Master of Science in Informatics at Grenoble
Master Informatique
Data Science

hW-inference of non Resettable State Machines: Theory, Limits and Improvements

Research project performed at the
Laboratoire d'Informatique de Grenoble

Moritz Halm

June 21, 2021

Supervisors:	*Jury:*
Roland Groz	Massih-Reza Amini
Catherine Oriat	Clovis Galiez
	Franck Iutzeler

Abstract

hW-inference is a deterministic black-box, active inference algorithm for learning non–resettable finite state machines [Gro+20]. Although hW-inference is designed to work under the usual assumption of a minimally adequate teacher, its termination could not yet have been proven.

We show that hW-inference does not always terminate. We propose a modification of hW-inference, h-forests, and prove that hW-inference with h-forests terminates given a homing sequence for the system under inference. We also sketch how a homing sequence can be found with a probabilistic method.

Further, we evaluate hW-inference in a case study of a supermarket scanner system. We observe that the overall trace length varies significantly depending on the size of the characterization set. We propose an heuristic optimization to hW-inference to tackle this issue and evaluate its effect.

Resumé

hW-inference [Gro+20] est un algorithme pour l'apprentissage actif des automates finis, qui ne peuvent pas être réinitialisés pendant l'apprentissage. Jusqu'ici, on n'avait pas pu prouver que l'algorithme termine toujours. Dans le cadre de ce rapport, nous examinons cette question en profondeur.

Nous démontrons avec un exemple détaillé que l'algorithme ne termine pas toujours. Pour résoudre ce problème de terminaison, nous proposons une modification de hW-inference, qu'on appelle h-forests. Nous démontrons que hW-inference avec h-forests termine, s'il trouve une *homing sequence* correcte. On esquisse également comment une *homing sequence* peut être obtenue au moyen d'un algorithme probabiliste.

De plus, on évalue la performance de hW-inference dans le cadre d'une étude de cas. On observe que le nombre total d'interactions varie fortement en fonction de la taille du *characterization set.* Afin de réduire ce effet, nous proposons une optimisation heuristique pour hW-inference et évalue son effet.

Contents

1 Introduction

When learning how to use an unknown device or computer program, many people follow an intuitive strategy: Press random buttons and see what happens. By interacting with the system they learn a mental model of it.

Analogously to this intuitive method, *active inference* algorithms build a formal model of an existing system through interacting with a running instance of it. If the learning algorithm does not presume any knowledge about the internals of the system under inference (SUI), i.e., its source code, we speak of *black box* inference. A common model that is simple, yet expressive enough to represent many real world applications is a *finite state machine* (FSM), most commonly a Mealy machine.

A related field of research concerns *passive* inference, which means learning a model from given execution traces, i.e. the log files of a program. Passive inference includes deterministic methods (state minimization, [AP18]) and statistical approaches (i.e, Hidden Markov Models, [CMR06]).

Active inference of finite state machines has been proven to be useful in several application domains, including network protocols, smartcard readers and embedded systems [Vaa17; BG19; PTR15]. First, learning yields a precise model for an already existing system, i.e. some software artifact. This model can then be used to verify the compliance of the software with a specification (model checking) or the generation of regression tests for later versions (model based testing). Second, the learning itself serves as an "intelligent monkey testing" to detect potential crashes in the software.

In this work, we focus on *hW-inference* [Gro+20], a recently proposed deterministic algorithm for active black box inference of finite state machines. Contrary to most other active inference algorithms [SG09; HMS03], hW-inference does not require that the system under inference can be reset to an initial state. Avoiding reset operations makes active inference more versatile. Consider for instance the case of learning a web application that runs on a remote server. While a single request has a round-trip-time of only a few milliseconds, restarting the application may take up to several minutes or be not possible at all. hW-inference significantly outperforms the few other algorithms [RS93; Gro+15; Pet+17] for non resettable systems.

The theoretical framework for active inference was set by Angluin's work on interactive learning of accepting automata for regular languages [Ang87]. It has later been adapted on Mealy machines [SG09]. In this framework, a learner can send output queries and equivalence queries to a *minimally adequate teacher* who knows the system under inference. In an output query, the learner sends an input symbol and receives the output of the SUI as response. This can change the internal state of the system. In an equivalence query, the learner sends a putative model, the conjecture, of the SUI. The teacher either confirms the correctness of this model or provides a counterexample. A counterexample is a sequence of inputs that, when applied to the SUI, yields an output different from the output predicted by the conjecture. The possibility to get counterexamples evades the need for an exponential runtime full search, that would otherwise be inherent in every learning algorithm [Moo56].

While hW-inference empirically works well, its correctness has yet only been proven under unrealistically strong assumptions on the length of the provided counterexamples. A minimally adequate teacher, however, is only guaranteed to provide *some* counterexample for the conjecture, if it is false. It is not clear whether in hW-inference the processing of every counterexample also leads to progress. There might be pathological cases, in which the learner requests an infinite number of counterexamples, but never converges towards a correct model. This fear is in particular founded on the observation that intermediate conjectures found by hW-inference can contain so-called "fake states", which do not match any state of the SUI. The possibility of fake states makes it tricky to find any bound for the number of counterexamples.

In this thesis we study hW-inference in depth towards answering the questions:

1. Does hW-inference always terminate when learning a deterministic FSM?
2. More precisely, under which assumptions may we prove termination?

Concretely, we make the following contributions:

Ineffective Counterexamples and Transfer Sequences We answer question 1 negatively. In Chapter 3, we show how counterexamples can be ineffective and hW-inference does not terminate. This example hinges on the fact that the choice of transfers from learned to unknown states within hW-inference is not sufficiently specified. We also sketch caveats future proof-attempts have to consider, if the choice of transfer sequences is repaired.

Fixing hW-inference: h-forests In Chapter 4, we propose h-forests, a modification to hW-inference, which partially addresses the problem of ineffective counterexamples. We prove that if hW-inference finds a homing sequence, h-forests infers a correct conjecture after at most n equivalence queries, n being the number of states in the system under inference. We then sketch how to resolve this constraint, by sh h-forests can be further extended to find a homing sequence with high probability.

Heuristic: W-set pruning In Chapter 5, we motivate and propose a heuristic, *W-set pruning*, that aims at reducing the number of output queries performed by hW-inference.

Application and Experiments We evaluate the effectiveness of W-set pruning on the case study of a supermarket self-service system in Chapter 6. Since the system under inference is not a finite state machine, we describe challenges in abstracting its behaviour in order to use finite state machine inference algorithms on it.

Note that in this work, we restrict ourselves on the non-adaptive version of hW-inference [Gro+20], setting aside the more involved version of the algorithm that uses adaptive sequences [GBS19].

The thesis starts by introducing basic concepts and recapitulating in detail how hW-inference works (Chapter 2). After presenting our contributions in Chapters 3 – 6, we conclude our findings and give an outlook on future work (Chapter 7).

2 Preliminaries

In this chapter we introduce basic notions of finite state machines (Sections 2.1 and 2.2). We present the problem of active inference of active inference of finite state machines (Section 2.3) and how it is solved by hW-inference (Section 2.4). We recapitulate the mechanics of the algorithm as it was proposed by Groz et al. [Gro+20] in detail, as a thorough understanding is essential for all the following chapters.

2.1 Finite State Machines

The type of systems that can be learned by hW-inference are finite state machines. A finite-state machine (FSM) or finite-state automaton is a model of computation. It describes a system that is in exactly one of several states at any given time. The system reacts to externally provided input by changing from its current state to another and by yielding some output. Although finite state machines are strictly less powerful than other models of computation, most notably Turing machines, many every day and industry technical systems can be modeled as finite state machines. Examples range from vending machines, voice menus of telephone hotlines, to microcontrollers embedded in car engines.

We formalize finite state machines as deterministic Mealy machines; a formalism which was introduced by Mealy to model sequential switching circuits [Mea55].

Definition 2.1 (Finite State Machine) *A* finite state machine *is a tuple* $\mathcal{M} = (Q, I, O, \lambda, \delta)$. *$Q$ is the finite set of* states. *I and O are the finite sets containing the* input symbols *and the* output symbols *of the machine, respectively. The function $\lambda : Q \times I \longrightarrow O$ denotes the output $\lambda(q, i)$ obtained when applying input i in state q. Likewise, the function $\delta : Q \times I \longrightarrow Q$ denotes the transition to state $\delta(q, i)$ when applying input i in state q.*

Since we defined λ and δ as functions with domain $Q \times I$ we implicitly ask, that every input can be applied in every state. Finite state machines with this property are said to be *complete.*

We overload the functions λ and δ to handle sequences of inputs, including the empty sequence ε: $\delta(q, \varepsilon) = q$, and recursively for $\alpha \in I^*, x \in I$: $\delta(q, \alpha x) = \delta(\delta(q, \alpha), x)$. Similarly, we have: $\lambda(q, \varepsilon) = \varepsilon$and $\lambda(q, \alpha x) = \lambda(q, \alpha).\lambda(\delta(q, \alpha), x)$.

We call the corresponding sequence of outputs $\lambda(q, \alpha)$ a *response* and the pair of an input sequence α and its response β a *trace*, notated as $\omega = \alpha/\beta$. Conversely, the operators $\overline{\omega} = \alpha$ and $\underline{\omega} = \beta$ project a trace variable $\omega = \alpha/\beta$ to its corresponding input and output sequences. When applying an input sequence α to a machine in state q, we refer to q and $\delta(q, \alpha)$ as *head* and *tail* states of the corresponding trace, respectively. We denote the concatenation of two traces α/β and γ/δ with a ".": $\alpha/\beta.\gamma/\delta$. If we the output of a trace is not important for our reasoning, we avoid naming it by writing $\alpha/-$ for the respective trace.

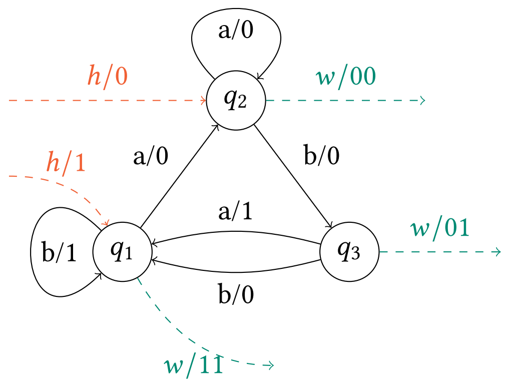


Figure 2.1 Example finite state machine with three states, input set $I = \{a, b\}$ output set $O = \{0, 1\}$. For a homing sequence $h = a$ and a characterization set $W = \{bb\}$ we indicate the responses to the homing and characterizign sequences in red and green, respectively.

The operator $|\cdot|$ indicates the length of a sequence of input or output symbols or a trace, respectively. Note that we also use $|\cdot|$ on sets to indicate their cardinality.

A finite state machine $\mathcal{M}$ is *strongly connected*, if and only if every state can be reached from any other state, i.e., $\forall q, q' \in Q : \exists \alpha \in I^* : \delta(q, \alpha) = q'$.

A sequence $\alpha \in I^*$ is said to *distinguish* the states q and q', if $\lambda(q, \alpha) \neq \lambda(q', \alpha)$. Two states q, q' are equivalent, if there is no sequence distinguishing q and q'. A finite state machine $\mathcal{M}$ is *minimal*, if there are no equivalent states, i.e., $\forall q, q' \in Q : \exists \alpha \in I^* : \lambda(q, \alpha) \neq \lambda(q', \alpha)$.

Typically, finite state machines are defined declaring some initial state. The machines are said to be equivalent, if their initial states are equivalent. However, since we are concerned with strongly connected machines that cannot be reset, the initial state is of less importance. We say two machines $\mathcal{M}, \mathcal{M}'$ are equivalent, if there is a state q of $\mathcal{M}$ and a state q' of $\mathcal{M}'$, such that q and q' are equivalent. We write $\mathcal{M} \equiv \mathcal{M}'$.

2.2 Homing and Characterizing Sequences

In hW-inference we often apply a specific sequence to infer the head or tail state by the response. A homing sequence determines its tail state, while multiple characterizing sequences determine a head state.

Definition 2.2 (homing sequence) *For a finite state machine $\mathcal{M} = (Q, I, O, \lambda, \delta)$, a sequence $h \in I^*$ is* homing, *if the response to h uniquely identifies its tail state. More formally, for any states $q, q' \in Q$, $\lambda(q, h) = \lambda(q', h) \implies \delta(q, h) = \delta(q', h)$.*

For example, in the finite state machine in Figure 2.1 the sequence $h = a$ is homing. Applying h to the states q_1 or q_2 leads to tails state q_2 with the response 0, respectively. The response 1 leads to tail state q_1 (from q_3).

