $q_i, q_j, q_{j+1}, \ldots, q_{n-1}, q_0, q_1, \ldots, q_i$ containing $n - j + i + 1$ transitions labeled by a. In particular, this means that the word $a^{n+n(n-j+i+1)}$ has two different outputs in A. The first one is obtained by going from q_0 to q_i, taking the alternative loop n times, and then going from q_i to q_0 using the $(a, 0, 1)$-loop. The second is obtained by going from q_0 to q_i, taking the $(a, 0, 1)$-loop $(n - j + i + 1)$ times, and then going from q_i to q_0 using the $(a, 0, 1)$-loop. This contradicts the output-uniqueness property of A.

A similar reasoning applies when $j < i$, by using $q_i, q_j, q_{j+1}, \ldots, q_i$ as the alternative loop.

(4) Due to the previous property, the only run labeled whose input is a^i for $0 \leq i \leq n-1$ is the one going through $q_0, q_1, \ldots, q_i$ in the $(a, 0, 1)$-loop. This entails that for $0 \leq i \leq n-1$, q_i is final and $F = Q$.

The following lemma states that multiple input/output examples may be encoded into just one example for f-NDMMs that have an $(a, 0, 1)$-loop.

Lemma 2. *Let $A = (\Sigma \times \Gamma, Q, q_{init}, \delta, F)$ be an f-NDMM with an $(a, 0, 1)$-loop. Let $u, v \in \Sigma^*$ and $u', v' \in \Gamma^*$ such that:*

$$A(u \cdot a \cdot v) = u' \cdot 1 \cdot v'.$$

Then $A(u \cdot a) = u' \cdot 1$ and $A(v) = v'$.

Proof. Using Lemma 1, we know that A has an $(a, 0, 1)$-loop. Therefore, the only transition labeled by $(a, 1)$ is the one leading to the initial state. Therefore, after reading $(u \cdot a) * (u' \cdot 1)$, A must be in the initial state. This entails that $A(u \cdot a) = u' \cdot 1$ and $A(v) = v'$.

5 NP-Hardness of the Minimization Problem with one Input/Output Example and Fixed Number of States

We prove the NP-hardness of Problem 2 by reducing the One-In-Three Positive SAT problem to it. This NP-hardness proof holds even when the target number of states for minimization is fixed to 3, the size of the output alphabet is fixed to 2, and there is single input/output example.

Theorem 1. *Problem 2 is NP-hard when the number of states is fixed, the output alphabet is fixed, and there is a single input/output example.*

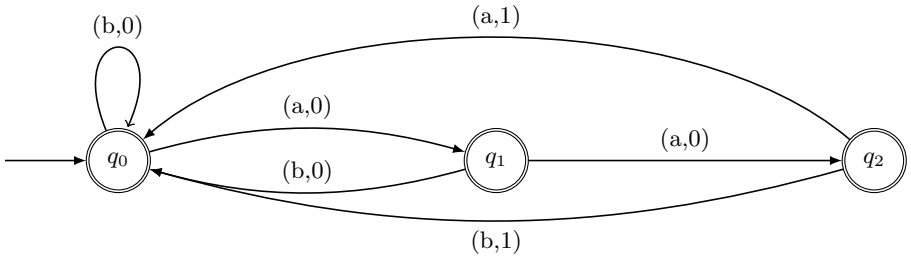

Fig. 3. f-NDMM used in the proof of Theorem 1.

Proof. Consider an instance φ of One-In-Three Positive SAT, with a set of variables V, and a set of clauses $C \subseteq V^3$. We reduce One-In-Three Positive SAT to Problem 2 as follows. We define $\Sigma = V \cup \{a, b\}$, where a and b are fresh symbols and $\Gamma = \{0, 1\}$. Moreover, we define $n = 3$ (fixed number of states).

Then, we define $E = \{w\}$ where w is one input/output example made of the concatenation of all the following words (the word $aaaaaa * 001001$ must go first in the concatenation, but the other words can be concatenated in any order):

- $aaaaaa * 001001$,
- $baaa * 0001$,
- $abaaa * 00001$,
- $aabaaa * 001001$,
- $xbaaa * 00001$ for all $x \in V$,
- $xxxaaa * 000001$ for all $x \in V$,
- $axxxaa * 000001$ for all $x \in V$,
- $aaxxxa * 000001$ for all $x \in V$,
- $xyzaa * 00001$ for all $(x, y, z) \in C$.

We prove that φ has a satisfying assignment if and only if there exists a total f-NDMM A, consistent with E, and with (at most) 3 states.

($\Rightarrow$) Let $f : V \to \{\bot, \top\}$ be a satisfying assignment for φ. We define $A = (\Sigma \times \Gamma, Q, q_{init}, \delta, F)$ following Figure 3 with $Q = F = \{q_0, q_1, q_2\}$ and $q_{init} = q_0$. The transitions involving $a \in \Sigma$ in A are: $(q_0, (a, 0), q_1), (q_1, (a, 0), q_2) \in \delta$, and $(q_2, (a, 1), q_0) \in \delta$.

Then, for each $x \in V$, if $f(x) = \top$, we add three transitions in δ, called *forward transitions*: $(q_0, (x, 0), q_1)$, $(q_1, (x, 0), q_2)$, and $(q_2, (x, 0), q_0)$. If $f(x) = \bot$, we add three transitions as well, called *looping transitions*: $(q_0, (x, 0), q_0)$, $(q_1, (x, 0), q_1)$, and $(q_2, (x, 0), q_2)$.

