Component Minimization and Synthesis in Sequential Compositions of Mealy Machines: Revisited

Anonymous author
Anonymous affiliation

Abstract

We consider a system consisting of a sequential composition of Mealy machines, called head and tail. We study two problems related to these systems. In the first problem, models of both head and tail components are available, and the aim is to obtain a replacement for either of them with the minimum number of states. We show that this minimization problem is NP-complete for both the head and the tail component. This contrasts with the state-of-the-art method for the optimization of the tail, whose time cost is doubly-exponential in the size of the composition, rather than only exponential. We identify the root cause for the additional blow-up and provide an extension of the existing theory which allows both the design of tail-minimization algorithms with the appropriate complexity, as well as the adaptation of existing techniques. In the second problem we study, either the head or the tail is known, and a desired model for the whole system is given. The objective is to build the missing component in a way that the composition behaves according to the given model. When synthesising the tail, we prove that it is possible to decide in polynomial time whether a solution to this problem exists, but obtaining it in the affirmative case has exponential cost. The reason is that for some heads and desired system models, all tails solving the synthesis problem have exponential size. Additionally, we also give tight lower bounds for the size of a solution when the missing component is the head. In this case the synthesis problem has polynomial complexity, as evidenced by existing methods. Finally, we discuss how our results relate to the problems of component minimization and synthesis in arbitrary synchronous compositions of Mealy machines. In particular, our findings imply that the general component-minimization problem is NP-complete.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Networks of Finite State Machines, Computational Complexity, Component-based design, Design Optimization, Synthesis of Unknown Component, Sequential Machines

1 Introduction

Two classical problems related to Finite State Machines are the optimization of a given machine [14, 9, 20], and the synthesis of a model satisfying some requirements [5, 6, 19, 24]. However, real-world systems usually consist of multiple interconnected components, rather than a single structure. These compositions lead to problems that have been studied intensively [11, 12, 28]. In this paper, we consider a cascade composition $T \circ H$ of two Mealy machines, H (head) and T (tail), where T receives inputs from H but not vice-versa. We focus on two tasks related to these systems: *component minimization* and *component synthesis*. In the Component Minimization Problem, the aim is obtain a smallest replacement for either H or T which does not alter the behaviour of the composition. To do this, one may try minimizing the desired component in isolation. However, there may be more room for optimization. This way, it is possible to find T and T' of different sizes that are not equivalent in isolation, but that can be used interchangeably in the context of another component H [27]. In the case of the Component Synthesis Problem, one of the components H, T is known and an overall desired model M is given. Here, the goal is to build the missing components in such a way that the resulting composition behaves according to M [4, 32, 28]. This problem may occur, for instance, in *rectification*, where a designer wants to repair or change the

behaviour of a system by only modifying a part of it. While the the Component Minimization Problem always admits a solution - if no smaller replacement exists, the component under consideration is already optimal -, this may not hold for the Component Synthesis Problem and thus deciding if an instance of the problem has a solution is also a question of interest.

The main contributions of this paper are as follows: First, we show that the state-of-the-art algorithm for optimizing the tail in a cascade composition are exponentially more costly than the actual complexity of the problem. Second, we describe a natural modification of this algorithm that avoids the complexity blow-up. Third, we give rigorous complexity analyses for both the Component Synthesis and Minimization problems in cascade compositions. We give a more detailed overview below.

The Tail Minimization Problem can be considered the most-studied case of the Component Minimization Problem. The exact algorithm for this scenario was given by Kim and Newborn [16]. Given a cascade $T \circ H$, they compute the smallest replacement T' by minimizing an incompletely specified (IS) Mealy machine N with size proportional to $2^{|H|}|T|$. Minimization of IS machines is known to be NP-hard [23], so known algorithms for this task take exponential time in the size of N [25, 1, 21], which cannot be improved under standard complexity-theoretic assumptions. This results in the complexity of the whole procedure being doubly exponential.

Surprisingly, a simple observation reveals that this approach cannot be efficient: Indeed, given a machine T' it is possible to decide in polynomial time whether its a valid replacement for T just by comparing $T' \circ H$ and $T \circ H$. This shows that the Tail Minimization Problem lies in NP, and exponential-time solutions should be expected. Moreover, this reasoning extends to more arbitrary compositions of Mealy Machines, implying that the general approach given in [32, 28, 12] is not optimal in many cases. A notable exception is the Head Minimization problem, where the main technique [4] has exponential cost, rather than doubly-exponential.

The reason for the additional blow-up in the complexity of the Kim-Newborn method lies in a determinization step that results in N being exponentially larger than H. Various non-exact methods have been studied to avoid this step. For instance, Rho and Somenzi [17] present an heuristic in which a "summarized" incompletely specified machine is obtained, and Wang and Brayton [15] avoid performing exact state minimization and in turn perform optimizations at the net-list logic level. We introduce the notion of a of observation machines (OMs), which can be regarded as IS machines with universal branching, and show that the determinization step in the K-N algorithm can be avoided by considering OMs instead of IS machines. Following this, we generalize the theoretical foundations of the minimization of IS machines [10] to the setting of OMs, allowing for the extension of existing techniques.

In addition to this study, we examine the complexity of the Minimization and Synthesis problems in the setting of cascade compositions in greater detail. We show that both the Head Minimization and the Tail Minimization problems are NP-complete. This is done through a reduction from the problem of minimizing an IS machine for the case of the tail, and a reduction from the Boolean satisfiability problem in the case of the head. When considering the synthesis task we show the surprising result that deciding the feasibility of the Tail Synthesis Problem takes polynomial time, but computing an actual solution T has exponential complexity, matching the bound given by the general approach [32]. This follows from the existence of instances of the synthesis problem where all solutions T have exponential size. We give a family of such instances constructively. Additionally, we show that it is possible to represent all solutions of the Tail Synthesis Problem via an OM. This representation avoids any kind of subset construction and hence is exponentially more succinct than the representation of the flexibility given in [32]. To complete our analysis, we prove

a lower complexity bound for the Head Synthesis problem that matches the cost of the procedure in [4], and is tight as a result.

Finally, we note that despite the simplicity of cascade compositions, in various cases it is possible to tackle problems in more complex two-component networks via simple reductions to this setting [29]. Moreover, cascade compositions can be seen as particular instances of general two-component architectures. An implication of this fact is that the Component Minimization Problem remains NP-complete when considering arbitrary non-trivial compositions.

2 Preliminaries

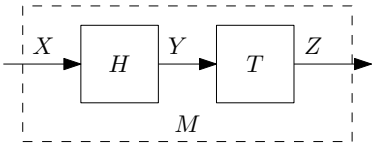
General Notation. We write $[k]$ for $\{0, \dots, k-1\}$, 2^X for the power set of X, and X^* for the set of finite words of arbitrary length over X. We use overlined variables $\overline{x} = x_0 \dots x_{n-1}$ for words, and write ϵ for the empty word. Given two words of the same length we define $\langle \overline{x}, \overline{y} \rangle \coloneqq (x_0, y_0) \dots (x_n, y_n) \in (X \times Y)^*$.

Mealy Machines. Let X and Y be finite alphabets. A **Mealy machine** M from X to Y is a tuple $(X, Y, S_M, D_M, \delta_M, \lambda_M, r_M)$, where S_M is a finite set of states, $D_M \subseteq S_M \times X$ is a specification domain, $\delta_M : D_M \to S_M$ is the next state function, $\lambda_M : D_M \to Y$ is the output function and $r_M \in S_M$ is the initial state. We say that an input string $\overline{x} \coloneqq x_0 x_1 \dots x_n \in X^*$ is **defined** at a state s_0 if there are states $s_1, \dots, s_{n+1}$ satisfying both $(s_i, x_i) \in D_M$ and $s_{i+1} = \delta_M(s_i, x_i)$ for all $0 \leq i \leq n$. We call the sequence $s_0, x_0, y_0, s_1, \dots, s_n, x_n, y_n, s_{n+1}$, where $y_i = \lambda_M(s_i, x_i)$, the **run** of M on $\overline{x}$ from s_0. When $s_0 = r_M$, we simply call this sequence the run of M on $\overline{x}$. We write $\Omega_M(s)$ for the set of input sequences defined at state s, and Ω_M for $\Omega_M(r)$. We lift δ_M and λ_M to defined input sequences in the natural way. We define $\delta_M(s, \epsilon) = s$, $\lambda_M(s, \epsilon) = \epsilon$ for all s. Given $\overline{x} \in \Omega_M(s)$, if $s' = \delta_M(s, \overline{x})$ and $(s', x') \in D_M$ then $\delta_M(s, \overline{x}x') = \delta_M(s', x')$ and $\lambda_M(s, \overline{x}x') = \lambda_M(s, \overline{x})\lambda_M(s', x')$. We write $\delta_M(\overline{x})$ and $\lambda_M(\overline{x})$ for $\delta_M(r_M, \overline{x})$ and $\lambda_M(r_M, \overline{x})$ respectively. We define $\mathrm{Out}(M)$ as the set of words $\lambda_M(\overline{x})$, for all $\overline{x} \in \Omega_M$. We define the set $\mathrm{Tr}(M) \subseteq (X \times Y)^*$ of **traces** produced by M as the one containing all words of the form $\langle \overline{x}, \overline{y} \rangle$, where $\overline{x} = \overline{x}^1\overline{x}^2$, $\overline{y} = \lambda_M(\overline{x})\overline{y}^2$, and $\overline{x}^1$ is the maximal prefix of $\overline{x}$ which belongs to Ω_M. Intuitively, Tr contains I/O words where the outputs are produced according to λ_M as long as the inputs are defined, and arbitrary outputs can be produced afterwards.

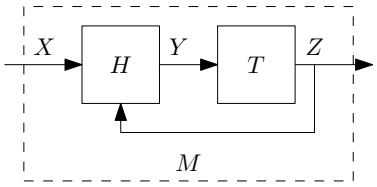
We say M is **completely specified**, if $D_M = S_M \times X$. Otherwise we say that M is **incompletely specified**. From now on, we refer to completely specified Mealy machines simply as Mealy machines, and to incompletely specified ones as IS Mealy machines. For considerations of computational complexity, we consider alphabets to be fixed and the size $|M|$ of a Mealy machine M to be proportional to its number of states.

We say that a (completely specified) Mealy machine N **implements** an IS Mealy machine M with the same input/output alphabets as N if $\lambda_N(\overline{x}) = \lambda_M(\overline{x})$ for all $\overline{x} \in \Omega_M$. Note that this is equivalent to $\mathrm{Tr}(N) \subseteq \mathrm{Tr}(M)$. The problem of minimizing an IS Mealy machine M consists of finding a minimal implementation for it. It is well known that this problem is computationally hard, in contrast with the minimization of completely specified Mealy machines. More precisely, we say that M is **n-implementable** if it has some implementation whose size is not greater than n. Given an IS machine M and some n, deciding whether M is n-implementable is an NP-complete problem [23].

Automata Over Finite Words. We consider automata over finite words where all states are accepting. Let Σ be a finite alphabet. A **non-deterministic finite automaton (NFA) A over** Σ, is a tuple $(\Sigma, S_A, \Delta_A, r_A)$, where S_A is a finite set of states, $\Delta_A : S_A \times \Sigma \to 2^{S_A}$ is the transition function, and $r_A \in S_A$ is the initial state. A **run** of A on a word $\overline{a}$ is defined



(a) A cascade composition of Mealy machines.