Note that the above definition states that same responses to a homing sequence must not lead to different tail states. On the other hand, a homing sequence might yield different responses that lead to the same tail state.

Every minimal finite state machine has a homing sequence. Rivest and Schapire [RS93] give the following constructive proof. If a sequence $h \in I^*$ is not homing, there exist two states $q_1, q_2 \in Q$ with $\lambda(q_1, h) = \lambda(q_2, h)$, but $q_1' = \delta(q_1, h) \neq \delta(q_2, h) = q_2'$. Because the machine is minimal, there exists a distinguishing sequence $g \in I^*$ for q_1' and q_2', such that $\lambda(q_1', g) \neq \lambda(q_2', g)$. We refine h as hg. This refinement increases the number of possible responses to h by 1. Also there can be no more than $n = |Q|$ such responses. Thus, after maximally n refinements, h is homing. Moreover, since there always exists a distinguishing sequence g shorter than n, the constructed h is not longer than n^2.

Definition 2.3 (characterization set) *A set of sequences $W \subset I^*$ is a* characterization set *(also called W-set) for a finite state machine $\mathcal{M} = (Q, I, O, \lambda, \delta)$, if any state of $\mathcal{M}$ is uniquely identified by its set of responses to the sequences $w \in W$. Formally, $\forall q, q' \in Q : q \neq q' \implies \exists w \in W : \lambda(q, w) \neq \lambda(q', w)$. For a characterization set W, we call the unique mapping from the sequences in W to output sequences for a state q its* characterization, *denoted by $\phi_W(q) := \{w \mapsto \lambda(q, w) \mid w \in W\}$.*

For the example in Figure 2.1, the set $W = \{bb\}$ characterizing, since $\phi(q_1) = \{bb \mapsto 11\}, \phi(q_2) = \{bb \mapsto 00\}$, and $\phi(q_3) = \{bb \mapsto 10\}$.

When the characterization set W is clear in a context, we just write ϕ instead of ϕ_W.

One can easily see that for any minimal finite state machine there exists a characterization set. It is the set that contains the distinguishing sequences which exist for any pair of states.

We call a tuple (h, W) of a homing sequence h and a characterization set W homing-characterizing.

2.3 Assumptions and Problem

We are now ready to define the problem that is solved by hW-inference. We assume that the system under inference exhibits the following properties:

1. The system's behaviour can be fully expressed by a deterministic Mealy machine (as defined in Definition 2.1). This implies in particular, that the system is deterministic and has a finite number of states.
2. The finite state machine describing the system is strongly connected.
3. Its input alphabet I is known a priori.

As for most parts of this work, we will say the system under inference *is* a deterministic Mealy machine (or finite state machine), although we are – strictly speaking – referring to the finite state machine describing the behaviour its system.

Our definition of Mealy machines require properties that might not be met by a world system, but can easily be ensured by the driver. The driver is the piece of software (or software and hardware) that connects the learner with the system under inference. If the system has multiple finite output variables $O_1, \dots, O_k$, we can consider their possible combinations as a single output alphabet $O = O_1 \times \cdots \times O_k$. If the system has no output, the driver can implement a timeout and return a special no-output symbol when the system does not output anything before the timeout expires.

Definition 2.4 (active inference) *Given a system with the above properties, learn a Mealy machine (a model) equivalent to the system by two types of queries: (1) Output queries consist of an input symbol $x \in I$. Upon an output query, x is applied to the system under inference, and its response $o \in O$ is returned. (2) Equivalence queries, as defined below.*

The possibility to ask output queries and equivalence queries is also called learning with a minimally adequate teacher [Ang87].

Counterexample Oracle We further assume that the learner has the ability to make equivalence queries to an external oracle. The oracle decides whether the conjecture model learned by hW-inference is equivalent to the model of the system under inference. If not, it also provides a counterexample, that is a sequence of inputs for which the real system yields a different output than the conjecture would suggest. Formally, we can define the oracle as a function $\mathcal{O}_{(\mathcal{M},q)}$, which is parameterized by the real model of the system $\mathcal{M} = (Q, I, O, \lambda, \delta)$ and its current state $q \in Q$. $\mathcal{O}$ takes as input a conjecture model $\mathcal{M}' = (\tilde{Q}, I, \tilde{O}, \tilde{\lambda}, \tilde{\delta})$ and its current state $q' \in \tilde{Q}$.

$$\mathcal{O}_{(\mathcal{M},q)} : (\mathcal{M}', q') \mapsto \begin{cases} \text{none} & \text{if } \mathcal{M} \equiv \mathcal{M}' \\ \alpha \in I^* \text{ s.t. } \lambda(q, \alpha) \neq \tilde{\lambda}(q', \alpha) & \text{else} \end{cases}$$

As often in computer science, the notion of oracle indicates a function whose computation requires further knowledge or time, i.e. a model of the system under inference. As having such knowledge subverts the idea of black-box-inference, the assumption to have a counterexample oracle might appear to be unrealistically strong. An oracle is, however, necessary, to ultimately decide the correctness of the inferred conjecture in absence of a bound. Moore showed that without an oracle the problem of learning a FSM is exponential [Moo56].

In practice, we can approximate the oracle by applying random inputs on both the conjecture and the real system. This method might use, however, some bound on the size of the system to determine a reasonable length of a error-free random walk after which the inferred model is considered correct.

2.4 hW-inference

As the name suggests, the two central variables of hW-inference are an input sequence h and a set of input sequences W. We first present a method, the backbone algorithm, that infers a correct model, if h, W is homing-characterizing for the system under inference. We then show how h and W can be constructed during multiple iterations of the backbone algorithm with tentative, non-homing h and non-characterizing W.

Algorithm 1 Backbone algorithm

```
procedure BACKBONE-INFERENCE($h, W$)
    $\tilde{Q}, \tilde{\lambda}, \tilde{\delta} \leftarrow \emptyset$
    $H \leftarrow \{(r, \emptyset) \mid r \in O^*\}$
    repeat
        apply $h$ and observe $r \in O^*$
        $p \leftarrow H(r)$
        if $p$ is partial then                                    ▷ Learn the state
            apply $w \in W \setminus \mathrm{dom}(p)$, observe $y$
            $p \leftarrow p \cup \{w \mapsto y\}$
            if $p$ is complete then $\tilde{Q} \leftarrow \tilde{Q} \cup \{p\}$ end if
        else                                                       ▷ $p$ is complete
            find shortest $\alpha x \in I^* \times I$, s.t. $p' = \tilde{\delta}(p, \alpha)$ complete and $p'' = \tilde{\delta}(p', x)$ partial
            apply $\alpha.x.w$ ($w \in W \setminus \mathrm{dom}(p'')$), observe $\beta.o.y$;        ▷ Learn a transition
            $\tilde{\lambda}(p', x) \leftarrow o$ and $p'' \leftarrow p'' \cup \{w \mapsto y\}$
            if $p''$ is complete then                        ▷ add tail state to conjecture
                $\tilde{Q} \leftarrow \tilde{Q} \cup \{p''\}$ and $\tilde{\delta}(p', x) \leftarrow p''$
            end if
        end if
    until $\mathcal{M}' = (\tilde{Q}, I, O, \tilde{\delta}, \tilde{\lambda})$ is complete
end procedure
```

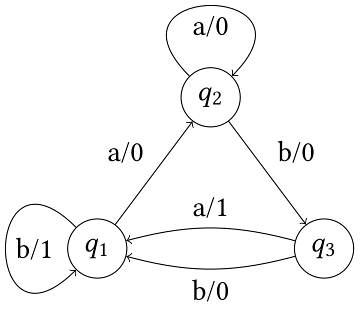
From now on, we refer to the inferred finite state machine as *conjecture* and denote it as $\mathcal{M}' = (\tilde{Q}, I, O, \tilde{\lambda}, \tilde{\delta})$. At some point, the algorithm presumes to be in a conjecture state $p \in \tilde{Q}$ and applies an input sequence $\alpha \in I^*$ and observes outputs z. We denote this with $p \,!\, \alpha/z$. Having such an operator is necessary, as z can deviate from $\tilde{\delta}(p, \alpha)$, if the conjecture is false.

2.4.1 The Backbone Algorithm

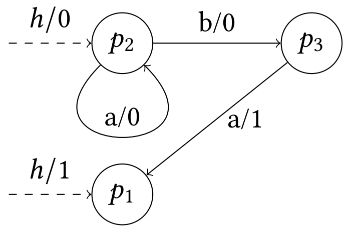
Let us for now assume, (h, W) is homing-characterizing. Recall that the homing property of h ensures that we are always in the same state after observing the same response to h. This allows us to use h as a surrogate for a reset operation. Recall also the that characterizing property of W means that each state can be uniquely identified by its characterization function $\phi(q) : W \rightarrow O^*$. To ease exposition of hW-inference, we represent inferred states only by their characterization instead of introducing arbitrary handles, i.e. for every conjecture state $p \in \tilde{Q} : \phi(p) = p$. The characterization of states is learned gradually. We say a state $p \in \tilde{Q}$ is *complete*, if we know its full characterization, i.e. $\mathrm{dom}(p) = W$, and is *partial*, otherwise.

We give pseudocode for the backbone algorithm in Algorithm 1.

The main loop of the backbone algorithm starts by applying the homing sequence h in order to transfer the system to a known state. For every observed response $r \in O^*$ to h, we store the homing tail state in a map $H : O^* \rightarrow W \times 2^{O^*}$. Initially, these states are partial since $\mathrm{dom}(H(r)) = \emptyset$. Thus, after observing h/r, we apply a missing characterization sequence



(a) Finite state machine to be learned.



(b) Conjecture after 14 transitions

Figure 2.2 Example for the backbone algorithm.

$w \in W \backslash \text{dom}(H(r))$ to learn the state's characterization. After having visited the same tail homing state $|W|$ times, it is complete. We can add the state to $\tilde{Q}$.

Once a state p is complete, we can learn transitions from it. We apply an input symbol $x \in I$, for which we do not know the reached state completely. The trace x/o allows to learn $\tilde{\lambda}(p,x)$. The tail state $\tilde{\delta}(p,x)$ is partial and we apply a sequence $w \in W \backslash \text{dom}(\tilde{\delta}(p,x))$. More generally, when all transitions from p are known, the algorithm chooses the shortest path α leading to state with an unknown transition in the conjecture. We then apply a missing sequence for the tail state of this transition.

Every time a state $p' = \tilde{\delta}(p,x)$ reached from p is fully characterized, we have either found a new state and add it to $\tilde{Q}$ or learned a transition to an already known state.

The algorithm stops when all reachable states are fully characterized. Since h is homing, the characterization sequences for a state in $H(r)$ are all applied in the same real state. Consequently, all input sequences applied to a conjecture state p' reached through a transfer sequence α after a state p are applied to the same real state as well. Every conjecture state thus has the characterization of a real state, and because W is characterizing, states are distinguished correctly. Since $\mathcal{M}$ is strongly connected, all states are learned once the conjecture is complete.

Example To illustrate the backbone algorithm, we consider the finite state machine in Figure 2.2a as an example. A possible homing sequence h is a. Also all states respond differently to the sequence ba, so $W = \{ba\}$ is a valid characterization set. Assume, the system is in state q_3 at the beginning of the inference.

We start by applying the homing sequence and observe $a/1$. The system is now in state q_1. Since we do not know the tail state for this response yet, i.e. $H(0) = \emptyset$, we apply $ba = w \in W$. We observe $ba/10$ and thus have learned a complete state p_1 with $\phi(p_1) = \{ba \mapsto 10\}$.

$$\underbrace{q_3 \xrightarrow{a/1}}_{h} q_1 \underbrace{\xrightarrow{ba/10}}_{w=ba} q_2$$

We again apply $h/0$ to localize. We do not know yet the tail state of h reached after 0, so we apply ba to learn it. We now know two states, $p_1 = H(1)$ and $p_2 = H(0)$ ($\phi(p_2) = \{ba \mapsto 01\}$) that can be reached after homing.

$$q_2 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{ba/01}}_{w=ba} q_1$$

After the next homing $h/0$ we arrive in the complete state p_2. We can thus learn a transition (line 11 in Algorithm 1). We apply $x = a \in I$, since the state $\tilde{\delta}(p_2, a)$ is yet undefined, and then $ba/01$ to characterize the state after the transition. We learned $\tilde{\lambda}(p_2, a) = 0$ and $\tilde{\delta}(p_2, a) = p_2$.

$$q_1 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{a/0}}_{x=a} q_2 \underbrace{\xrightarrow{ba/01}}_{w=ba} q_1$$

In the same way, we learn the next transition by b leading to a new state $p_3 = \tilde{\delta}(p_2, b)$ with $\phi(p_3) = \{ba \mapsto 00\}$:

$$q_1 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{b/0}}_{x=b} q_3 \underbrace{\xrightarrow{ba/00}}_{w=ba} q_2$$

After the next homing, we are in state p_2 again. All transitions of this state are known, so we choose a shortest transfer sequence $\alpha = b$ that leads to the complete state p_3, which has at least one partial subsequent state. The state $\tilde{\delta}(p_3, a)$ is partial for its known characterization is $\emptyset$. We learn that the state $\tilde{\delta}(p_3, a)$ is the already known state p_1 and that the output symbol of the transition is $\tilde{\lambda}(p_3, a) = 1$.

$$q_2 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{b/0}}_{\alpha=b} q_3 \underbrace{\xrightarrow{a/1}}_{x=a} q_1 \underbrace{\xrightarrow{ba/10}}_{w=ba} q_2$$

The conjecture learned by now is depicted in Figure 2.2b. The other transitions are learned in a similar manner. The algorithm stops, once the conjecture is complete.