A is a total f-NDMM, since all states are final, and for every state and every input in Σ, there is a unique outgoing transition labeled by this input (and some output in Γ). Moreover, we can verify that A is consistent with the input/output example w.

($\Leftarrow$) Let $A = (\Sigma \times \Gamma, Q, q_{init}, \delta, F)$ be a total f-NDMM with 3 states, and consistent with E. Our proofs goes as follows. First, using Lemma 1 and Lemma 2,

we deduce that A must have an $(a, 0, 1)$-loop, and must accept all the individual words that constitute the concatenation w. Then, using the facts that $A(baaa) = 0001$, $A(abaaa) = 00001$, $A(aabaaa) = 001001$, we deduce that A must contain the transitions present in Figure 3, and no other transitions labeled by b.

Then, for each variable $x \in V$, using the facts that $A(xbaaa) = 00001$ and $A(xxxaaa) = A(axxxaa) = A(aaxxxa) = 000001$, we show that x must either have looping transitions, or forward transitions, as described in the first part of the proof. We then use this fact to define f that assigns $\top$ to variables that have forward transitions, and $\bot$ to variables that have looping transitions.

Finally, for each clause $(x, y, z) \in C$, and using $A(xyzaa) = 00001$, we deduce that exactly one variable out of x, y and z must have forward transitions, and conclude that f is a satisfying assignment for φ.

We now give more details for each step of the proof. Our first goal is to prove that A must contain the transitions given in Figure 3. Since $A(baaa) = 0001$, we know that after reading $(b, 0)$, A must be in state q_0, and therefore there exists a transition $(q, 0, (b, 0), q_0) \in \delta$. Using $A(abaaa) = 00001$ and $A(aabaaa) = 001001$ respectively, we deduce that there exist transitions $(q, 1, (b, 0), q_0)$ and $(q, 2, (b, 1), q_0)$ in δ. Using the output-uniqueness property of A, we can verify that there can be no other transitions labeled by b in A.

Our next goal is to prove that for each variable $x \in V$, x must either have looping transitions or forward transitions.

Since $xbaaa * 00001 \in A$ and the only transitions labeled by $(b, 0)$ are the ones from states q_0 and q_1, we deduce that from the initial state, reading $(x, 0)$ must lead either to q_0 or q_1, and therefore there should either exist a transition $(q_0, (x, 0), q_1) \in \delta$ or a transition $(q_0, (x, 0), q_0) \in \delta$.

Assume $(q_0, (x, 0), q_1) \in \delta$. In that case, we prove that x has forward transitions, in the sense that there are transitions $(q_1, (x, 0), q_2)$ and $(q_2, (x, 0), q_0)$ in δ. We know $xxxaaa * 0000001 \in A$. Since the only state from which the word $aaa * 001$ is accepted is q_0, the automaton A must end in q_0 after reading $xxx * 000$. Moreover, since $(q_0, (x, 0), q_1) \in \delta$, we know A ends in state q_1 after reading $(x, 0)$ in the initial state. Therefore, when reading $xx * 00$ from state q_1, A must end in state q_0. The only way this is possible is by having transitions $(q_1, (x, 0), q_2)$ and $(q_2, (x, 0), q_0)$ in δ.

The other case we consider is when $(q_0, (x, 0), q_0) \in \delta$. Here, we want to prove that x has looping transitions, with $(q_1, (x, 0), q_1)$ and $(q_2, (x, 0), q_2)$ in δ. We know $axxxaa*000001 \in A$. The only state from which $aa*01$ can be accepted is q_1. Moreover, A ends in state q_1 after reading $(a, 0)$. Therefore, A must go from state q_1 to q_1 by reading $xxx * 000$. Due to the self-loop $(q_0, (x, 0), q_0) \in \delta$, the only possibility for this is to have a loop $(q_1, (x, 0), q_1) \in \delta$. Similarly, using $aaxxxa * 000001 \in A$, we deduce there is a loop $(q_1, (x, 0), q_1) \in \delta$.

Overall, we have shown that each variable $x \in V$ either has forward transitions, or looping transitions. We now define the assignment f that assigns $\top$ to variables that have forward transitions, and $\bot$ to variables that have looping transitions. Let $(x, y, z) \in C$. We know $xyzaa * 00001 \in A$. The only state from

which $aa * 01$ can be accepted is q_1. Therefore, A must end in state q_1 after reading $xyz * 000$. The only way for this to be the case is that exactly one of x, y, z has forward transitions, while the two others have looping transitions.

6 NP-Hardness Proofs for Other Variants

In this section, we give two other NP-hardness proofs, that cover instances of the problem which are not comparable to the ones treated in Section 5.

These proofs also follow the idea of reducing from the One-In-Three Positive SAT problem, but require new encodings. The proofs are deferred to the long version [14].

6.1 NP-Hardness of the Minimization Problem with One Input/Output Example and Fixed Alphabets

Our second NP-hardness proof holds for the case where the sizes of both input and output alphabets are fixed, and there is a single input/output example. When the input and output alphabets are fixed, we can no longer use the encoding given in the previous section, where we could associate to each variable of the SAT formula a letter in our alphabet. Instead, we here rely on the fact that the target number of states is not fixed. As such, this theorem is complementary to Theorem 1.