(b) A composition between Mealy machines with two-way communication.

Figure 1 Two different compositions of Mealy machines.

as usual. We say that A **accepts** a word $\overline{a}$ if there is a run of A on $\overline{a}$. The **language** of A is the set $\mathcal{L}(A) \subseteq \Sigma^*$ containing the words accepted by A. Note that $\mathcal{L}(A)$ is prefix-closed. We lift Δ_A to words $\overline{a} \in \Sigma^*$ and sets of states $Q \subseteq S_A$ in the natural way. The set $\Delta_A(s, \overline{a})$ consists of all s' such that some run of A on $\overline{a}$ from s finishes at s', and we write $\Delta_A(Q, \overline{a})$ for $\cup_{s \in Q} \Delta_A(s, \overline{a})$. We write $\Delta_A(\overline{a})$ for $\Delta_A(r_A, \overline{a})$. We say that A is **deterministic (a DFA)**, if $|\Delta(s, a)| \leq 1$ for all s, a. For considerations of computational complexity, we consider the alphabet to be fixed, as before, and the size $|A|$ of an automaton A to be proportional to $|S_A| + \sum_{s,a} |\Delta_A(s, a)|$.

We note that Mealy machines can be seen as particular cases of DFAs, in the sense that any Mealy machine M from X to Y corresponds naturally to a DFAs A over $X \times Y$ with (almost) the same number of states satisfying $\mathrm{Tr}(M) = \mathcal{L}(A)$. If M is completely specified, A can be obtained just by coping M and rewriting the I/O x/y pairs in the transitions as inputs. That is, whenever $\delta_M(s, x) = t$ and $\lambda_M(s, x) = y$, we define $\Delta_A(s, (x, y)) = \{t\}$. If M is incompletely specified, in addition to this step we also add outgoing transitions $\Delta_A(s, (x, y)) = \{*\}$ for all undefined pairs $(s, x) \notin D_M$ and all $y \in Y$. Here $*$ denotes an additional sink state added to A.

3 Problem statements

We define a **cascade composition of Mealy machines** as a system consisting of two Mealy machines H, the **head**, and T, the **tail**, that work in sequential composition as shown in Figure 1a. We write $T \circ H$ to refer to this sequential composition. The behaviour such cascade composition can be described via another Mealy machine M resulting from a standard product construction: Set $S_M := S_H \times S_T$ and $r_M := (r_H, r_T)$. Let $s_H \in S_H, s_T \in S_T, x \in X$ and $y := \lambda_H(s_H, x)$. We define $\delta_M((s_H, s_T), x) := (s'_H, s'_T)$ where $s'_H := \delta_H(s_H, x)$ and $s'_T := \delta_T(s_T, y)$, and $\lambda_M((s_H, s_T), x) := \lambda_T(s_T, y)$. Given a cascade composition $T \circ H$, we say that a Mealy machine T' is a **replacement** for T if $T' \circ H \equiv T \circ H$. We say that T is n**-replaceable** in $T \circ H$ if there is a replacement T' for T with at most n states.

We study two problems related to this composition. In the first, both components H and T are given and the goal is to find a minimal replacement for either of them that leaves the behaviour of the system unaltered. In the second, only one of the components is given, together with an additional machine M, and one is asked to build the missing component in such a way that $T \circ H \equiv M$. We also consider two related decision problems.

▶ **Problem 1** (Tail (Head) Minimization Problem)**.** *Given a cascade composition $T \circ H$, find a replacement T' (H') for T (H) with the minimum number of states.*

▶ **Problem 2** (Tail (Head) Synthesis Problem). *Given Mealy machines H (T) and M sharing the same input (output) alphabet, construct a Mealy machine T (H) so that $T \circ H \equiv M$.*

▶ **Problem 3** (n-Replaceabilty of the Tail (Head)). *Given a cascade composition $T \circ H$ and a number $n \in \mathbb{N}$, decide whether there is a replacement T' (H') for T (H) with at most n states.*

▶ **Problem 4** (Feasibility of the Tail (Head) Synthesis Problem). *Given Mealy machines H (T) and M with the same input (output) alphabet, decide whether there exists some Mealy machine T (H) such that $T \circ H \equiv M$.*

3.1 General Method

Here we give a high-level description of the general method for component minimization and synthesis in arbitrary synchronous compositions shown in [32, 28, 12]. The goal in both the minimization and synthesis tasks is to construct a model for a component C in a composition in such a way that the behaviour of the whole system matches a desired one. The two main differences are that (1) in the optimization problem one aims for a smallest model, while in the synthesis one any model suffices, and (2) a solution for the synthesis problem may not exist at all for some instances.

In the general method a DFA over I/O words F called the "flexibility" of the component C is computed (the equivalent formalism of "pseudo non-deterministic FSMs" is used in [32, 28, 12]). This automata, also known as the the E-machine, satisfies the key property that a model C' for C leads to desirable system behaviours if and only if $\mathrm{Tr}(C') \subseteq \mathcal{L}(F)$. An important aspect of this construction is that T is exponentially larger than the system components for arbitrary compositions. In the Minimization Problem one proceeds to obtain a smallest C' satisfying $\mathrm{Tr}(C') \subseteq \mathcal{L}(F)$. This last optimization step is as hard as minimizing a IS Mealy machine (given that DFAs are more general models), so it is expected to take exponential time in the size of F (e.g., [30]), hence resulting in a total doubly-exponential cost for the Component Minimization procedure. In the Synthesis Problem any machine C' satisfying $\mathrm{Tr}(C') \subseteq \mathcal{L}(F)$ suffices. Deciding whether such C' exists and computing it in the affirmative case amounts to solving a two player game over F called a **safety game** [5, 6, 19]. This has linear complexity in the size of F, so the total cost of the synthesis procedure is exponential.

As stated during the introduction, the case where the component under minimization/-synthesis is the head H of a cascade $T \circ H$ deserves special mention. Here the flexibility F can be computed in polynomial time through a simple product construction [4]. This makes the minimization and synthesis procedures exponentially more efficient in this situation.

Finally, we note that there is a different strain of algorithms for the Tail Minimization Problem, represented by [22, 2]. The aim of these is component learning, rather than component minimization, but they can be used for the second task. These methods are inspired in the L* algorithm for the active learning of regular languages [3], and their general structure is as follows: they maintain a set O of input/output traces of the form $\langle \overline{y}, \overline{z} \rangle$, where $\overline{y} \in \mathrm{Out}(H)$ and $\overline{z} = \lambda_T(\overline{y})$. In the first step they enlarge the set O adding traces until some criteria is met. In the second, they compute smallest machine T' consistent with the observations in O, and they check whether T' is a suitable replacement for T. If it is, the procedure is finished. If not, a counterexample is added to O and the first step is carried out again. This is an interesting direction to explore. However, the time-cost of these learning-based methods and their comparison to the ones represented by [32, 16] is unclear, so they may share the same worst case complexity.

4 Observation Machines

We introduce now observation machines, which can be regarded as IS Mealy machines [12] with universal branching. This construction allows us to express the solutions of the Tail Minimization Problem and the Tail Synthesis Problem exponentially more succinctly than [16] and [32]. An **observation machine (OM)** from X to Y is a tuple $M = (X, Y, S_M, D_M, \Delta_M, \lambda_M, r_M)$ defined the same way as a Mealy machine except for the next-state function Δ_M, which now maps elements of the specification domain to sets of states $\Delta_M : D_M \to 2^{S_M} \setminus \{\emptyset\}$. A **run of M over $\overline{x} \in X^*$ starting from $s_0 \in S_M$**, is a sequence $s_0, x_0, y_0, s_1, \ldots, s_n, x_n, y_n, s_{n+1}$, where $(s_i, x_i) \in D_M$, $s_{i+1} \in \Delta_M(s_i, x_i)$ and $y_i = \lambda(s_i, x_i)$ for all $0 \leq i \leq n$. We call this run simply a run of M on $\overline{x}$ when $s_0 \coloneqq r_M$. We say that a sequence $\overline{x}$ is defined at a state s if there is a run of M on $\overline{x}$ starting from s. As with Mealy machines, we put $\Omega_M(s)$ and Ω_M for the sets of defined sequences at s and at r_M respectively. We say that M is **consistent** if all runs of M on any given defined input sequence $\overline{x} \in \Omega_M$ have the same output. Unless otherwise specified, we assume all OMs to be consistent. In this case we can lift λ to defined input sequences, witting $\lambda_M(\overline{x})$ for the unique output sequence $\overline{y}$ corresponding to all runs of M over $\overline{x}$. We also lift the transition function Δ_M to defined sequences and sets of states as done previously for NFAs. Note that when M has no branching, i.e. $|\Delta_M(s, x)| = 1$ for all $(s, x) \in D_M$, we end up with a construction equivalent to an IS Mealy machine. Again, for considerations of computational complexity, we consider alphabets to be fixed, and the size $|M|$ to be proportional to $|S_M| + \sum_{(s,a)\in D_M} |\Delta_M(s, a)|$.

We can use consistent OMs to represent specifications over Mealy machines. We say that a machine N **implements** a OM M with the same input/output alphabets as N if $\lambda_N(\overline{x}) = \lambda_M(\overline{x})$ for all $\overline{x} \in \Omega_M$. Note that it is straightforward to check this property in $O(|N||M|)$ time via a product construction. We can also define the set $\mathrm{Tr}(M) \subseteq (X \times Y)^*$ of traces produced by M the same way as with Mealy machines. This way, the fact that N implements M is equivalent to $\mathrm{Tr}(N) \subseteq \mathrm{Tr}(M)$.We say that M is n-implementable if it has an implementation with at most n states. The problem of **minimizing** an OM M is the one of finding an implementation for it with the minimum number of states.

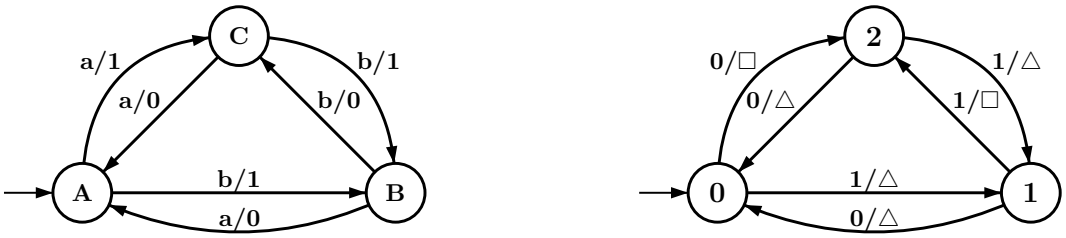
Informally, the reason we call the branching "universal" is that M does not correspond to an NFA in the same way that Mealy machines do to DFAs. Instead, the traces $\mathrm{Tr}(M)$ of an OM M are given by a universal automaton of the same size, rather than a non-deterministic one. The intuitive argument is that once an input word leaves Ω_M, all behaviours are allowed. In a way, this means that M accepts the complement of Ω_M, which is given by an NFA.

5 Revising The Kim-Newborn Algorithm

In this section we describe an exponentially more efficient modification of the Kim-Newborn (K-N) method [16], which represents the state-of-the-art exact solution for the Tail Minimization Problem. This modification improves over the original method by avoiding an expensive determinization step, and in exchange utilizes OMs instead of IS Machines.