2.4.2 Finding h and W: Nondeterminism

So far we have seen how finite state machines can be inferred if we know sequences (h, W) that are homing-characterizing. However, we do not know such (h, W) a priori for a system under inference. The key idea in hW-inference is to start with tentative h and W that are not homing-characterizing. With such (h, W) the system under inference will appear to behave non-deterministically. We leverage this apparent nondeterminism to refine h and W until (h, W) is eventually homing-characterizing.

h-ND Let us suppose h is not homing. That is, for the same response r to h the system might be in different states q and q'. Since these states are assumed to be one state $H(r)$ in the conjecture, this conjecture state can appear to be non-deterministic. Formally, an *h-inconsistency* occurs, if the global trace (all interactions with the system during inference) contains the traces

Algorithm 2 hW-infernce

```
1: procedure hW-INFERENCE
2:     $h \leftarrow \varepsilon, W \leftarrow \emptyset$
3:     repeat
4:         try
5:             $\mathcal{M} \leftarrow$ BACKBONE-ALGORITHM($h, W$)          ▷ infer model
6:             $c \leftarrow$ query oracle with $\mathcal{M}$          ▷ ask for counterexample
7:             if counterexample $c$ was found then
8:                 apply $c$          ▷ yields non-determinism
9:             end if
10:        catch h-ND or W-ND
11:            refine $h$ and $W$ accordingly
12:        end try
13:    until no counterexample is found
14:    return $\mathcal{M}$
15: end procedure
```

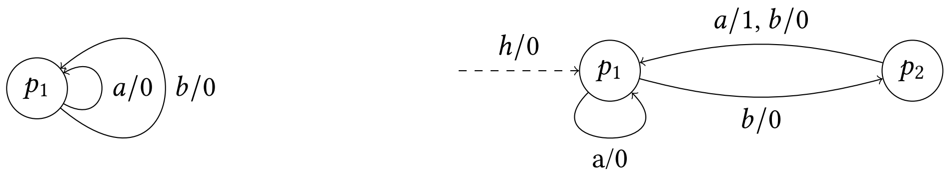
$h/r.\beta/y$ and $h/r.\beta/y'$ with $y \neq y'$. This observation is twofold: (1) h is not homing. (2) The input sequence β distinguishes two tail states of h/r. Hence, whenever an *h-inconsistency* occurs, we refine $h^{\text{new}} := h^{\text{old}}.\beta$, such that q and q' are distinguished.

W-ND If W is not characterizing, there are different states q and q' that have the same response to all sequences in W. During inference, they are characterized as the same conjecture state p. However, there exists a sequence β with $\lambda(q, \beta) \neq \lambda(q', \beta)$. Consequently, the response observed when applying β to p appears to be non-deterministic depending on the current real state. We call this a *W-inconsistency.* Formally, it occurs when the traces $h/r_1.\alpha_1/-.\beta/y_1$ and $h/r_2.\alpha_2/-.\beta/y_2$ with $\tilde{\delta}(H(r_1), \alpha_1) = \tilde{\delta}(H(r_2), \alpha_2)$ and $y_1 \neq y_2$ are observed.

The sequence β is a distinguishing sequence for the possible states of the real system in conjecture states $\tilde{\delta}(H(r_1), \alpha_1)$ and $\tilde{\delta}(H(r_2), \alpha_2)$. Moreover, all real states passed while applying β are distinguished by some suffix of β. There are different strategies to refine W (cf. [IOG10]). Groz et al. propose to add the shortest suffix β' of β that is not yet in W to W. All prefixes of β' in W can then be removed, as they distinguish states that are distinguished by β anyhow.

The backbone algorithm is augmented in two ways in order to detect and handle non-determinism as just described. First, every interaction with the real machine is wrapped by a proxy component, that keeps track of the global trace. As soon as an h- or W- inconsistency is detected, an exception is raised. h or W are refined as presented above and the backbone-algorithm is restarted with an empty conjecture and the refined h and W.

Otherwise the backbone algorithm terminates with an inferred conjecture. We then ask the oracle for a counterexample. If there is no counterexample, the conjecture is correct. Otherwise, we apply the counterexample on the system under inference. This will yield an h-inconsistency or a W-inconsistency. We refine h and W and restart the backbone algorithm.



(b) $h = a, W = \{a\}$

(a) $h = \varepsilon, W = \emptyset$

Figure 2.3 Conjecture after the first (2.3a) and second subinference (2.3b) on the machine in Figure 2.1.

Each round of the backbone algorithm with an updated h and W is called a *subinference*. The complete hW-inference is shown in Algorithm 2.

An incorrect conjecture might not be strongly connected. We thus adapt line 12 to stop the backbone algorithm if no transfer sequence to a partial state can be found. In this case, we ask for a counterexample as well.

Note, that we defined inconsistencies with respect to traces, but counterexamples with respect to conjectures. However, during a counterexample at some point the actual output diverges from the output predicted by the conjecture. This transition was traversed at least once during learning, so the respective trace will yield an inconsistency with the currently applied trace.

It remains to see that h and W are homing-characterizing after a finite number of such refinements. The refinements of h conform to the constructive proof that a homing sequence exists we gave in Section 2.4. Thus, h is homing after extending it at most $n = |Q|$ times. To show that W is eventually characterizing, Groz et al. make the additional assumption that the counterexamples provided by the oracle only lead to refinements in W of length shorter than n. Since any non-equivalent states can be distinguished by some sequence shorter than n, there surely is a characterization set containing only such sequences. Thus, after adding a potentially exponentially large yet finite number of characterizing sequences, W is a characterization set.

Example Consider again the finite state machine in Figure 2.2a. At the start of hW-inference, we have $h = \varepsilon$ and $W = \emptyset$. Suppose the system under inference is in state q_2.

$$q_2 \underbrace{\xrightarrow{\varepsilon}}_{h} q_2 \underbrace{\xrightarrow{a/0}}_{x=a} q_2 \underbrace{\xrightarrow{\varepsilon}}_{w=\varepsilon} q_2 \underbrace{\xrightarrow{\varepsilon}}_{h} q_2 \underbrace{\xrightarrow{b/0}}_{x=b} q_3$$

We apply $h = \varepsilon$ for homing. Since $W = \emptyset$, the state $p_1 = H(\emptyset)$ is complete ($\phi(p_1) = \emptyset$) and we learn immediately the first transition $a/0$. Again due to $W = \emptyset$ there is only the state p_1, and we learn a loop. Similarly, we learn a loop for $b/0$. We end up with the "daisy" machine in Figure 2.3a. This is always the case after the first subinference with empty h and W.

We now query the oracle for a counterexample, which we suppose to be a.

$$q_3 \xrightarrow{a/1} q_1$$

Applying the counterexample triggers an h-inconsistency, because we observed $h/\varepsilon.a/0$ before and $h/\varepsilon.a/1$ now. h is extended by a. At the same time, we have a W-inconsistency, since we observed different traces $a/0$ and $a/1$ from state p_1. So we set $W = \{a\}$.

$h = a$ is already homing. The next subinference proceeds as usual:

$$q_1 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{a/0}}_{w=a} q_2 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{a/0}}_{x=a} q_2 \underbrace{\xrightarrow{a/0}}_{w=a} q_2 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{b/0}}_{x=b} q_3 \underbrace{\xrightarrow{a/1}}_{w=a} q_1$$

We learned a state $p_1 = H(0)$ with characterization $\phi(p_1) = \{a \mapsto 0\}$ and the transitions $a/0$ and $b/0$ from it. a is a loop and b leads to another state p_2 with $\phi(p_2) = \{a \mapsto 1\}$. We now learn transitions from p_2.

$$q_1 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{b/0}}_{\alpha=b} q_3 \underbrace{\xrightarrow{a/1}}_{x=a} q_1 \underbrace{\xrightarrow{a/0}}_{w=a} q_2 \underbrace{\xrightarrow{a/0}}_{h} q_2 \underbrace{\xrightarrow{b/0}}_{\alpha=b} q_3 \underbrace{\xrightarrow{b/0}}_{x=b} q_1 \underbrace{\xrightarrow{a/0}}_{w=a} q_2$$

By now we have learned the conjecture in Figure 2.3b, which is complete. Since we learned the transition from p_2 to p_1 by $b/0$ and then looped with $a/0$, we are in conjecture state q_2. Assume, we get the following counterexample by the oracle:

$$q_2 \xrightarrow{b/0} q_3 \xrightarrow{a/1} q_1 \xrightarrow{b/1} q_1$$

This yields a W-inconsistency, as we have observed $b/0$ from state p_1. We extend W by b. $W = \{a, b\}$ is characterizing, so in the next subinference we infer a correct conjecture.

Optimizations

There are several heuristic optimizations of hW-inference, which aim to decrease either the total trace length or the number of calls to the counterexample oracle. For a detailed expostion of these optimizations, we refer the reader to [Gro+20].

3 Towards showing Termination of hW-inference

Groz et al. showed the correctness of the hW-inference method under quite a strong assumption: All counterexamples must have a distinguishing suffix that is no longer than the number of states of the system n. While such counterexamples do always exist, it might be hard to find them. An oracle that yields minimal counterexample would be more powerful than what we demand from a "minimally adequate teacher". In particular, an oracle with this guarantee is hard to implement or approximate. Random walks, for instance, may contain loops of arbitrary length before eventually yielding an inconsistency. Suppose now, that hW-inference is equipped with a minimal adequate oracle (as formally defined in Section 2.3). Does hW-inference still terminate after a finite number of counterexample queries?

In Section 3.1, we sketch how there might be counterexamples on which hW-inference makes no progress. We illustrate, how these ineffective counterexamples can in fact lead to the non-termination of hw-inference in Section 3.2. To this end, we also exploit the fact that the choice of transfer sequences in hW-inference is not sufficiently specified. Section 3.3 discusses how termination might be proven or disproven, once the choice of transfer sequences is fixed.

3.1 Fake States and Ineffective Counterexamples

When running hW-inference on a system we know and comparing the intermediate conjectures with the real system, we observe an interesting phenomenon: There are conjecture states with a characterization that is impossible for any real state of the system. We call these states *fake states*. Formally, $p \in \tilde{Q}$ is a fake state, if $\nexists q \in \tilde{Q} : \forall w \in W : \phi(p)(w) = \lambda(q, w)$. How can fake states occur? Recall there are two possibilities, how new states are learned in hW-inference: (a) As tail state of a homing sequence or (b) at the end of a transition. In both cases, we can learn fake states.

(a) *h-ambiguous:* If h is not homing, there is a response r, such that the trace h/r has different tail states $q_1, ...q_k$. We then call the conjecture state $p = H(r)$ is *ambiguous*. Every time a sequence w is applied to learn its characterization, we are in one of the states $q_1, ...q_k$. For the characterization of p we have $\phi(p)(w) \in \{\lambda(q_i, w) \mid i = 1, \dots, k\}$ for all $w \in W$ (see Figure 3.1a).

