ChatGPT

Reconstructing a Finite-State Automaton from Experiments

1. Theoretical Foundations: Automaton Inference and Partial Observability

Reconstructing a deterministic finite-state machine (FSM) from input-output experiments is a classic problem in automata theory and learning. As early as 1956, Moore described "Gedanken-experiments" for **state identification** – designing input sequences to deduce the state of a black-box sequential machine [1]. In our scenario, each state produces a 2-bit output label (one of 4 values), so different states may *appear identical* if they share the same output. This is a case of **partial observability**, meaning the current state cannot be directly observed, only its label. Theoretical work on *testing and identification* of FSMs addresses exactly this: how to use input sequences and observed outputs to distinguish states and infer the transition structure [1] [2]. An extensive body of research exists for Moore/Mealy machines (where outputs are emitted synchronously with inputs) on finding **distinguishing sequences** and other identifying experiments [3]. A *distinguishing sequence (DS)* is a single input sequence that produces a unique output sequence from each state of the machine [2]. If such a sequence exists, running that one experiment would reveal the initial state by its output pattern. However, not every machine has a single DS (and if one exists it can be extremely long) [4]. More generally, one uses a collection of shorter sequences, called a **characterizing set** (or *W-set*), which collectively differentiate every pair of states [5]. Each state yields a unique "signature" of outputs in response to the sequences in W, allowing the states to be distinguished [6]. These concepts – distinguishing sequences, homing sequences (which let you determine the *ending* state), synchronizing sequences (which drive the machine to a known state), etc. – form the theoretical foundation for FSM inference under ambiguous observations [2]. In summary, theory guarantees that if the machine is minimal (has no redundant states) and we know an upper bound on the number of states, we *can* design experiments to uniquely identify the state labeling and transition structure (up to isomorphism), despite partial output information [1] [6].

2. Algorithms and Complexity for Automaton Reconstruction

Over the decades, researchers have developed both passive and active algorithms for inferring finite automata. **Passive** (offline) inference algorithms (e.g. Regular Grammar Induction) require a large representative set of input/output traces, whereas **active** algorithms query the system with strategically chosen inputs. Gold's seminal result (1967) showed that inferring an exact DFA from only examples (without queries) is not algorithmically solvable in general, but with queries or complete data the problem becomes tractable. In our active setting (we can reset to the initial state and experiment freely), classic results apply. Angluin's **L** *algorithm (1987) demonstrated that a minimal DFA with n states can be learned in polynomial time using membership and equivalence queries* [7]*. That algorithm, originally for accepting automata, has since been adapted to Mealy/Moore machines** that produce outputs on each input [8]. In practice, this means we can reconstruct a state machine by asking on the order of $O(n^2 \cdot |\Sigma|)$ queries (where $|\Sigma|=6$ in our case) in the worst case – a quantity that is polynomial in the number of states. The known upper bounds are quite moderate; for example, Angluin's approach will make at most $n-1$ wrong conjectures and a polynomial number of membership queries before converging on the correct automaton [9] [10]. Modern improvements like

the TTT algorithm and others further optimize query count [11], but all rely on the same principle: systematically distinguishing states via queries.

It's important to note complexity differences between *single-sequence* identification and *multiple adaptive experiments*. Checking whether a single preset distinguishing sequence (PDS) exists for an FSM is PSPACE-complete [4], and indeed the shortest such sequence can be exponentially long in n [4]. In contrast, allowing adaptive testing (or simply multiple experiments with resets) makes the task much easier: **adaptive distinguishing sequences (ADS)** or sets of separating sequences can always be constructed for minimal deterministic machines in *polynomial* time [12]. Lee and Yannakakis (1994) proved that while a universal DS may be infeasible, one can *always* build an experiment strategy that identifies all states with polynomial effort [4]. One such strategy is the classic **partition-refinement algorithm** based on Hopcroft's DFA minimization: essentially, start by grouping states by their output label, then iteratively refine these groups by applying input symbols that reveal different outputs [12]. This yields an adaptive decision tree (a splitting tree) that separates all states. Recent research has extended these ideas – e.g. constructing splitting trees for machines that lack a single global DS [13] – but the core complexity result is that reconstructing an n-state deterministic FSM with k inputs is polynomial-time achievable with active queries. In the worst case, the number of experiments (input sequences tried) might be on the order of $O(n^2)$ or $O(n \cdot k \cdot n)$, but not exponential, and each experiment's length is also bounded by a polynomial in n (often $O(n)$ or $O(n \log n)$) in these algorithms [4].

