A Simple Algorithmic Framework for Disambiguation of Finite Automata

Mauricio Cari
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Martín Muñoz
Univ. Artois, CNRS, UMR 8188, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data (IMFD), Chile

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Abstract

We study the task of disambiguation of finite state automata, namely, converting an automaton into an equivalent, unambiguous one. We do this by developing a novel and simple algorithmic framework that generalizes the subset construction for determinization, and that satisfies some desirable properties: (1) it preserves the original automaton if it was already unambiguous, (2) it computes the successor states on-the-fly and (3) computes each new state in polynomial time – this last point is crucial as it guarantees that the running time is polynomial in the size of the output automaton. Then, we show how to apply this framework for *partial* disambiguation: by changing the criterion that builds the new states, we develop algorithms for different levels of ambiguity, namely, finitely ambiguous, and polynomially ambiguous automata. These algorithms also satisfy condition (1) for their respective levels, and also (2) and (3). Finally, we show that the disambiguation framework can easily be extended to other models of automata like weighted automata.

2012 ACM Subject Classification Theory of computation → Automata extensions

Keywords and phrases Algorithmic automata theory, unambiguous automata models, degree of ambiguity, determinization, disambiguation, weighted automata

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.121

Related Version *Extended Version*: `https://arxiv.org/abs/2607.06894` [9]

Funding The work of Cari and Riveros was supported by ANID Fondecyt Regular project 1230935 and ANID – Millennium Science Initiative Program – Code ICN17_002. This work was partly supported by the ANR CERADOC project (ANR-25-CE23-3078).

1 Introduction

Automata theory is today one of the main areas of theoretical computer science, studying computational models with restricted resources and having applications in different areas of computer science. For these applications, finding automata models with good algorithmic properties is a crucial task, and *determinism* is probably the most well-known condition for reaching them. For example, deterministic finite automata are efficiently closed under several operations (i.e., with a polynomial-size output); the evaluation problem can be efficiently computed; equivalence and containment problems are tractable (and intractable in general) [42]; the number of states can be efficiently minimized [22]; and learning can be efficiently performed in an active setting [6], among other results [36].

In practice, systems employ *non-deterministic* automata models that are common in user applications. These are, in general, exponentially more succinct than their deterministic counterparts, but unfortunately do not possess such good algorithmic properties. To address this problem, systems typically rely on *determinizing* non-deterministic models; that is, converting a non-deterministic finite automaton into a deterministic one. The *subset construction*, introduced by Rabin and Scott [37], is the most used and best-known determinization procedure, in which non-determinism is simulated by maintaining subsets of states. Although the output of determinizing the automata could have up to exponential size [32], the subset construction has the advantage that it is simple and easy to implement, and can be computed on-the-fly (i.e., only the necessary part of the determinization is expanded). Indeed, this is probably the reason why the procedure is used so extensively (see, e.g., [21, 16, 1, 38, 8]). As an example, in regular expression (regex) evaluation, a regex engine (without backtracking) [16] usually evaluates a deterministic finite automaton on-the-fly: the running of the automaton is simulated by maintaining the current set of active states, caching the previously computed sets, and computing the next set for each new arriving character (if it was not already computed). This procedure allows for exploring only the sets of states reached by the data, and the next state can be easily computed from the previous state.

An alternative notion to determinism are *unambiguous* automata models, which are non-deterministic machines with at most one successful run per input. Unambiguous finite automata generalize deterministic finite automata, offering a good balance between efficiency and succinctness [13]. Specifically, they usually admit efficient algorithms; for example, the equivalence and containment problems are tractable [41], one can compute the number of accepted words (of a fixed length) in polynomial time [7], or efficiently enumerate them with constant delay [5]. Sometimes, these properties also extend to more general notions of ambiguity, like *finitely ambiguous* models (i.e., each input is accepted by at most a finite number of accepting runs) or *polynomially ambiguous* models (i.e., each input is accepted by at most a polynomial number of accepting runs in the size of the input) for which researchers have also found interesting results [45, 19, 13].

The disambiguation of finite automata has been stated as a relevant algorithmic problem in automata theory [14, 12]. Unlike the determinization procedure for finite automata, there is no standard or well-known procedure that converts any given non-deterministic automata into an equivalent, unambiguous one (besides the determinization itself). Like determinization, one would also desire a disambiguation procedure that is simple and easy to implement, and can be efficiently computed on-the-fly (i.e., computing the next set of states in polynomial time). In the past, researchers have proposed procedures for disambiguating finite automata and other models [39, 34, 35, 46, 20]; however, these procedures are often cumbersome, cannot be computed on-the-fly, and may even encounter technical difficulties (see Appendix A). Indeed, today there is no well-known disambiguation procedure that is used in practice. Furthermore, none of these procedures can be extended to other levels of ambiguity (e.g., finitely ambiguous) or other automata models (e.g., weighted automata). An important exception to this rule is the *unambiguous subset construction* of Weber and Klemm [44] which, as far as we know, has been largely overlooked by the community.

In this work, we generalize the approach of Weber and Klemm and propose an algorithmic framework for disambiguation of NFA that is arguably simple and can be computed on-the-fly (Section 3). This framework allows for the presentation of efficient disambiguation algorithms for unambiguous, finitely ambiguous, and polynomially ambiguous automata. Furthermore, we show that these constructions preserve the original automata if it already had the target ambiguity, and are optimal in some precise sense (Section 4). Finally, we show that this framework can be extended to other automata models by presenting general disambiguation algorithms for the model of weighted automata (Section 5).

Related work. Unambiguous, finitely ambiguous, and polynomially ambiguous finite state automata have been extensively studied in the literature for various automata models (see, e.g., [13, 45, 36]). Our work complements this literature, where we seek useful algorithms for constructing such models.

Algorithms for disambiguating finite state automata, transducers or weighted automata have been proposed in the literature [39, 34, 35, 46, 20]. In general, all of them rely on maintaining the active state and a subset of competing states (i.e., runs) plus some conditions to prune undesirable runs. Some of them even require some additional intermediate steps. For example, [39] needs to compute first the determinization of the automaton, and [35, 46] require a pre-disambiguation step. As discussed above, these conditions can be rather obscure and their correctness is hard to prove. Indeed, in Appendix A we show that the construction presented in [34] does not work in general. As we mentioned, our approach is inspired by the construction of Weber and Klemm [44]. To the best of our knowledge, previous approaches have not studied the disambiguation of finite automata as a general framework, and their construction does not extend to other levels of ambiguity.

In the context of weighted automata, ambiguity can define a strict hierarchy of classes of functions, and researchers have studied the problem of deciding whether a given weighted automaton (e.g., over the tropical semiring) is equivalent to a deterministic or unambiguous one [27, 26, 25, 2, 3]. These articles studied only the decision problem of determinization, and the constructions are usually non-trivial. In contrast, the present work aims for a simple and modular algorithm that could lead to useful solutions in practice.

Outline of the paper. We start with some preliminaries in Section 2. We present the disambiguation framework in Section 3 and use it for finding disambiguation schemes for different levels of ambiguity in Section 4. We extend this framework to the disambiguation of weighted automata in Section 5. Finally, we discuss some future work in Section 6. Due to space constraints, the proofs of the results can be found in the extended version [9].

2 Preliminaries

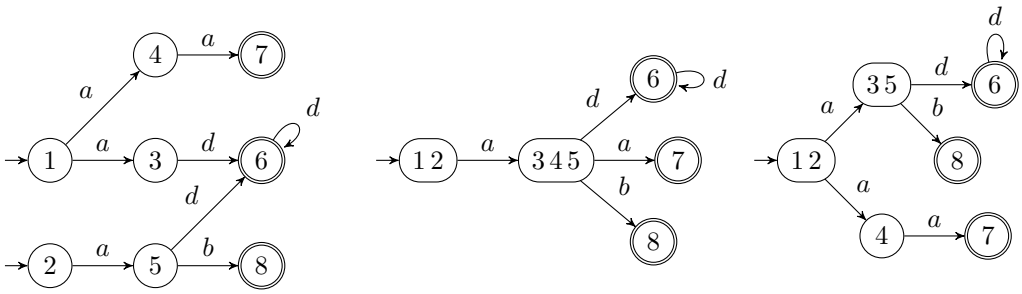
Non-deterministic finite automata. A *non-deterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $I \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of final states. A *partial run* of $\mathcal{A}$ over a word $w = a_1 \dots a_n$ is a sequence:

$$\rho \;:=\; p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n \qquad (*)$$

where $(p_{i-1}, a_i, p_i) \in \Delta$ for every $i \in [n]$. We say that ρ is a *run* of $\mathcal{A}$ over w if additionally $p_0 \in I$. Further, we say that a (partial) run ρ is *accepting* if $p_n \in F$. The language of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is the set of all $w \in \Sigma^*$ such that there exists an accepting run of $\mathcal{A}$ over w.

One can see the transition relation Δ as a function $\Delta : Q \times \Sigma \to 2^Q$ where $\Delta(p, a) = \{q \mid (p, a, q) \in \Delta\}$. We extend Δ from letters to words (i.e., $\Delta : Q \times \Sigma^* \to 2^Q$) as usual: $\Delta(p, \varepsilon) = \{p\}$ and $\Delta(p, a \cdot w) = \bigcup_{q \in \Delta(p,a)} \Delta(q, w)$. Furthermore, we can extend Δ from single states to sets of states $\Delta\colon 2^Q \times \Sigma^* \to 2^Q$ defined as $\Delta(S, w) = \bigcup_{p \in S} \Delta(p, w)$. One can note that $w \in \mathcal{L}(\mathcal{A})$ iff $\Delta(I, w) \cap F \neq \emptyset$. For the sake of presentation, we will use $\Delta(q, a)$, $\Delta(q, w)$, or $\Delta(S, w)$ with the same Δ when the input is clear from the context.

We define the size of $\mathcal{A}$ as $|\mathcal{A}| = |Q| + |\Delta|$. Further, we say that two NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *isomorphic* if $\mathcal{A}_1$ and $\mathcal{A}_2$ are identical up to renaming the states.



(a) finFA $\mathcal{A}$ (b) DFA $\mathcal{A}_{\text{det}}$ (c) UFA $\mathcal{A}_{\mathcal{O}_{\mathsf{CF}}}$

Figure 1 (a) An example of an NFA $\mathcal{A}$ with $\mathsf{da}(\mathcal{A}) = 2$. (b) The determinization $\mathcal{A}_{\text{det}}$ of $\mathcal{A}$. (c) The disambiguation $\mathcal{A}_{\mathcal{O}_{\mathsf{CF}}}$ of $\mathcal{A}$.

Trimming. We say that $q \in Q$ is *reachable from* $p \in Q$ if there exists a partial run ρ like $(*)$ of $\mathcal{A}$ over some word w such that $p_0 = p$ and $p_n = q$ (i.e., $q \in \Delta(p, w)$ for some w). A state $q \in Q$ is *reachable* if q can be reached from a state in I (i.e., $q \in \Delta(I, w)$ for some w) and is *co-reachable* if some state in F is reachable from q (i.e., $\Delta(q, w) \cap F \neq \emptyset$ for some w). In this work, we assume all NFAs are *trimmed*, namely, that every $q \in Q$ is (co-)reachable.