(b) *W-ambiguous*: If W is not characterizing, multiple real states can be considered as one conjecture state as well. Consider the conjecture in Figure 3.1b. Let $q_1, q_2 \in Q$ have the same characterization $\phi(q_1) = \phi(q_2)$ with respect to W. Learning the transition from q_3 to q_2 will thus learn the conjecture transition $p_2 \longrightarrow p_1$. p_1 is now an ambiguous

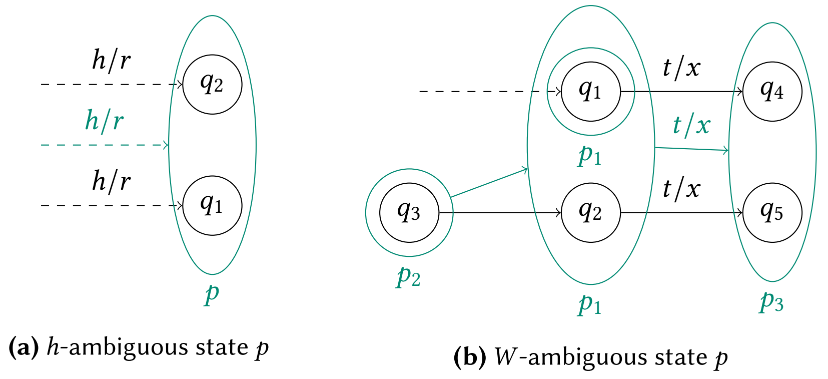


Figure 3.1 Initial ambiguous states appear during inference, if (h, W) is not homing-characterizing. The real states and transition are drawn in black, the conjecture machine in green.

state with $\phi(p_1) = \phi(q_1) = \phi(q_2)$. When learning a transition from p_2, the characterization of the tail state p_3 depends on whether we are in q_1 or q_2, when applying tw to learn p_3. In the general case with states $q_1, ...q_k$ in state p, we have $\phi(p_3)(w) \in \{\lambda(\delta(q_i, t), w) \mid i = 1, \ldots, k\}$ $(w \in W)$. p_3 can thus be a fake state.

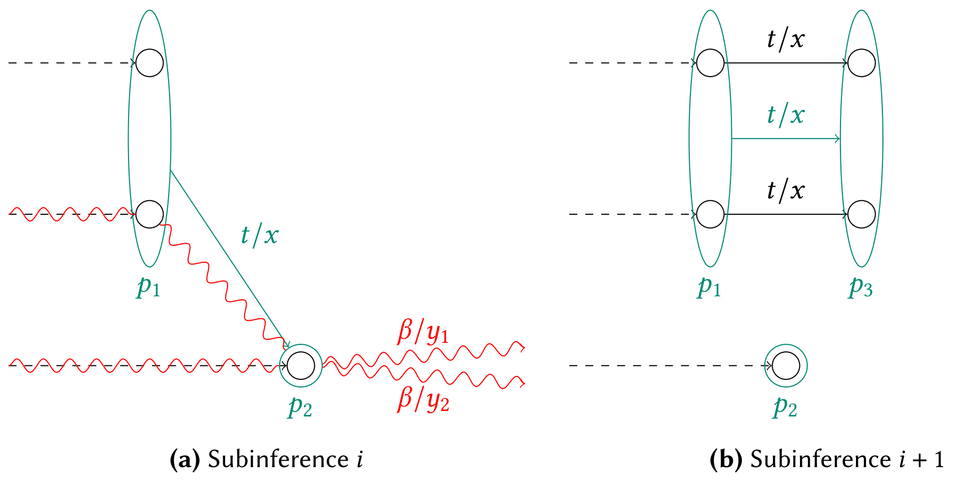


Figure 3.2 Example of how a counterexample can lead to the discovery of a new fake state p_3 $(\phi(p_3) = \phi(p_2) \cup \{\beta \mapsto y_1\})$ instead of making progress.

The possibility of fake states makes it difficult to account for the progress of counterexamples. Every increase of W, increases the upper bound for the number of fake states to $n^{|W|}$. Every counterexample can lead to the discovery of a new fake state. Assume the example sketched in Figure 3.2 a counterexample disproves that a transition $\tilde{\delta}(p_1, t) = p_2$ exists, by observing two traces β/y_2 from p_2 and $t/x.\beta/y_1$ from p_1 with $y_1 \neq y_2$. Thus, p' cannot be the state reached after p by t. However, if p is ambiguous, we can discover a fake state $p_3 = \tilde{\delta}(p_1, t)$ with $\phi(p_3)(\beta) = y_2$. p_3 is a new state. This counterexample did not eliminate the ambiguity in state p_1. We term this phenomenon *ineffective counterexamples*.

A major challenge in proving (or disproving) termination of hW-inference is to show that ineffective counterexamples can only appear a finite number of time. As seen above, the characterization of states depends on the order in which the different incoming paths (and thus real states) are used when learning the state.

3.2 Infinite Machines with Bad Transfer Sequences

We now give an an concrete example to illustrate that the problem of ineffective counterexamples can prevent termination of hW-inference. To have ever new fake states, ambiguous states are reached in a specific order, so as to have fake states with different characterizations. We ensure this order by the choice of transfer sequences from complete to partial states.

Recall that when we are in a complete state after homing, we choose a transfer sequence α to another complete state that has a transition to a partial state (line 12 in Algorithm 1). The only requirement is to choose a state and a transfer sequence such that the latter is as short as possible. However, there can be multiple states at the same distance, and, moreover, multiple transfer sequences leading to the same state. At first glance, it might seem indifferent which transfer sequence we choose apart from privileging short ones. We take advantage of the missing specification. We assume that transfer sequences are chosen in the most "unlucky" or adversarial way possible.

Certainly, there is no reason to assume that transfer sequences are chosen in a very specific, disadvantageous way. Most implementations implicitly choose the same transfer sequence for a pair of states. The purpose of our case is, however, to illustrate, how counterexamples can be ineffective.

We first describe the system under inference. We then show inductively, that a larger conjecture is obtained with every subinference, and that the characterization set $W^{(i+1)}$ obtained after subinference i and the counterexample processing is $W^{(i+1)} = \left\{b^j c \,\middle|\, 0 \leq j \leq i\right\}$.

System under inference

Consider the finite state machine $\mathcal{M}$ shown in Figure 3.3. It consists of five states $Q = \{q_h, q_c, q_0, q_1, q_x\}$, the input alphabet $I = \{a, b, c\}$ and the output alphabet $O = \{0, 1, 2, 3\}$. From every state there is a transition to state q_h by input c . The sequence $h \coloneqq c$ is thus trivially a homing sequence and $\forall r\ :\ H(r) = q_h$. c is also a distinguishing sequence for all states except for q_0 and q_1, since $\lambda(q_0, c) = 0 = \lambda(q_1, c)$. Inference starts in state q_h with empty h and c. In the first iteration, a daisy machine with the loops $a/0$, $b/0$ and $c/0$ is inferred and the machine is again in state q_h. Let the first counterexample be $c/2$. After this, we have $h = c$ and $W = \{c\}$.

3.2.1 Subinference 0

In the next subinference, which we indicate as subinference with index $i = 0$, the states q_h, q_c and q_x are correctly characterized as conjecture states p_h, p_c and p_x.

The states q_0 and q_1 are merged as a conjecture state $p_0 \coloneqq \{c \mapsto 0\}$. Since homing always leads to p_h, learning transitions from p_0 requires a transfer from p_h to p_0. Two transfer se-

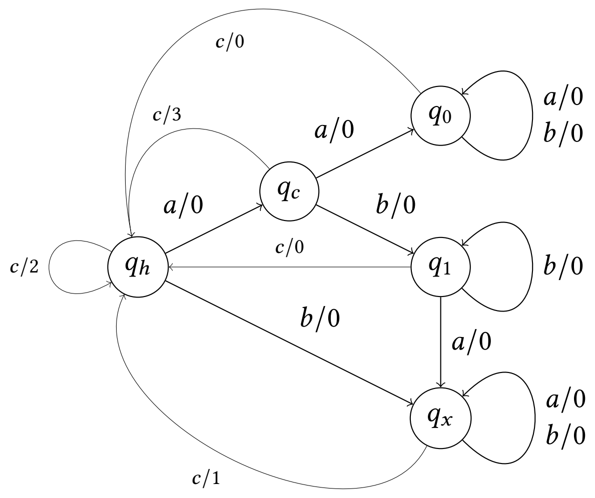


Figure 3.3 Example FSM for infinite inference.

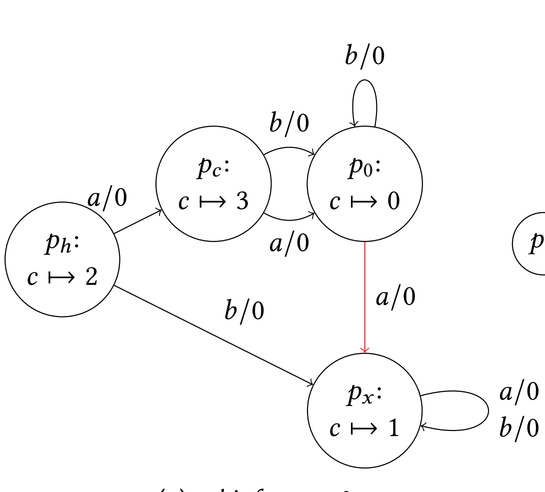
quences of shortest length come in question: $\alpha_1 = aa$ and $\alpha_2 = ab$, where α_1 leads in fact to q_0, and α_2 to q_1. Depending on the choice of the transfer sequence, we get different results for the transition, namely $\alpha_1/00.ac/00$ and $\alpha_2/00.ac/01$. That is, if we choose α_1 we learn a loop $p_0 \xrightarrow{a/0} p_0$, whereas with α_2 we learn $p_0 \xrightarrow{a/0} p_x$. Suppose we always choose α_1 for transfers to p_0. We end up with the conjecture depicted in Figure 3.4a.

At this point, hW-inference queries the counterexample oracle. Let the first counterexample be $c_0 = h/-.aaa/000.bc/00$. This triggers a W-inconsistency with the sequence $h/-.bbc/001$ that was applied before when learning transition b from state p_x, as $\tilde{\delta}(p_h, aaa) = \tilde{\delta}(p_h, b) = p_x$ and we observe $p_x \,!\, bc/00$ and $p_x \,!\, bc/01$. As a consequence, W is extended by bc.

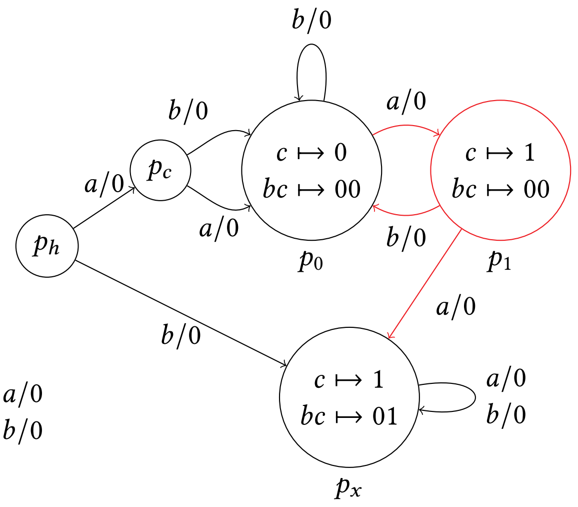
The key idea is this: The last counterexample showed that $\tilde{\delta}(p_0, a) \neq p_x$, but not that $\tilde{\delta}(p_c, a) \neq \tilde{\delta}(p_c, b)$. Since the ambiguous state p_0 persists, we can construct a fake state p_1 with $\tilde{\delta}(p_1, a) = p_x$ in the next subinference (see Figure 3.4b). The next counterexample will then disprove $\tilde{\delta}(p_1, a) = p_x$. It is ineffective and we can continue forever.

3.2.2 i-th subinference ($i > 0$)

In sub inference i, we know by the induction hypothesis, that the new characterization set is $W^{(i)} = \{w_j\} = \{b^j c\}_{0 \leq j \leq i}$.



(a) subinference 0



p_h
p_c
$c \mapsto 0$ $bc \mapsto 00$
p_0
$c \mapsto 1$ $bc \mapsto 00$
p_1
$c \mapsto 1$ $bc \mapsto 01$
p_x
$b/0$
$a/0$
$b/0$
$a/0$
$a/0$
$b/0$
$a/0$
$b/0$
$a/0$
$b/0$

(b) subinference 1

Figure 3.4 Conjecture after subinference 0 with $h = c$ and $W = \{c\}$. From every state, there is a transition leading to p_h by input c, which is not drawn. The characterizations of the states learned during inference are noted inside each state. The red parts of the conjecture are subject to the choice of a transfer sequence between p_h and p_0.

Conjecture

Before we describe the inferred conjecture, observe a property of $\mathcal{M}$: If an input that obeys the regular expression $t = axa(a|b)^*c$ with $x \in \{a, b\}$ is applied to state q_h, the output is

$$\lambda(q_h, t) = \begin{cases} 0^{|t|-1}0 & \text{if } x = a \\ 0^{|t|-1}1 & \text{if } x = b \end{cases}$$

When hW-inference infers states that are only reached by passing through p_0 and are connected by transitions of the inputs a and b, the transfer and transitions sequences only contain the inputs a and b. Since all characterization sequences are of the form b^*c, the characterization of states depends entirely on the choice of the transfer x from p_c to p_0.