Two important observations are the following.

▶ **Observation 1.** *The n-replaceability decision problem Problem 3 is in NP: Given a candidate T' with $|S_{T'}| \leq n$ it takes polynomial time to build Mealy models for $T \circ H$ and $T' \circ H$, and to decide whether they are equivalent. The analogous statement clearly holds as well when considering replacements for the head H.*



(a) A Mealy machine H from $X \coloneqq \{a, b\}$ to $Y \coloneqq \{0, 1\}$.

(b) A Mealy machine T from $Y \coloneqq \{0, 1\}$ to $Z \coloneqq \{\triangle, \square\}$.

Figure 2 The head H and the tail T of a cascade composition.

▶ **Observation 2.** *A Mealy machine T' is a replacement for T if and only if $\lambda_T(\overline{y}) = \lambda_{T'}(\overline{y})$ for all $\overline{y} \in \mathrm{Out}(H)$*

Because of Observation 1, there are exponential-time algorithms for Problem 1 and Problem 3: there is a straightforward ("naive") polynomial reduction of the n-replaceability problem into a satisfiability problem along the lines of bounded synthesis [8]. This encoding can be used to optimize the tail of a cascade composition by finding the minimum n for which the resulting CNF formula is satisfiable. The exponential complexity of this procedure contrasts with the double-exponential complexity of the K-N algorithm. This reasoning also applies to more general networks of Mealy machines [12, Chapter 6], implying that the approach for component minimization based on optimizing the flexibility [12, 32] is not optimal in theory, as it takes doubly exponential time.

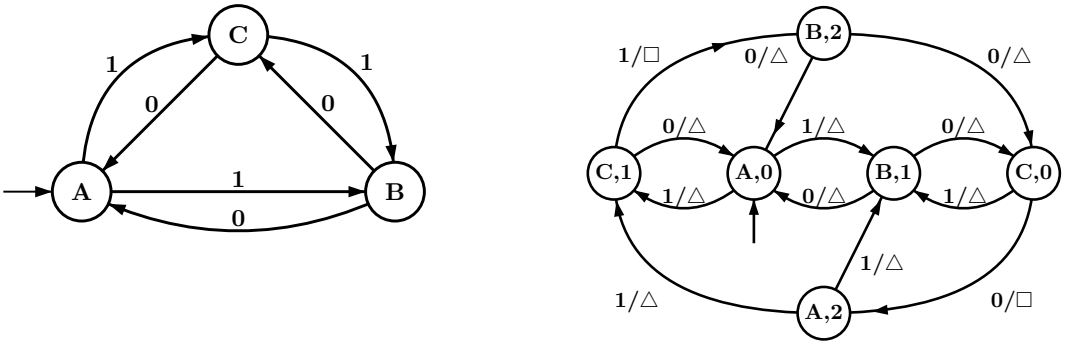
5.1 Proposed Modification

We give an overview of our minimization method here. The algorithm is divided in three steps: (1) We compute an NFA A which accepts the language $\mathrm{Out}(H)$, (2) using A and T we build an OM M whose set of implementations is precisely the set of replacements for T. By Observation 2, this is ensured by $\Omega_M = \mathrm{Out}(H)$ and $\lambda_M(\overline{y}) = \lambda_T(\overline{y})$ for all $\overline{y} \in \Omega_M$. Lastly, (3) we find an implementation T' of M with the minimum number of states. This machine T' is a minimal replacement for T.

This algorithm follows the K-N procedure but skips an expensive determinization step. Indeed, the difference is that in the K-N method the NFA A obtained in (1) is determinized before performing (2). This removes the branching from M, making it an IS machine, but it yields $|M| = O(2^{|H|}|T|)$ rather than $|M| = O(|H||T|)$ in our method. Note that this IS Machine M plays the role of the flexibility in the general approach (Section 3.1). Step (3), the last one, is responsible for the overall complexity of our algorithm as well as the K-N one, and takes $2^{|M|^{O(1)}}$ time. This makes the total time costs of our procedure and the K-N one $2^{(|H||T|)^{O(1)}}$ and $2^{(2^{|H|}|T|)^{O(1)}}$, respectively. We illustrate our method in Figure 3 and Figure 4, where we minimize the tail of the cascade composition shown in Figure 2.

The Image Automaton

Let H be a Mealy machine from X to Y. In this section we describe how to obtain an NFA whose language is $\mathrm{Out}(H)$ [13, 16]. The **image automaton** of H (sometimes called the inverse automaton), written $\mathrm{Im}(H)$, is the NFA over Y defined as follows: Let $S_{\mathrm{Im}(H)} \coloneqq S_H$ and $r_{\mathrm{Im}(H)} \coloneqq r_H$. We set $\Delta_{\mathrm{Im}(H)}(s_1, y) \coloneqq \{s_2 \in S_H \mid \exists x \in X \text{ s.t. } \lambda(s_1, x) = y, \delta(s_1, x) = s_2\}$. It holds that $\mathcal{L}(\mathrm{Im}(H)) = \mathrm{Out}(H)$. Essentially, to obtain $\mathrm{Im}(H)$ one deletes the input



(a) The image automaton $\mathrm{Im}(H)$ of the machine H in Figure 2a.

(b) The restriction $M \coloneqq T|_{\mathrm{Im}(H)}$ of the Machine T in Figure 2b to the automaton A in Figure 3a.

Figure 3 First and second step of the minimization of T in Figure 2

labels from H's transitions, as shown in Figure 3a. The time and space complexity of this construction is $O(|H|)$.

The Restriction Machine

Let A be an NFA over Y, and T a Mealy machine from Y to Z. In this section our goal is to build an OM M whose set of defined sequences Ω_M is precisely $\mathcal{L}(A)$ and that satisfies $\lambda_M(\overline{y}) = \lambda_T(\overline{y})$ for all $\overline{y} \in \Omega_M$. The **restriction of T to A**, denoted by $T|_A$, is the OM M from Y to Z defined as follows. Let $S_M \coloneqq S_T \times S_A$ and $r_M \coloneqq (r_T, r_A)$. Given a state $s_M \coloneqq (s_T, s_A) \in S_M$ and an input $y \in Y$ there are two possibilities: (1) $\Delta_A(s_A, y) \neq \emptyset$. In this case we mark the transition as defined $(s_M, y) \in D_M$, and set $\Delta_M(s_M, y) = \{(s'_T, s'_A) \mid s'_T = \delta_T(s_T, y), s'_A \in \Delta_A(s_A, y)\}$. Alternatively, (2) $\Delta_A(s_A, y) = \emptyset$. Here we just mark the transition as undefined $(s_M, y) \notin D_M$. It is direct to see that both $\Omega_M = \mathcal{L}(A)$ and $\lambda_M(\overline{y}) = \lambda_T(\overline{y})$ for all $\overline{y} \in \Omega_M$.

An example is given in Figure 3b. This product construction generalizes the one in the K-N algorithm: when A is deterministic, so is M, yielding an IS Mealy machine. The construction of $M \coloneqq T|_A$ can be performed in $O(|T||A|)$ time. In the case where $A \coloneqq \mathrm{Im}(H)$ results from the head H of a cascade, we can substitute $|A| = O(|H|)$.

Reduction to a Covering Problem

Let M be an OM from Y to Z. In this section we describe how to find a minimal implementation of M. In order to do this, we generalize the theory of [10] for minimization of IS machines in a natural way. This generalization can be used to extend existing minimization algorithms to our setting. To illustrate this point, in Appendix C we describe a modification of MeMin, an algorithm for the minimization of IS machines. Additionally, we describe how to use this modification in the Tail Minimization Problem following our approach and provide some preliminary experimental evaluation.

Following [10] the basic idea to obtain a smallest implementation of M is to define a compatibility relation $\sim$ over S_M and then use it to reduce the task to a covering problem over S_M. Two states $s_1, s_2 \in S_M$ are **compatible**, written $s_1 \sim s_2$, if $\lambda_M(s_1, \overline{y}) = \lambda_M(s_2, \overline{y})$ for all $\overline{y} \in \Omega_M(s_1) \cap \Omega_M(s_2)$. We note that this relation between states is symmetric and reflexive, but not necessarily transitive. A set $Q \subseteq S_M$ is called a **compatible** if all its states are pairwise compatible. We say that Q is **incompatible at depth** k if k is the

length of the shortest word $\overline{y} \in Y^*$ such that $\overline{y}$ is defined for two states $s_1, s_2 \in Q$, and $\lambda_M(s_1, \overline{y}) \neq \lambda_M(s_2, \overline{y})$. A convenient characterization is as follows:

▶ **Lemma 3.** *Let $Q \subseteq S_M$. The following statements hold: (1) Q is incompatible at depth 1 if and only if for some $s_1, s_2 \in Q$ there is a defined input $y \in Y$ with $\lambda_M(s_1, y) \neq \lambda_M(s_2, y)$. (2) If Q is incompatible at depth $i > 1$ then $\Delta_M(Q, y)$ is incompatible at depth $i - 1$ for some $y \in Y$.*

Lemma 3 gives a straightforward way to compute the compatibility relation $\sim$ over S_M. We begin by finding all pairs s_1, s_2 incompatible at depth 1. Afterwards we propagate the incompatible pairs backwards: if $s_1 \nsim s_2$ and $s_1 \in \Delta_M(s_1', y), s_2 \in \Delta_M(s_2', y)$, then $s_1' \nsim s_2'$. This process can be carried out in $O(|M|^2)$ time (see Appendix A for a reduction to Horn SAT). When $M = T|_{\mathrm{Im}(H)}$, it holds $|M| = O(|H||T|)$, and the required time to compute the $\sim$ relation over B is $O(|H|^2|T|^2)$.

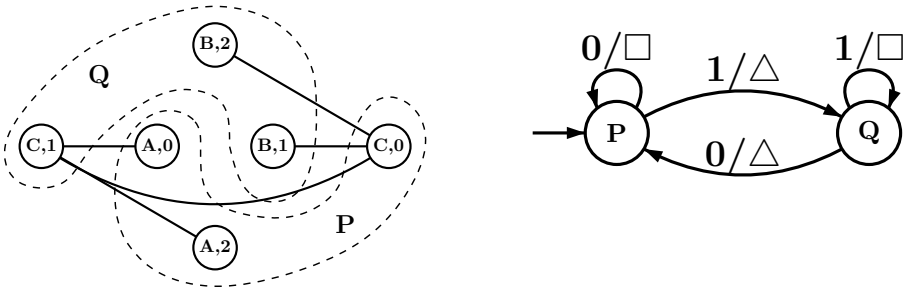
Let $F \subseteq 2^S_M$ be a family where all $C \in F$ are compatibles. We call F a **closed cover of compatibles over** M if the following are satisfied: (1) $r_M \in C$ for some $C \in F$, and (2) for any $C \in F$ and $y \in Y$ there is some $C' \in F$ such that $\Delta(C, y) \subseteq C'$. The following theorem implies that the problem of finding a minimal implementation for M is polynomially equivalent to the problem of finding a closed cover of compatibles over it with the minimum size. This equivalence is illustrated in Figure 4a.