Ambiguity. We say that a finite automaton $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ is *deterministic* (DFA) iff $|I| = 1$ and for every $p \in Q$ and $a \in \Sigma$ there exists *exactly one* state $q \in Q$ such that $(p, a, q) \in \Delta$. In other words, Δ forms a function $\Delta : Q \times \Sigma \to Q$. We say that $\mathcal{A}$ is *unambiguous* (UFA) iff for every $w \in \Sigma^*$ there exists at most one accepting run of $\mathcal{A}$ over w. One can check that if $\mathcal{A}$ is deterministic then $\mathcal{A}$ is unambiguous, whereas the converse does not necessarily hold. In Figure 1 (b) and (c), we display a DFA and UFA, respectively.

Given an NFA $\mathcal{A}$, the *degree of ambiguity* $\mathsf{da}_{\mathcal{A}}(w)$ of a word w is the number of different accepting runs of $\mathcal{A}$ over w [45]. Similarly, the *degree of ambiguity* $\mathsf{da}(\mathcal{A})$ of $\mathcal{A}$ is the maximum degree of ambiguity over all $w \in \mathcal{L}(\mathcal{A})$, that is, $\mathsf{da}(\mathcal{A}) = \max_{w \in \mathcal{L}(\mathcal{A})} \mathsf{da}_{\mathcal{A}}(w)$. If no such maximum exists, then $\mathsf{da}(\mathcal{A}) = \infty$. We say that an NFA $\mathcal{A}$ is *finitely ambiguous* (finFA) if $\mathsf{da}(\mathcal{A}) \leq k$ for some $k \in \mathbb{N}$. In this case, we say that $\mathcal{A}$ is k-*ambiguous* (k-ambFA). Instead, $\mathcal{A}$ is called *infinitely ambiguous* if $\mathsf{da}(\mathcal{A}) = \infty$. Note that $\mathcal{A}$ is unambiguous iff $\mathsf{da}(\mathcal{A}) = 1$ (i.e., 1-ambiguous). Further, the level of ambiguity forms a hierarchy, namely, every ℓ-ambFA is also $\ell + 1$-ambFA, and so on. In Figure 1 (a), we show a finFA $\mathcal{A}$ with $\mathsf{da}(\mathcal{A}) = 2$.

For infinitely ambiguous automata, one can characterize how the ambiguity increases with the size of the input word [45]. The *degree of growth of ambiguity*, denoted as $\mathsf{deg}(\mathcal{A})$, is defined as the smallest degree of a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for all $w \in \Sigma^*$, we have $\mathsf{da}_{\mathcal{A}}(w) \leq p(|w|)$, if such a polynomial exists. We say that $\mathcal{A}$ is *polynomially ambiguous* (polyFA) if $\mathsf{deg}(\mathcal{A}) = k$ for some $k \in \mathbb{N}$. Otherwise, if no such polynomial exists, we define $\mathsf{deg}(\mathcal{A}) = \infty$, and say that $\mathcal{A}$ is *exponentially ambiguous* (expFA). In Figure 4 (c) and (a), we show examples of a polyFA and expFA, respectively.

The work by Weber and Seidl [45] defines criteria that characterize the degree and the degree of growth of ambiguity of an NFA. By these criteria, deciding whether an NFA $\mathcal{A}$ is deterministic, unambiguous, finitely, or polynomially ambiguous can be done in polynomial time over the size of $\mathcal{A}$. Conversely, computing the exact degree of ambiguity $\mathsf{da}(\mathcal{A})$ is PSPACE-complete [10] and the degree of growth of ambiguity can be computed in polynomial time [45].

We denote by DFA, UFA, finFA, and polyFA the class of all finite automata that are deterministic, unambiguous, finitely ambiguous, and polynomially ambiguous, respectively. One can note that these classes form a strict hierarchy where:

$$\text{DFA} \subsetneq \text{UFA} \subsetneq \text{finFA} \subsetneq \text{polyFA} \subsetneq \text{NFA}$$

Furthermore, for each pair of classes $\mathcal{C}_1$ and $\mathcal{C}_2$ with $\mathcal{C}_1 \subsetneq \mathcal{C}_2$ where $\mathcal{C}_1, \mathcal{C}_2 \in \{\text{DFA}, \text{UFA}, \text{finFA}\}$ there exists a family $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ where each $\mathcal{A}_n$ has $O(n)$ number of states and $\mathcal{A}_n \in \mathcal{C}_2$ such that for every $\mathcal{A} \in \mathcal{C}_1$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_n)$, it holds that $|\mathcal{A}| \in \Omega(2^n)$ [29, 31]. Similarly, there exists a family $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ in polyFA where each $\mathcal{A}_n$ has $O(p(n))$ states for some polynomial p, such that for every equivalent automaton in finFA it holds that $|\mathcal{A}| \in \Omega(2^{n^{1/3}})$ [23]; also, NFA can be super-polynomially more succinct than polyFA (see [24, 30]).

3 The automata disambiguation framework

The goal of this section is to define a framework suited for disambiguation of finite state automata, namely, algorithms that convert any NFA into a UFA, finFA, or polyFA. We start by introducing the motivation of the framework and then provide the formal definitions.

Determinization. Let us highlight the classical *subset construction* for determinization, first described by Rabin and Scott [37], that receives $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ and produces a DFA:

$$\mathcal{A}_{\mathsf{det}} := (2^Q, \Sigma, \Delta_{\mathsf{det}}, \{I\}, \{S \mid S \cap F \neq \emptyset\}),$$

its states are the subsets of Q, its single initial state is I, and its final states are the sets that contain at least one state from F. Further, $(S_1, a, S_2) \in \Delta_{\mathsf{det}}$ iff $S_2 = \Delta(S_1, a)$ for every $S_1, S_2 \in 2^Q$ and $a \in \Sigma$. The NFA $\mathcal{A}_{\mathsf{det}}$ is deterministic and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_{\mathsf{det}})$.

It is important to note that, since we assume that all our NFA are trimmed, we assume that $\mathcal{A}_{\mathsf{det}}$ is trimmed after the construction (i.e., we only keep the subsets in 2^Q that reach F and are reachable from I). One can prove that, given that $\mathcal{A}$ is initially trimmed, all the states reachable from I are reachable and co-reachable in $\mathcal{A}_{\text{det}}$. In Figure 1 (b), we display the determinization $\mathcal{A}_{\text{det}}$ of $\mathcal{A}$ where we only have the states reachable from $\{1, 2\}$.

Although the automaton $\mathcal{A}_{\mathsf{det}}$ could be of size exponential with respect to $|Q|$, we claim that it has three algorithmic properties that makes it suitable to be used in practice.

1. (**idempotent**) If $\mathcal{A}$ is deterministic, then $\mathcal{A}_{\mathsf{det}}$ is *isomorphic* to $\mathcal{A}$.
2. (**on-the-fly**) Δ_{det} can be computed *on-the-fly*, namely, for any $S \in Q_{\mathsf{det}}$ one can compute $\Delta_{\mathsf{det}}(S, a)$ only from Δ and S.
3. (**efficient**) Given an $S \in Q_{\mathsf{det}}$ and $a \in \Sigma$, one can compute $\Delta_{\mathsf{det}}(S, a)$ in *polynomial time* in $|S|$ and $|\Delta|$.

The first property ensures that the procedure preserves the automata if it already possesses the desired properties. The second and third are used in practice (see, e.g., [21, 16, 1, 38, 8]) since one can keep a cache of explored sets and efficiently compute the next one from a given set.

We want to have the same properties for a disambiguation procedure that takes any NFA $\mathcal{A}$ and generates an equivalent NFA with the desired ambiguity. It is important to remark that all disambiguation procedures proposed in the literature [39, 34, 35, 46, 20] fail to achieve at least one of them (and most fail all of them). For instance, Mohri's procedure [34] and Schützenberger's construction [39] cannot be computed on-the-fly, since they require knowledge about previously computed states to discard transitions and disallow finality of new states. Moreover, all of them work for converting NFA to UFA, and none of them translates into finFA or polyFA.

The framework. Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be an NFA. We define a *partition oracle* $\mathcal{O}$ for $\mathcal{A}$ as a function $\mathcal{O} : 2^Q \mapsto 2^{2^Q}$ that receives an $S \subseteq Q$, and outputs a partition of S, namely, $\mathcal{O}(S) = \{S_1, \ldots, S_m\}$ where $S = S_1 \uplus \cdots \uplus S_m$. Given a partition oracle $\mathcal{O}$ for $\mathcal{A}$, we define the NFA:

$$\mathcal{A}_{\mathcal{O}} := (2^Q, \Sigma, \Delta_{\mathcal{O}}, \mathcal{O}(I), \{S \mid S \cap F \neq \emptyset\}).$$

This is similar to Rabin-Scott's $\mathcal{A}_{\text{det}}$, except the transition relation and the set of initial states depend on $\mathcal{O}$. Specifically, we define the transition relation $\Delta_{\mathcal{O}}$ as:

$$\Delta_{\mathcal{O}} \;:=\; \big\{\, (S_1, a, S_2) \;\mid\; S_2 \in \mathcal{O}(\Delta(S_1, a)) \,\big\},$$

and more succinctly, $\Delta_{\mathcal{O}}(S, a) = \mathcal{O}(\Delta(S, a))$ for $S \subseteq Q$ and $a \in \Sigma$. In other words, finding the transitions $(S_1, a, S_2) \in \Delta_{\mathcal{O}}$ for S_1 and a can be done by first computing $\Delta(S_1, a)$ and then applying the partition oracle $\mathcal{O}$ over it. We recall again that, similar to $\mathcal{A}_{\text{det}}$, we assume that $\mathcal{A}_{\mathcal{O}}$ is trimmed after the construction by keeping only the states reachable from $\mathcal{O}(I)$.

A crucial property of $\mathcal{A}_{\mathcal{O}}$ is that it preserves equivalence for every partition oracle $\mathcal{O}$.

▶ **Proposition 1.** *For every partition oracle $\mathcal{O}$ of $\mathcal{A}$, it holds that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_{\mathcal{O}})$.*

Given this construction, one could intuit that there exists a disambiguation *spectrum* from oracles that partition their sets more finely or more coarsely. Indeed, the trivial partition oracles $\mathcal{O}_{\mathsf{single}}$ and $\mathcal{O}_{\mathsf{full}}$, where $\mathcal{O}_{\mathsf{single}}(S) := \{\{q\} \mid q \in S\}$ and $\mathcal{O}_{\mathsf{full}}(S) := \{S\}$, define two extremes: $\mathcal{A}_{\mathcal{O}_{\mathsf{single}}}$ is isomorphic to $\mathcal{A}$ and $\mathcal{A}_{\mathcal{O}_{\mathsf{full}}}$ is isomorphic to $\mathcal{A}_{\mathsf{det}}$.