Theorem 2. *Problem 2 is* NP*-hard when the alphabets Σ and Γ are fixed, and there is a single input/output example.*

Remark 3. Note that if the input and output alphabets as well as the target number of states are fixed, then Problem 2 can be solved in polynomial time. The reason is that when all these parameters are constants, then there is only a constant number of f-NDMMs to explore.

6.2 NP-Hardness of the Minimization Problem for Layered Automata

In this section, we cover automata that only recognize words of the same length. An NFA $A = (\Sigma, Q, q_{init}, \delta, F)$ is said to be *l-layered* for $l \in \mathbb{N}$ if A only accepts words of length l, i.e. $\mathcal{L}(A) \subseteq \Sigma^l$. An l-layered f-NDMM $A = (\Sigma \times \Gamma, Q, q_{init}, \delta, F)$ is called *l-total* if the domain of the function associated with A is Σ^l.

We then adapt Problem 2 for this setting.

Problem 4. Let Σ be an input alphabet, Γ an output alphabet, and $l \in \mathbb{N}$. Let $u_1 * v_1$, ..., $u_k * v_k$ be a set of input/output examples, with $u_i \in \Sigma^l$ and $v_i \in \Gamma^l$ for all $1 \leq i \leq k$. Let $n \in \mathbb{N}$.

Does there exist an l-layered and l-total f-NDMMs that accepts $u_i * v_i$ for all $1 \leq i \leq k$, and that has at most n states.

The following theorem proves that Problem 4 is NP-hard, even when the alphabets are fixed. In this theorem, we can no longer rely on Lemmas 1 and 2, since layered automata cannot contain cycles. Instead, we have to use multiple input/output examples in our encoding.

Theorem 3. *Problem 4 is* NP*-hard when the alphabets Σ and Γ are fixed.*

Remark 4. When there is a single input/output example, Problem 4 can be solved in polynomial time. The reason is that, in a layered f-NDMM, we need at least as many states as the size of the example (plus one) to recognize it. Therefore, the minimal layered f-NDMM that recognizes one given input/output example is easy to construct, by using that many states.

7 Solving the Minimization Problem in NP

We now focus on finding an algorithm for solving the minimization problems 1 and 2. In this section, we propose an approach which solves the problem in non-deterministic polynomial-time. Combined with the proofs in the previous sections, we can deduce that Problem 2 is NP-complete.

The key is to prove (see Lemma 3) that for any valid set of input/output examples, there exists a total f-NDMM, consistent with E, and whose size is at most $2 + \sum_{w \in E} |w|$. Then, a naive minimization approach can iterate through all integers i between 1 and this bound, guess non-deterministically a DFA A of size i, and check whether A is a total f-NDMM consistent with E. We prove that this final check can be done in polynomial time (see Lemma 4), meaning that the whole procedure has non-deterministic polynomial time.

Lemma 3. *Let $E \subseteq (\Sigma \times \Gamma)^*$ be a valid set of input/output examples. There exists a total f-NDMM, consistent with E, with at most $2 + \sum_{w \in E} |w|$ states.*

Proof. We define $T = (\Sigma \times \Gamma, Q^T, q^T_{init}, \delta^T, F^T)$ to be a tree-shaped (partial) f-NDMM consistent with E, as follows:

- Q^T is the set of all prefixes of E,
- $q^T_{init} = \varepsilon$,
- $\delta^T = \{(q_1, (a, b), q_2) \mid q_1, q_2 \in E \wedge q_2 = q_1 \cdot (a, b)\}$,
- $F^T = E$.

By construction, T has at most $1 + \sum_{w \in E} |w|$ states.

Let $P = \mathit{Prefixes}(\mathit{dom}(E)) \subseteq \Sigma^*$ be the set of all prefixes of $\mathit{dom}(E)$. For each $u \in P$, we choose $v \in \Gamma^*$ as follows:

- if $u \in \mathit{dom}(E)$, choose v as the unique word such that $u * v \in E$,
- otherwise, choose any v such that $u * v \in \mathit{Prefixes}(E)$.

We denote by $P' \subseteq \mathit{Prefixes}(E)$ the set of pairs (u, v) where $u \in P$ and v is the corresponding word, chosen in the previous step. Let $b_0 \in \Gamma$ be a letter of the output alphabet. We define the automaton $A = (\Sigma \times \Gamma, Q, q_{init}, \delta, F)$, which is a total f-NDMM consistent with E, as follows:

- $Q = Q^T \cup \{q_f\}$ where q_f is a new state,
- $q_{init} = q^T_{init}$,
- $\delta = \delta^T \cup$
  $\{(q_f, (a, b_0), q_f) \mid a \in \Sigma\} \cup$
  $\{(q, (a, b_0), q_f) \mid q \in P' \wedge a \in \Sigma \wedge \mathit{input}(q) \cdot a \notin P\}$
- $F = P' \cup \{q_f\}$.

It remains to prove three things: (1) A is an f-NDMM, (2) A is total, and (3) $E \subseteq \mathcal{L}(A)$.