▶ **Theorem 4.** *Let M be an OM from Y to Z. Let $|M| \coloneqq \sum_{s,y} |\Delta_B(s, y)|$ be the number of transitions in M. The following two statements hold: (1) Let N be an implementation of M. Then it is possible to build a closed cover of compatibles F over M with $|F| \leq |S_N|$ in $O(|M||N|)$ time. (2) Let F be a closed cover of compatibles over M. Then it is possible to build an implementation N of M with $|S_M| = |F|$ in $O(|M||F|)$ time.*

Proof. We prove (1) and (2) separately. **(1):** Let N be an implementation of M. For each $s_N \in S_N$ we define the set $Q(s_N) \subseteq S_M$ as follows: $Q(s_N) = \{\, s_M \in S_M \mid \exists \overline{y} \in \Omega_M \text{ s.t. } \delta_N(\overline{y}) = s_N,\ s_M \in \Delta_M(\overline{y}) \,\}$. It holds that each $Q(s_N)$ is a compatible, as the output of each state $s_M \in Q(s_N)$ over a defined input sequence has to coincide with the output of s_N in N. Moreover, $r_M \in Q(r_N)$, and for any $s_N \in S_N$, $y \in Y$ it holds that $\Delta_M(Q(s_N), y) \subseteq Q(\delta_M(s_N, y))$. Thus, the family $F \coloneqq \{Q(s_N) \mid s_N \in S_N\}$ is a closed cover of compatibles over N, and $|F| \leq |S_N|$. Note that $|F|$ may be strictly lesser than $|S_N|$, as for some $s_N^1, s_N^2 \in S_M$ it may happen that $Q(s_M^1) = Q(s_M^2)$. The sets $Q(s_N)$, and with them the family F, can be derived from a product construction between M and N which takes $O(|M||N|)$ time. **(2):** Let F be a closed cover of compatibles over M. We build an implementation N of M: Set $S_N \coloneqq F$. Thus, each state in N corresponds to compatible $C \in F$. We define r_N as an arbitrary $C \in F$ satisfying $r_M \in C$. Let $C \in F$ and $y \in Y$. We define $\delta_N(C, y)$ as an arbitrary $C' \in F$ such that $\Delta_M(C, y) \subset C'$. To define $\lambda_N(C, y)$ we take into account two possibilities: (1) $(s_M, y) \in D_M$ for some $s_M \in C$. Then we set $\lambda_N(C, y) = \lambda_M(s_M, y)$. Note that $\lambda_N(C, y)$ is independent of the particular choice of s_M, as M being consistent implies that all choices yield the same output. (2) $(s_M, y) \notin D_M$ for all $s_M \in C$. In this case put an arbitrary output for $\lambda_N(C, y)$. It can be checked that N is indeed an implementation of M (Appendix B). The Mealy machine N given by this construction satisfies $|S_N| = |F|$, and can be built in $O(|M||N|)$ time and space. ◀

6 Complexity the Minimization Problems

As evidenced in Observation 1, deciding whether the tail (head) in a cascade composition is n-replaceable is an NP problem. Here we show that this bound is tight, meaning that the



(a) The relation of the OM M in Figure 3b, together with a closed cover of compatibles P and Q

(b) A Mealy machine T' corresponding to the cover in Figure 4a.

Figure 4 Final step of the minimization of T in Figure 2. The machine T' in (b) is a minimal replacement for T.

problem is NP-hard as well. To the best of our knowledge, these complexity results have not been shown elsewhere.

6.1 Tail Minimization

▶ **Theorem 5.** *Deciding whether the tail in a cascade composition is n-replaceable is an NP-hard problem.*

Proof. Let N be an IS machine and let $n \in \mathbb{N}$. We build machines H and T in polynomial time satisfying that N is n-implementable if and only if T is n-replaceable in $T \circ H$. As deciding whether N is n-implementable is an NP-complete problem [23], this reduction proves the theorem. Informally, the aim is to build H whose output language coincides with Ω_N, and use an arbitrary implementation of N as T. This idea does not quite work because it requires Ω_N to have no maximal words, which may not be the case, but this can be solved adding some extra output symbols and transitions. Let Y and Z be the input and output alphabets of N respectively. Let $\hat{Y} \coloneqq Y \cup \{\bot\}$, $\hat{Z} \coloneqq Z \cup \{\bot\}$, where $\bot$ is a fresh symbol. We begin by building a machine H from Y to $\hat{Y}$. We set $S_H \coloneqq S_N \cup \{*\}$, where $*$ is a fresh state, and $r_H \coloneqq r_N$. Given $(s, y) \in S_N \times Y$, we define $\delta_H(s, y) \coloneqq \delta_N(s, y)$ and $\lambda_H(s, y) \coloneqq \lambda_N(s, y)$ if $(s, y) \in D_N$, and $\delta_H(s, y) \coloneqq *$ together with $\lambda_H(s, y) \coloneqq \bot$ otherwise. We also define $\delta_H(*, y) \coloneqq *$ and $\lambda_H(*, y) \coloneqq \bot$ for all y. It is direct to see that $\mathrm{Out}(H) = \Omega_N\{\bot\}^*$. Now we build another machine T from $\hat{Y}$ to $\hat{Z}$. Let N' be an arbitrary implementation of N, built in polynomial time simply by adding the missing transitions. To construct T we add self loops $\delta_T(s, \bot) = s$, $\lambda_T(s, \bot) = \bot$ to N' at each state $s \in S_{N'}$.

Now we show that T is n-replaceable in $T \circ H$ if and only if N is n-implementable. As exposed in Observation 2, a machine T' is a valid replacement for T if and only if for all $\overline{y} \in \mathrm{Out}(H)$ $\lambda_{T'}(\overline{y}) = \lambda_T(\overline{y})$. Moreover, any word $\overline{y} \in \mathrm{Out}(H)$ is of the form $\overline{u}(\bot)^k$, where $\overline{u} \in \Omega_N$. By construction $\lambda_T(\overline{u}(\bot)^k) = \lambda_N(\overline{u})(\bot)^k$. This implies the following: (1) Let T' be a replacement for T. Then removing all transitions on input $\bot$ from T' yields an implementation of N of the same size. (2) Let T' be an implementation of N. Then adding self-loops $\delta_T(s, \bot) = s$, $\lambda_T(s, \bot) = \bot$ to all states of T' yields a replacement for T in $T \circ H$ with the same number of states. This proves the result. ◀

6.2 Head Minimization

Our main result here, Theorem 8, is a polynomial-time reduction from the Boolean satisfiability problem. We review the necessary concepts now. A **literal** is an expression of the form x or $\neg x$, for some Boolean variable x. A **clause** C is just a (finite) set of literals. A **CNF formula** $\mathcal{F}$ over n variables is a (finite) family $\{C_0, \ldots, C_{m-1}\}$ of clauses where the Boolean variables are among $x_0, \ldots, x_{n-1}$. We consider only formulas $\mathcal{F}$ whose clauses are **non-tautological**, meaning that no clause in $\mathcal{F}$ contains a pair of literals of the form $x, \neg x$, for some variable x. We define an **assignment** of n Boolean variables as word $\overline{\sigma} \in \{-1, 1\}^n$. We say that this assignment **satisfies** a clause C over $x_0, \ldots, x_{n-1}$ if either $\sigma_i = 1$ for some $x_i \in C$, or $\sigma_i = -1$ for some $\neg x_i \in C$. A CNF formula $\mathcal{F}$ over n variables is **satisfiable** if there is some assignment that satisfies all its clauses.

Fix a number of variables $n > 0$ for the rest of the section. The idea behind our reduction in Theorem 8 is that we can represent assignments $\overline{\sigma}$ as heads $H_{\overline{\sigma}}$ and formulas $\mathcal{F}$ as tails $T_{\mathcal{F}}$ in a way that the composition $T_{\mathcal{F}} \circ H_{\overline{\sigma}}$ exhibits a unique special behaviour when $\overline{\sigma}$ satisfies $\mathcal{F}$. We define the following alphabets $X := \{\bot\}$, $Y := \{-1, 1, \bot\}$, $Z := \{\bot, \top\}$. In this section we consider compositions $T \circ H$ where H is a machine from X to Y and T from Y to Z. Note that both H and the cascade $T \circ H$ can only produce a unique string of outputs each. We give now two mappings: one from assignments $\overline{\sigma} \in \{-1, 1\}^n$ to machines $H_{\overline{\sigma}}$ from X to Y, and another from formulas $\mathcal{F}$ over n variables to machines $T_{\mathcal{F}}$ from Y to Z.

The machine $H_{\overline{\sigma}}$ simply outputs the values σ_i in succession, before producing $\bot$ in the $(n+1)$-th step and returning to its initial state. More formally, we define $S_{H_{\overline{\sigma}}} := [n+1]$, $r_{H_{\overline{\sigma}}} := 0$. For the transition and output functions we define $\delta_{H_{\overline{\sigma}}}(i, \bot) = i + 1$, $\lambda_{H_{\overline{\sigma}}}(i, \bot) = \sigma_i$ for all $i \in [n]$, together with $\delta_{H_{\overline{\sigma}}}(n, \bot) = 0$ and $\lambda_{H_{\overline{\sigma}}}(n, \bot) = \bot$.

The construction of $T_{\mathcal{F}}$ is more involved. We give here a high-level description. The details follow in a fairly direct way, and can be found at Appendix H. Let $\mathcal{F} = \{C_0, \ldots, C_{m-1}\}$. To describe $T_{\mathcal{F}}$ we specify its behaviour over an arbitrary input string $\overline{y} \in Y^*$ of length $\ell := (m+2)(n+1)$. We say that $\overline{y} := y_0 \ldots y_{\ell-1}$ has the **intended format** if $y_i = \bot$ whenever $i = n \pmod{n+1}$, and $y_i \in \{-1, 1\}$ for the other values of i. Whenever $\overline{y}$ has the intended format, it can be written as $\overline{\sigma}^0 \bot \overline{\sigma}^1 \bot \ldots \bot \overline{\sigma}^{m+1} \bot$, where each $\overline{\sigma}^i$ is an assignment of n variables. In this case, we write $y = [\overline{\sigma}^0, \ldots, \overline{\sigma}^{m+1}]$. We specify various properties of $T_{\mathcal{F}}$ now: (I) The machine $T_{\mathcal{F}}$ has two sink states Sat and UnSat, which only output $\top$ and $\bot$ respectively. (II) The other states in $T_{\mathcal{F}}$ only output $\bot$ as well, so $T_{\mathcal{F}}$ only can produce $\top$ from the state Sat. (III) When $T_{\mathcal{F}}$ reads a string $\overline{y} = [\overline{\sigma}^0, \ldots, \overline{\sigma}^{m+1}]$ with the intended format, it ends up in Sat if $\overline{\sigma}^{i+2}$ satisfies the clause C_i in $\mathcal{F}$, for all $0 \leq i < m$. If this does not hold, $T_{\mathcal{F}}$ reaches UnSat. (IV) In the alternative case, whenever the string $\overline{y} \in Y^\ell$ deviates from the intended format, $T_{\mathcal{F}}$ immediately moves to UnSat. It holds that a machine $T_{\mathcal{F}}$ satisfying (I)-(IV) with no more than 2ℓ states can be built in time polynomial in ℓ (see the explicit construction at Appendix H).