Given an NFA $\mathcal{A}$, our main goal is to find a partition oracle $\mathcal{O}$ for which $\mathcal{A}_{\mathcal{O}}$ has a desired property (e.g., being unambiguous) and we want to compute $\mathcal{O}(S)$ from $\mathcal{A}$ and S efficiently. For this purpose, we define a *disambiguation scheme* as any function $\mathcal{D}$ that, given an NFA $\mathcal{A}$, defines a partition oracle $\mathcal{D}(\mathcal{A})$ for $\mathcal{A}$. Furthermore, we say that a disambiguation scheme $\mathcal{D}$ is *efficient* if there is an algorithm that receives as input any NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ and $S \subseteq Q$ and outputs $[\mathcal{D}(\mathcal{A})](S)$ in polynomial time in $|\mathcal{A}|$ and $|S|$. As an example, there exists an efficient disambiguation scheme $\mathcal{D}_{\mathsf{det}}$ given by defining $\mathcal{D}_{\mathsf{det}}(\mathcal{A}) = \mathcal{O}_{\mathsf{full}}$ for every NFA.

Our disambiguation framework involves identifying efficient disambiguation schemes for various levels of ambiguity. Given a class $\mathcal{C}$ of NFA (e.g., $\mathcal{C} = \text{UFA}$), we say that a disambiguation scheme $\mathcal{D}$ is a *$\mathcal{C}$-disambiguation scheme* iff $\mathcal{A}_{\mathcal{O}} \in \mathcal{C}$ and $\mathcal{A}_{\mathcal{O}}$ is isomorphic to $\mathcal{A}$ whenever $\mathcal{A} \in \mathcal{C}$, for every NFA $\mathcal{A}$ and $\mathcal{O} = \mathcal{D}(\mathcal{A})$. Continuing the example, $\mathcal{D}_{\mathsf{det}}$ is a DFA-disambiguation scheme. Notice that if we find an efficient $\mathcal{C}$-disambiguation scheme $\mathcal{D}$ then $\mathcal{D}$ will satisfy our desirable algorithmic properties 1. to 3. for the class $\mathcal{C}$ (i.e., similarly to the Rabin-Scott subset construction), namely, (1) if $\mathcal{A} \in \mathcal{C}$, then $\mathcal{A}_{\mathcal{O}}$ is isomorphic to $\mathcal{A}$, (2) $\Delta_{\mathcal{O}}$ can be computed on-the-fly, and (3) $\Delta_{\mathcal{O}}$ can be computed efficiently.

Discussion. The proposed framework for disambiguating NFA is *a* generic strategy for this task, but, of course, it is not the only one. One could also disambiguate NFA through other strategies (e.g. [39, 34]) with their own desirable properties. The advantages of our approach are its simplicity and generality; and, since each new state is created efficiently and on-the-fly, it also offers runtime guarantees that depend on the size of the resulting automaton. Having a generic framework for disambiguation is also quite versatile: we can now define a wide array of algorithms to reach different levels of ambiguity, and compare their properties.

4 Disambiguation schemes for different levels of ambiguity

We present disambiguation schemes for constructing UFA, finFA, and polyFA, and study their properties. We start by presenting our general strategy for finding partition oracles, which is based on binary relations over states. Then, we instantiate this strategy for each class.

Relational partition oracles. First, we introducing some useful notation. Let $R \subseteq A \times A$ be a symmetric relation (i.e., $(p, q) \in R$ implies $(q, p) \in R$) over some non-empty set A. We define by $\mathsf{dom}(R) = \{p \mid (p, q) \in R\}$ the active domain of R (note that $\mathsf{dom}(R)$ might be a strict subset of A). We denote by $\kappa(R)$ the set of connected components of the undirected graph (A, R) where $\kappa(R)$ always forms a partition of $\mathsf{dom}(R)$.

Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be an NFA. Given a symmetric and reflexive (i.e., $(p,p) \in R$ for every $p \in Q$) relation R over Q, we define the oracle $\mathcal{O}_R : 2^Q \mapsto 2^{2^Q}$ by $\mathcal{O}_R(S) = \kappa(R \cap (S \times S))$. Since R is reflexive, $\mathsf{dom}(R \cap (S \times S)) = S$ and then $\kappa(R \cap S \times S)$ forms a partition of S. Therefore, $\mathcal{O}_R$ is well defined as a partition oracle of $\mathcal{A}$. We say that $\mathcal{O}$ is a *relational partition oracle* of $\mathcal{A}$ if there exists a relation R over Q with $\mathcal{O} = \mathcal{O}_R$. For example, for $R_1 = \{(p,p) \mid p \in Q\}$ and $R_2 = Q \times Q$, one can check that $\mathcal{O}_{\mathsf{single}} = \mathcal{O}_{R_1}$ and $\mathcal{O}_{\mathsf{full}} = \mathcal{O}_{R_2}$. Therefore, the partition oracles $\mathcal{O}_{\mathsf{single}}$ and $\mathcal{O}_{\mathsf{full}}$ are relational as well.

We say that $\mathcal{D}$ is a *relational disambiguation scheme* if $\mathcal{D}(\mathcal{A})$ is relational for every $\mathcal{A}$. Note that if for every $\mathcal{A}$ we can compute a relation $R_{\mathcal{A}}$ in polynomial time such that $\mathcal{D}(\mathcal{A}) = \mathcal{O}_{R_{\mathcal{A}}}$, then $\mathcal{D}$ is efficient. Thus, our strategy in the sequel is to provide relations $R_{\mathcal{A}}$ that can be computed in polynomial time from $\mathcal{A}$, leading to the desired disambiguation schemes.

UFA disambiguation. Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be an NFA. We say that two states $p, q \in Q$ *share a common future* in $\mathcal{A}$ iff there exists $v \in \Sigma^*$ such that $\Delta(p, v) \cap F \neq \emptyset \neq \Delta(q, v) \cap F$. In other words, there exist two accepting partial runs ρ_p and ρ_q of $\mathcal{A}$ over the same word v starting from p and q, respectively. We define the relation $\mathsf{CF}_{\mathcal{A}} \subseteq Q \times Q$ where $(p,q) \in \mathsf{CF}_{\mathcal{A}}$ iff p and q share a common future. By definition, $\mathsf{CF}_{\mathcal{A}}$ is symmetric. Further, since $\mathcal{A}$ is trimmed, $\mathsf{CF}_{\mathcal{A}}$ is reflexive. Then, we denote by $\mathcal{O}_{\mathsf{CF}}$ the relational partition oracle of $\mathcal{A}$ defined by $\mathsf{CF}_{\mathcal{A}}$.

Intuitively, if p and q share $v \in \Sigma^*$ as a common future, then, whenever a disambiguation scheme reaches p and q from the initial states (i.e., $p, q \in \Delta(I, u)$ for some u), p and q must be part of the same state in the resulting automata so that it stays unambiguous (otherwise, $u \cdot v$ will have two accepting runs). For this reason, the partition oracle $\mathcal{O}_{\mathsf{CF}}$ will keep p and q together along with all states in their corresponding connected component. Indeed, $\mathcal{O}_{\mathsf{CF}}$ is a relational partition oracle for producing UFA and can be computed efficiently.

▶ **Theorem 2.** *For every NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, the following properties hold: (1) $\mathcal{A}_{\mathcal{O}_{\mathsf{CF}}}$ is always unambiguous; (2) if $\mathcal{A}$ is unambiguous, then $\mathcal{A}_{\mathcal{O}_{\mathsf{CF}}}$ and $\mathcal{A}$ are isomorphic; and (3) the relation $\mathsf{CF}_{\mathcal{A}}$ can be computed in time $O(|\mathcal{A}|^2)$. That is, the disambiguation scheme $\mathcal{D}_{\mathsf{CF}}$ that assigns the relational partition oracle $\mathcal{O}_{\mathsf{CF}}$ is an efficient UFA-disambiguation scheme.*

The time to compute $\mathsf{CF}_{\mathcal{A}}$ matches the classical algorithm for testing if an NFA is unambiguous, and the exponent cannot be lowered, under fine-grained assumptions [17].

Coming back to our example NFA $\mathcal{A}$ in Figure 1(a), one can check that its common future relation is $\mathsf{CF}_{\mathcal{A}} = \{(1,2), (2,1), (3,5), (5,3), (4,4), \ldots\}$. We show its disambiguation $\mathcal{A}_{\mathcal{O}_{\mathsf{CF}}}$ by the oracle $\mathcal{O}_{\mathsf{CF}}$ in Figure 1 (c).

While $\mathcal{D}_{\mathsf{CF}}$ fulfils our desirable algorithmic properties, one can easily devise "efficient UFA-disambiguation schemes" by an ad-hoc partition oracle. For instance, one may use $\mathcal{O}_{\mathsf{single}}$ when $\mathcal{A}$ is unambiguous, and $\mathcal{O}_{\mathsf{full}}$, otherwise (in other words, if $\mathcal{A}$ is not unambiguous, determinize it). So, what makes $\mathcal{D}_{\mathsf{CF}}$ so special as a disambiguation procedure? We argue that $\mathcal{D}_{\mathsf{CF}}$ is minimal in the following sense. Let π_1, π_2 be two partitions of the same set A. We say that π_1 *refines* π_2 if for every $P_1 \in \pi_1$ there is a $P_2 \in \pi_2$ such that $P_1 \subseteq P_2$. That is, π_1 breaks A into smaller pieces than π_2. Let us define $\mathsf{Reach}(\mathcal{A}) = \bigcup_{w \in \Sigma^*} \Delta(I, w)$ as the set of all states reachable from I. For two partition oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ of $\mathcal{A}$, we say that $\mathcal{O}_1$ *refines* $\mathcal{O}_2$ over $\mathcal{A}$ if $\mathcal{O}_1(I)$ refines $\mathcal{O}_2(I)$ and, for every $S \in \{\Delta(S', a) \mid S' \in \mathsf{Reach}(\mathcal{A}_{\mathcal{O}_2}) \wedge a \in \Sigma\}$, $\mathcal{O}_1(S)$ refines $\mathcal{O}_2(S)$.

▶ **Theorem 3.** *Let $\mathcal{O}$ be any partition oracle of an NFA $\mathcal{A}$. If $\mathcal{A}_{\mathcal{O}}$ is unambiguous, then $\mathcal{O}_{\mathsf{CF}}$ refines $\mathcal{O}$ over $\mathcal{A}$.*

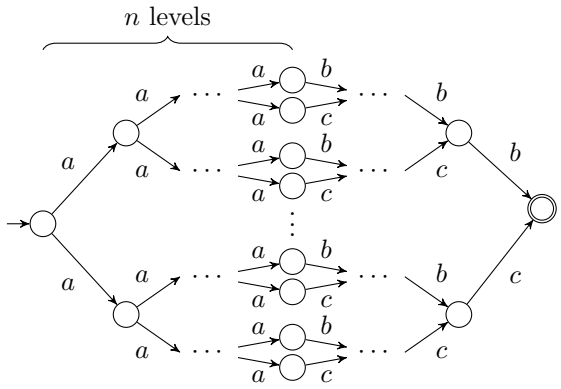


Figure 2 Example of an UFA with $\Omega(2^n)$ states whose determinization is of size $O(n)$.