Using this observation, we can infer arbitrary states with characterizations $\{b^jc \mapsto 0^jc_j\}_{0\le j\le i}$ where $c_j \in \{0, 1\}$ by choosing the right transfer sequences. Figure 3.5 shows the structure of the conjecture. We learn the same states as in the $(i-1)$-th subinference and a new state p_i, which is reached by the transition $a/0$ from p_{i-1}. The characterization of the new state is $\phi(p_i) = \left\{b^lc \mapsto 0^l1 \,\middle|\, 0 \le l \le i\right\} \cup \{b^ic \mapsto 0^i0\}$. It thus is in line with the counterexample after the previous subinference, in which we observed the trace $a/0.0b^i/0^i.c/0$ from state p_{i-1}. The characterizations of the other states p_j $(j < i)$ are extended by $w_i \mapsto 0^i0$.

For every state p_j $(j < i)$, the transition $a/0$ leads to the state p_{j+1}, and the transition $b/0$ leads to p_{j-1}, except for p_0, where we loop for b, i.e. $\tilde{\delta}(p_0, b) = p_0$ and $\tilde{\lambda}(p_0, b) = 0$. For every state there is a transition $\tilde{\delta}(p, c) = p_h$.

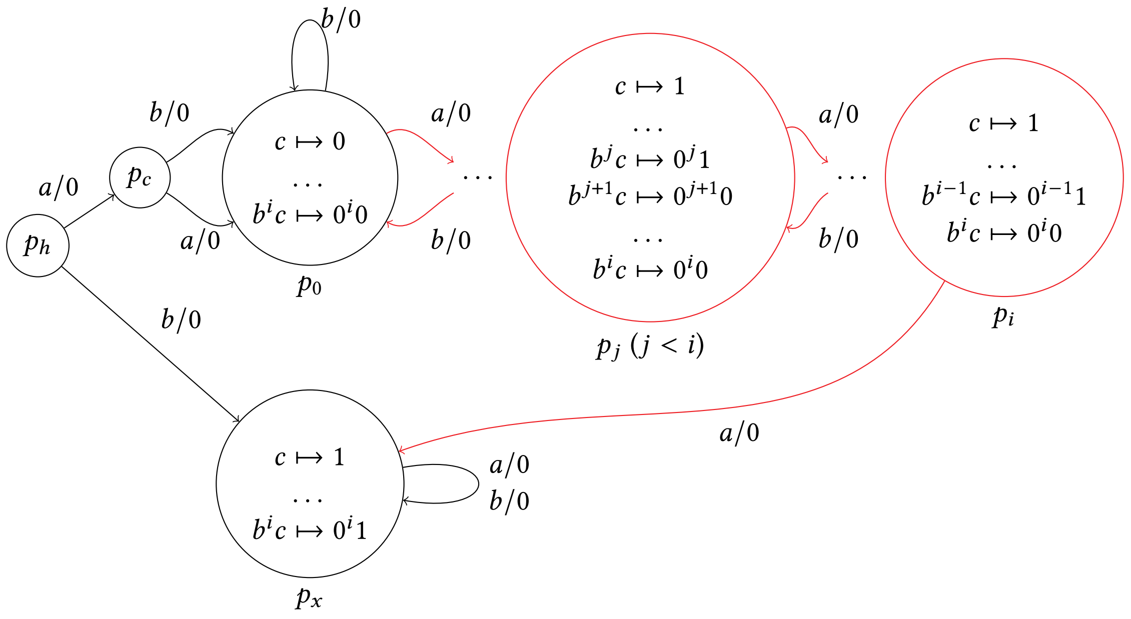


Figure 3.5 Conjecture in subinference i (transitions by c to p_h are not drawn). The characterization ϕ of nodes are written within the states, fake states are drawn in red. The counterexample is $aa/00.a^i/0^i.a/0.b^ic/0^{i+1}$

Counterexample

After subinference i, we provide as counterexample $c_i = aaa^iab^ic$ from state p_h. The counterexamples transfers to conjecture state p_x by p_i and then applies b^ic.

$$p_h \xrightarrow{aa/00} p_0 \xrightarrow{a^i/0^i} p_i \xrightarrow{a/0} p_x \xrightarrow{b^i} p_x \xrightarrow{b^i/0^i} p_x \xrightarrow{c/0} \text{inconsistent}$$

Since the real system is in state p_0 after the first aa, the output to the suffix $b^{i-1}c$ is 0^i. The counterexample hence triggers a W-inconsistency with the trace $h/-.b/0.b/0.b^ic/0^i1$ that was applied when learning transition $b/0$ from state p_x. W is thus extended by $b^{i+1}c$.

Consistency

It is left to show that the construction we described does not lead to any inconsistencies other than the inconsistencies triggered by the counterexamples.

First of all, h-inconsistencies cannot occur, as $h = c$ is a valid homing sequence.

Since in subinference i we discover the same states p_h, p_c, p_x and $p_0, \dots p_{i-1}$ as in subinference $i - 1$, the traces to learn the characterizations $w_0 \dots w_{i-1}$ are identical. For the new characterization sequence w_i, for any state p_j the application of w_i is identical with the application bw_{i-1} executed in subinference $i-1$ to learn $\tilde{\delta}(p_j, b)$. Moreover, the optimization *Search Counterexamples in the Trace* [Gro+20] checks whether the traces applied during learning are consistent with the conjecture. This is ensured by the backward transitions $\tilde{\delta}(p_j, b) = p_{j-1}$, because $\phi(p_j)(b^kc) = 0.\phi(p_j - 1)(b^{k-1}c)$.

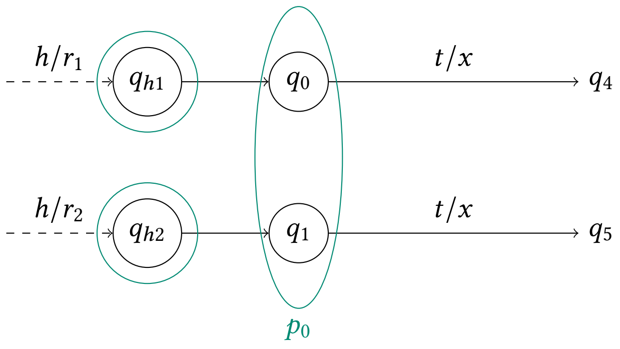


Figure 3.6

Finally, the new conjecture $i+1$ must be consistent with the $(i-1)$-th counterexample. The $(i-1)$-th counterexample observed $p_{i-1} \,!\, a/0.b^i/0^i.c/0$. The new state $p_i = \tilde{\delta}(p_{i-1}, a)$ with $\phi(p_i)(b^i c) = 0^i 0$ thus maintains consistency with the counterexample trace.

3.3 Preliminary Findings and Outlook

Groz et al. did not specify which transfer sequence is applied if there are multiple sequences of the same length, as it seemed to be indifferent. We have now seen that in fact choosing transfer sequences in a very specific way can enable an infinite inference and thus threatens the algorithm's correctness. There is nevertheless no reason why one would allow to use different transfer sequence between the same states at different time, i.e. by choosing a transfer sequence randomly. We say the choice of transfer sequences is *consistent*, if we always take the same transfer path for every pair of states. A straightforward implementation is to sort the transfer sequences of shortest length lexicographically and to always take the first one. This prohibits constructions as the one we presented, since the algorithm would always reach p_0 by aa and be in the state q_0.

While a consistent choice of transfer sequences is a necessary condition of termination, we do not know if it is also sufficient. Our example illustrated in detail the problem with ineffective counterexamples, that we sketched in Section 3.1: A W-inconsistency may lead to a refinement of W without distinguishing more real states than before. In our case, all sequences of form b^*c do not distinguish q_0 and q_1, while all other states in $Q\backslash\{q_0, q_1\}$ were already distinguished by W.

There could be machines similar to $\mathcal{M}$, where q_0 and q_1 are reached from different homing-tail states (see the sketch in Figure 3.6). The characterization of states reached through p_0 depends on the order in which the traces h/r_1 and h/r_2 occur. If h is not homing, the analogous problem might occur with an h-ambiguous state such as in Figure 3.1a. In both cases, hW-inference would not terminate, even though transfer sequences are chosen consistently.

So a proof of correctness of hW-inference must in particular rule out the possibility that we visit an ambiguous state in an order that leads to the construction of arbitrarily many fake states. We neither found an example in which such a bad visiting order did occur, nor could we exclude its possibility. So this remains an open challenge for any further research on hW-inference.

One possible attempt we pursued was to show that the number of different orders in which two real states (such as q_0 and q_1 in the previous example) can be visited during hW-inference is bounded. The order in which we visit states depends on the order in which homing responses $r \in \mathrm{dom}(H(r))$ occur. The next homing response is determined by the tail state of the application of one sequence $w \in W$ at the end of previous iteration of the backbone algorithm. For each partial state, there is an index pointing to the next sequence in $w \in W$ to be applied to this state. If there are k partially characterized states, we have $|W|^k$ different combinations of next sequences w to apply. Hence the number of possible orders in which homing responses occur, grows with the size of W. This indicates that the number of visiting orders of ambiguous states and thus the number of different characterizations of fake states is unbounded. We were however not able to construct an example that leverages this effect to learn a conjecture with size increasing in $|W|$, which is also consistent with the trace.

4 h-forests

In Chapter 3 we saw that our present understanding of hW-inference is still incomplete. We could not rule out the possibility of finite state machines on which inference does not terminate for an infinite sequence of counterexamples. To mitigate this issue we propose an adapted version of hW-inference, h-forests, that infers provably a correct model, if h is homing.

Ineffective counterexamples require the existence of fake states. A necessary condition of fake states is to learn states passing through ambiguous states. The idea of h-forests is to choose transfer sequences in a more conservative way than hW-inference and thus to avoid passing potentially W-ambiguous states are avoided. Under the assumption that h is homing, we present the algorithm (Section 4.1) and prove its termination (Section 4.2). In Section 4.3, we sketch how to possibly overcome this assumption with a probabilistic search for h-inconsistencies.

4.1 Method

From now on, we assume that h is a correct homing sequence. Consequently, for any sequence $\alpha \in I^*$, the trace $h/r.\alpha/x$ has a unique tail state. The core idea of h-forests is to access every conjecture state only by a single path along a tree rooted in $H(r)$, hence the name. This idea is implemented by the following adaptions to hW-inference.

1. Every conjecture state is labeled with a response to h. The function $\text{root}(p)$ indicates the label for state p. Whenever we observe a full characterization for the first time, viz. learn a new state, this state is labeled. A new tail state $p = H(r)$ after homing is labeled with $\text{root}(p) = r$. A new state after a transition $p = \tilde{\delta}(p', t)$ is labeled as its predecessor, i.e. $\text{root}(p) = \text{root}(p')$. The transition from p' to p is called a *tree edge*. We call the set of nodes with label r the *tree* of r: $T(r) = \left\{p \in \tilde{Q} \,\middle|\, \text{root}(p) = r\right\}$.

   If we learn a homing tail state $H(r')$ and there already exists a state p with $\phi(p) = \phi(H(r'))$, we set $H(r') =\perp$ to indicate that $T(r') = \emptyset$.

2. When choosing a shortest transfer sequence $\tau = (H(r), ..., p)$ from $H(r)$ to learn a transition $\tilde{\delta}(p, t)$, τ must only contain tree edges. If there are multiple shortest transfer sequences of the same length, choose the lexicographically smallest sequence.

3. A tree $T(r)$ is complete, if no state in $T(r)$ has a transition to a partial state. If we are in state $H(r)$ after homing, and $T(r)$ is complete, we apply a sequence $\alpha/-.h/r'$, such that $T(r')$ is incomplete. If the conjecture is not connected, ask for a counterexample.

4. There are at least two characterization sequences in W, i.e., $|W| \geq 2$. Initially, these can be symbols in I, h or random input sequences.

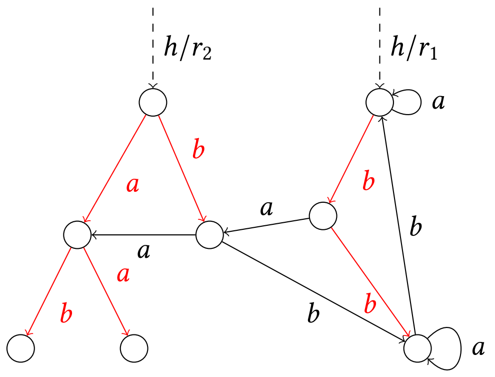


Figure 4.1 Sample conjecture searched with the h-forests method. Tree-edges are drawn in red. Only those are used in transfer sequences to partial states.