Besides Angluin's query-learning approach, there are other algorithms and lower-bound results worth noting. **Trakhtenbrot and Barzdin (1973)** developed an early algorithm that, given an *upper bound* on states, could identify the exact automaton by systematically testing all input sequences up to a certain length. Their approach, while correct, had exponential worst-case cost (essentially trying all distinguishing experiments) unless a helpful "complete" sample is provided [14]. On the other hand, **Rivest and Schapire (1993)** showed that if one knows a *homing sequence* for the machine (an input sequence that always leads to a known state, regardless of start state), one can leverage it to learn the automaton with only membership queries (no equivalence queries) by an interactive procedure – an important result that influenced later practical algorithms. In general, the **query complexity** of active automaton learning is polynomial, but if we restrict to passive observation or insist on a single long test sequence, the problem becomes computationally hard [4].

3. Heuristics for Partial Observability: State Merging and Minimization Techniques

When outputs are ambiguous (as with 2-bit labels), a crucial heuristic is to exploit partial observations to incrementally refine the hypothesis model. One family of approaches builds a hypothesis FSM by **state merging** in an observation tree. For example, the classic RPNI algorithm (Oncina & García, 1992) takes a set of input-output traces and merges states that are *compatible* (i.e. no observed sequence distinguishes them) [15]. In our active context, we can generate traces and apply similar merging heuristics: start with a tree where each distinct observed output sequence corresponds to a node (state), then *merge nodes* that show no discrepancy on all experiments performed so far. As we perform more experiments, we either discover a new output pattern (indicating a truly new state) or we find that an unexplored path leads to an output sequence that matches an existing state's signature, in which case we collapse those nodes as one state. This is akin to **graph merging** or the "blue-fringe" heuristic in grammar inference, which greedily merges states and backtracks if a contradiction appears. Such heuristics can dramatically cut down the search space, especially when the automaton has some structure or when the number of outputs is much smaller than the number of states (our case, 4 labels vs. n states).

Another important technique under partial observability is to use **unique input/output sequences (UIOs)** for states, if they exist. A UIO for a state s is a short input sequence that, when executed from s, produces an output sequence that no other state produces [6]. If each state can be given a unique "fingerprint" this way, identification becomes easier: one can drive the machine to a state and then apply the state's UIO to confirm that you're indeed in that state (by checking the output). In practice, not every state has a very short UIO, but many testing methods attempt to find UIOs or smaller separating sequences for each state and use them in combination. When UIOs are not available, one falls back to using a **characterizing set W** (as mentioned earlier) which distinguishes all states in a group-wise fashion [6]. The **W-method** in conformance testing, for instance, constructs a test suite by taking every state-covering prefix and appending every sequence in W to it – thereby probing each state with all distinguishing sequences [16]. This method can be used for inference too: treat the unknown machine as the implementation and systematically explore prefixes and suffixes to identify transitions.

Heuristics also leverage classic **minimization algorithms** as part of inference. A minimal deterministic automaton is defined by the invariant that no two distinct states are equivalent (produce the same outputs for all future inputs) [5]. During reconstruction we can continuously check for state equivalence under the experiments done so far. If two hypothesized states have produced identical output responses on every tested sequence, one might tentatively consider them the same state. Partition-refinement (the basis of Hopcroft's minimization) can be applied on-the-fly: initially group states by their current output label; then refine partitions by splitting a group whenever we find an input that yields different next-output behavior for some members [12]. This gradually pushes the process toward the minimal, correct partition of the state space. In fact, Lee & Yannakakis's result showed that an *adaptive* strategy based on refining equivalence partitions will eventually distinguish all states in polynomial time [4]. Modern research, like Soucha & Bogdanov (2020), refines these ideas by constructing *splitting trees* and **harmonized state identifiers (HSI)** – optimized sets of sequences that distinguish states more efficiently [13]. These advanced heuristics aim to reduce test lengths and counts, but they build on the same principles: merge states that appear equivalent; split them when a new output difference is observed; and try to find short separating sequences for efficiency.