In other words, the disambiguation scheme $\mathcal{D}_{\mathsf{CF}}$ provides the *finest* possible partitions on state sets $S \subseteq Q$ while $\mathcal{A}_{\mathcal{O}}$ is still unambiguous. We argue that fineness is a reasonable criterion for minimality. Indeed, finer partitions, in a sense, better preserve the original automaton, and the finest partition is $\mathcal{O}_{\mathsf{single}}$ which leaves the automaton unchanged.

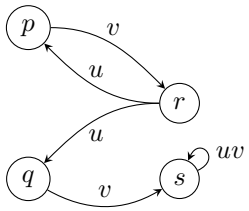
Discussion. The reader may have noted that, although the partition oracle $\mathcal{O}_{\mathsf{CF}}$ finds the finest possible partition, the determinization $\mathcal{A}_{\mathrm{det}}$ of $\mathcal{A}$ in Figure 1 (b) has less states compared to the disambiguation $\mathcal{A}_{\mathcal{O}_{\mathsf{CF}}}$ in Figure 1 (c) (i.e., one state less). Therefore, it is essential to discuss that, regarding minimizing the size of the resulting automaton, the disambiguation schemes $\mathcal{D}_{\mathsf{CF}}$ and $\mathcal{D}_{\mathsf{det}}$ (i.e., the determinization procedure) are incomparable. On the one hand, we know that there exists a family $\{\mathcal{A}^n\}_{n\in\mathbb{N}}$ of UFA such that every DFA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^n)$ has $|\mathcal{A}| \in \Omega(2^n)$ [13]. This implies that the determinization $\mathcal{A}^n_{\mathcal{O}_{\mathsf{full}}}$ by $\mathcal{D}_{\mathsf{det}}$ can be of exponential size with respect to $\mathcal{A}^n_{\mathcal{O}_{\mathsf{CF}}}$. On the other hand, in Figure 2 we show an example of a family of UFA with $\Omega(2^n)$ states whose Rabin-Scott determinization is of size $O(n)$. Finding an optimal UFA-disambiguation scheme that minimizes the number of explored states is an interesting and relevant open problem that we leave for future work.

finFA and polyFA disambiguation. Similar to UFA, we provide relations to define partition oracles for the finFA and polyFA disambiguation. Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be an NFA.

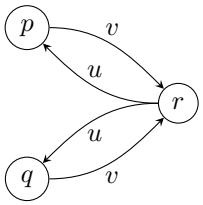
- *Infinite common future (ICF)*: We say that a pair of states (p, q) share an *infinite common future* in $\mathcal{A}$ iff there exists $r, s \in Q$ and $u, v \in \Sigma^*$ such that $p, q \in \Delta(r, u)$, $r \in \Delta(p, v)$, $s \in \Delta(q, v)$, and $s \in \Delta(s, uv)$. We define the relation $\mathsf{ICF}_{\mathcal{A}} \subseteq Q \times Q$ where $(p, q) \in \mathsf{ICF}_{\mathcal{A}}$ if (p, q) or (q, p) share an infinite common future.
- *Exponentially infinite common future (ECF)*: We say that two states $p, q \in Q$ share an *exponentially infinite common future* in $\mathcal{A}$ iff there exists $r \in Q$ and $u, v \in \Sigma^*$ such that $r \in \Delta(p, v)$, $r \in \Delta(q, v)$, and $p, q \in \Delta(r, u)$. Similarly, we define the relation $\mathsf{ECF}_{\mathcal{A}} \subseteq Q \times Q$ where $(p, q) \in \mathsf{ECF}_{\mathcal{A}}$ if p, q share a exponentially infinite common future.

We illustrate the infinite and exponentially infinite common future conditions in Figure 3 (a) and (b), respectively. Note that ICF and ECF are stronger conditions than regular common future and, indeed, ECF implies ICF. Further, $\mathsf{ICF}_{\mathcal{A}}$ and $\mathsf{ECF}_{\mathcal{A}}$ are reflexive and symmetric. Thus, they induce relational partition oracles that we denote by $\mathcal{O}_{\mathsf{ICF}}$ and $\mathcal{O}_{\mathsf{ECF}}$, respectively.

Similar to the common future condition, ICF and ECF identify pairs of states in $\mathcal{A}$ that can lead to infinitely and exponentially ambiguous behaviour, respectively. Indeed, they lead to efficient finFA- and polyFA-disambiguation schemes as expected.

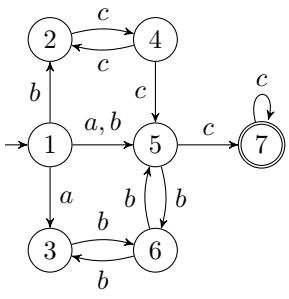


(a) Infinite common future

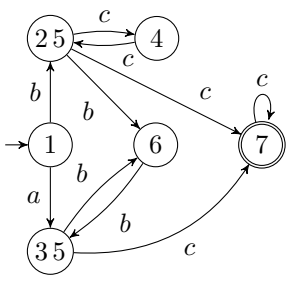


(a) Exponentially infinite common future

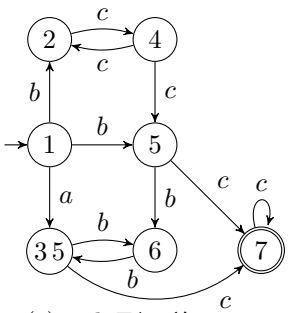
Figure 3 Graphical representation of infinite and exponentially infinite common future conditions.



(a) expFA $\mathcal{A}'$



(b) finFA $\mathcal{A}'_{\mathcal{O}_{\mathsf{ICF}}}$



(c) polyFA $\mathcal{A}'_{\mathcal{O}_{\mathsf{ECF}}}$

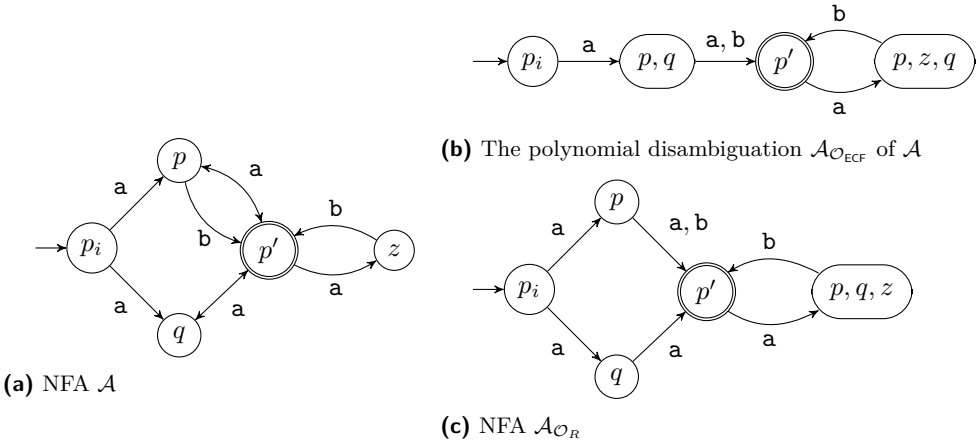
Figure 4 (a) An example of an exponentially ambiguous NFA $\mathcal{A}'$. (b) The finFA disambiguation $\mathcal{A}'_{\mathcal{O}_{\mathsf{ICF}}}$ of $\mathcal{A}'$. (c) The polyFA disambiguation $\mathcal{A}'_{\mathcal{O}_{\mathsf{ECF}}}$ of $\mathcal{A}'$.

▶ **Theorem 4.** *For every NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, the following properties hold: (1) $\mathcal{A}_{\mathcal{O}_{\mathsf{ICF}}}$ and $\mathcal{A}_{\mathcal{O}_{\mathsf{ECF}}}$ are always finitely ambiguous and polynomially ambiguous, respectively; (2) if $\mathcal{A}$ is finitely ambiguous, then $\mathcal{A}_{\mathcal{O}_{\mathsf{ICF}}}$ is isomorphic to $\mathcal{A}$; if $\mathcal{A}$ is polynomially ambiguous then $\mathcal{A}_{\mathcal{O}_{\mathsf{ECF}}}$ isomorphic to $\mathcal{A}$; and (3) the relations $\mathsf{ICF}_{\mathcal{A}}$ and $\mathsf{ECF}_{\mathcal{A}}$ can be computed in time $O(|\mathcal{A}|^5)$ and $O(|\mathcal{A}|^3)$, respectively. In other words, the disambiguation schemes $\mathcal{D}_{\mathsf{ICF}}$ and $\mathcal{D}_{\mathsf{ECF}}$ that assign the relational partition oracles $\mathcal{O}_{\mathsf{ICF}}$ and $\mathcal{O}_{\mathsf{ECF}}$ for each NFA are efficient finFA- and polyFA-disambiguation schemes, respectively.*

An example of finFA and polyFA disambiguation of an expFA $\mathcal{A}'$ is shown in Figure 4. The infinite and exponentially infinite common future relations on the NFA $\mathcal{A}'$ are $\mathsf{ICF}_{\mathcal{A}'} = \{(2,5),(5,2),(3,5),(5,3),(1,1),\ldots\}$ and $\mathsf{ECF}_{\mathcal{A}'} = \{(3,5),(5,3),(1,1),\ldots\}$, respectively. Note that $\mathcal{A}'_{\mathcal{O}_{\mathsf{ICF}}}$ in Figure 4 (b) and $\mathcal{A}'_{\mathcal{O}_{\mathsf{ECF}}}$ in Figure 4 (c) are exactly finFA and polyFA, respectively, achieving partial disambiguation depending on the chosen relation.

Similar to the UFA case, one would like to show that $\mathcal{O}_{\mathsf{ICF}}$ and $\mathcal{O}_{\mathsf{ECF}}$ are minimal in the sense that they refine any other partition oracle that produces finitely ambiguous or polynomially ambiguous NFA, respectively. Unfortunately, this is not the case for both, since one can find partition oracles that produce the required level of ambiguity, but they are not refined by $\mathcal{O}_{\mathsf{ICF}}$ or $\mathcal{O}_{\mathsf{ECF}}$.

▶ **Example 5.** Consider the NFA $\mathcal{A}$ and its polynomial disambiguation $\mathcal{A}_{\mathcal{O}_{\mathsf{ECF}}}$ in Figure 5. Its exponentially infinite common future relation is $\mathsf{ECF}_{\mathcal{A}} = \{(p,q),(q,p),(p,z),(z,p)\}$. There exist a relation $R = \{(p,z),(q,z),(z,p),(z,q)\}$ such that $\mathcal{O}_R(\{p,q\})$ refines $\mathcal{O}_{\mathsf{ECF}}(\{p,q\})$, while $\mathcal{A}_{\mathcal{O}_R}$ is polynomially ambiguous. In $\mathcal{A}_{\mathcal{O}_R}$, states p and q are only merged in the set of states that contains z, effectively preventing exponential degree of ambiguity.