One can think of the conjecture as of a forest of BFS-trees (Breadth First Search) that are rooted in the homing tail states. Figure 4.1 contains an example. Conditions 1 and 2 ensure that there is a uniquely defined tree-path to reach every node from a homing tail state. This path only consists of tree-edges of the BFS-tree. Cross-edges or backward-edges within a BFS-tree as well as transitions between trees are not traversed in transfer sequences. This is the main difference to hW-inference, which allows to use all kind of edges in transfer sequences.

4.1.1 Find Transfers

Compared to original hW-inference, h-forests restricts the way to transfer to a state with unknown transitions. Condition 3 demands that if the current BFS-tree is complete, we can transfer to the root of another partial tree. Suppose, for instance, we are in state $H(r_1)$ in the conjecture drawn in Figure 4.1. Then we may not traverse directly to one of the states with unknown transitions, but must first apply an input sequence α with $H(r_1) \,!\, \alpha/-.h/r_2$. We now show that condition 3 is satisfiable, if the conjecture is connected.

Suppose we have observed h/r_1 and all states reachable by tree-edges are fully characterized. Then there must be a transition $p_2 = \tilde{\delta}(p_1, t)$ ($\mathrm{root}(p_1) = r_1$ and $\mathrm{root}(p_2) = r_2 \neq r_1$) to another tree or the conjecture is not connected. Let τ be the unique tree-path from $H(r_2)$ to p_2. Since p_2 was labeled with r_2, and $|W| \geq 2$, we have already traversed two times from $H(r_2)$ to p_2. We thus observed $h/r_2.\tau/-.w_1/-.\alpha/-.h/r_2.\tau/-.w_2/-$ before with some sequence α. We can now transfer to the other tree, by going from $H(r_1)$ to p_1 and then applying $t.w_1.\alpha.h$. This yields h/r' and transfers to the other tree. If $T(r_2)$ is complete, there must be a transfer to the root of another tree with label r_3 by the same argument. Eventually we must reach the tree containing a partial state.

4.2 Termination

We show that hW-inference h-forests terminates and infers a correct conjecture, by showing two Lemmas.

Bounded Conjecture Size

Lemma 4.1 *For every conjecture state $p \in \tilde{Q}$, there is at least one real state $q \in Q$, with $\phi(p) = \phi(q)$. There is a mapping $\pi : \tilde{Q} \rightarrow Q$, that maps a conjecture state to a corresponding real state.*

Proof: Since h is homing, h/r has a unique tail state. Consequently, for every input sequence τ, the trace $h/r.\tau/-$ has a unique tail state as well. When learning a state p for the first time, we go by a unique path $h/r.\tau/-$ of tree-edges before applying each $w \in W$. As a consequence, $\phi(p) = \phi(q)$, where q is the tail state of $h/r.\tau/-$ and we can set $\pi(p) = q$. □

A direct consequence of Lemma 4.1 is that the number of conjecture states is bounded by the number of states of the real system: $|\tilde{Q}| \leq n$. The conjecture cannot have any fake states.

Finding New States

The intuitive idea of W-inconsistencies is to unveil that at some point during inference two states were perceived as one state. The refinement of W aims at "splitting up" the conjecture state by distinguishing real states that were not distinguished before. In order to formally proof the notion of splitting up states, we have to define a relation between states in different subinferences.

Definition 4.1 (Child states) *Let $\tilde{Q}^{(i)}$ and $W^{(i)}$ be the set of states and the characterization set in the i-th subinference. For any following subinference $j > i$, we define the* child states *$c(p)$ of a state $p \in \tilde{Q}$ as the states that have the same characterization as p with respect to $W^{(i)}$, i.e. $c(p) = \left\{p' \in \tilde{Q}^{(j)} \,\middle|\, \phi_{|W^{(i)}}(p') = \phi(p)\right\}$.*

Note that a state $p \in \tilde{Q}^{(i)}$ does not necessarily have a child state in subinference $i+1$. This is possible, if we observe completely different homing responses in subinference $i+1$ than in subinference i and the state machine is not connected for the current W. However, because of Lemma 4.1 the possible characterizations of conjecture states are restricted to characterizations of the real states. So for some later inference $j > i$, there must be a child state of p.

If a state in subinference i has more than a single child state in subinference j, this means that $W^{(j)}$ distinguishes more states than $W^{(i)}$. On the other hand, if $W^{(i)}$ already distinguishes a state q from all other states, the state p with $\pi(p) = q$ can only have a single child state for every refinement $W^{(j)}$ of $W(i^)$ (otherwise there would be two states with $q_1 \neq q_2$ with $\phi_{|W^{(i)}}(q_1) = \phi_{|W^{(i)}}(q_2)$). We can use this observation to detect W-inconsistencies: If a state $p^{(i)} \in \tilde{Q}^{(i)}$ has only a single child state $p^{(j)} \in \tilde{Q}^{(j)}$ ($j > i$), and we observe sequences $p^{(i)}\,!\,\beta/y_1$ and $p^{(j)}\,!\,\beta/y_2$ with $y_1 \neq y_2$, we extend $W^{(j)}$ by the shortest suffix in β, that is not yet in $W^{(j)}$.

With this extended notion of W-inconsistencies, we can show:

Lemma 4.2 *Every counterexample and the ensuing refinement of W increases the number of states distinguished by W.*

Suppose a counterexample in subinference i triggers the W-inconsistency $\sigma_1 = h/r_1.\alpha_1/-.\beta/y_1$ and $\sigma_2 = h/r_2.\alpha_2/-.\beta/y_2$ with $y_1 \neq y_2$ and $\tilde{\delta}(H(r_1), \alpha_1) = \tilde{\delta}(H(r_2), \alpha_2)$. As a consequence $W^{(i)}$ is refined: The shortest suffix γ of β that is not yet in $W^{(i)}$ is added to $W^{(i)}$.

After γ was added to W, we are in one of the following cases. We show that in every case either $W^{(i)} \cup \{\gamma\}$ distinguishes more states, or another W-inconsistency is triggered and a longer suffix $t\gamma$ of σ_1 or σ_2 is added to W. After this refinement we are again in one of the cases below, until eventually W distinguishes more states or contains $\alpha_1\beta$ or $\alpha_2\beta$.

Let $p \in \tilde{Q}^{(i)}$ be the state, in which γ was applied during the counterexample.

1. p has the same predecessor state p' in both traces σ_1 and σ_2 with $\tilde{\delta}(p', t) = p$. This is necessarily the case for the common tail of both traces, i.e. if γ is a real suffix of β.

   In this case, let $j > i$ be the next subinference, in which a child state $p'^{(j)}$ of p' appears. If p' has more than one child state, $W^{(j)}$ distinguishes more states than $W^{(i)}$ and we are done. Otherwise, we can assume that $p'(j)$ is the single child of p'. We thus check for inconsistencies between both. Because $\gamma \in W^{(j)}$, we apply $t\gamma$ to $p'(j)$ to learn the tail state of transition t and get response z. During the counterexample we have observed two different traces $p' \,!\, t\gamma/z_1$ and $p' \,!\, t\gamma/z_2$ ($z_1 \neq z_2$) in p'. So z is unequal with either z_1 or z_2, which yields another W-inconsistency.

2. p has a different predecessor states p_1 in σ_1 and p_2 in σ_2, with $\tilde{\delta}(p_1, t_1) = p$ and $\tilde{\delta}(p_2, t_2) = p$ This is the case in the state, where both traces meet.

   Let $j > i$ be the next subinference, in which child states $p_1^{(j)}$ and $p_2^{(j)}$ of both, p_1 and p_2 occur. As in case 1 we can assume that $p_1^{(j)}$ and $p_2^{(j)}$ are single child states.

   Both states must have a subsequent state $p_1'^{(j)} = \tilde{\delta}(p_1^{(j)}, t_1)$ and $p_2'^{(j)} = \tilde{\delta}(p_2^{(j)}, t_2)$. $p_1'^{(j)}$ must be a child state of p. If it is not a child state of p, then for some $w \in W^{(i)}$ the responses for $p_1 \,!\, tw$ and for $p_1^{(j)} \,!\, tw$ deviate. By Lemma 4.1 there are then two states $q, q' \in Q$ with $\phi_{|W^{(i)}}(q) = \phi_{|W^{(i)}}(q')$, but $\lambda(q', t_1 w) \neq \lambda(q, t_1 w)$. In this case, we extend W by tw_1 and increase the number of states distinguished by W. Analogously, $p_2'^{(j)}$ must be a child state of p as well. If p does not have more than a single child, this means that $p_1'^{(j)} = p_2'^{(j)}$.

   The transitions from $p_1^{(j)}$ and $p_2^{(j)}$ lead to the same state and this state only has one response to $\gamma \in W^{(j)}$. Thus, either learning the transition t_1 from $p_1^{(j)}$ is inconsistent with σ_1 or learning the transition t_2 from $p_2^{(j)}$ is inconsistent with σ_2. $W^{(j)}$ is extended by $t_1\gamma$ or $t_2\gamma$.

3. p has only a predecessor state p' in one of the traces σ_1 or σ_2, with $p = \tilde{\delta}(p', t)$. This is the case for the part of the traces, before they reach the same state.

   Without loss of generality, assume that p' only appears in σ_1. The inconsistency in p then must have occurred when learning a transition. There are thus traces $h/r_1.\alpha_1'/-.\gamma/y_1$, which is a suffix of σ_1 and $h/r_3.\tau/-.\gamma/y_2$, which was executed when learning the transition t from p', and $\tilde{\delta}(H(r_1), \alpha'_1) = \tilde{\delta}(H(r_3), \tau) = p$.

As in the other cases, let $p'^{(j)}$ be the single child state of p' in a later subinference. We apply $p'^{(j)} \,!\, t/-.\gamma/z$ to learn transition t from $p'^{(j)}$. If $z = y_1$, the number of states distinguished by W was increased by adding γ to W, since we had $\phi(p)(\gamma) = y_2$ in the previous subinference. If $z = y_1$, there is an inconsistency with the trace σ_1 and we extend $W^{(j)}$ by $t\gamma$.

If the number of states distinguished by W does not increase, we thus keep adding longer suffixes of σ_1 or σ_2 to W. Eventually, there is a W-inconsistency with $\alpha_1\beta$ or $\alpha_2\beta$ right after h/r_1 or h/r_2, respectively. However, this would be an h-inconsistency which contradicts the fact, that h is homing. So the refinement of adding some suffix of σ_1 or σ_2 must have had increased the number of states distinguished by W. □

Therefore, every counterexample increases the number of conjecture states by Lemma 4.2. Thus, after at most n counterexamples, W distinguishes n conjecture states. Due to Lemma 4.1, every conjecture state has the characterization of a real state and thus the conjecture is correct.

4.3 Find h

Suppose now, that h is not homing, but we are provided with an upper bound $n \leq |Q|$ on the number of states of the system under inference. Since h is not homing, Lemma 4.1 does not hold true anymore and there can be fake states in the conjecture.

We can use a similar procedure as Rivest and Schapire [RS93]. As soon as there are $n + 1$ conjecture states, we know that h is not homing and that there must be at least one fake state. At this point, we can stop inference and start searching actively for an h-inconsistency.

Let p_f be a fake state and τ/x the tree path from the root $H(r_f)$ to p_f ($r_f = \mathrm{root}(p_f)$). Then there must be at least two states $q_1, \ldots, q_k \in Q$ such that $\forall 1 \leq i \leq k : \lambda(q_i, h\tau) = r_f x$ and $q'_1 = \delta(q_1, h\tau), \ldots, q'_k = \delta(q_2, h\tau)$ are pairwise distinct. Because p_f is a fake state, for every state q'_i ($1 \leq i \leq k$), there is a sequence $w \in W$, such that $\lambda(q'_i, w) \neq \phi(p_f)(w)$. This observation suggests a strategy to find an h-inconsistency:

1. Choose at random a conjecture state p. Apply a transfer sequence to p, such that the trace ends with $h/\mathrm{root}(p).\tau/-$, where τ is tree path of p.

2. Choose at random a sequence $w \in W$ and apply it to p. If we observe a response different to $\phi(p)(w)$ an h-inconsistency is found.

With probability of at least $\frac{1}{n+1}$ we pick the fake state $p = p_f$ and with probability $\frac{1}{|W|}$ we choose a sequence, that triggers an h-inconsistency. If we repeat the procedure $|W|(n+1)\ln(1/\epsilon)$ times, the probability to find an h-inconsistency is:

$$1 - \left(1 - \frac{1}{|W|(n+1)}\right)^{|W|(n+1)\ln(1/\epsilon)} \geq 1 - \frac{1}{e}^{\ln(1/\epsilon)} = 1 - \epsilon$$

After every h-inconsistency, h is refined. After at most n refinements, h is homing (see Section 2.2).

