Minimum Deterministic Finite Automaton
SAT Modeling and Solution

David Garcia Tejeda

May 2025

Abstract

We present a SAT-based method for solving the Minimum Consistent Finite Automaton problem: finding the smallest deterministic finite automaton (DFA) that accepts all positive and rejects all negative binary sequences in a given labeled sample. The encoding, variables, constraints, and solving strategy are detailed.

1 Problem Description

A deterministic finite automaton (DFA) is defined by a set of states, an initial state, transitions for each input bit (0 or 1), and a set of accepting states. Given two disjoint sets of binary sequences A (accepted) and R (rejected), the task is to find a minimal-state DFA consistent with both.

A DFA accepts a sequence if, after processing all bits starting from the initial state, it reaches an accepting state; otherwise, it rejects it.

2 SAT Modeling

The problem is encoded into propositional satisfiability (SAT) as follows.

2.1 Variables

Let n be the number of states in the candidate DFA.

We define the following boolean variables:

- $T_{i,b,j}$: True if the DFA transitions from state i to state j on input bit $b \in \{0,1\}$.
- A_i: True if state i is an accepting state.
- $S_{w,p,i}$: True if after reading the first p bits of word w, the DFA is in state i.

2.2 Constraints

1. **Deterministic Transitions:** For each state i and input bit b, exactly one $T_{i,b,j}$ is true for $j \in \{0,\ldots,n-1\}$:
$$\sum_{j=0}^{n-1} T_{i,b,j} = 1$$
2. **Initial State:** For each word w, the empty prefix $p = 0$ leads to the initial state 0:
$$S_{w,0,0} = \text{true}, \quad S_{w,0,i} = \text{false} \quad \forall i \neq 0$$

3. **Unique State per Prefix:** For each word w and prefix length p, the DFA is in exactly one state:
$$\sum_{i=0}^{n-1} S_{w,p,i} = 1$$
4. **Transition Consistency:** For each word w, prefix length p, bit $b = w[p]$, and states i, j:
$$S_{w,p,i} \wedge T_{i,b,j} \implies S_{w,p+1,j}$$
5. **Acceptance and Rejection:** For each accepted word w, at least one accepting state is active after reading all bits:
$$\bigvee_{i=0}^{n-1} (S_{w,|w|,i} \wedge A_i)$$
For each rejected word w, at least one non-accepting state is active after reading all bits:
$$\bigvee_{i=0}^{n-1} (S_{w,|w|,i} \wedge \neg A_i)$$

2.3 Encoding Details

All the above logical constraints are translated to Conjunctive Normal Form (CNF) using standard techniques, such as:

- Exactly-one constraints are encoded with pairwise mutual exclusion or more efficient encodings such as sequential counters.
- Implications are converted to CNF clauses.
- State variables $S_{w,i}$ are introduced for all prefixes of sequences in $A \cup R$.

3 Implementation and Solving

The solver is implemented in C++ as a single-binary project with Kissat as its only dependency. It reads the problem instance from `stdin`, dynamically encodes the constraints into CNF based on the input sample and current DFA size n, and calls the SAT solver.

Constraint encoding is performed incrementally: the system generates only the clauses required for the current n, leveraging the SAT solver's internal mechanisms (e.g., clause learning, propagation) to manage the problem's combinatorial complexity.

The search proceeds by incrementing n from 1 upward until a satisfying assignment is found, which guarantees minimality. The resulting model is decoded to reconstruct a valid DFA, printed in the specified format.

4 Remarks and Observations

- **Dynamic Constraint Encoding:** The model supports generating alternative versions of the `atMostOne` constraint encoding dynamically, adapting heuristics such as the number of involved variables. Constraints are produced on demand for each candidate size n, avoiding the overhead of monolithic encodings and enabling scalable, efficient exploration of the search space.

- **Solver-Aided Complexity Management:** By offloading the combinatorial reasoning to the SAT solver, the system benefits from efficient clause learning, propagation, and conflict analysis.
- **Minimal DFA Guarantee:** The incremental search over increasing n ensures the first solution found is minimal by construction.
- **Resource Efficiency:** Only necessary state and transition variables are introduced based on actual sequence prefixes, minimizing redundant logic.
- **Timeout Handling:** A solver-level timeout (e.g., 60 seconds) ensures practical runtime bounds without modifying core logic.
- **Extensibility:** The modular encoding strategy allows future integration of richer constraints (e.g., transition symmetry, forbidden patterns) without altering the solving backbone.
- **Future Work - Search Optimization:** A more efficient search can be implemented by first performing an exponential (doubling) search to find an upper bound where a solution exists, and then switching to a binary (dichotomous) search between the last unsuccessful and first successful bounds. This hybrid approach reduces the number of solver invocations to roughly $O(\log n)$, improving overall performance compared to a linear search. It can be tested that trying to find a suboptimal solution (a DFA much larger than the minimum) is not a very costly operation as expected, since although we are generating more variables the search space is more rich in possible solutions and their isomorphisms.

5 Conclusion

The SAT-based modeling provides a clear and flexible framework for solving the Minimum Consistent Finite Automaton problem. By leveraging state-of-the-art SAT solvers, this approach can find minimal DFAs consistent with given samples, supporting the combinatorial problem solving objectives.

Solver Comparison: Kissat, Gecode, and CPLEX (ILP)

The SAT-based version using **Kissat** proved to be the most performant among all approaches. This is likely due to the highly combinatorial nature of the problem, which makes linear programming approaches less suitable, and reduces the usefulness of constraint programming systems like **Gecode**.

Gecode high-level global constraints were not that useful to exploit the structure of the problem, and the problem was already so constrained that propagation was not very effective.

Linear programming using **CPLEX** also underperformed, likely due to the difficulty in expressing the problem efficiently and the unsuitability of LP relaxations in such discrete, combinatorial domains.

6 Final remarks

All code can be found at `https://github.com/DavidNexuss/cps_3` or in the attached zip.