(a) NFA $\mathcal{A}$

(b) The polynomial disambiguation $\mathcal{A}_{\mathcal{O}_{\mathsf{ECF}}}$ of $\mathcal{A}$

(c) NFA $\mathcal{A}_{\mathcal{O}_R}$

Figure 5 Counterexample for the minimality of ECF.

To determine that states p and q should not be merged, the procedure needs knowledge of their future behavior on $\mathcal{A}_{\mathcal{O}_R}$ to verify they are not part of an (EDA) condition. Consequently, no disambiguation procedure that cannot be computed on-the-fly can refine the states formed by oracle ECF and define a polynomially ambiguous automaton. Note that in the example, the relation $\mathsf{ICF}_{\mathcal{A}}$ is equivalent to $\mathsf{ECF}_{\mathcal{A}}$, then the same NFA serves as a counter example for the minimality of ICF.

Nevertheless, we can show that among all *relational* partition oracles, both $\mathcal{O}_{\mathsf{ICF}}$ and $\mathcal{O}_{\mathsf{ECF}}$ are minimal in terms of refinement; namely, $\mathsf{ICF}_{\mathcal{A}}$ and $\mathsf{ECF}_{\mathcal{A}}$ are the minimal relations for which the disambiguation schemes can reach the required level of ambiguity.

▶ **Theorem 6.** *Let $\mathcal{A}$ be an NFA and R be a reflexive and symmetric relation between states of $\mathcal{A}$. (1) If $\mathcal{A}_{\mathcal{O}_R}$ is finitely ambiguous and $R \subseteq \mathsf{ICF}_{\mathcal{A}}$, then $\mathcal{O}_{\mathsf{ICF}}$ refines $\mathcal{O}_R$ over $\mathcal{A}$; and (2) if $\mathcal{A}_{\mathcal{O}_R}$ is polynomially ambiguous and $R \subseteq \mathsf{ECF}_{\mathcal{A}}$, then $\mathcal{O}_{\mathsf{ECF}}$ refines $\mathcal{O}_R$ over $\mathcal{A}$.*

We end this section by recalling that the notion of a common future between states was previously used in [34], and the disambiguation scheme $\mathcal{D}_{\mathsf{CF}}$ was introduced in [44], but with a different presentation.The novelty of Theorem 2 relies on understanding this result in the context of a larger disambiguation framework (the one introduced in this paper) and studying its minimality (e.g., Theorem 3). Moreover, to the best of our knowledge, disambiguation procedures for finding finFA and polyFA have not been studied before.

5 Disambiguation of weighted automata

Weighted finite automata are more complex than previous automata models and have numerous use cases [18]. Moreover, they do not always admit a deterministic equivalent automata. Therefore, generalizing our algorithmic framework for disambiguation to this model would be useful for practical applications. In this section, we extend the algorithmic framework for disambiguation to the model of weighted automata. We start by recalling some algebraic notions and the model of weighted automata. We then present the framework and the disambiguation results over weighted automata.

Semirings. A *monoid* is a triple $(\mathbb{M}, \oplus, \overline{0})$ where $\mathbb{M}$ is a non-empty set, $\oplus$ is an associative binary operation, and $\overline{0} \in \mathbb{M}$ is the identity of $\oplus$ over $\mathbb{M}$ (i.e., $\overline{0} \oplus m = m \oplus \overline{0} = m$). We say that $(\mathbb{M}, \oplus, \overline{0})$ is *commutative* if, additionally, $\oplus$ is commutative. A *semiring* is a tuple

$(\mathbb{S}, \oplus, \odot, \overline{0}, \overline{1})$ such that $(\mathbb{S}, \oplus, \overline{0})$ is a commutative monoid, $(\mathbb{S}, \odot, \overline{1})$ is a monoid, $\odot$ distributes over $\oplus$ and $\overline{0}$ annihilates with $\odot$. For simplicity, we will refer to such a semiring by its underlying set $\mathbb{S}$. Examples of semirings are the *boolean semiring* $(\{0,1\}, \vee, \wedge, 0, 1)$, the *natural numbers* $(\mathbb{N}, +, \cdot, 0, 1)$, the *tropical semiring* $(\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, and the *artic semiring* $(\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$, among others.

Given non-empty sets A and B, we denote by $\mathbb{S}^{A \times B}$ the set of all *matrices* indexed by $A \times B$ over $\mathbb{S}$. Further, we denote by $\mathbb{S}^A$ the set of (column) *vectors* (i.e., $\mathbb{S}^A = \mathbb{S}^{A \times 1}$). For $M \in \mathbb{S}^{A \times B}$ and $V \in \mathbb{S}^A$, we write $M[a,b]$ and $V[a]$ for the entries at positions $(a,b) \in A \times B$ and $a \in A$, respectively. Given two matrices $M \in \mathbb{S}^{A \times B}$ and $N \in \mathbb{S}^{B \times C}$, their product $P = M \odot N \in \mathbb{S}^{A \times C}$ is defined by $P[a,c] = \bigoplus_{b \in B} M[a,b] \odot N[b,c]$ for all $a \in A$ and $c \in C$. We write $M^t \in \mathbb{S}^{B \times A}$ for the transpose of $M \in \mathbb{S}^{A \times B}$ where V^t corresponds to a row vector when $V \in \mathbb{S}^A$. For a value $s \in \mathbb{S}$ and vector $V \in \mathbb{S}^A$, we define $s \odot V \in \mathbb{S}^A$ ($V \odot s \in \mathbb{S}^A$) by $(s \odot V)[a] = s \odot V[a]$ ($(V \odot s)[a] = V[a] \odot s$, resp.) for each $a \in A$. Given a vector $V \in \mathbb{S}^A$, we define its support as $\mathsf{supp}(V) = \{a \in A \mid V[a] \neq \overline{0}\}$. Given a value $s \in \mathbb{S}$, we denote by s^A the vector in $\mathbb{S}^A$ such that $s^A[a] = s$ for every $a \in A$.

Weighted automata. A weighted finite automaton [40] (WFA) over a semiring $\mathbb{S}$ is a tuple:

$$\mathcal{W} \;:=\; (Q, \Sigma, \Delta, I, F) \tag{$\dagger$}$$

where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times \mathbb{S} \times Q$ is a finite transition relation, and $I : Q \to \mathbb{S}$, $F : Q \to \mathbb{S}$ are the initial and final weighted functions, respectively. We assume that for every $p, q \in Q$ and $a \in \Sigma$ there exists at most one transition $(p, a, s, q) \in \Delta$ for some $s \in \mathbb{S}$ (i.e., there cannot be multiple transitions between states for the same letter). We say that $q \in Q$ is an initial (final) state if $I(q) \neq \overline{0}$ ($F(q) \neq \overline{0}$, resp.). A *partial run* ρ of $\mathcal{W}$ over a word $w = a_1 \dots a_n \in \Sigma^*$ is a sequence:

$$\rho \;:=\; p_0 \xrightarrow{a_1/s_1} p_1 \xrightarrow{a_2/s_2} \cdots \xrightarrow{a_n/s_n} p_n \tag{$\ddagger$}$$

where $(p_{i-1}, a_i, s_i, p_i) \in \Delta$ for each $i \in [n]$. We say ρ is a *run* if p_0 is an initial state, and it is *accepting* if additionally p_n is a final state. We denote by $\mathsf{runs}_{\mathcal{W}}(w)$ the set of all accepting runs of $\mathcal{W}$ over w. For a partial run ρ like ($\ddagger$) we define $\omega(\rho) := I(p_0) \odot s_1 \odot \cdots \odot s_n \odot F(p_n)$ as the *weight of* ρ. Each weighted automaton $\mathcal{W}$ defines a function $[\![\mathcal{W}]\!] : \Sigma^* \to \mathbb{S}$ such that $[\![\mathcal{W}]\!](w) := \bigoplus_{\rho \in \mathsf{runs}_{\mathcal{W}}(w)} \omega(\rho)$ for every $w \in \Sigma^*$. Similar to NFA, sometimes we also interpret Δ as a function $\Delta : Q \times \Sigma \to 2^{\mathbb{S} \times Q}$ such that $\Delta(p,a) = \{(s,q) \mid (p,a,s,q) \in \Delta\}$.

In the sequel, it will be useful to consider the *matrix representation* of a weighted automaton $\mathcal{W}$. For every $a \in \Sigma$, the transition relation naturally defines the matrix $\Delta_a \in \mathbb{S}^{Q \times Q}$ such that $\Delta_a[p,q] = s$ if $(p,a,s,q) \in \Delta$ and $\overline{0}$, otherwise. We extend this matrix from Σ to Σ^* as $\Delta_w = \Delta_{a_1} \odot \cdots \odot \Delta_{a_n}$ for every $w = a_1 \dots a_n \in \Sigma^*$. Further, we can represent the initial and final functions I and F as vectors $\vec{I}, \vec{F} \in \mathbb{S}^Q$ such that $\vec{I}[q] = I(q)$ and $\vec{F}[q] = F(q)$ for every $q \in Q$. Then, the function $[\![\mathcal{W}]\!]$ can be equivalently defined as $[\![\mathcal{W}]\!](w) = \vec{I}^t \odot \Delta_w \odot \vec{F}$ for every $w \in \Sigma^*$. We will also use the notation $\vec{\Delta}(V, a) := (V^t \odot \Delta_a)^t$ which intuitively is the aggregate vector reached by starting at V and moving through a.

Given a WFA $\mathcal{W}$ like ($\dagger$), we define its *underlying NFA* $\mathcal{A}_{\mathcal{W}} = (Q, \Sigma, \Delta_{\mathsf{FA}}, I_{\mathsf{FA}}, F_{\mathsf{FA}})$ such that $I_{\mathsf{FA}} = \{p \in Q \mid I(p) \neq \overline{0}\}$, $F_{\mathsf{FA}} = \{p \in Q \mid F(p) \neq \overline{0}\}$, and $\Delta_{\mathsf{FA}} = \{(p,a,q) \mid (p,a,s,q) \in \Delta\}$. Similar to NFA, we assume that all WFA considered in this paper are *trimmed*, namely, every state in its underlying NFA is reachable and co-reachable simultaneously. Further, we say that $\mathcal{W}$ is *deterministic* (DWFA), *unambiguous* (UWFA), *finitely ambiguous* (finWFA), or *polynomially ambiguous* (polyWFA) if the underlying NFA $\mathcal{A}_{\mathcal{W}}$ is deterministic, unambiguous,

finitely ambiguous, or polynomially ambiguous, respectively. On some semirings, these classes define a strict *hierarchy of functions* [28, 11], namely, for every pair of classes $\mathcal{C}_1, \mathcal{C}_2 \in \{\text{DWFA}, \text{UWFA}, \text{finWFA}, \text{polyWFA}, \text{WFA}\}$ with $\mathcal{C}_1 \subsetneq \mathcal{C}_2$ there exists $\mathcal{W} \in \mathcal{C}_2$ such that $[\![\mathcal{W}]\!] \neq [\![\mathcal{W}']\!]$ for every $\mathcal{W}' \in \mathcal{C}_1$.