1. By construction, A is a DFA. Let $u * v_1 \in A$, and $u * v_2 \in A$, with $u \in \Sigma^*$ and $v_1, v_2 \in \Gamma^*$. Our goal is to prove that $v_1 = v_2$. We consider several cases:
   (a) $u*v_1$ and $u*v_2$ are both accepted in q_f: By construction of A, q_f is a state from which a run can never get out (a sink state). Consider the accepting run of $u * v_1$ in A and let $q_1 \in Q^T$ be the last state of Q^T before reaching q_f. There is a prefix $u_1 * v'_1$ of $u * v_1$ that corresponds to q_1. Similarly, let $q_2 \in Q^T$ be the last state of Q^T in the run of $u * v_2$ in A, and let $u_2 * v'_2$ be the prefix of $u * v_2$ that corresponds to state q_2. Without loss of generality, we can assume that u_1 is a prefix of u_2.
   Moreover, we prove that u_1 is in fact equal to u_2. Assume by contradiction that u_1 is a strict prefix of u_2, and let $u_2 = u_1 \cdot a \cdot u'_1$. Therefore, there is a transition from q_1 to q_f whose input letter is a, which is not possible since $u_1 \cdot a \in P$. Therefore, $u_1 = u_2$.
   So far, we know $u_1 * v'_1$ goes to state q_1, and $u_1 * v'_2$ goes to state q_2. By construction, the only transitions leading to q_f are from states of P'. So we have $q_1, q_2 \in P'$. We know P' is a function relation, and only associates to each word in Σ^* at most one word in Γ^*. We deduce that $v'_1 = v'_2$, and that $q_1 = q_2$.
   Since the runs then join q_f, where the only possible output letter is b_0, we deduce that $v_1 = v_2$.
   (b) $u * v_1$ is accepted in q_f, while $u * v_2$ is accepted in P' (the case where v_1 and v_2 are interchanged is symmetrical): Consider the accepting run of $u * v_1$ in A and let $q_1 \in Q^T$ be the last state of Q^T before reaching q_f. Let $u_1 * v'_1$ be the prefix of $u * v_1$ that corresponds to q_1. Let $u = u_1 \cdot a \cdot u'_1$ with $a \in \Sigma$ and $u'_1 \in \Sigma^*$. By construction of q_1, there is a transition from q_1 to q_f whose input letter is a. However, this is a contradiction, as $u_1 \cdot a \in P$.
   (c) $u*v_1$ and $u*v_2$ are both accepted in P'. P' has been built as a functional relation, therefore we must have $v_1 = v_2$.
2. Let $u \in \Sigma^*$. We want to prove that there exists $v \in \Gamma^*$ such that $u * v \in A$. Let $u = u' \cdot u''$ where u' is the longest prefix of u that belongs to P. Let $v' \in \Gamma^*$ be the unique word such that $u' * v' \in P'$. By defining $v = v' \cdot (b_0)^{|u''|}$, and by construction of A, we have $u * v \in A$.
3. Since A is obtained from T by adding one state, some transitions, and by making some states accepting, we have $\mathcal{L}(T) \subseteq \mathcal{L}(A)$. Moreover, by construction of T, we have $E = \mathcal{L}(T)$, so we have $E \subseteq \mathcal{L}(A)$.

Checking whether a DFA A is a total f-NDMM can be done in polynomial time, as shown in Lemma 4. In addition, checking whether an f-NDMM A is consistent with E, can be done by doing membership checks $w \in A$ for each $w \in E$.

Lemma 4. *Let A be a DFA over the alphabet $\Sigma \times \Gamma$. We can check in polynomial time whether A is a total f-NDMM.*

Proof. Let A' be the projection of A over the input part of the alphabet Σ. The output-uniqueness property of A is equivalent to the fact that A' is unambiguous. Checking whether an NFA is unambiguous can be done in polynomial time [23].

For the output existence property, we check whether $\Sigma^* = A'$, which can be done in polynomial time [25] since A' has been verified to be unambiguous.

Using these lemmas, we conclude with the main result of this section.

Theorem 4. *The minimization problems (1, 2, and 4) can be solved in* NP.

8 Algorithm for Solving the Minimization Problem

8.1 Description of the Algorithm

The algorithm given in the previous section is not applicable in practice, as it requires *guessing* a total f-NDMM that satisfies the constraints. On a computer, this would require enumerating all automata of a certain size until we find one that satisfies the constraints.

In this section, we instead propose to encode the constraints in a logical formula, and let an SMT solver check satisfiability of the formula. More precisely, given a set of input/output examples $E \subseteq (\Sigma \times \Gamma)^*$, and $k \geq 1$, we define a formula $\varphi_{E,k}$ which is satisfiable if and only if there exists a total f-NDMM with k states and that is consistent with E.

Then, in order to find the minimal total f-NDMM with a given set of examples E, our algorithm checks satisfiability of $\varphi_{E,1}$, then $\varphi_{E,2}$, and so on, until one of the formula is satisfiable and the automaton is found.

Encoding all the constraints of the problem in a logical formula is challenging. The main reason is that SMT solver are best suited for dealing with logical formula written in purely existential form, while the constraints that we want to express (totality and output-uniqueness for f-NDMMs) are naturally expressed using alternations between *for all* and *exists* quantifiers. Still, we were able to find a purely existential encoding of the problem, in (quantifier-free) linear arithmetic, which we describe below.

8.2 Encoding

The free variables of $\varphi_{E,k}$ are (bounded) integers and booleans. They are setup so that a valuation of the free variables represent an f-NDMM $(\Sigma \times \Gamma, Q, q_{init}, \delta, F)$

with k states ($Q = \{q_0, \ldots, q_{k-1}\}$). More precisely, $\varphi_{E,k}$ contains for every $q \in Q$ and $\sigma \in \Sigma, \gamma \in \Gamma$, a integer variable

$$0 \leq \delta_{(q,\sigma,\gamma)} < k$$

to represent the value $\delta(q, (\sigma, \gamma))$.