However, we do not know a bound on $|W|$. In Section 4.2 we saw, that if h is homing, the number of counterexamples is bound by n and we add not more than all suffixes of a counterexample to W. This would imply a bound of $|W| \leq n\theta$, where θ is the length of the longest counterexample. Nevertheless, we do not know whether this fact still holds true if h is not homing.

In order to use the approach sketched above one has to show, that if h is not homing after a bounded number of counterexamples there are more than $n + 1$ conjecture states. We suppose that this is true, but have not worked out a technical proof yet.

4.3.1 Extending to Unknown n

In black-box inference, we do not know a priori an upper bound on the number of states n. We can work around this restriction by starting with an initial estimate for n. Every time the conjecture has $n + 1$ states, but we cannot find an h-inconsistency, we double the estimate of n. After at most $\log_2(n)$ refinements, the estimated bound is greater or equal to the real bound.

Summary

We conclude the main contributions of this chapter.

1. The h-forests modification ensures termination in the setting of a minimally adequate teacher, if it finds a correct homing sequence. This partially solves the problem of ineffective counterexamples.

2. If we can show that the number of conjecture states necessarily exceeds n, if h is not homing, we can find homing sequence with a probabilistic strategy.

5 Heuristic: W-set pruning

5.1 Motivation

In the example in Chapter 3 new sequences are added to W, without distinguishing more states of the system under inference. This observation raises a more general problem: With every W-inconsistency either a new sequence is added to W or an already present sequence is extended. However, we never remove sequences in W that are distinguished by sequences that were added later. As we will later see, experiments confirm that the size of W varies a lot depending on the provided counterexamples (Section 6.2.2). This translates to a large variation regarding the length of the total trace (see Figure 6.2).

There are two measures to which refer by the term "size of W": the number of sequences in it and their average length. To learn a transition, we apply an input sequence $h.\alpha.w$ for every sequence $w \in W$, so the number of input-output-queries depends linearly on both measures.

5.2 Strategy

Our idea is to reduce the W-set based on the conjecture. We demand that the reduced W-set is still a characterizing set for the inferred conjecture. There are two approaches how to obtain such a reduced W-set.

Compute W from Scratch One approach is to compute a characterization set for the current conjecture from scratch. Although the problem of finding a minimal characterization set for a FSM is PSPACE-Complete [BJT20], there are heuristic approaches to obtain a small W-set. Indeed, we were able to compute a W-set with a total length of no more than 12 symbols for the complete Scanette FSM with such a heuristic. This is half the size of the smallest W-set for Scanette found by hW.

However, in hW-inference using an entirely new characterization set in a later subinference drastically limits the use of the dictionary optimization. The dictionary caches the responses for input sequences $h.\alpha.w$. Since the same transfer sequences and characterization sequences are applied across subinferences, this reduces the number of output queries significantly. By replacing all sequences in W, most dictionary entries become irrelevant. We found that on average about 60 % of all transitions are looked up in the dictionary. In view of this impact of the dictionary, we suppose that the advantage of having a smaller W would be outweighed by the drastic reduce of dictionary look-ups.

Prune Existing W A different approach is to take the current W as a starting point and greedily shrink it while preserving its characterizing property (see pseudocode in Algorithm 3).

Algorithm 3 W-set pruning

```
procedure PRUNEW(conjecture $\mathcal{M}'$, current $W$-set $W$)
    while $\exists w \in W\colon W \leftarrow W \setminus \{w\}$ is characterizing for $\mathcal{M}'$ do
        $W \leftarrow W \setminus \{w\}$
    end while
    while $\exists w \in W\colon (W \setminus \{w\}) \cup \{w.popLast()\}$ is characterizing for $\mathcal{M}'$ do
                                        ▷ $w.popLast()$ returns a copy of $w$ without the last symbol
        $W \leftarrow (W \setminus \{w\}) \cup \{w.popLast()\}$
    end while
return $W$
end procedure
```

In a first phase, we remove sequences from W one-by-one while the remaining W is still characterizing for $\mathcal{M}'$. In the second phase, we remove repeatedly the last input symbol of some sequence in W, while the remaining W is still characterizing. The advantage of this approach is that every sequence in the reduced W-set is a prefix of some sequence in the prior W-set. We can thus reuse the entries in the dictionary.

We can reduce W with this procedure every time the backbone algorithm returns a conjecture and before we apply the counterexample. This order is essential because the counterexample contradicts by definition the conjecture. The characterization sequence found by the counterexample thus would be removed during the pruning, which uses the conjecture as a reference.

Moreover, sometimes an inconsistency leads to increasing the W-set but the same conjecture is inferred in the next subinference. This can happen in particular for Moore locks [Moo56]. In a Moore lock there is a pair of states that can only be distinguished by one specific sequence. When a counterexample contains this sequence, we may only extend W by a suffix of it, which will not distinguish more states. In this case, multiple counterexamples are needed until W distinguishes a new sequence and the conjecture changes. We address this issue by only calling the reduce W procedure when the conjecture has increased in size.

6 Experiments

Previous experiments with hW-inference focused on randomly generated machines with up to 3000 states and two inputs [Gro+20; GBS19] or case studies on real systems of small size ($|Q| \cdot |I| \approx 100$) [BG19]. In this chapter, we evaluate hW-inference on *Scanette*, a supermarket scanner system that can be modeled by an FSM with 121 states and 15 inputs.

Before we present the numerical results in Section 6.2, we discuss challenges inferring an abstract finite state model from a non-finite state system, such as Scanette (Section 6.1).

6.1 Case Study: Scanette

Scanette models a self-service system of scanners that are typically found in French supermarkets. Customers have a handheld scanning device, they use to scan the barcode on every item they intend to buy. At the end of their shopping, they connect the scanner to a checkout and pay the required amount on the checkout.

There are two complications to this standard procedure: Some products, e.g., loose fruits, are unknown to the scanner and must be manually rescanned and weighed by a cashier. If such a product is among the courses, a cashier logs into the checkout and scans the product. Furthermore, there are controls on a random basis. With a certain probability the checkout demands for verification upon connection of the scanner. During the verification phase, a cashier rescans a sample of the items in the customer's caddie.

The PHILAE project implemented a simulation of such a shop-scanner system in Java, which is called Scanette. The classes `Scanette` and `Checkout` as listed in 6.1 give an overview of the possible interactions of a client or cashier with the system. For a detailed specification of Scanette, we refer the reader to [Phi19][1]. In the following, we focus on the parts that are challenging to write a proxy for finite state machine inference.

6.1.1 Abstracting FSM

Scanette is not equivalent to a finite state machine and thus we have to abstract from its behaviour in order to infer a model with hW-inference. This abstraction mainly happens in the driver, the component that proxies the system under inference towards the learner.

Instantiation A Scanette system can consist of multiple scanner instances and checkout instances. In our driver implementation we keep a single scanner s and a single checkout c. For each method in the Scanette and the Checkout class, the driver has one input symbol

[1]There is also a Javascript implementation to interact with Scanette in a game-like manner: `https://fdadeau.github.io/scanette/`

```
class Scanette -
    int unlock()
    int scan(long ean13)
    int remove(long ean13)
    void abandon()
    int transmission(Checkout c)
```

```
class Checkout -
    int connect(Scanette s)
    double pay(double somme)
    void abandon()
    int openSession()
    int closeSession()
    int scan(long ean13)
    int remove(long ean13)
```

Figure 6.1 The interfaces of the classes `Scanette` and `Checkout` document the possible interaction with the Scanette system.

upon which the method is called. Since `Scanette#transmission` calls internally the `Checkout#connect` method of the parameter, it suffices to have one input symbol, that calls s.`transmission`(c).

Input Concretization and Output Abstraction Most methods return status codes (0 for "OK", -1 for "wrong state", etc.) that can be symbols in the output alphabet. The method `Checkout#pay` gives the change or the remainder, if the sum payed was below the balance. The driver abstracts the output by applying the signum function to the return value.

For methods, that take the bar code of an item as parameter, we add one input symbol for the bar code of every item. To this end, we chose two representative items, i.e., one item that is known to Scanette and one that has to be scanned by a cashier. For `Checkout#pay` we add two input symbols, one that pays 0, and another one that pays exactly the balance.

Externalize Non-determinism When connecting s to c, the Scanette implementation decides randomly, whether a verification is required. We add a boolean parameter to `Scanette #transmission` that controls whether the verification phase is entered or not. Accordingly, there are two inputs.

Although a client should not have the choice between these kinds of transition, it makes sense, that a learner of the system can externally control the randomness in order to be able to explore all states.

In total, the driver for Scanette has 15 different input symbols.

Counters Having a finite input and output alphabet does not yet guarantee, that the system under inference is functionally equivalent to a finite state machine. In the case of Scanette, implicit counters are a problem. Scanette counts how often an item was scanned. If a client calls `Scanette#remove` with the article number of an item, that it is not in the basket, an error code is returned. Thus, for any m `Scanette#scan(itemA)`^{m} leads to a state, that can be identified by applying the sequence `Scanette#remove(itemA)`$^{m+1}$. Technically, this number is only bounded by the size of Java integers used for counting, but clearly this number of states is by far to high.

To avoid a state explosion, we limit the number of times an item can be scanned by the driver. The driver maintains a counter for every item, that is updated according to the `Scanette#scan` and `Scanette#remove` inputs. If the counter for item *a* is equal to 2, the driver outputs an error code for the `Scanette#scan(itemA)` input and does not pass the `Scanette#scan` call to the simulator. The counter in the driver is reset, whenever `Scanette#unlock` is executed successfully, which indicates the beginning of shopping.

We entirely omit the `Checkout#remove` functionality to hide a similar counter for scanning items at the checkout.

6.1.2 Discussion

By the adjustments presented above, the behaviour of Scanette is restricted to a finite state machine. We believe these restrictions preserve the most important aspects. For one, it is intuitive to think of a system of Scanette in terms of states. The restricted Scanette system still maintains properties such as "Scanning items has a different effect during shopping than during verification" or "The checkout can only scan items after a cashier has logged in". Moreover, if we think of inference as testing, the restrictions make sense. Specifying test cases manually would also implicitly limit the number of items that are scanned.

However, in the case of Scanette, developing a driver required detailed knowledge of the specification of the system. For instance, the driver must reflect whether the internal state of the system is “Shopping” or “Verification”, since only during “Shopping” the number of `Scanette#scan` is limited. If the driver already involves such a precise understanding of the system to learn, this threatens the idea of black-box inference.

6.2 Results

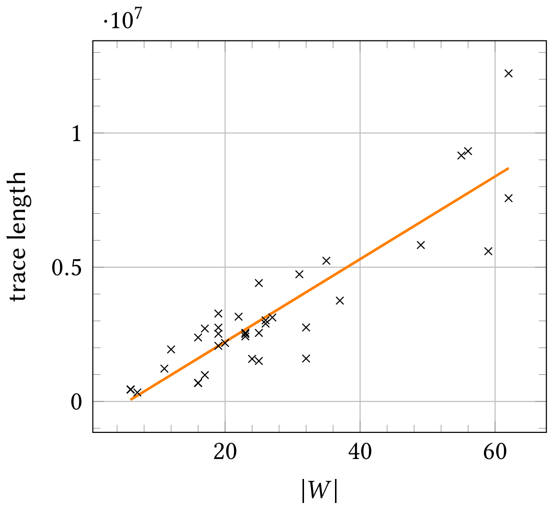
6.2.1 Model of Scanette

For the experiments with the Scanette driver, we used the implementation of hW-inference in the SIMPA tool[2]. We enabled the optimizations *add h in W*, *dictionary*, *search counterexamples in trace* and *check for inconsistency between H and the conjecture.* We use a random walk as counterexample oracle. Since *h* and *W* are constructed from counterexamples, the course of inference depends on counterexamples. We used 40 different seeds for the random walks.

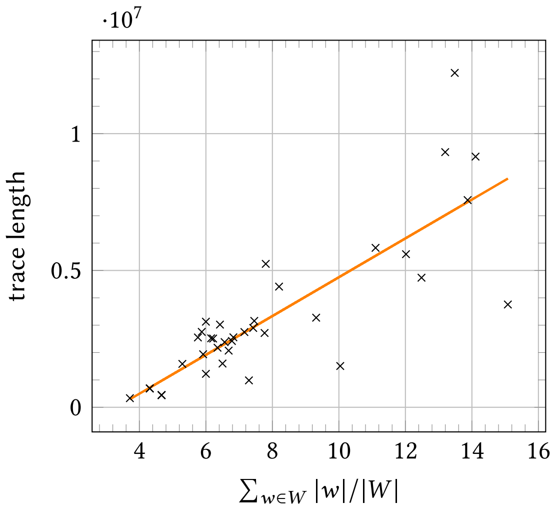
The inferred machine has 121 states. In around 30% of the inferences, hW-inference found only 120 states. This is presumably due to the counters built into Scanette. To discover all states of a counter – similar as for Moore-Locks [Moo56] – unique input sequences are needed. Indeed, after increasing the maximum length of random walks from 75,000 to 300,000 transitions, the rate of wrong machines dropped to 5%.