▶ **Lemma 6.** *Let $\mathcal{F} = \{C_0, \ldots, C_{m-1}\}$ be a CNF over n variables, let $\overline{\sigma} \in \{-1, 1\}^n$ be an assignment, and let $\ell := (m+1)(n+2)$. Then the following two statements are equivalent: (1) the cascade $T_{\mathcal{F}} \circ H_{\overline{\sigma}}$ outputs $\bot$ during the first ℓ steps, and $\top$ from that point on, and (2) $\overline{\sigma}$ satisfies $\mathcal{F}$.*

Proof. Let $\overline{y} := \lambda_{H_{\overline{\sigma}}}(\bot^\ell)$ be the first ℓ outputs of $H_{\overline{\sigma}}$. Clearly (1) holds if and only if $\delta_{T_{\mathcal{F}}}(\overline{y}) = \text{Sat}$. By construction of $H_{\overline{\sigma}}$, $\overline{y} = [\overline{\sigma}, \overline{\sigma}, ..., \overline{\sigma}]$. That is, $\overline{y}$ consists of $m+2$ copies of $\overline{\sigma}$ with $\bot$ in between as a separating character. This sequence follows the intended format, so $\delta_{T_{\mathcal{F}}}(\overline{y}) = \text{Sat}$ if and only if $\overline{\sigma}$ satisfies each of the clauses $C_i \in \mathcal{F}$, meaning that $\overline{\sigma}$ satisfies $\mathcal{F}$ itself. This proves the result. ◀ ◀

▶ **Lemma 7.** *Let $\mathcal{F} = \{C_0, \ldots, C_{m-1}\}$ be a CNF over n variables and $\ell := (m+1)(n+2)$. The following statements are equivalent: (1) there exists a Mealy machine H from X to Y with $|S_H| = n+1$ satisfying that $T_{\mathcal{F}} \circ H$ outputs $\top$ from the $(\ell+1)$-th step on, and (2) the formula $\mathcal{F}$ is satisfiable.*

Proof. The fact that (2) implies (1) follows directly from last lemma: given a satisfying assignment $\overline{\sigma}$, clearly (1) holds for $H = H_{\overline{\sigma}}$. We show that (1) implies (2) as well. Let H be a Mealy machine witnessing (1). Let $\overline{y} = \lambda_H(\bot^{2(n+1)})$. Necessarily $\overline{y}$ follows the intended format. Otherwise we would have $\delta_{T_{\mathcal{F}}}(\overline{y}) = \textsc{UnSat}$. Thus $\overline{y}$ is of the form $[\overline{\sigma}, \overline{\sigma}']$ for two assignments $\overline{\sigma}, \overline{\sigma}' \in \{-1, 1\}^n$. As H has $n+1$ states, it has to be that $\overline{\sigma} = \overline{\sigma}'$ and H equals $H_{\overline{\sigma}}$ (up to isomorphism). As $H_{\overline{\sigma}}$ witnesses (1), using last lemma we obtain that $\overline{\sigma}$ satisfies $\mathcal{F}$ and (2) holds, showing the result. ◀

▶ **Theorem 8.** *Deciding whether the head in a cascade composition is n-replaceable is an NP-hard problem.*

Proof. As stated in the beginning of this section, we give a reduction from the Boolean satisfiability problem. Given a CNF formula $\mathcal{F} := \{C_0, \ldots, C_{m-1}\}$ over n variables we give a polynomial-time construction of Mealy machines H and T, from X to Y and from Y to Z respectively, such that H is $n+1$-replaceable in $T \circ H$ if and only if $\mathcal{F}$ is satisfiable. For T we simply take $T_{\mathcal{F}}$, as defined above. As stated during its description, the construction of $T_{\mathcal{F}}$ takes time polynomial in the size of $\mathcal{F}$ and the number of variables n. Let $\mathcal{F} = \{C_0, \ldots, C_{m-1}\}$ Let $\ell := (m+2)(n+1)$, as before. For H, we take an arbitrary machine where $\lambda_H(\bot^{\ell}) = [\overline{\sigma}^0, \ldots, \overline{\sigma}^{m+1}]$ for some assignments $\overline{\sigma}^i \in \{-1, 1\}^n$, where $\overline{\sigma}^{2+i}$ satisfies the clause C_i for each $0 \leq i < m$. By construction, $T_{\mathcal{F}}$ reaches the state $\textsc{Sat}$ after reading the first ℓ outputs from H, and so $T_{\mathcal{F}} \circ H$ emits the symbol $\top$ from step $\ell+1$ onwards. By the last lemma H is $(n+1)$-replaceable in $T_{\mathcal{F}} \circ H$ if and only if $\mathcal{F}$ is satisfiable. ◀

7 Complexity of the Synthesis Problems

During this part of the paper we study the Tail and Head Synthesis Problems, giving lower complexity bounds for both of them that coincide with the cost of existing methods.

7.1 Tail Synthesis

We consider the Tail Synthesis Problem, where both the head H and the system model M are given. The state-of-the art for this problem is represented by the general approach described in Section 3.1. Here the flexibility F is a DFA exponentially larger than H, M and both deciding the feasibility of the synthesis problem as well as computing a solution when one exists take time linear in the size of F. Our main result here is the fact that while the Feasibility Problem has polynomial time complexity, there are instances of the Tail Synthesis Problem where minimal solutions have exponential size, and hence the problem itself has exponential complexity, matching the cost of the general method. The proof of this result relies on a construction that, given a feasible instance of the Synthesis Problem, produces an OM that represents all its solutions.

Feasibility of the Synthesis Problem

We begin by characterizing the feasible instances of the Tail Synthesis Problem. Let H and M be Mealy machines from X to Y and from X to Z respectively. A solution T to the synthesis problem (that is, $T \circ H \equiv M$) must satisfy $\lambda_T(\lambda_H(\overline{x})) = \lambda_M(\overline{x})$ for all $\overline{x} \in X^*$. In

particular, if a solution T exists, then there cannot be two words $\overline{x}, \overline{x}'$ with $\lambda_H(\overline{x}) = \lambda_H(\overline{x}')$ but $\lambda_M(\overline{x}) \neq \lambda_M(\overline{x}')$. We argue that the converse holds as well.

▶ **Proposition 9.** *There exists a Mealy machine T with $T \circ H \equiv M$ if and only if for any two words $\overline{x}, \overline{x}' \in X^*$ with $\lambda_H(\overline{x}) = \lambda_H(\overline{x}')$ it holds $\lambda_M(\overline{x}) = \lambda_M(\overline{x}')$ as well.*

Proof. See Appendix D. ◀

As a consequence of this result, deciding the feasibility of the Tail Synthesis Problem given by H and M is equivalent to checking whether $\lambda_H(\overline{x}) = \lambda_H(\overline{x}')$, but $\lambda_T(\overline{x}) \neq \lambda_T(\overline{x}')$ for some $\overline{x}, \overline{x}' \in X^*$. The existence of such words $\overline{x}, \overline{x}'$ can be easily computed in $O(|H|^2|M|^2)$ time via a fixed point procedure on the synchronous product of H and M.

Representing all Solutions

We give the construction of an OM with size $O(|H||M|)$ encoding all solutions for a feasible instance of the Tail Synthesis Problem. We note that this is an exponentially more succinct representation of the solutions than the E-machine from [32]. In fact, this construction is equivalent to the NDE-machine introduced in [12, 31] but with universal acceptance conditions, rather than non-deterministic ones.

We define an OM N from Y to Z as follows. Let $S_N \subseteq S_H \times S_M$ be the set of pairs (s_H, s_M) that are reachable in the synchronous product $H \times M$. That is, those satisfying $\delta_H(\overline{x}) = s_H$ and $\delta_M(\overline{x}) = s_M$ for some $\overline{x} \in X^*$. Let $r_N \coloneqq (r_H, r_M)$. We define the transition and output functions Δ_N, λ_N for N. Fix a transition $(s_H, s_M) \in S_N, y \in Y$. Let V be the set of $x \in X$ satisfying $\lambda_H(s_H, x) = y$. We take two cases into consideration. If the set V is empty, then we set the transition as undefined $((s_H, s_M), y) \notin D_N$. Otherwise, if $V \neq \emptyset$, we mark the transition as defined $((s_H, s_M), y) \in D_N$. In this case $\lambda_M(s_M, x)$ takes a unique value z for all $x \in V$. Indeed, the opposite would yield two sequences $\overline{x}, \overline{x}' \in X^*$ with $\lambda_H(\overline{x}) = \lambda_H(\overline{x}')$ and $\lambda_H(\overline{x}) \neq \lambda_M(\overline{x}')$, making the Tail Synthesis Problem given by H and M infeasible. Hence, we can define $\lambda_N((s_H, s_M), y)$ as z, and $\Delta_N((s_H, s_M), y)$ as $\{ (s'_H, s'_M) \mid \exists x \in V \text{ s.t. } s'_H = \delta_H(s_H, x),\ s'_M = \delta_M(s_M, x) \}$.

▶ **Proposition 10.** *Let N be the OM defined above. A machine T satisfies $T \circ H \equiv M$ if and only if T implements N.*

Proof. The machine T satisfies $T \circ H \equiv M$ if and only if for all $\overline{y} \in \mathrm{Out}(H)$ and $\overline{x} \in X^*$ with $\lambda_H(\overline{x}) = \overline{y}$, it holds $\lambda_T(\overline{y}) = \lambda_M(\overline{x})$. Note that by construction $\Omega_N = \mathrm{Out}(H)$. Moreover, there is a run of N over $\overline{y}$ whose output is $\overline{z}$ if and only if there is some $\overline{x} \in X^*$ satisfying $\lambda_H(\overline{x}) = \overline{y}$ and $\lambda_M(\overline{x}) = \overline{z}$. This shows the result. ◀

Exponentially Large Tails

In this section we show that there are instances of the synthesis problem (Problem 2) where all solutions have at least exponential size.

▶ **Theorem 11.** *There exist finite alphabets X, Y, Z and families of increasingly large Mealy machines $\{M_n\}_n$, from X to Y, and $\{H_n\}_n$, from X to Z, for which the size of any T_n satisfying $T_n \circ H_n \equiv M_n$ is bounded from below by an exponential function of $|M_n||H_n|$.*

The proof of this result has two parts. First, we show an infinite family of OMs for which all implementations have exponential size (Lemma 12). Afterwards we prove that any OM N can be "split" into Mealy machines M and H with the same number of states

(plus one) as N for which any T with $T \circ H \equiv M$ provides an implementation of N (Lemma 13). These two results together prove Theorem 11. Some care has to be employed in order to obtain fixed alphabets in Theorem 11. The alphabets for H and T in the splitting construction of Lemma 13 depend on the alphabets of N and its **degree**, defined as $d(N) \coloneqq \max_{(s,y)\in D_N} |\Delta_N(s,y)|$. Hence, it is necessary to ensure in Lemma 12 that the resulting OMs have bounded degree.