In summary, under partial observability one typically uses a combination of **state exploration** and **state distinction** heuristics. Exploration finds candidates for states (often by treating each new unexplained output sequence as a new state), while distinction ensures that any candidates that are actually the same state get merged (or that truly different states are eventually told apart by some experiment). Graph-based algorithms that merge equivalent nodes in a prefix-tree model [15], as well as adaptive sequence generation that targets currently ambiguous state groups [13], are practical heuristics to tackle the inference problem when outputs don't directly reveal states.

4. Practical Strategies for Efficient State Space Exploration

To actually **recover the transition structure and labeling** of the automaton in practice, one should plan a systematic exploration strategy. Below is a recommended approach for the base problem, assuming we can reset to the initial state for each experiment and want to minimize the number of experiments:

1. **Cover the State Space:** First, generate input sequences to ensure all n states are reached at least once. This can be done via a breadth-first search of the input space. Start from the initial state and apply each input (0–5), observe the output label, and record the resulting sequence of labels. This reveals the labels of whatever states those one-step transitions reach. Continue extending sequences stepwise until you have encountered n distinct states (as judged by differing sequences of outputs or by provisional state IDs). Ensuring a *state cover* (a set of input

prefixes reaching every state) is fundamental before trying to differentiate transitions. If outputs were fully unique per state this is trivial; under ambiguous outputs you may need to go a bit further – e.g. you might treat two sequences that produce the same output trace as reaching the *same* state unless/until a longer extension disproves that. Keep track of a representative input sequence for each discovered state.

2. **Identify Distinguishing Sequences:** Next, determine a set of suffix sequences that can distinguish all states from each other. Using the theory above, construct a **characterizing set** W of inputs [6]. One way is: for each pair of hypothesized states, try to find a short input sequence that produces different output responses when starting from those two states. You can do this by experiment: take two candidate states (reachable by prefixes p_1 and p_2) and try various suffixes t (sequences of inputs) to see if p_1t and p_2t yield different output label sequences. If so, t can serve as a distinguishing sequence for that pair. You can build a set of such sequences that collectively separate all states. In practice, a smaller approach is to find, for each state, a unique "fingerprint" sequence (similar to a UIO if available) that no other state reproduces. For example, you might find that from state $S3$, input sequence `301` produces outputs `2,3,1` which is different from the outputs that sequence produces from any other state – thus marking state $S3$. Gathering these distinguishing suffixes might be done with guidance from algorithms (e.g. using adaptive splitting: apply inputs step by step that split the largest remaining group of indistinguishable states [13]). The end result is a set W such that for every pair of states s_i, s_j, there is some $w \in W$ yielding different output sequences from s_i vs. s_j.

3. **Map Out Transitions:** Now systematically map the transitions of the automaton using the information above. For each discovered state S and each input symbol $a \in {0,1,2,3,4,5}$, perform an experiment to see what happens *from* S on input a. Concretely, take the known prefix p that brings the machine to state S, then extend it with a (i.e. input sequence $p a$), and observe the output label sequence. The immediate output tells you the label of the next state, but that next state could be one of several states sharing that label. To resolve which one, append one of the distinguishing suffixes from W and run the full sequence $p a w$. By comparing the output sequence of $p a w$ to the known "signatures" (output responses) of each state for suffix w, you can identify exactly which state $p a$ leads to [6]. This way, you determine the transition $S \xrightarrow{a} S'$ and also learn the output label of S' (which should match the first output seen after input a). By doing this for all states S and all inputs 0–5, you will have the complete transition table of the automaton. Many of these experiments can be reused or combined to reduce total number: for example, a single long sequence can test multiple transitions in a chain, as long as you can still interpret the results state-by-state. (In practice, test algorithms often create a single **checking sequence** that covers all transitions by concatenating these prefix–suffix combinations in a clever order, using resets or homing sequences in between as needed.)