The weighted disambiguation framework. The new challenge in defining a disambiguation algorithm for weighted automata lies in handling weights during this process. Mohri's determinization algorithm [33] does this by adding a residual weight on subsets. Kirsten and Mäurer [27] later generalized this approach by introducing the weight factorization. We use these ideas to generalize the algorithmic framework of previous sections to weighted automata.

Fix a semiring $\mathbb{S}$ and a set Q. A *weight factorization* [27] over $\mathbb{S}^Q$ is a pair of functions $(\mathsf{fac}, \mathsf{res})$, called the *residual* and *factor* functions, respectively, with $\mathsf{fac} : \mathbb{S}^Q \setminus \{\overline{0}^Q\} \mapsto \mathbb{S}$ and $\mathsf{res} : \mathbb{S}^Q \setminus \{\overline{0}^Q\} \mapsto \mathbb{S}^Q$ such that for every vector $V \in \mathbb{S}^Q \setminus \{\overline{0}^Q\}$ it holds that $V = \mathsf{fac}(V) \odot \mathsf{res}(V)$. In other words, $(\mathsf{fac}, \mathsf{res})$ defines a strategy to factorize a vector V into a single factor common to all components ($\mathsf{fac}(V)$) and a vector ($\mathsf{res}(V)$).

Let $\mathcal{W}$ be an WFA like (†) over $\mathbb{S}$. We define a *partition-factorization oracle* $\mathcal{F}$ for $\mathcal{W}$ (called PF-oracle for short) as a triple $\mathcal{F} = (\Pi, \mathsf{fac}, \mathsf{res})$ where $(\mathsf{fac}, \mathsf{res})$ is a weight factorization over $\mathbb{S}^Q$ and $\Pi : \mathbb{S}^Q \mapsto 2^{\mathbb{S}^Q \setminus \{\overline{0}^Q\}}$ is a *weight partition function* such that $\Pi(V)$ is a finite set and $\bigoplus_{V' \in \Pi(V)} V' = V$ for every $V \in \mathbb{S}^Q$. That is, $\Pi(V)$ distributes the values of vector V over a set of vectors that has to $\oplus$-aggregate back to V. Indeed, one can see Π as a generalization of the partition oracle $\mathcal{O}$ from the boolean semiring to any semiring $\mathbb{S}$. Given a PF-oracle $\mathcal{F} = (\Pi, \mathsf{fac}, \mathsf{res})$ for $\mathcal{W}$, we define the (infinite) WFA:

$$\mathcal{W}^\infty_{\mathcal{F}} \;:=\; (\mathbb{S}^Q, \Sigma, \Delta_{\mathcal{F}}, I_{\mathcal{F}}, F_{\mathcal{F}})$$

such that $\mathsf{supp}(\vec{I}_{\mathcal{F}}) = \{\mathsf{res}(V) \mid V \in \Pi(\vec{I})\}$ and $I_{\mathcal{F}}(\mathsf{res}(V)) = \mathsf{fac}(V)$ for all $V \in \Pi(\vec{I})$; $F_{\mathcal{F}}(V) = V^t \odot \vec{F}$ for every $V \in \mathbb{S}^Q$; and:

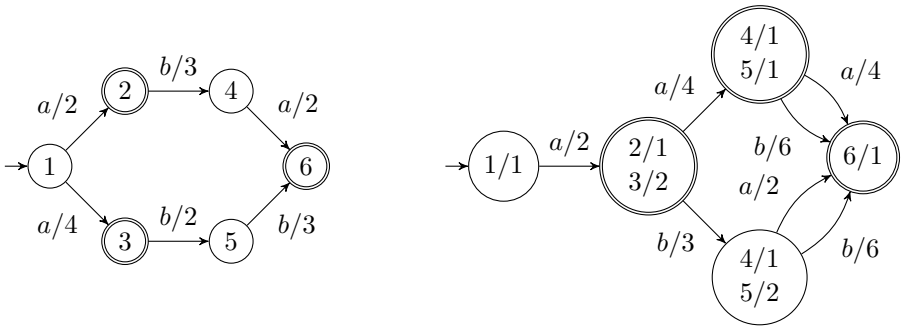
$$\Delta_{\mathcal{F}} \;:=\; \big\{\, (V, a, \mathsf{fac}(V'), \mathsf{res}(V')) \;\mid\; V' \in \Pi(\vec{\Delta}(V, a)) \,\big\}.$$

Note that $\vec{I}_{\mathcal{F}}$ has finite support and, for every $V \in \mathbb{S}^Q$, $\Delta_{\mathcal{F}}(V, a)$ is finite. So, although $\mathcal{W}^\infty_{\mathcal{F}}$ has an infinite number of states when $\mathbb{S}$ is infinite, there is always a finite number of initial states and every state/letter has a finite number of next configurations. We define $\mathcal{W}_{\mathcal{F}}$ as the *trimmed version of* $\mathcal{W}^\infty_{\mathcal{F}}$. Notice that $\mathcal{W}_{\mathcal{F}}$ could lead to a finite WFA but not always.

▶ **Proposition 7.** *For every PF-oracle $\mathcal{F}$ of $\mathcal{W}$, if $\mathcal{W}_{\mathcal{F}}$ is finite, then $[\![\mathcal{W}]\!] = [\![\mathcal{W}_{\mathcal{F}}]\!]$.*

For a class $\mathcal{C}$ of WFA, the notion of a *$\mathcal{C}$-disambiguation scheme* $\mathcal{D}$ naturally extends from NFA to WFA where $\mathcal{D}(\mathcal{W})$ must output a PF-oracle for a WFA $\mathcal{W}$. The notion of an *efficient* disambiguation scheme $\mathcal{D}$ also extends to WFA where now, for every vector V, $\Pi(V)$, $\mathsf{fac}(V)$, and $\mathsf{res}(V)$ must be computed in polynomial time over $|\mathcal{W}|$ whenever $\mathcal{D}(\mathcal{W}) = (\Pi, \mathsf{fac}, \mathsf{res})$.

One can check that the construction $\mathcal{W}_{\mathcal{F}}$ is a generalization of Mohri's determinization algorithm [33]. Specifically, for the tropical semiring $\mathbb{T} = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$ and a WFA $\mathcal{W}$ like (†) over $\mathbb{T}$, consider the PF-oracle $\mathcal{F}_{\mathsf{M}} = (\Pi_{\mathsf{id}}, \mathsf{fac}_{\mathsf{M}}, \mathsf{res}_{\mathsf{M}})$ where $\Pi_{\mathsf{id}}(V) = \{V\}$, $\mathsf{fac}_{\mathsf{M}}(V) = min_{q \in Q} V[q]$ and $\mathsf{res}_{\mathsf{M}}(V) = V - \mathsf{fac}_{\mathsf{M}}(V)$ for every $V \in \mathbb{T}^Q$. Then $\mathcal{W}_{\mathcal{F}_{\mathsf{M}}}$ indeed corresponds to Mohri's determinization construction. Similarly, consider now any semiring $\mathbb{S}$ and a PF-oracle $\mathcal{F}_{\mathsf{K}} = (\Pi_{\mathsf{id}}, \mathsf{fac}, \mathsf{res})$ for any weight factorization $(\mathsf{fac}, \mathsf{res})$ over $\mathbb{S}^Q$. And now $\mathcal{W}_{\mathcal{F}_{\mathsf{K}}}$ is equivalent to Kirsten and Mäurer's determinization construction [27]. What our approach adds is that we can consider different weight partition functions Π, which could lead to different levels of ambiguity.



(a) finWFA $\mathcal{W}$ (b) UWFA $\mathcal{W}_{\mathcal{F}}$

Figure 6 WFA $\mathcal{W}$ and its disambiguation $\mathcal{W}_{\mathcal{F}}$. Every vector V state in $\mathcal{W}_{\mathcal{F}}$ is shown as pairs q/v, where $V[q] = v$, states q that are not part of the support of V are omitted.

Weighted disambiguation. Within this framework, we can combine various factorizations, partitions, and semirings to define disambiguation algorithms for WFA. However, the effectiveness of these algorithms depends on whether $\mathcal{W}_{\mathcal{F}}$ is finite for all input WFA $\mathcal{W}$. Furthermore, we aim to ensure the desired properties of idempotent, on-the-fly, and efficiency discussed in the previous sections. We address this by introducing a specific class of functions Π that, combined with the oracles from Section 4, will guarantee the desired level of ambiguity.

Given a WFA $\mathcal{W}$ like (†) over a semiring $\mathbb{S}$ and a PF-oracle $\mathcal{F} = (\Pi, \mathsf{fac}, \mathsf{res})$, we say that Π has *disjoint vector support* iff $\mathsf{supp}(V_1) \cap \mathsf{supp}(V_2) = \emptyset$ for every pair $V_1, V_2 \in \Pi(V)$ with $V_1 \neq V_2$ and $V \in \mathbb{S}^Q$. Further, we say that $(\mathsf{fac}, \mathsf{res})$ has *identity factorization* if $\mathsf{fac}(s \odot V) = s$ and $\mathsf{res}(s \odot V) = V$ whenever V is a zero-one vector (i.e., $V \in \{\overline{0}, \overline{1}\}^Q$). For example, the PF-oracle $\mathcal{F}_{\mathsf{M}} = (\Pi_{\mathsf{id}}, \mathsf{fac}_{\mathsf{M}}, \mathsf{res}_{\mathsf{M}})$ of Mohri's determinization satisfies both properties.

As we will see, the identity factorization of $(\mathsf{fac}, \mathsf{res})$ will ensure the idempotent property of disambiguation algorithms; whereas the disjoint vector support of Π will allow to extend the partition oracles found for NFA to WFA. Specifically, let $\mathcal{O}$ be a partition oracle for the underlying NFA $\mathcal{A}_{\mathcal{W}}$ of a WFA $\mathcal{W}$. Then $\mathcal{O}$ naturally defines a weight partition function $\Pi_{\mathcal{O}}$ over $\mathbb{S}^Q$ given by $\Pi_{\mathcal{O}}(V) = \{V_S \mid S \in \mathcal{O}(\mathsf{supp}(V))\}$ where $V_S \in \mathbb{S}^Q$ is defined as $V_S[q] = V[q]$ if $q \in S$, and $V_S[q] = \overline{0}$ otherwise. Note that $\Pi_{\mathcal{O}}$ has disjoint vector support.

▶ **Theorem 8.** *Let $\mathcal{W}$ be a WFA over $\mathbb{S}$ and $(\mathsf{fac}, \mathsf{res})$ be a weight factorization. For a PF-oracle $\mathcal{F}$ equal to $(\Pi_{\mathcal{O}_{\mathsf{CF}}}, \mathsf{fac}, \mathsf{res})$, $(\Pi_{\mathcal{O}_{\mathsf{ICF}}}, \mathsf{fac}, \mathsf{res})$, or $(\Pi_{\mathcal{O}_{\mathsf{ECF}}}, \mathsf{fac}, \mathsf{res})$, if $\mathcal{W}_{\mathcal{F}}$ is finite, then $\mathcal{W}_{\mathcal{F}}$ is unambiguous, finitely ambiguous, or polynomially ambiguous, respectively. Furthermore, if $\mathcal{W}$ is unambiguous, finitely ambiguous, or polynomially ambiguous and $(\mathsf{fac}, \mathsf{res})$ is an identity factorization, then $\mathcal{W}_{\mathcal{F}}$ is isomorphic to $\mathcal{W}$.*

The previous result shows that the partition oracles for disambiguation introduced in Section 4 naturally extend to the weighted case – but only if the resulting disambiguation $\mathcal{W}_{\mathcal{F}}$ is finite. Furthermore, these PF-oracles define the corresponding efficient disambiguation schemes when the weighted factorization $(\mathsf{fac}, \mathsf{res})$ can be computed in polynomial time.