For each state $q \in Q$, $\varphi_{E,k}$ also contains a boolean variable $\mathsf{isFinal}_q$ which is true when state $q \in F$. By convention, q_0 is the initial state, and q_{k-1} is the (non-accepting) sink state.

For states $p, q \in Q$ and input letter $\sigma \in \Sigma$, we also add boolean variables $\delta_{in}^{p,\sigma,q}$ describing the transition relation of the projection A' of A over the input alphabet Σ. The variable δ_{in} is expressed as a relation rather than as a function, since in general, A' can be non-deterministic.

The formula $\varphi_{E,k}$ is then composed of multiple components:

$$\mathsf{AcceptExamples} \wedge \mathsf{Projection} \wedge \mathsf{Unambiguous} \wedge \mathsf{Total}.$$

The formula $\mathsf{AcceptExamples}$ constrains the variables $\delta_{(q,\sigma,\gamma)}$ and $\mathsf{isFinal}_q$ ($q \in Q, \sigma \in \Sigma, \gamma \in \Gamma$) to make sure that every input/output example in E is accepted by A.

The formula $\mathsf{Projection}$ ensures that the variable $\delta_{in}^{p,\sigma,q}$ indeed represents the projection of δ on the input alphabet Σ.

The formulas $\mathsf{Unambiguous}$ and Total correspond to the approach described in Lemma 4. The formula $\mathsf{Unambiguous}$ is a constraint over the variables $\delta_{in}^{p,\sigma,q}$ and $\mathsf{isFinal}_q$. It states that A' is a UFA, which ensures that A accepts every input word at most once. Being unambiguous is naturally stated using quantifiers: for every word w, if w is accepted by two runs r_1 and r_2 in A', then r_1 and r_2 must be identical runs (i.e. going through identical states). However, writing this condition as is would make it hard for the SMT solver to check satisfiability of the formula, due to the universal quantification.

Instead, our formula $\mathsf{Unambiguous}$ is inspired from the algorithm that checks whether a given NFA is unambiguous [23]. This algorithm constructs inductively the pairs of states (q_i, q_j) that are reachable by the same word, but with distinct runs. Then, the NFA is unambiguous if and only if there are no pairs (q, q') in that inductive construction where q and q' are both final states.

The construction starts with the empty set, and adds, for each state q which is reachable, and for every letter $a \in \Sigma$, the pairs (q_1, q_2), with $q_1 \neq q_2$ such that $\delta_{in}(q, a, q_1)$ and $\delta_{in}(q, a, q_2)$ hold. Then, for every (q_i, q_j) and every $a \in \Sigma$, we add the pairs (q_i', q_j') such that $\delta_{in}(q_i, a, q_i')$ and $\delta_{in}(q_j, a, q_j')$ hold.

Therefore, to ensure the unambiguity A', the formula $\mathsf{Unambiguous}$ states that there exists a fixed point (a set of pairs of states represented by boolean variables $r_{q,q'}$ for $q, q' \in Q$) to that construction, i.e. a set which is closed under adding new pairs according to the rules above. Finally, for every $q, q' \in Q$, we add a clause stating that two final states should not belong to the fixed point:

$$\mathsf{isFinal}_q \wedge \mathsf{isFinal}_{q'} \implies \neg r_{q,q'}.$$

The formula Total is also a constraint over the variables $\delta_{in}^{p,\sigma,q}$ and $\mathsf{isFinal}_q$., and states that A' recognizes every string in Σ^*. This ensures that the f-NDMM A accepts every input string at least once. Again, this constraint is naturally expressed using quantifiers: for every word w, there exists a run for w in A'. Such formulas are challenging for SMT solvers. Instead, our formula relies on the fact that A' is ensured to be unambiguous by the formula Unambiguous. More precisely, to check that A' accepts every string of Σ^*, it suffices to check that A' has $|\Sigma|^l$ accepting runs, for every $l \geq 0$. Moreover, it was shown that it is enough to do this check for $l \leq |Q|$ (see [25]).

Our formula Total introduces free variables $c_{l,q}$, for each $0 \leq l \leq |Q|$, and $q \in Q$, and constrains them so that they count how many runs of length l end in state q. By definition, the variable c_{0,q_0} equals 1 (only one word of length 0 is accepted in the initial state), and every other c_{0,q_i} $(i > 0)$ equals 0 (the empty word is not accepted in non-initial states).

Then, using a linear arithmetic formula, we express every $c_{l,q}$ (with $l > 0$) in terms of the variables $c_{l-1,p}$ for $p \in Q$:

$$c_{l,q} = \sum_{\sigma \in \Sigma, p \in Q} \texttt{if } \delta_{in}^{p,\sigma,q} \texttt{ then } c_{l-1,p} \texttt{ else } 0.$$

Total then states, again using linear constraints, that for every $0 \leq l \leq |Q|$, the number of accepting runs of length l equals $|\Sigma|^l$, i.e.

$$\sum_{q \in Q} \texttt{if } \mathsf{isFinal}_q \texttt{ then } c_{l,q} \texttt{ else } 0 = |\Sigma|^l.$$

9 Experimental Evaluation

We implemented our algorithm in Scala, using Z3 [19] as our backend.