▶ **Lemma 12.** *There are finite alphabets Y, Z and increasingly large OMs $\{N_n\}_n$ from Y to Z for which the size of a machine T_n implementing N_n is bounded from below by an exponential function of $|N_n|$. Moreover, it is possible to build N_n in such a way that* $\max_n d(N_n) = 2$

Proof. See Appendix E. ◀

▶ **Lemma 13.** *Let N be a consistent OM from Y to Z, and let $k \coloneqq d(N)$. Let $X \coloneqq Y \times [k]$, $\hat{Y} = Y \cup \{\bot\}$ and $\hat{Z} = Z \cup \{\bot\}$, where $\bot$ is a fresh symbol. Then there exist Mealy machines H from X to $\hat{Y}$, and M from X to $\hat{Z}$, such that (1) $|S_H|, |S_M| = |S_N| + 1$, (2) there are machines T with $T \circ H \equiv M$, and (2) any of those machines T satisfy $\lambda_T(\overline{y}) = \lambda_N(\overline{y})$ for all $\overline{y} \in \Omega_N$.*

Proof. We give an explicit construction for H and M. We define the edge set of N, $G_N \subset S_N \times Y \times S_N$, as the set consisting of the triples (s, y, s') where $(s, y) \in D_N$ and $s' \in \Delta_N(s, y)$. By the definition of k, we can build a map $L : G_N \to [k]$ satisfying $L(s, y, t_1) \neq L(s, y, t_2)$, for any $(s, y) \in D_T$ and any two different states $t_1, t_2 \in \Delta_N(s, y)$. The map L assigns a number $0 \leq i < k$ to each edge $(s, y, s') \in G_N$, giving different labels to each edge corresponding to a pair $(s, y) \in D_N$. We define H and M at the same time. Set $S_H, S_M \coloneqq S_N \cup \{*\}$, where $*$ is a fresh state, and $r_H, r_M \coloneqq r_N$. Let $s \in S_N \cup \{*\}$ and let $x \coloneqq (y, i) \in X = Y \times [k]$. We take into account three cases: (1) If $s = *$, then $\delta_H(*, x), \delta_M(*, x) = *$, and $\lambda_H(*, x), \lambda_M(*, x) = \bot$. (2) If $s \in S_N$ and there exists some $t \in S_N$ with $L(s, y, t) = i$, then $\delta_H(s, x), \delta_M(s, x) = t$, $\lambda_H(s, x) = y$, and $\lambda_M(s, x) = \lambda_N(s, y)$. Finally, (3) if $s \in S_N$ and there is no t with $L(s, y, t) = i$, then $\delta_H(s, x), \delta_M(s, x) = *$, $\lambda_H(s, x), \lambda_M(s, x) = \bot$. We claim that H and M built this way satisfy the theorem's statement (see Appendix F). ◀

7.2 Head Synthesis

Next, we study the Head Synthesis Problem with a given tail T and a system model M. For completeness, we show that the cost of the existing techniques matches the complexity of the problem. Here, the procedure shown in [4] obtains the flexibility F for H through a product construction between T and M. This way, obtaining a solution for the synthesis problem, if it exists, takes $O(|S_T||S_M|)$ time. We show that there are instances of the problem where solutions H have at least $|S_T||S_M|$ states, giving a tight complexity bound for the problem.

▶ **Theorem 14.** *Let $X \coloneqq \{\bot\}, Y, Z \coloneqq \{0, 1\}$. There are sequences of increasingly large machines $\{T_n\}_n$ and $\{M_n\}_n$ from Y to Z and from X to Y respectively satisfying the following: (1) there exists a machine H_n from X to Y satisfying $T_n \circ H_n \equiv M_n$, and (2) all machines H witnessing (1) have at least $|S_{T_n}||S_{M_n}|$ states.*

Proof. Let $\{p_n\}_n$ and $\{q_n\}_n$ be sequences of increasingly large natural numbers satisfying both that p_n and q_n are co-primes as well as $p_n < q_n$ for all n. The machine M_n is a cycle of size q_n that emits 1 every q_n steps and 0 otherwise. That is, (1) $S_{M_n} \coloneqq \{s_0, \dots s_{q_n-1}\}$, (2) $r_{M_n} \coloneqq s_0$, (3) $\delta_{M_n}(s_i, \bot) \coloneqq s_j$ where $j = i + 1 \pmod{q_n}$ for all i, and (4) $\lambda_{M_n}(s_i, \bot) \coloneqq 0$,

if $i \neq q_n - 1$, or 1 if $i = q_n - 1$. The machine T_n is defined similarly. It consists of a p_n-cycle where each state outputs whichever input it receives except for the last one, which flips the input. This way, (1) $S_{T_n} \coloneqq \{t_0, \dots t_{p_n-1}\}$, (2) $r_{T_n} \coloneqq t_0$, (3) $\delta_{T_n}(t_i, \bot) \coloneqq t_j$ where $j = i + 1 \pmod{p_n}$ for all i, and (4) for all $y = 0, 1$ $\lambda_{M_n}(s_i, y) \coloneqq y$, if $i \neq p_n - 1$, or $1 - y$ if $i = p_n - 1$.

Fix $n > 0$. We define the sequence $\overline{y} \in \{0,1\}^{p_n q_n}$ by setting $y_i \coloneqq 0$ whenever both $i \neq -1 \pmod{q_n}$ and $i \neq -1 \pmod{p_n}$ or both $i = -1 \pmod{q_n}$ and $i = -1 \pmod{p_n}$. Otherwise we set $y_i \coloneqq 1$. A machine H_n satisfies $T_n \circ H_n \equiv M_n$ if and only if it just outputs $\overline{y}$ periodically. This clearly can be done, but $\overline{y}$ has no periodic sub-sequence, so H has to contain at least $p_n q_n$ states. This shows the result. ◀

8 Other Compositions

The results and techniques exposed so far pertain to component optimization and synthesis in cascade compositions. We briefly mention here how these findings relate to the more general setting of arbitrary compositions.

Firstly, our results about the Tail Minimization and Synthesis extend to any composition of two components H and T where the outputs of the component under consideration T are fully observable. Without loss of generality we can assume that (1) T's input signals coincide with H's output signals, (2) the system's output signals coincide with T's output signals, and (3) H's input signals are the system's input signals plus T's output ones. This situation is depicted in Figure 1b. We claim that the minimization and synthesis of T in this context can be polynomially reduced to those of the tail in a cascade composition. The reductions are similar to the ones in [29], and can be found in Appendix G.

The Component Minimization Problem lies also in NP for arbitrary two-component compositions by the same reasoning as in Observation 1. When the composition is non-trivial (i.e., there is some communication between the components), the problem is easily seen to be NP-complete. For this, note that it generalizes either the Tail Minimization Problem, if the component under optimization has some observable outputs, or the Head Minimization Problem, in the alternative case.

9 Conclusions and Future Work

In this paper we have shown that there was a surprising discrepancy between the complexity of the Tail Minimization Problem and the cost of current solutions, and we have proposed a modification of the existing main technique which is exponentially less costly. This study can also be carried out in more general versions of the Component Minimization Problem where this mismatch also occurs. Additionally, one of the main desired outcomes of this research should be the development of practically efficient solutions.

Additionally, we have shown tight complexity bounds for both the Minimization and the Synthesis problem in cascade compositions. Among other things, we proved that the Component Synthesis Problem has polynomial cost for the head, but for the tail this complexity is exponential. However, the complexity of the problem in other architectures is left unaddressed. Whether those are the two only possible complexity classes and the exact conditions under which each of them occurs are open questions. These should be the object of future research.

References

1 Andreas Abel and Jan Reineke. MEMIN: SAT-based exact minimization of incompletely specified Mealy machines. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 94–101. IEEE, 2015. doi:10.1109/ICCAD.2015.7372555.

2 Andreas Abel and Jan Reineke. Gray-box learning of serial compositions of mealy machines. In Sanjai Rayadurgam and Oksana Tkachuk, editors, *NASA Formal Methods*, Lecture Notes in Computer Science, page 272–287. Springer International Publishing, 2016. doi:10.1007/978-3-319-40648-0_21.

3 Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *IJCAI*, pages 3308–3314, 2015.

4 M. D. Di Benedetto, A. Sangiovanni-Vincentelli, and T. Villa. Model matching for finite-state machines. *IEEE Transactions on Automatic Control*, 46(11):1726–1743, 2001. doi:10.1109/9.964683.

5 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking*, pages 921–962. Springer, 2018.

6 Roderick Bloem, Robert Könighofer, and Martina Seidl. Sat-based synthesis methods for safety specs. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 1–20. Springer, 2014.

7 William F Dowling and Jean H Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984.

8 Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013. doi:10.1007/s10009-012-0228-z.

9 Seymour Ginsburg. On the reduction of superfluous states in a sequential machine. *J. ACM*, 6(2):259–282, 1959. doi:10.1145/320964.320983.

10 A. Grasselli and F. Luccio. A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks. *IEEE Transactions on Electronic Computers*, EC-14(3):350–359, June 1965. doi:10.1109/PGEC.1965.264140.

11 Gary D. Hachtel and Fabio Somenzi. *Logic synthesis and verification algorithms*. Kluwer, 1996.

12 M.S. Harris. Synthesis of finite state machines: Functional optimization. *Microelectronics Journal*, 29(6):364–365, 1998. doi:10.1016/S0026-2692(97)00075-X.

13 Frederick C Hennie. *Finite-state models for logical machines*. Wiley, 1968.

14 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

15 Huey-Yih Wang and R. K. Brayton. Multi-level logic optimization of FSM networks. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 728–735, November 1995. doi:10.1109/ICCAD.1995.480254.

16 Joonki Kim and M.M. Newborn. The Simplification of Sequential Machines with Input Restrictions. *IEEE Transactions on Computers*, C-21(12):1440–1443, 1972. doi:10.1109/T-C.1972.223521.

17 June-Kyung Rho and F. Somenzi. Don't care sequences and the optimization of interacting finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(7):865–874, 1994. doi:10.1109/43.293943.

18 T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. A fully implicit algorithm for exact state minimization. In *31st Design Automation Conference*, page 684–690, Jun 1994. doi:10.1145/196244.196615.

19 Andreas Morgenstern, Manuel Gesell, and Klaus Schneider. Solving games using incremental induction. In *International Conference on Integrated Formal Methods*, pages 177–191. Springer, 2013.

20 Marvin C. Paull and Stephen H. Unger. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Trans. Electron. Comput.*, 8(3):356–367, 1959. doi:10.1109/TEC.1959.5222697.

21 J.M. Pena and A.L. Oliveira. A new algorithm for exact reduction of incompletely specified finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(11):1619–1632, Nov./1999. doi:10.1109/43.806807.

22 Alexandre Petrenko and Florent Avellaneda. Learning communicating state machines. In Dirk Beyer and Chantal Keller, editors, *Tests and Proofs*, Lecture Notes in Computer Science, page 112–128. Springer International Publishing, 2019. doi:10.1007/978-3-030-31157-5_8.

23 C. P. Pfleeger. State reduction in incompletely specified finite-state machines. *IEEE Transactions on Computers*, C–22(12):1099–1102, Dec 1973. doi:10.1109/T-C.1973.223655.

24 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '89, page 179–190. Association for Computing Machinery, Jan 1989. doi:10.1145/75277.75293.

25 June-Kyung Rho, Gary D Hachtel, Fabio Somenzi, and Reily M Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE transactions on computer-aided design of integrated circuits and systems*, 13(2):167–177, 1994.

26 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

27 Stephen H Unger. Flow table simplification-some useful aids. *IEEE Transactions on Electronic Computers*, EC-14(3):472–475, 1965.