We do not have an explicit model of the Scanette-FSM as a finite state machine to compare against the model learned by hW-inference. However, its number of states seems plausible. The implementations of Scanette and Checkout distinguish explicitly 5 and 4 states, respectively.

[2]The source code of SIMPA is publicly available: `https://gricad-gitlab.univ-grenoble-alpes.fr/SIMPA/SIMPA`



(a) number of sequences in W



(b) average length of sequences in W

Figure 6.2 Relation between the size of the characterization set and the trace length.

Then we have two items that can be scanned 0, 1 or 2 times each. This totals to an upper bound of $4 \cdot 5 \cdot 3^2 = 180$ states, of which not all are in fact reachable.

6.2.2 Trace Length

The key measure to asses the performance of any active inference method is the length of the global trace, viz. the number of input output queries it takes to learn the system under inference. The global trace does not count interactions that are performed as part of random walks to find counterexamples. These are attributed to the counterexample oracle, which is not part of the algorithm. In theory, there could be other oracles, that do not require interaction with the system under inference.

We observed an average trace length of 3.09 million sequences, ranging from $3.33 \cdot 10^5$ to $1.22 \cdot 10^7$ with a standard deviation of $2.66 \cdot 10^6$. Depending on the counterexample provided by an oracle, the trace length can thus vary by an order of magnitude.

Figure 6.2 plots the total trace length in relation to the size of the characterization set (6.2a) and the average length of a characterization sequence (6.2b) of the final W of an inference. We observe a Pearson correlation coefficient (PCC) of 0.90 between $|W|$ and l and of 0.83 for $|\bar{w}|$ and all. The length of the homing sequence correlates less with the trace length (PCC 0.59).

One possible reason for the variation of trace length could be the presence of counters in the FSM. Since only a specific input sequence can distinguish the states within a counter, the total trace is longer, if this sequence is provided by very late counterexamples. In particular, since some counterexamples can be ineffective (see Chapter 3).

We summarize our findings:

1. For counterexamples found by random walks, the size of the final W-set varies widely.

	trace length (rel. diff.)			#subinf.		$\|W\|$ ($\sum_W \|w\|$)		#prun.
	445237	*370866*	*(16.7%)*	18	*18*	6 (28)	*3 (23)*	3
	1508355	*1389921*	*(7.9%)*	35	*35*	25 (251)	*22 (246)*	4
	2521299	*2496875*	*(1.0%)*	31	*32*	19 (117)	*10 (47)*	5
	3156945	*2868067*	*(9.2%)*	35	*35*	22 (164)	*13 (90)*	7
	9161031	*7222532*	*(21.1%)*	70	*69*	55 (776)	*14 (114)*	8
Avg.[1]	3093769	*2396832*	*(13.3%)*	38.2	*39.7*	25.7 (236)	*12.1 (100)*	6.0
stddev.[1]	2660656	*1916275*	*(36.1%)*	17.1	*15.8*	15.6 (240)	*6.6 (90)*	2.2

Table 6.1 Key figures on a sample of 5 inferences with hW-inferences without and with W-set pruning (in *italics*). From right to left: the trace length and the relative reduction by W-set pruning, the number of subinferences, the number of sequences in the final W-set and their total size, and the number of times W-set pruning was triggered during inference.

[1]The average and standard deviation are over all 40 inferences.

2. The total trace length depends linearly on the number of sequences in W and their total length, and thus varies widely as well.

6.2.3 W-set Pruning

The impact of the W-set on the trace length motivates reducing the size of W. To this end, we augmented the SIMPA implementation of hW-inference with W-set pruning as described in Chapter 5. For our experiments, we inferred the Scanette FSM using the 40 same seeds for random walks as in Section 6.2.1. Table 6.1 compares key figures between both version on a representative sample, as well as the average of all 40 runs.

On average, W-set pruning was triggered sparsely during hW-inference. The traces of hW-inference with W-set pruning were on average 13 % shorter than those of the normal hW-inference. This decrease is consistent, i.e. in almost every case, W-set pruning resulted in a shorter trace length. The trace length with W-set pruning show a similar variation coefficient as the trace length without pruning.

We observed one exceptional case, where W-set pruning almost tripled the trace length and increased the number of subinference from 24 to 46. In this case, the original trace was quite short ($6.8 \cdot 10^5$ symbols), and we pruned a useful sequence from W that distinguished many states of the real system. However, the fact that on average the number of subinferences is hardly higher for W-set pruning indicates, that we usually do not prune many useful sequences.

Note, that the number of sequences in the final W-set and their total length is on average reduced by about 50 %. The smaller W-set does not translate, however, proportionally to a shorter trace length (as it does without pruning, cf. Figure 6.2).

We summarize the effectiveness of W-set pruning as follows:

1. Extending W-set pruning on average reduce the trace length moderately but consistently.

2. However, the advantage through W-set pruning is still outweighed by far by the different quality of random counterexamples.

7 Conclusion

In this thesis, we examined hW-inference in detail. hW-inference is a deterministic algorithm that learns finite state machines through interaction. As common in active inference, the learning relies on externally provided counterexamples. We found the following theoretical and experimental results.

Theory

- hW-inference does not always terminate. We gave a concrete example of non-termination. This example exploits that transfer sequences within hW-inference might be chosen in an adversarial or "unlucky" way. This can easily be repaired by specifying that transfer sequences are chosen in lexicographic order. More generally, the example we gave illustrated that due to fake states counterexamples may be ineffective.

- The h-forests modification to hW-inference ensures termination, if a homing sequence is found or provided externally. We also sketched a probabilistic strategy to find a homing sequence. The latter method is subject to the supposition, that if h is not homing, the number of fake states increases with processing counterexamples.

Application The observation that we do not progress on some counterexamples raises awareness, that even in terminating inferences, hW-inference might use an unnecessarily large number of characterizing sequences. Motivated by this fact, we proposed a heuristic to decrease the size of the W-set.

- For the experiments, we conducted a case study on Scanette, a supermarket scanner system. Inferring Scanette showed in particular, that the trace length can vary by an order of magnitude when inferring the same system, with random counterexamples. This behaviour of hW-inference was not observed before. It indicates that ineffective counterexamples might indeed threaten performance.

- W-set pruning is a greedy strategy to remove redundant sequences form the characterization set. Experiments shows that this decreases the overall trace length by 13% on average.

Further Research First of all, the question remains open, if hW-inference always terminates, if we choose transfer sequences in lexicographic order. A potential future proof will have bound somehow the number of ineffective counterexamples.

A different approach is continuing work on h-forests. h-forests can be seen as a more conservative version of hw-inference, as it does not use learned transitions to find new states. This will necessarily increase the length of transfer sequences and thus the total trace length. However, the fact that every counterexample makes progress once *h* is homing, might reduce the number of subinferences and outweigh the effect of longer transfer sequences. This could be evaluated experimentally.

Bibliography

[Ang87] Dana Angluin. "Learning regular sets from queries and counterexamples". In: *Information and Computation* 75.2 (1987), pp. 87–106. ISSN: 0890-5401. DOI: `https://doi.org/10.1016/0890-5401(87)90052-6`. URL: `https://www.sciencedirect.com/science/article/pii/0890540187900526`.

[AP18] Florent Avellaneda and Alexandre Petrenko. "FSM Inference from Long Traces". In: *Formal Methods.* Ed. by Klaus Havelund et al. Cham: Springer International Publishing, 2018, pp. 93–109. ISBN: 978-3-319-95582-7.

[BG19] Nicolas Bremond and Roland Groz. "Case Studies in Learning Models and Testing Without Reset". In: Apr. 2019, pp. 40–45. DOI: `10.1109/ICSTW.2019.00030`.

[BJT20] Kadir Bulut, Guy Vincent Jourdan, and Uraz Cengiz Türker. "Minimizing Characterizing Sets". In: *Formal Aspects of Component Software.* Ed. by Farhad Arbab and Sung-Shik Jongmans. Cham: Springer International Publishing, 2020, pp. 72–86. ISBN: 978-3-030-40914-2.

[CMR06] Olivier Cappé, Eric Moulines, and Tobias Rydén. *Inference in hidden Markov models.* Springer Science & Business Media, 2006.

[GBS19] Roland Groz, Nicolas Bremond, and Adenilso Simao. "Using Adaptive Sequences for Learning Non-Resettable FSMs". In: *Proceedings of The 14th International Conference on Grammatical Inference 2018.* Ed. by Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek. Vol. 93. Proceedings of Machine Learning Research. PMLR, Feb. 2019, pp. 30–43. URL: `http://proceedings.mlr.press/v93/groz19a.html`.

[Gro+15] Roland Groz et al. "Inferring Finite State Machines Without Reset Using State Identification Sequences". In: *Testing Software and Systems.* Ed. by Khaled El-Fakih, Gerassimos Barlas, and Nina Yevtushenko. Cham: Springer International Publishing, 2015, pp. 161–177. ISBN: 978-3-319-25945-1.

[Gro+20] Roland Groz et al. "hW-inference: A heuristic approach to retrieve models through black box testing". In: *Journal of Systems and Software* 159 (2020), p. 110426. ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2019.110426`. URL: `https://www.sciencedirect.com/science/article/pii/S0164121219302006`.

[HMS03] H. Hungar, T. Margaria, and B. Steffen. "Test-based model generation for legacy systems". In: *International Test Conference, 2003. Proceedings. ITC 2003.* Vol. 1. 2003, pp. 971–980. DOI: `10.1109/TEST.2003.1271084`.

[IOG10] Muhammad Naeem Irfan, Catherine Oriat, and Roland Groz. "Angluin Style Finite State Machine Inference with Non-Optimal Counterexamples". In: *Proceedings of the First International Workshop on Model Inference In Testing*. MIIT '10. Trento, Italy: Association for Computing Machinery, 2010, pp. 11–19. ISBN: 9781450301473. DOI: 10.1145/1868044.1868046. URL: https://doi.org/10.1145/1868044.1868046.

[Mea55] George H. Mealy. "A method for synthesizing sequential circuits". In: *The Bell System Technical Journal* 34.5 (1955), pp. 1045–1079. DOI: 10.1002/j.1538-7305.1955.tb03788.x.

[Moo56] Edward F. Moore. "Gedanken-Experiments on Sequential Machines". In: *Automata Studies. (AM-34), Volume 34*. Ed. by C. E. Shannon and J. McCarthy. Princeton University Press, 1956, pp. 129–154. DOI: doi:10.1515/9781400882618-006. URL: https://doi.org/10.1515/9781400882618-006.

[Pet+17] Alexandre Petrenko et al. "From Passive to Active FSM Inference via Checking Sequence Construction". In: *Testing Software and Systems*. Ed. by Nina Yevtushenko, Ana Rosa Cavalli, and Hüsnü Yenigün. Cham: Springer International Publishing, 2017, pp. 126–141. ISBN: 978-3-319-67549-7.

[Phi19] ANR Philae. *Data gathering, preparation and format*. Tech. rep. WP1/D1.1. Grenoble INP - LIG, UBFC - Femto-ST, July 2019. URL: https://projects.femto-st.fr/philae/en/download/file/fid/93.

[PTR15] Alexandre Petrenko, Omer Nguena Timo, and S. Ramesh. "Model-Based Testing of Automotive Software: Some Challenges and Solutions". In: *Proceedings of the 52nd Annual Design Automation Conference*. DAC '15. San Francisco, California: Association for Computing Machinery, 2015. ISBN: 9781450335201. DOI: 10.1145/2744769.2747935. URL: https://doi.org/10.1145/2744769.2747935.

[RS93] R. Rivest and R. Schapire. "Inference of Finite Automata Using Homing Sequences". In: *Inf. Comput.* 103 (1993), pp. 299–347.

[SG09] Muzammil Shahbaz and Roland Groz. "Inferring Mealy Machines". In: *FM 2009: Formal Methods*. Ed. by Ana Cavalcanti and Dennis R. Dams. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 207–222. ISBN: 978-3-642-05089-3.

[Vaa17] Frits Vaandrager. "Model Learning". In: *Commun. ACM* 60.2 (Jan. 2017), pp. 86–95. ISSN: 0001-0782. DOI: 10.1145/2967606. URL: https://doi.org/10.1145/2967606.