9.1 Discovering Small Automata for Common Functions

We give in this section a few examples that we ran using our algorithm. We focus on examples that have small automata, whether or not the input examples are small. Indeed, the combinatorial explosion makes it hard for the SMT solver to find solutions for automata that have more than ~10 states. The results are shown in Figure 4. The arithmetic examples operate on binary representations of numbers, truncating the output to the length of inputs where needed. We note that simple relations such as addition are recovered from examples without the need to specify any expression grammars as in Syntax-Guided Synthesis [2], because automaton minimality provides the needed bias towards simple solutions. Adding more examples than needed (e.g. 22 examples of length 22) keeps the synthesis time manageable, which is useful for cases of automatically generated examples.

We give in the last column (Time2) of the table the times for an enumeration algorithm which does not use SMT solvers. Our algorithm enumerates all transducers by order of the number of states, and prune the search when it encounters

transducers that are not compatible with the input/output examples. When we find a transducer that accepts all input/output examples, we use a *completion* procedure to attempt to make the transducer total by adding transitions. Our implementation should not be considered heavily optimized; we believe that there is space for improvement both in terms of internal data structures and heuristics.

9.2 Evaluating Usefulness of Minimality on Random Automata

The next set of experiments evaluate the likelihood that our algorithm finds the automaton that the user has is mind, depending on the number and size of the input/output examples provided. We generated 100 random minimal total f-NDMMs with 5 states, where the input and output alphabet were both of size 2. For each f-NDMM A, and for every $1 \leq i, j \leq 15$, we generated i random words in Σ^*, of length j. For each such word, we looked up the corresponding output in A, thereby constructing a set of input/output examples E for A. Then, we used our algorithm on E to see whether the obtained automaton would be A. In Table 2, we summarized, for every i and j, out of the 100 automata, how many we were able to reobtain using that method. Overall, the experiments ran for about 3 hours, for $15*15*100 = 22500$ queries. The 3 hours also include the time taken to generate the random automata. To generate a random minimal total f-NDMM, we generated a random sample, and applied our algorithm. Then, if the obtained automaton had 5 states, we kept it for our experiment. Our selection for the choice of the random automata is therefore biased, as the automata are found by our tool in the first place.

Discussion. Generally, the results show that the greater the number of examples given, and the longer they are, the more likely we are to find the automaton

Problem	#Ex.	Ex.Len.	States	Alphabet	Time (s)	Time2 (s)
$x, y \mapsto x+y$	1	17	3	8	0.23	1.090
$x, y \mapsto x+y$	5	4	3	8	0.20	0.090
$x, y \mapsto x+y$	22	22	3	8	0.19	17.610
xor	1	4	2	8	0.07	0.002
and	1	4	2	8	0.08	0.004
or	1	4	2	8	0.05	0.002
not	1	4	2	4	0.06	0.002
$x \mapsto 2x + 1$	1	5	3	4	0.41	0.160
$(p \vee q) \wedge (r \vee s) \wedge \neg t$	1	32	2	64	0.14	3.960
$(p \vee q) \wedge (r \vee s) \wedge \neg t$	32	1	2	64	0.14	0.240
overwrite	10	2	6	4	0.42	0.150
overwrite (3)	39	3	8	9	4.41	4.310

Fig. 4. Synthesis of some common functions from examples, showing successful discovery of minimal automata and tolerance to many long examples and larger alphabets.

i \ j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
2	0	0	0	0	0	0	5	3	8	16	21	16	15	18	22
3	0	0	0	3	1	10	16	32	24	36	35	33	44	44	41
4	0	0	4	7	20	25	37	45	51	53	52	52	51	56	65
5	0	0	6	20	35	46	57	63	59	64	67	62	60	60	64
6	0	0	8	34	43	59	58	67	60	73	75	68	67	66	69
7	0	0	17	37	61	65	70	70	81	76	78	72	75	73	75
8	0	0	22	46	74	79	73	77	78	79	74	77	75	76	78
9	0	0	22	63	67	76	86	80	78	79	82	83	84	82	80
10	0	0	34	59	72	82	86	81	85	80	79	83	84	84	84
11	0	0	36	73	82	86	83	85	85	89	88	86	91	82	83
12	0	0	32	66	86	83	83	86	88	85	86	87	89	88	88
13	0	0	41	83	85	85	89	87	89	85	93	89	88	89	89
14	0	0	41	78	83	88	93	93	92	88	88	87	88	88	91
15	0	0	51	83	87	87	88	84	91	87	91	91	90	87	88

Table 2. In a given cell, the number represents, out of 100 random automata, how many we were able to reobtain using our algorithm, with a random sample with i input/output examples of length j.

that we want. More interestingly, we note that we are more likely to find the automaton we want with a large number of small examples (e.g. $i = 15, j = 5$) than with a small number of large examples (e.g. $i = 5, j = 15$).

Another interesting observation is that the likelihood of finding the automaton increases sharply when using examples of size $j = 4$ rather than $j = 3$. Without counting the sink state, the automata we considered have 4 states. This suggests that in general, a good strategy is to give multiple examples which are at most as long as the number of states (though the user giving the examples may not know how many states are required for the minimal automaton).

10 Related Work

In [16], we studied the problem of synthesizing tree-to-string transducers from examples. Here, instead of having the user provide input/output examples, we proposed an algorithm that generates particular inputs, and asks the user what are the corresponding outputs. We show that, when the algorithm is allowed to analyze previous answers in order to generate the next question, then the number of questions required to determine the transducer that the user has in mind is greatly reduced (compared to an approach without interaction, where the algorithm would ask for all outputs at once).