28 Tiziano Villa, Nina Yevtushenko, Robert K Brayton, Alan Mishchenko, Alexandre Petrenko, and Alberto Sangiovanni-Vincentelli. *The unknown component problem: theory and applications*. Springer Science & Business Media, 2011.

29 H. Wang and R. K. Brayton. Input don't care sequences in fsm networks. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, page 321–328, Nov 1993. doi:10.1109/ICCAD.1993.580076.

30 Y. Watanabe and R. K. Brayton. State minimization of pseudo non-deterministic FSMs. In *Proceedings of European Design and Test Conference EDAC-ETC-EUROASIC*, pages 184–191, 1994. doi:10.1109/EDTC.1994.326878.

31 Yosinori Watanabe. *Logic Optimization of Interacting Components in Synchronous Digital Systems*. PhD thesis, EECS Department, University of California, Berkeley, Apr 1994. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1994/2554.html.

32 Yosinori Watanabe and Robert K. Brayton. The maximum set of permissible behaviors for FSM networks. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '93, pages 316–320. IEEE Computer Society Press, 1993.

A Horn CNF Encoding for the Incompatibility Relation

Let M be an OM from Y to Z. In this section we describe a way of computing the compatibility relation over M defined in Section 5.1 by finding the minimal satisfying assignment of a Horn formula. Let $m \coloneqq |S_M|$. We identify states in S_M with numbers $j \in [m]$. We describe our Horn encoding as follows. For each $i, j \in [m]$ with $i \leq j$ we introduce a variable $X_{i,j}$ meaning $i \nsim j$. We give now the clauses of the formula. For each pair $i \leq j$ we consider two scenarios: (1) If i and j are incompatible at depth one, we add the clause $X_{i,j}$. (2) Otherwise, for all $y \in Y, i' \in \Delta_M(i, y), j' \in \Delta_M(j, y)$ we add clauses $X_{i',j'} \implies X_{i,j}$, where the values of i' and j' are swapped if necessary to ensure $i' \leq j'$.

It follows from Lemma 3 that two states $i \leq j$ are compatible if and only if $X_{i,j}$ is true in the minimal satisfying assignment for this encoding. It is well-known that assignment can be obtained in time linear in the size of the formula [7]. Moreover, the encoding has size $O(|M|^2)$. Hence, the compatibility relation over M can be computed in $O(|M|^2)$ time.

B Correctness of the Covering Reduction

Proof that the N constructed in Theorem 4 implements M: Let $\overline{y} \in \Omega_M$. Consider a run $s_0 \coloneqq r_M, y_0, z_0, s_1, \ldots, s_n, y_n, z_n, s_{n+1}$ of M on $\overline{y}$, and let $C_0 \coloneqq r_N, y_0, z'_0, C_1, \ldots, C_n, y_n, z'_n, C_{n+1}$ be the run of N on $\overline{y}$. By construction, $s_i \in C_i$ for all $0 \leq i \leq n+1$. Moreover, as $\lambda_M(s_i, y_i) = z_i$ it also holds $\lambda_N(C_i, y_i) = z_i$. Thus, $\lambda_M(\overline{y}) = \lambda_N(\overline{y})$ and N implements M. ◀

C Generalization of *MeMin*

Let M be an OM from Y to Z. Further suppose that $M = T|_{\mathrm{Im}(H)}$ for some machines T, H. Replacements of T in $T \circ H$ are precisely the implementations of M. Thus, solving the Tail Minimization Problem amounts to finding a minimal implementation of M. As shown in the previous section, this is equivalent to finding a minimal closed cover of compatibles over M. This reduction has been widely employed in the study of the analogous minimization problem for IS Mealy machines [25, 1, 18]. We show how to adapt MeMin [1] for finding a minimal implementation of M by making use of the theory developed in Section 5.1.

Given a bound n, we reduce the problem of finding a closed cover $\mathcal{F}$ of n compatibles over M to a SAT instance. The CNF encoding follows closely the one in [1], and the details can be found at Appendix C.1. As them, we also compute in advance a so-called partial solution. This clique of pairwise incompatible states $Cl \subseteq S_M$ is obtained via a greedy algorithm. Each state in Cl must belong to a different compatible in $\mathcal{F}$, so the clique can be used for adding symmetry breaking predicates to the encoding and thus to reduce solving times.

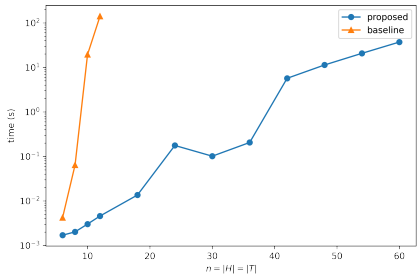
In order to obtain the minimal replacement for a tail T in $T \circ H$ we look for the minimum n that yields a satisfiable CNF encoding. It is clear that such n must lie in the interval $[|Cl|, |S_T|]$. In particular, when $|Cl| = |S_T|$, the machine T is already optimal and no encoding is needed. Otherwise, any searching strategy for n in $[|Cl|, |S_T|]$ may be employed. In our case, we simply employ linear search from $|Cl|$ upwards, which is expected to perform well when $|Cl|$ is a good estimate for the optimal n, as is the case in [1]. A preliminary experimental evaluation of this approach can be found in Appendix C.2.

C.1 CNF Encoding for OM Minimization

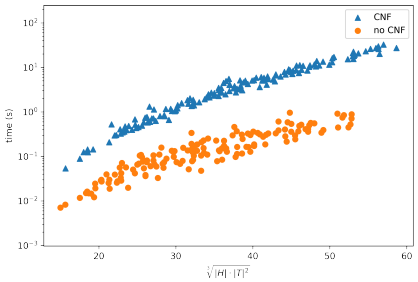
We describe how to reduce the problem of finding a closed cover of n over M to SAT, following the CNF encoding given in [1]. We identify the set S_M with the set of integers $[m]$, where $m \coloneqq |S_M|$. Finding a closed cover of n compatibles is equivalent to finding two maps $C : [n] \to 2^{[m]}$ and $\text{Succ} : [n] \times Y \to 2^{[n]}$ that satisfy: (1) for each $i \in [n]$ the set $C(i) \subseteq [m]$ is a compatible, (2) $r_M \in C(i)$ for some i, (3) $\Delta_M(C(i), y) \subseteq C(j)$ for all $j \in \text{Succ}(i, y)$, and (4) $\text{Succ}(i, y) \neq \emptyset$ for each $i \in [n], y \in Y$. The propositional variables of the CNF encoding are the following: A variable $L_{s,i}$ for each $s \in [m], i \in [n]$, encoding that $s \in C(i)$. A variable $N_{i,j,y}$ for each pair $i, j \in [n], y \in Y$, encoding that $j \in Succ(i, y)$. The clauses of our encoding are as follows: A clause $\neg L_{s_1,i} \vee \neg L_{s_2,i}$, for each $i \in [n], s_1, s_2 \in [m]$ with $s_1 \leq s_2$ and $s_1 \nsim s_2$. This encodes condition (1). A clause $\vee_{i \in [n]} L_{r,i}$, which encodes condition (2). We have a clause $\vee_{j \in [n]} N_{i,j,y}$ for each $i \in [n], y \in Y$, encoding condition (3). Finally, there is a clause $(N_{i,j,y} \wedge L_{s,i}) \implies \vee L_{s',j}$ for each $s \in [m]$, $y \in Y$ with $(s, y) \in D_M, s' \in \Delta_M(s, y)$, $i, j \in [n]$. These clauses encode condition (4). The CNF obtained so far already encodes the desired problem. However, we also use a partial solution $Cl \subseteq S_M$ for adding symmetry breaking predicates. If $Cl = \{s_1, \ldots, s_l\}$ is a set of $l \leq n$ pair-wise incompatible states, we can add the clauses $L_{s_i,i}$ for each $0 \leq i \leq l$.

C.2 Experimental Evaluation

Since the Tail Minimization Problem is in NP, it allows for a straightforward reduction to SAT, which already improves over the doubly-exponential complexity of the K-N algorithm. We show in Figure 5a preliminary experimental results comparing this solution with our proposed one. The baseline method uses a "naive" CNF encoding to decide whether the tail component is n-replaceable. Like in our proposed approach, this is done for increasing values of n until a satisfiable CNF is obtained. The encoding can be seen as analogous to ours but without the information about the incompatibility graph and the partial solution. The experiments were run on cascade compositions consisting of independently randomly generated machines M and T with $n = |S_M| = |S_T|$ and input/output alphabets of size 4. Mean CPU times of all runs for each n are plotted for both the baseline algorithm and ours. The baseline implementation is not able to complete any instance with $n \geq 12$ with a timeout of 10 minutes, while our algorithm solved all instances in under a minute. We conclude that our approach has a clear benefit over a straightforward approach for this class of random instances. Both the implementation of our algorithm and the baseline algorithm use CryptoMiniSat [26]. All experiments were run on a Intel Core i5-6200U (2.30GHz) machine.



(a) A comparison between our proposed algorithm and the baseline one.



(b) Random experiments run with our proposed minimization method.

One reason for the good performance is that our algorithm skips the CNF encoding entirely in about half the instances by using the size of the partial solution, as discussed in Section 5.1. To illustrate this we ran 200 additional experiments where T and H were generated independently with random sizes between 12 and 60 states. The running times are shown in Figure 5b, where orange points correspond to instances where it was possible to avoid any CNF encoding, and the blue ones correspond to the rest. It can be observed that orange and blue points form two separate clouds of points. It is apparent that the algorithm is significantly more efficient when it does not call the SAT solver, but the approach uses under a minute regardless of whether the SAT solver is employed.

D Characterization of Feasible Synthesis Instances

Proof of Proposition 9. The fact that T's existence implies the second part of the statement is straightforward, as discussed above. We prove the other implication. Suppose that any two words $\overline{x}, \overline{x}'$ with $\lambda_H(\overline{x}) = \lambda_H(\overline{x}')$ also satisfy $\lambda_M(\overline{x}) = \lambda_M(\overline{x}')$. This defines a map $F : \mathrm{Out}(H) \to Z^*$ by setting $F(\overline{y}) = \lambda_M(\overline{x})$ if $\overline{y} = \lambda_H(\overline{x})$. Let us define the language $\mathcal{L}_F \subseteq (Y \times Z)^*$ as the one consisting of the words $\langle \overline{y}, F(\overline{y}) \rangle$ for all $\overline{y} \in \mathrm{Out}(H)$. Clearly this language is regular and prefix-closed. Hence, there is some Mealy machine T form Y to Z satisfying $F(\overline{y}) = \lambda_T(\overline{y})$ for all $\overline{y} \in \mathrm{Out}(H)$. By definition of F, we have $\lambda_T(\lambda_H(\overline{x})) = \lambda_M(\overline{x})$ for all $\overline{x} \in X^*$, and $T \circ H \equiv M$. This proves the result. ◀

E Observation Machines with no Small Implementations

Fix $n > 0$. Figure 6 shows the construction of an OM M (bottom right) from $Y := \{a, b\}$ to $Z := \{a, b, \top\}$ for which all implementations have at least 2^n states. Moreover, $d(M) = 2$. Let $y_0 y_1 \dots y_{2n-1}$ be a word in Y^*. M behaves the following way: (1) It outputs $\top$ in response to the first n inputs $y_0, \dots, y_{n-1}$. (2) Starting from y_n, M responds with $\top$ until b is received as an input. (3) If y_{n+i} is the first input equal to b since y_n, then M outputs y_i in response. Intuitively, an implementation of M has to have at least 2^n states because it has to store the first n inputs in order to carry out (3).