One example of disambiguation is shown in Figure 6 with WFA $\mathcal{W}$ over the natural numbers semiring $\mathbb{S} = (\mathbb{N}, +, \cdot, 0, 1)$ and PF-oracle $\mathcal{F} = (\Pi_{\mathcal{O}_{\mathsf{CF}}}, \mathsf{fac}, \mathsf{res})$, where $\mathsf{fac}(V) = \mathsf{GCD}(V)$ and $\mathsf{res}(V) = \frac{V}{\mathsf{GCD}(V)}$ for all $V \in \mathbb{S}^Q$. $\mathsf{GCD}(V)$ is the *greatest common divisor* of all elements in V. $\mathcal{W}_{\mathcal{F}}$ is unambiguous and equivalent to $\mathcal{W}$.

Generalization of the *twins property*. For this subsection, fix the tropical semiring $\mathbb{T} = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$. In [33], Mohri studied the *twins property*, a condition towards characterizing termination for his algorithm for determinizing weighted automata over $\mathbb{T}$.

This condition is sufficient in general, and also necessary when the weighted automata is unambiguous. Here, we demonstrate how to generalize the twins property to provide sufficient conditions for finite disambiguation across different levels of ambiguity.

Let $\mathcal{W}$ be a WFA of the form (†) over $\mathbb{T}$. Given a relation $R \subseteq Q \times Q$, we say that two states $p, q \in Q$ are *R-twins* iff for every $u, w \in \Sigma^*$, whenever $p, q \in \Delta_{\mathsf{FA}}(I_{\mathsf{FA}}, u)$, $p \in \Delta_{\mathsf{FA}}(p, w)$, $q \in \Delta_{\mathsf{FA}}(q, w)$, and $(p, q) \in R$, then $\Delta_w[p, p] = \Delta_w[q, q]$. In other words, if w forms a loop in p and in q and these nodes are related by R, then the weight of the loops must be the same. Then, we say that $\mathcal{W}$ has the *R-twins property* if every pair of states in Q are R-twins.

Note that the twins property in [33] is the special case when $R = Q \times Q$. We generalize this property for the relations introduced in Section 4, providing sufficient conditions for termination of the disambiguation algorithms.

▶ **Theorem 9.** *Let $\mathcal{W}$ be a WFA over $\mathbb{T}$ and $R \in \{\mathsf{CF}_{\mathcal{W}}, \mathsf{ICF}_{\mathcal{W}}, \mathsf{ECF}_{\mathcal{W}}\}$. If $\mathcal{W}$ satisfies the R-twins property, then $\mathcal{W}_{\mathcal{F}}$ is finite when $\mathcal{F} = (\Pi_{\mathcal{O}_R}, \mathsf{fac}_{\mathsf{M}}, \mathsf{res}_{\mathsf{M}})$ is a PF-oracle.*

Similarly to [33], the previous result provides sufficient conditions for termination when disambiguating WFA for each level of ambiguity. We leave as an open problem to understand when the R-twins property is a *necessary* condition in each case.

6 Future work

As an initial paper on this algorithmic framework, numerous approaches remain to be explored for further study. Besides the open problems we have stated throughout the paper, some natural follow-up work would be extending the framework onto tree automata [15], cost-register automata [4], or Büchi automata [43]. In the last case, it is an open problem to find good disambiguation algorithms, since deterministic Büchi automata are less expressive than unambiguous ones. It will be interesting to see if the algorithmic framework presented here could lead to new insights on the disambiguation of Büchi automata. We highlight one open problem, posed in Section 4, we find particularly interesting, which is to find optimal disambiguation schemes that minimize the number of explored states. Finally, our disambiguation framework is simple enough to be implemented and used in practice. We are interested in seeing it being explored as an alternative to determinization in real-life cases where an unambiguous, or a boundedly ambiguous automaton, is good enough.

References

1 Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *CIAA*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007. doi:10.1007/978-3-540-76336-9_3.

2 Shaull Almagor, Guy Arbel, and Sarai Sheinvald. Determinization of min-plus weighted automata is decidable. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 247–257. SIAM, 2026. doi:10.1137/1.9781611978971.11.

3 Shaull Almagor, Guy Arbel, and Sarai Sheinvald. Unambiguisability and register minimisation of min-plus models. In *53rd International Colloquium on Automata, Languages, and Programming, ICALP 2026, Royal Holloway, University of London, Egham, United Kingdom, July 7-10, 2026*, volume 374 of *LIPIcs*, pages 159:1–159:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.ICALP.2026.159.

4 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *LICS*, pages 13–22, 2013. doi:10.1109/LICS.2013.65.

5 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *ICALP*, volume 80 of *LIPIcs*, pages 111:1–111:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.111.

6 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.

7 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM*, 68(6):48:1–48:40, 2021. doi:10.1145/3477045.

8 Marco Bucchi, Alejandro Grez, Andrés Quintana, Cristian Riveros, and Stijn Vansummeren. CORE: a complex event recognition engine. *VLDB*, 15(9):1951–1964, 2022. doi:10.14778/3538598.3538615.

9 Mauricio Cari, Martín Muñoz, and Cristian Riveros. A simple algorithmic framework for disambiguation of finite automata, 2026. doi:10.48550/arXiv.2607.06894.

10 Tat-hung Chan and Oscar H. Ibarra. On the finite-valuedness problem for sequential machines. *Theor. Comput. Sci.*, 23:95–101, 1983. doi:10.1016/0304-3975(88)90012-6.

11 Agnishom Chattopadhyay, Filip Mazowiecki, Anca Muscholl, and Cristian Riveros. Pumping lemmas for weighted automata. *Log. Methods Comput. Sci.*, 17(3), 2021. doi:10.46298/lmcs-17(3:7)2021.

12 Thomas Colcombet. Forms of determinism for automata (invited talk). In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, Paris, France, February 29 - March 3, 2012*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.1.

13 Thomas Colcombet. Unambiguity in automata theory. In *DCFS*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3_1.

14 Thomas Colcombet, Karin Quaas, and Michał Skrzypczak. Unambiguity in automata theory (dagstuhl seminar 21452). *Dagstuhl Reports*, 11(10):57–71, 2022. doi:10.4230/DagRep.11.10.57.

15 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications, 2008.

16 Russ Cox. Regular expression matching can be simple and fast (but is slow in java, perl, php, python, ruby,...). URL: https://swtch.com/~rsc/regexp/regexp1.html, 2007.

17 Karolina Drabik, Anita Dürr, Fabian Frei, Filip Mazowiecki, and Karol Węgrzycki. Fined-grained complexity of ambiguity problems on automata and directed graphs. *CoRR*, abs/2501.14725, 2025. doi:10.48550/arXiv.2501.14725.

18 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

19 Nathanaël Fijalkow, Cristian Riveros, and James Worrell. Probabilistic automata of bounded ambiguity. *Inf. Comput.*, 282:104648, 2022. doi:10.1016/j.ic.2020.104648.

20 Tero Harju, H. C. M. Kleijn, and Michel Latteux. Compositional representation of rational functions. *RAIRO Theor. Informatics Appl.*, 26:243–255, 1992. doi:10.1051/ita/1992260302431.

21 Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on software engineering*, 23(5):279–295, 1997. doi:10.1109/32.588521.

22 John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

23 Juraj Hromkovic and Georg Schnitger. Ambiguity and communication. *Theory Comput. Syst.*, 48(3):517–534, 2011. doi:10.1007/s00224-010-9277-4.

24 Juraj Hromkovic, Sebastian Seibert, Juhani Karhumäki, Hartmut Klauck, and Georg Schnitger. Communication complexity method for measuring nondeterminism in finite automata. *Inf. Comput.*, 172(2):202–217, 2002. doi:10.1006/inco.2001.3069.

25 Ismaël Jecker, Filip Mazowiecki, and David Purser. Determinisation and unambiguisation of polynomially-ambiguous rational weighted automata. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, 2024. doi:10.1145/3661814.3662073.

26 Daniel Kirsten. A burnside approach to the termination of mohri's algorithm for polynomially ambiguous min-plus-automata. *RAIRO Theor. Informatics Appl.*, 42(3):553–581, 2008. doi:10.1051/ita:2008017.

27 Daniel Kirsten and Ina Mäurer. On the determinization of weighted automata. *J. Autom. Lang. Comb.*, 10(2/3):287–312, 2005. doi:10.25596/jalc-2005-287.

28 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theor. Comput. Sci.*, 327(3):349–373, 2004. doi:10.1016/j.tcs.2004.02.049.

29 Ernst L. Leiss. Succinct representation of regular languages by boolean automata II. *Theor. Comput. Sci.*, 38:133–136, 1985. doi:10.1016/0304-3975(85)90215-4.

30 Hing Leung. Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. *SIAM J. Comput.*, 27(4):1073–1082, 1998. doi:10.1137/S0097539793252092.

31 Hing Leung. Descriptional complexity of nfa of different ambiguity. *Int. J. Found. Comput. Sci.*, 16(5):975–984, 2005. doi:10.1142/S0129054105003418.

32 Albert R Meyer and Michael J Fischer. Economy of description by automata, grammars, and formal systems. In *12th annual symposium on switching and automata theory (swat 1971)*, pages 188–191. IEEE Computer Society, 1971. doi:10.1109/SWAT.1971.11.

33 Mehryar Mohri. Finite-state transducers in language and speech processing. *Comput. Linguistics*, 23(2):269–311, 1997.

34 Mehryar Mohri. On the disambiguation of finite automata and functional transducers. *Int. J. Found. Comput. Sci.*, 24(6):847–862, 2013. doi:10.1142/S0129054113400224.

35 Mehryar Mohri and Michael D. Riley. A disambiguation algorithm for weighted automata. *Theor. Comput. Sci.*, 679:53–68, 2017. doi:10.1016/j.tcs.2016.08.019.

36 Jean-Éric Pin, editor. *Handbook of Automata Theory*. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/Automata.

37 Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, 1959. doi:10.1147/rd.32.0114.

38 Cristian Riveros, Nicolás Van Sint Jan, and Domagoj Vrgoč. Rematch: a novel regex engine for finding all matches. *VLDB*, 16(11):2792–2804, 2023. doi:10.14778/3611479.3611488.

39 Jacques Sakarovitch. A construction on finite automata that has remained hidden. *Theor. Comput. Sci.*, 204(1-2):205–231, 1998. doi:10.1016/S0304-3975(98)00040-1.