The results obtained in [16] do not directly apply here, as they were for single-state transducers. However, some of the techniques are fundamental and could be reused here. In that respect, we could generate questions for the users,

and guarantee that the generated f-NDMM is indeed the one that the user had in mind (given some bound on the number of states).

Our paper is similar in spirit to [10], where the author proves that Problem 2 is NP-complete for *deterministic* Mealy machines. Their NP-hardness holds even when the alphabets' sizes are fixed to 2, but the case where the number of states is fixed is not treated. Moroever, even though f-NDMMs are a more general model than deterministic Mealy machines, the NP-hardness of [10] cannot be directly applied to f-NDMMs.

There is a long line of work devoted to learning *deterministic* finite state transducers (see e.g. [6, 20, 1, 18]). Algorithms for learning deterministic finite automata (e.g. [4]) or finite transducers do not directly translate to our setting, since we need to consider functionality and totality constraints, as shown in Section 8.2. Methods for learning non-deterministic automata (e.g. [7]) do not directly apply to our setting either, for the same reasons.

A particular case of learning transducers is an interpolation problem, that consists in learning a finite automaton that accepts some given inputs (i.e. outputs 1) and rejects some other inputs (i.e. outputs 0) (see e.g. [21, 8, 11]).

In [15], the authors present an algorithm for learning *non-deterministic* Mealy machines. They are interested in non-determinism to represent unknown components of reactive systems, and as such do not focus on *functional* non-deterministic Mealy machines. Moreover, their focus is rather on the algorithmic aspect of the problem rather than on complexity classes.

In [12], the author proposes an efficient synthesis procedure from examples for a language that does string transformations, but does not deal with the issue of synthesizing finite-state transducers.

Our algorithm in Section 8 is inspired from the bounded synthesis approach of [9]. There, the authors suggest that bounding the number of states is a good strategy to synthesize reactive systems. They also propose a reduction from the bounded synthesis problem for reactive systems to SMT solvers.

In [13], we presented a way to synthesize string-to-string functions given any specification written in weak monadic second-order logic. Using these techniques, it would be possible to synthesize an f-NDMM consistent with input/output examples, by writing the input/output examples as a logical formula. However, this approach would not yield the minimal f-NDMM consistent with the examples. For example, regardless of how many input/output examples we give for the function $(\{0,1\} \times \{0,1\})^* \to \{0,1\}^*$ which xor's two streams of bits, this approach would not yield the 1-state automaton that we are expecting. Instead, the method will generate large automata that are consistent with the given examples, but do not recognize the xor operation for other input strings. On the other hand, our approach can find this automaton with only a few small examples.

The automata we consider in this paper are closely related to the notion of *thin language* (see e.g. [22]). A language L is called thin if for every $n \in \mathbb{N}$, it contains at most one word of length n. Moreover, L is called length-complete if for every $n \in \mathbb{N}$, L contains at least one word of length n. When $|\Sigma| = 1$, i.e. when only the length of the input matters, our minimization problem corresponds

exactly to finding a minimal DFA that contains a given set of examples, which is both thin and length-complete. We left this question open in Section 3.1, and leave it for future work. This analogy with thin languages breaks when using a non-unary input alphabet.

In [24], the authors encode the problem of learning DFAs in an SMT solver. As is the case with our algorithm, such encodings only perform well for finding automata with a small number of states (up to 10 or 15).

11 Conclusions

f-NDMMs are a form of functional non-deterministic one-way finite-state transducers (see e.g. [23, 5]) where each transition is forced to produce exactly one letter (instead of 0 or more in the general case). The term functional corresponds to the output uniqueness property of f-NDMMs, and ensures that despite the non-determinism, at most one output string is produced for each input string. The non-determinism here refers to the input part of the alphabet, and f-NDMMs, even though they are deterministic on $\Sigma \times \Gamma$, can indeed be non-deterministic in the input alphabet Σ. In that sense, f-NDMMs can define transformations that are not captured by deterministic one-way transducers, such as the function that maps a word w to $l^{|w|}$ where l is the last letter of w. On the other hand, deterministic one-way transducers can recognize transformations not recognized by f-NDMMs, since they do not require the output to have the same length as the input. This can be circumvented by padding the input and output strings using a dummy letter. Existing synthesis algorithms generally target classes of deterministic transducers, such as subsequential transducers (see e.g. [26]). Our results about f-NDMMs are a first step towards synthesis algorithm for larger classes of deterministic or functional non-deterministic transducers, such as two-way finite-state transducers, or streaming string transducers [3]. We have shown that most variants of synthesis for f-NDMMs are NP-complete, and presented a promising approach using an encoding into SMT formulas.

12 Acknowledgement

We thank Rupak Majumdar for references on hardness of the interpolation problem in the settings of automata. We thank anonymous reviewers of previous submissions for their thorough comments and for relevant references related to learning finite automata and regarding the interpolation problem.