4. **Merge and Minimize (if needed):** During the above process, you may encounter situations where two hypothesized states produce identical output behavior for *all* experiments performed so far. This suggests they are actually the same physical state (or are indistinguishable given the experiment set). You should merge such states in your hypothesis to avoid redundant duplicates. If you followed steps 1–3 rigorously with a correct W set, you should end up with exactly n states that are pairwise distinguishable by outputs on W, which means your model is minimal and consistent. It is still wise to perform a **minimization** check: treat your constructed graph of n states with labeled transitions and apply a DFA minimization algorithm (e.g. partition refinement based on output equivalence) – no states should collapse since the machine is minimal by assumption, but this double-checks that your inferred structure has no symmetry or

redundancy. If any mergeable states are found, it indicates the experiment set was insufficient to tell them apart, and you should design an additional experiment (a new distinguishing sequence) for that case.

5. **Extended Variant – Label Overwrites:** In the extended problem, you have an extra capability: temporarily overriding labels (perhaps denoted by square bracket operations). This can be exploited to simplify the inference. Essentially, a *label overwrite* lets you tag a state with a unique marker visible in the outputs. A practical strategy is to mark each new state upon discovery and then trace whether other paths lead to the same marked state. For example, if you reach a new state S_x via some input sequence, you could use an operation `[X]` that forces the state's output to a distinctive value (without changing its transitions). Thereafter, run other test sequences and see if the output `[X]` appears – if it does, it means those sequences also ended up in $S_x`. This approach can **bypass the need for complex distinguishing sequences**, since you are actively labeling states to recognize them. Using label overwrites, one can iteratively: mark a state, explore all inputs from it to label where they go, unmark (restore) it, move to the next state. This yields a correct solution with far less thinking about minimal experiment count (we trade optimality for convenience). The downside is that it's not something you can always do in a real system (it's like a cheat), but since the extended task allows it, it provides a straightforward path: essentially make the machine fully observable by unique labels, then do a simple BFS exploration of transitions. Each transition can be identified in one experiment because the destination state will *tell you* its identity via the override label. While this may not minimize the number of experiments, it guarantees a working reconstruction of the automaton.

Key References: Foundational work by Moore [1] and others established the viability of state-identification experiments. Lee & Yannakakis analyzed the complexity of finding distinguishing sequences and showed the polynomial-time construction of adaptive strategies [4]. Active learning algorithms like Angluin's L^* and its Mealy-machine adaptations provide polynomial bounds on queries [7]. In testing theory, methods such as Chow's W-method and HSI (harmonized state identification) leverage characterizing sets to differentiate states [5] [13]. Heuristic approaches, including state merging algorithms from grammatical inference [15] and adaptive splitting trees [13], offer practical ways to handle partial observability. By combining these insights – covering states, distinguishing outputs, and efficient experiment design – one can confidently recover the complete transition graph and labeling of the unknown automaton with minimal experiments in the base scenario, and even more easily in the extended scenario with state-marking operations.

---

[1] [3] www-verimag.imag.fr
https://www-verimag.imag.fr/TR/TR-2005-5.pdf

[2] [6] Microsoft PowerPoint - FSM.theory
https://www.site.uottawa.ca/~bob/csi5118/fsmtheory.pdf

[4] [12] FSM Sequences
https://cw.fel.cvut.cz/wiki/_media/courses/b2m32dsaa/lectures/p12.fsmsequences.pdf

[5] archiv.infsec.ethz.ch
https://archiv.infsec.ethz.ch/intranet_secured/r/1/chow-testingFSMs.pdf

[7] [8] [11] A Quick Survey of Active Automata Learning # | Active-Automata-Learning
https://wcventure.github.io/Active-Automata-Learning/

[9] [10] PII: 0890-5401(87)90052-6
https://people.eecs.berkeley.edu/~dawnsong/teaching/s10/papers/angluin87.pdf

13 State identification sequences from the splitting tree - White Rose Research Online
https://eprints.whiterose.ac.uk/id/eprint/158336/

14 [PDF] Learning DFA from Simple Examples*
https://faculty.ist.psu.edu/vhonavar/Papers/parekh-dfa.pdf

15 [1605.07805] Learning Moore Machines from Input-Output Traces
https://arxiv.org/abs/1605.07805

16 [PDF] A new Method for Incremental Testing of Finite State Machines
https://ntrs.nasa.gov/api/citations/20100018542/downloads/20100018542.pdf