40 Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control.*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

41 Richard Edwin Stearns and Harry B Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985. doi:10.1137/0214044.

42 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

43 Wolfgang Thomas. Automata on infinite objects. In *Formal Models and Semantics*, pages 133–191. Elsevier, 1990. doi:10.1016/b978-0-444-88074-1.50009-3.

44 Andreas Weber and Reinhard Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, 118(2):327–340, 1995. doi:10.1006/inco.1995.1071.

45 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991. doi:10.1016/0304-3975(91)90381-B.

46 Shuly Wintner. Emmanuel roche and yves schabes, editors, *Finite-State Language Processing*. MIT press, cambridge, ma, 1997. ISBN 0-262-18182-7. ix+464 pages. *Nat. Lang. Eng.*, 7(1):87–97, 2001. URL: http://journals.cambridge.org/action/displayAbstract?aid=73677.

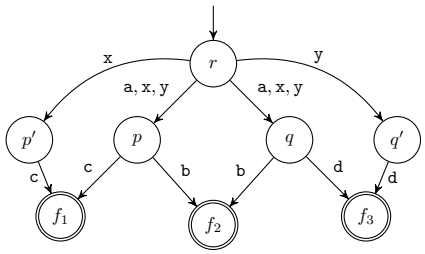


Figure 7 Example of an automaton $\mathcal{A}$ such that is not equivalent to $\mathcal{A}_{\mathsf{M}}$ for some order of the states added to the queue.

A A counterexample to Mohri's disambiguation algorithm

A disambiguation algorithm for finite automata was proposed by Mehryar Mohri in [34]. The main idea of Mohri's disambiguation is to work over an equivalent automaton $\mathcal{A}_{\mathsf{M}}$ that simulates the runs of the original automaton $\mathcal{A}$, and the states keep track of competing runs (i.e., runs that have a common future). $\mathcal{A}_{\mathsf{M}}$ is equivalent to $\mathcal{A}$ but it can still be ambiguous. For this reason, Mohri's disambiguation traverses the automaton $\mathcal{A}_{\mathsf{M}}$ in a depth-search manner, and prunes transitions that lead to ambiguous runs. The resulting unambiguous automaton depends on the order in which $\mathcal{A}_{\mathsf{M}}$ is traversed, that is, the order in which the transitions are chosen. For this reason, Mohri's disambiguation is not unique for each finite automaton $\mathcal{A}$. Moreover, as we will show below, there exists a finite automaton $\mathcal{A}$ such that for some order traversal of $\mathcal{A}_{\mathsf{M}}$, the resulting disambiguation is not equivalent to $\mathcal{A}$. In other words, Mohri's disambiguation is not correct for all input.

The counterexample is the NFA $\mathcal{A}$ shown in Figure 7. Next, we present the result of applying Mohri's disambiguation procedure over $\mathcal{A}$. For a detailed presentation of Mohri's algorithm, we refer the reader to [34].

In the following, we denote by $\mathcal{A}_{\mathsf{M}} = (Q_{\mathsf{M}}, \Sigma, \Delta_{\mathsf{M}}, I_{\mathsf{M}}, F_{\mathsf{M}})$ be the result of Mohri's procedure over $\mathcal{A}$. During the construction, we assume that the states are enqueued in $\mathcal{Q}$ in the following order:

$$(r, \{r\}),\ (p, \{p, p', q\}),\ (q, \{p, q, q'\}),$$
$$(p, \{p, q\}),\ (q, \{p, q\}),\ (p', \{p, p'\}),\ (q', \{q, q'\}),$$
$$(f_1, \{f_1\}),\ (f_2, \{f_2\}),\ (f_3, \{f_3\}).$$

Construction of $\mathcal{A}_{\mathsf{M}}$. The set of initial states and the queue are computed:

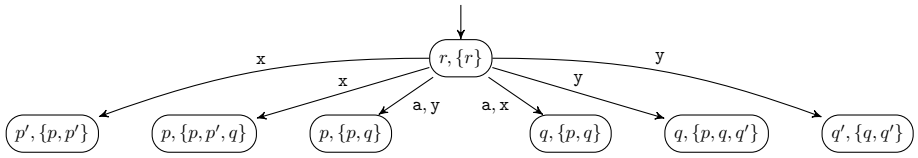
$$\mathcal{Q} = I_{\mathsf{M}} = \{(r, \{r\})\}.$$

The set of transitions Δ_{M}, the set of states Q_{M} and set of final states F_{M} starts empty. Next, the common future relation $\mathsf{CF}_{\mathcal{A}}$ is computed. In this case, $\mathsf{CF}_{\mathcal{A}}$ contains the pairs of states:

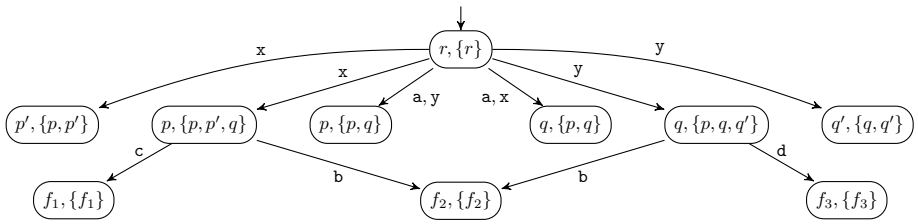
$$(p', p), \quad (p, q), \quad (q, q').$$

The first state $(r, \{r\})$ is dequeued, and all outgoing transitions from r are examined. For each transition $(r, a, q) \in \Delta$, the algorithm constructs the state

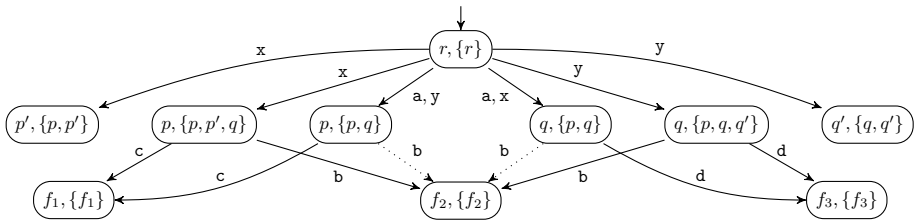
$$(q, T) \quad \text{where} \quad T = \{s \in \Delta(\{r\}, a) \mid (q, s) \in \mathsf{CF}_{\mathcal{A}}\}.$$



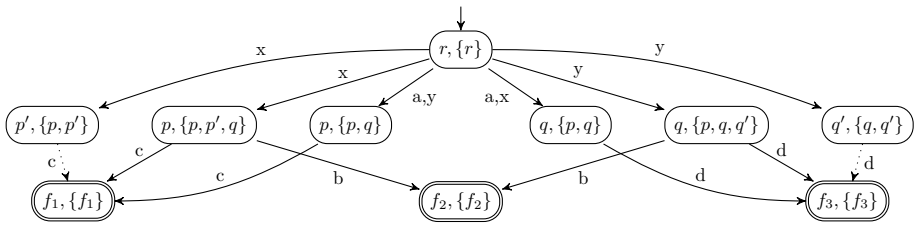
(a) Partial automaton $\mathcal{A}_\mathsf{M}$ after dequeing and processing the state $(r, \{r\})$.



(b) Partial automaton $\mathcal{A}_\mathsf{M}$ after dequeing and processing the states $(p, \{p, p', q\})$ and $(q, \{p, q, q'\})$.



(c) Partial automaton $\mathcal{A}_\mathsf{M}$ after dequeing and processing the states $(p, \{p, q\})$ and $(q, \{p, q\})$, the dotted transitions were discarded by the algorithm.



(d) Automaton $\mathcal{A}_\mathsf{M}$, the dotted transitions were discarded by the algorithm.

Figure 8 Mohri's disambiguation steps of the NFA $\mathcal{A}$ shown in Figure 7.

Then, the transition $((r, \{r\}), a, (q, T))$ is added to Δ_M, unless there already exists a state $(p, S) \in Q_\mathsf{M}$ such that (p, S) and $(r, \{r\})$ are both reachable by the same word w (in this case $w = \varepsilon$), and the transition $((p, S), a, (q, T))$ is already in Δ_M. This restriction avoids having two distinct runs over the same word from I_M to (q, T) [34].

For example, consider the transition $(r, x, p) \in \Delta$. The algorithm creates the state $(p, \{p, p', q\})$, which is reachable by x. Here, the states p, p', q are all reachable by x and share a common future with p. Similarly, for the transition $(r, y, p) \in \Delta$, the state $(p, \{p, q\})$ is created. Although p, q, q' are reachable by y, only p and q share a common future with p.

The state (q, T) is then added to Q_M. In this first step, no transitions are discarded, and all newly created states are queued in $\mathcal{Q}$. The resulting partial automaton $\mathcal{A}_\mathsf{M}$ is shown in Figure 8a.

Processing $(p, \{p, p', q\})$ and $(q, \{p, q, q'\})$. These two states are dequeued next. All outgoing transitions from p and q are processed, producing new transitions in Δ_{M}, none of which are discarded. As a result, the states $(f_1, \{f_1\}), (f_2, \{f_2\}), (f_3, \{f_3\})$ are queued. Since f_1, f_2, and f_3 do not share a common future with any other state, they remain as singleton states. The resulting automaton is shown in Figure 8b.

Processing $(p, \{p, q\})$ and $(q, \{p, q\})$. For the state $(p, \{p, q\})$, p has two outgoing transitions. The transition

$$((p, \{p, q\}), c, (f_1, \{f_1\}))$$

is added to Δ_{M}, but the transition

$$((p, \{p, q\}), b, (f_2, \{f_2\}))$$

is discarded, since $(q, \{p, q, q'\})$ and $(p, \{p, q\})$ are both reachable by y, and $(q, \{p, q, q'\})$ already has a transition over b to $(f_2, \{f_2\})$. Analogously, when processing $(q, \{p, q\})$, only one of its two transitions is added. The resulting automaton is shown in Figure 8c.

Final steps. The states $(p', \{p, p'\})$ and $(q', \{q, q'\})$ are dequeued, but their outgoing transitions are discarded. The states $(f_1, \{f_1\})$ and $(f_2, \{f_2\})$ are already reachable by xc from $(p, \{p, p', q\})$ and by yd from $(q, \{p, q, q'\})$, respectively. Finally, the states $(f_1, \{f_1\})$, $(f_2, \{f_2\})$, and $(f_3, \{f_3\})$ are dequeued and added to F_{M}. The resulting automaton $\mathcal{A}_{\mathsf{M}}$ is shown in Figure 8d. One can easily note that the word $ab \in \mathcal{L}(\mathcal{A})$ is not accepted by $\mathcal{A}_{\mathsf{M}}$, thus $\mathcal{A}_{\mathsf{M}}$ is not equivalent to $\mathcal{A}$.

In conclusion, Mohri's algorithm is not correct for all finite automata. Specifically, the step of discarding outgoing transitions from states $(p, \{p, q\})$ correctly discards an already existing accepting run over yb, but also discards the accepting run over ab. Therefore, disambiguation procedures that are not computed on the fly needs to track a significant amount of information to ensure that no words in $\mathcal{L}(\mathcal{A})$ are discarded.