References

1. Aarts, F., Kuppens, H., Tretmans, J., Vaandrager, F.W., Verwer, S.: Improving active mealy machine learning for protocol conformance testing. Machine Learning **96**(1-2), 189–224 (2014). https://doi.org/10.1007/s10994-013-5405-0, https://doi.org/10.1007/s10994-013-5405-0
2. Alur, R., Bodik, R., Juniwal, G., Martin, M.M., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Formal Methods in Computer-Aided Design (FMCAD), 2013. pp. 1–17. IEEE (2013)
3. Alur, R., Cerný, P.: Expressiveness of streaming string transducers. In: Lodaya, K., Mahajan, M. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India. LIPIcs, vol. 8, pp. 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
4. Angluin, D.: Learning regular sets from queries and counterexamples. Information and computation pp. 87–106 (1987)
5. Berstel, J.: Transductions and context-free languages, Teubner Studienbücher : Informatik, vol. 38. Teubner (1979), http://www.worldcat.org/oclc/06364613
6. Bojańczyk, M.: Transducers with origin information. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8573, pp. 26–37. Springer (2014). https://doi.org/10.1007/978-3-662-43951-7_3, https://doi.org/10.1007/978-3-662-43951-7_3
7. Bollig, B., Habermehl, P., Kern, C., Leucker, M.: Angluin-style learning of NFA. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 1004–1009 (2009), http://ijcai.org/Proceedings/09/Papers/170.pdf
8. Chen, Y., Farzan, A., Clarke, E.M., Tsay, Y., Wang, B.: Learning minimal separating DFA's for compositional verification. In: Kowalewski, S., Philippou, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5505, pp. 31–45. Springer (2009). https://doi.org/10.1007/978-3-642-00768-2_3, https://doi.org/10.1007/978-3-642-00768-2_3
9. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT **15**(5-6), 519–539 (2013). https://doi.org/10.1007/s10009-012-0228-z, https://doi.org/10.1007/s10009-012-0228-z
10. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37**(3), 302–320 (1978). https://doi.org/10.1016/S0019-9958(78)90562-4, https://doi.org/10.1016/S0019-9958(78)90562-4
11. Grinchtein, O., Leucker, M., Piterman, N.: Inferring network invariants automatically. In: Furbach, U., Shankar, N. (eds.) Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4130, pp. 483–497. Springer (2006). https://doi.org/10.1007/11814771_40, https://doi.org/10.1007/11814771_40

12. Gulwani, S.: Automating string processing in spreadsheets using input-output examples. In: Ball, T., Sagiv, M. (eds.) Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011. pp. 317–330. ACM (2011). https://doi.org/10.1145/1926385.1926423, http://doi.acm.org/10.1145/1926385.1926423
13. Hamza, J., Jobstmann, B., Kuncak, V.: Synthesis for regular specifications over unbounded domains. In: Bloem, R., Sharygina, N. (eds.) Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23. pp. 101–109. IEEE (2010), http://ieeexplore.ieee.org/document/5770938/
14. Hamza, J., Kuncak, V.: Minimal synthesis of string to string functions from examples. CoRR **abs/1710.09208** (2017), http://arxiv.org/abs/1710.09208
15. Khalili, A., Tacchella, A.: Learning nondeterministic mealy machines. In: Clark, A., Kanazawa, M., Yoshinaka, R. (eds.) Proceedings of the 12th International Conference on Grammatical Inference, ICGI 2014, Kyoto, Japan, September 17-19, 2014. JMLR Workshop and Conference Proceedings, vol. 34, pp. 109–123. JMLR.org (2014), http://jmlr.org/proceedings/papers/v34/khalili14a.html
16. Mayer, M., Hamza, J., Kuncak, V.: Proactive synthesis of recursive tree-to-string functions from examples. In: Müller, P. (ed.) 31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain. LIPIcs, vol. 74, pp. 19:1–19:30. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017). https://doi.org/10.4230/LIPIcs.ECOOP.2017.19, https://doi.org/10.4230/LIPIcs.ECOOP.2017.19
17. Mealy, G.H.: A method for synthesizing sequential circuits. Bell Labs Technical Journal **34**(5), 1045–1079 (1955)
18. Merten, M.: Active automata learning for real life applications. Ph.D. thesis, Dortmund University of Technology (2013), http://hdl.handle.net/2003/29884
19. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24
20. Oncina, J., García, P., Vidal, E.: Learning subsequential transducers for pattern recognition interpretation tasks. IEEE Trans. Pattern Anal. Mach. Intell. **15**(5), 448–458 (1993). https://doi.org/10.1109/34.211465, https://doi.org/10.1109/34.211465
21. Pitt, L., Warmuth, M.K.: The minimum consistent DFA problem cannot be approximated within any polynomial. J. ACM **40**(1), 95–142 (1993). https://doi.org/10.1145/138027.138042, http://doi.acm.org/10.1145/138027.138042
22. Păun, G., Salomaa, A.: Thin and slender languages. Discrete Applied Mathematics **61**(3), 257–270 (1995). https://doi.org/10.1016/0166-218X(94)00014-5, https://doi.org/10.1016/0166-218X(94)00014-5
23. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press (2009), http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521844253
24. Smetsers, R., Fiterau-Brostean, P., Vaandrager, F.W.: Model learning as a satisfiability modulo theories problem. In: Klein, S.T., Martín-Vide, C.,

Shapira, D. (eds.) Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10792, pp. 182–194. Springer (2018). https://doi.org/10.1007/978-3-319-77313-1_14, https://doi.org/10.1007/978-3-319-77313-1_14

25. Stearns, R.E., III, H.B.H.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM J. Comput. **14**(3), 598–611 (1985). https://doi.org/10.1137/0214044, https://doi.org/10.1137/0214044
26. Vilar, J.M.: Query learning of subsequential transducers. In: Miclet, L., de la Higuera, C. (eds.) Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, September 25-27, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1147, pp. 72–83. Springer (1996). https://doi.org/10.1007/BFb0033343, https://doi.org/10.1007/BFb0033343