F Correctness of the splitting construction

Correctness of the constructions in Lemma 13. Here we show that the given constructions for H and M indeed fulfill the result. We have to show that (1) some T satisfies $T \circ H \equiv M$ and (2) any such T also satisfies $\lambda_N(\overline{y}) = \lambda_T(\overline{y})$ for all $\overline{y} \in \Omega_N$. Remember that characters $x \in X$ are (input,label) pairs $(y, l) \in Y \times [k]$. We say that a word $\overline{x} := (y_0, l_0) \dots (y_n, l_n) \in X^*$ labels a run of N if N has a run $r_N = s_0, y_0, z_0, s_1, \dots, s_n, y_n, z_n, s_{n+1}$ where $L(s_i, y_i, s_{i+i}) = l_i$ for all $0 \leq i \leq n$. Given such $\overline{x}$ by construction $s_0, (y_0, l_0), y_0, s_1, \dots, s_n, (y_n, l_n), y_n, s_{n+1}$ and $s_0, (y_0, l_0), z_0, s_1, \dots, s_n, (y_n, l_n), z_n, s_{n+1}$ are the runs of H and M on $\overline{x}$, respectively. Let $\overline{x} \in X^*$ be arbitrary. We can write $\overline{x} = \overline{u}\overline{v}$, where $\overline{u} = \langle \overline{y}, \overline{l} \rangle$ is the largest prefix of $\overline{x}$ which labels a run of N. The previous observation yields $\lambda_H(\overline{u}) = \overline{y}$, $\lambda_M(\overline{u}) = \lambda_N(\overline{y})$. Moreover, by construction $\lambda_H(s, \overline{v}), \lambda(s, \overline{v})$ are both sequences of only $\perp$ symbols, where $s = \delta_H(\overline{u}) = \delta_M(\overline{u})$. This way we have shown that the language $E(H, M) = \{ \langle \lambda_H(\overline{x}), \lambda_M(\overline{x}) \rangle \mid \overline{x} \in X^* \}$ equals $\{ \langle \overline{y}, \lambda_N(\overline{y}) \rangle \mid \overline{y} \in \Omega_N \} \{ \perp \}^*$. This identity proves both (1) and (2). Indeed, if $\lambda_H(\overline{x}) = \lambda_H(\overline{x}')$, then necessarily $\lambda_M(\overline{x}) = \lambda_M(\overline{x}')$. By Proposition 9 this implies (1). Also, if $\lambda_H(\overline{x}) = \overline{y}$ and $\overline{y} \in \Omega_N$, then $\lambda_M(\overline{x}) = \lambda_N(\overline{y})$, which shows (2). ◀

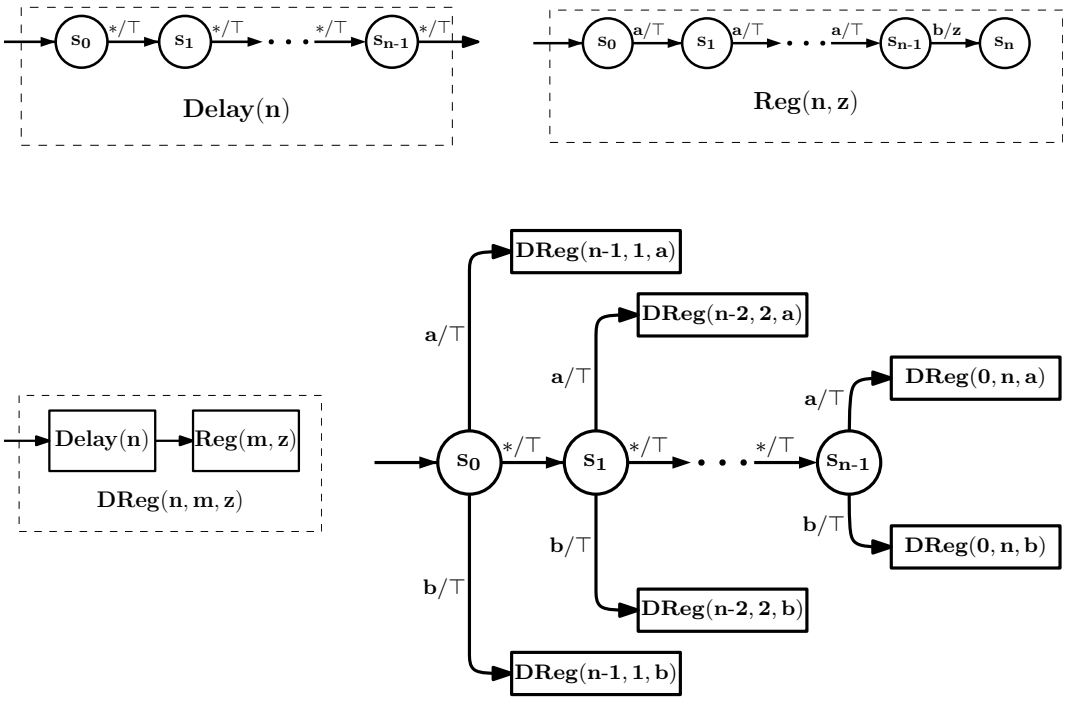


Figure 6 The construction of an OM with no small implementations (bottom right). The symbol $*$ stands for both a, b.

G Reductions to Cascade Compositions

We put $T \odot H$ for the composition between two Mealy machines H and T shown in Figure 1b. This composition is not well-defined in general [12, Section 6.2], but a sufficient requirement is that H is a Moore machine, for example. Analogously to Problem 1, we consider the problem of finding a minimal replacement for T in $T \odot H$ when both T and H are given. In Figure 7a it is shown how to build a machine H' such that finding a minimal replacement for T in $T \odot H$ is equivalent to finding a minimal replacement for T in the cascade composition $T \circ H'$. Similarly, when M and H are given, we can study the problem of finding T with $T \odot H \equiv M$. In Figure 7b it is shown how to build H' in a way that T satisfying $T \odot H \equiv M$ is equivalent to $T \circ H' \equiv M$.

H Construction of a Tail Representing a CNF Formula

The set $S_{T_\mathcal{F}}$ contains the following states: (1) a state $\textsc{Init}(i)$ for each $0 \leq i < 2(n+1)$, (2) a state $\textsc{Clause}(k, i, b)$ for each $0 \leq k < m$, $0 \leq i < n+1$, $b \in \{\mathrm{T}, \mathrm{F}\}$, and (3) two additional states called $\textsc{Sat}$ and $\textsc{UnSat}$. As mentioned before, $\textsc{Sat}$ and $\textsc{UnSat}$ are sink states, $\lambda_{T_\mathcal{F}}(\textsc{Sat}, y) = \top$ for all $y \in Y$ and $\lambda_{T_\mathcal{F}}(s, y) = \bot$ for all $s \neq \textsc{Sat}$, $y \in Y$. Hence, we only need to define $\delta_{T_\mathcal{F}}$. The first $2(n+1)$ inputs are read using the states $\textsc{Init}(i)$, which ensure that the intended format is followed. Let $r_{T_\mathcal{F}} \coloneqq \textsc{Init}(0)$. For all $0 \leq i \leq< 2(n+1)$ with $i \neq n, 2n+1$ we define and $\delta_\mathcal{F}(\textsc{Init}(i), y) \coloneqq \textsc{Init}(i+1)$ for all $y = -1, 1$ and $\delta_\mathcal{F}(\textsc{Init}(i), \bot) \coloneqq \textsc{UnSat}$. For $i = n, 2n+1$ we set $\delta_\mathcal{F}(\textsc{Init}(i), y) \coloneqq \textsc{UnSat}$ if $y = -1, 1$. We also define $\delta_\mathcal{F}(\textsc{Init}(n), \bot) \coloneqq \textsc{Init}(n+1)$, and $\delta_\mathcal{F}(\textsc{Init}(2n+1), \bot) \coloneqq$

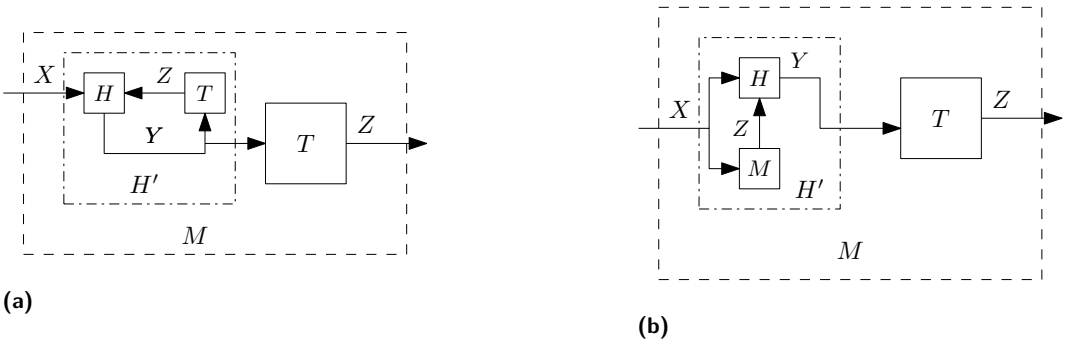


Figure 7 Polynomial transformations to cascade compositions.

Clause$(0, 0, \mathrm{F})$ if $m > 0$ (i.e., $\mathcal{F}$ contains some clause) or $\delta_{\mathcal{F}}(\textsc{Init}(2n+1), \bot) \coloneqq \textsc{Sat}$ if $\mathcal{F}$ is empty. Now we describe the transition function on the states Clause(k, i, b). Let $0 \leq k < m$ and $0 \leq i < n$. Then $\delta_{\mathcal{F}}(\textsc{Clause}(k, i, b), \bot) \coloneqq \textsc{UnSat}$ for both $b = \mathrm{F}, \mathrm{T}$. Whenever $y \in \{-1, 1\}$ we set $\delta_{\mathcal{F}}(\textsc{Clause}(k, i, \mathrm{T}), \bot) \coloneqq \textsc{Clause}(k, i+1, \mathrm{T})$. If $y = -1$, resp. $y = 1$, and the clause C_k contains the literal $\neq x_i$, resp. x_i, then $\delta_{\mathcal{F}}(\textsc{Clause}(k, i, \mathrm{F}), y) \coloneqq \textsc{Clause}(k, i+1, \mathrm{T})$. Now the only transitions left to define are those coming out of the states Clause(k, n, b). For all b, y we set $\delta_{\mathcal{F}}(\textsc{Clause}(k, n, \mathrm{F}), y) \coloneqq \textsc{UnSat}$. Additionally, if $y \neq \bot$ we also define $\delta_{\mathcal{F}}(\textsc{Clause}(k, n, \mathrm{T}), y) \coloneqq \textsc{UnSat}$. Finally, $\delta_{\mathcal{F}}(\textsc{Clause}(k, n, \mathrm{T}), \bot) \coloneqq \textsc{Clause}(k, 0, \mathrm{F})$ when $k < m - 1$, and $\delta_{\mathcal{F}}(\textsc{Clause}(m-1, n, \mathrm{T}) \coloneqq \textsc{Sat}$.