Solving Language Equations Using Flanked Automata

Florent Avellaneda, Silvano Dal Zilio, Jean-Baptiste Raclet

Solving Language Equations using Flanked Automata

Florent Avellaneda[1], Silvano Dal Zilio[2], and Jean-Baptiste Raclet[3]

[1] CRIM Montréal, Canada
[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[3] IRIT, Université de Toulouse, CNRS, Toulouse, France

Abstract. We define a new subclass of nondeterministic finite automata for prefix-closed languages called *Flanked Finite Automata* (FFA). Our motivation is to provide an efficient way to compute the quotient and inclusion of regular languages without the need to determinize the underlying automata. These operations are the building blocks of several verification algorithms that can be interpreted as language equation solving problems. We provide a construction for computing a FFA accepting the quotient and product of languages that is compositional and that does not incur an exponential blow up in size. This makes flanked automata a good candidate as a formalism for compositional design and verification of systems.

1 Introduction

A very common problem in system design is to solve equations of the form $C \parallel X \preceq G$, where C is the specification of a given system and G is the overall behavior (the goal) that we want to implement. The objective is to compute a subsystem X that, when composed with C, produces a system which conforms to the specification G. We are generally interested in the maximal solution. When it exists, this solution is denoted G/C, also called the quotient of G by C.

Solving language equations is a problem that appears in many different domains, with different choices for the composition operator ($\parallel$) and for the conformance relation ($\preceq$). For example, this problem has been studied by the discrete-event systems community under the name *controller synthesis* [19]. In other works, finding X is sometimes referred to as computing a *protocol converter* or an *adaptor* [24]. In this context, the goal is to correct some mismatches between a set of n interacting subsystems in order to satisfy a compatibility property (deadlock freeness, for instance) specified by G. Likewise, the quotient G/C can be seen as the implementation of a subsystem that needs to realize a given specification G while reusing a trustworthy *off-the-shelf* component C [18]. Finally, computing the language quotient is a stepping stone to verify contract satisfaction [6]. The links between all these problems has been clearly highlighted in the literature [23, 11].

Our interest in the quotient operator is motivated by our interest in contract-based design. Contracts have recently been identified as a key element for the

modular design of complex systems [7]. Fundamentally, a contract for a system S can be viewed as a pair (A, G) of two specification requirements, where A is an assumption on the environment where S executes and G is a guarantee on the behavior of the system (given that the assumptions in A are met). Namely, with our notations, the pair (A, G) is a contract for S if and only if $A \parallel S \preceq G$. In this case, when we fix the guarantee G, the best possible assumption is given by the quotient G/S.

Contracts, and the use of the quotient operator, arises naturally in the context of compositional verification. For example, when we consider the simplest instance of the Assume-Guarantee law (see for example [12]):

$$\frac{A \parallel P_1 \preceq G \qquad P_2 \preceq A}{P_1 \parallel P_2 \preceq G}$$

then a natural choice for the assumption A is to find a contract of the form (A, G) for P_1. Also, the quotient operator is central when computing the contract of a compound system $P_1 \parallel P_2$. Indeed, if (A_1, G_1) and (A_2, G_2) are contracts for the processes P_1, P_2, then a sensible contract for $P_1 \parallel P_2$ is given by the pair $(A_1/G_2 \wedge A_2/G_1, G_1 \parallel G_2)$. As a consequence, it is clear that any tool based on the use of contract theory needs to compute quotients efficiently.

In this paper, we propose a new method to compute the quotient and composition of two or more specifications in a compositional way. We describe our approach by choosing the simplest possible instantiation for the language equation problem. We consider that the semantics of a system is given by a regular and prefix-closed set of traces. Likewise, we use language intersection for the composition of systems ($\parallel$) and language inclusion for conformance ($\preceq$). In this simple context, the quotient of two prefix-closed regular languages G/C can be defined as the biggest prefix-closed language included in $G \cup \overline{C}$, where $\overline{C}$ is the complement of language C. While we concentrate on regular languages in this paper, our approach can be extended to more general composition operators, like synchronous product, and to more complex formalisms.

Contributions. Since we want to solve a problem on regular languages, the simplest choice would be to select either deterministic (DFA) or nondeterministic finite automata (NFA); but this is not satisfying. While the problems of checking universality or language inclusion are known to be computationally easy for DFA, they are PSPACE-complete for NFA. On the other hand, the size of a NFA can be exponentially smaller than the size of an equivalent minimal DFA. This gap in complexity between the two models can be problematic in practice. This is the case when using finite state automata for system verification, where we need to manipulate a very large number of states.

To solve this problem, we need an extension of finite automata that share the same complexity properties as DFA while being, as much as possible, as succinct as NFA. In this paper, we define a new class of finite state automata called *Flanked Finite Automata* (FFA) that has good complexity and closure properties. With our approach, it is possible to efficiently compute the quotient

of two languages without relying on the use of deterministic automata or on the determinization of automata. We also prove that FFA can be exponentially more succinct than an equivalent DFA. We give some examples of the gain of performance brought by this new approach with a simple use case (Sect. 6).

In Sect. 3, we show that the universality problem for FFA is in linear-time while testing the language inclusion between two FFA $\mathcal{A}$ and $\mathcal{B}$ is in time $O(|\mathcal{A}| \cdot |\mathcal{B}|)$. In Sect. 4, we define several operations on FFA. In particular we describe how to compute a flanked automaton for the intersection, union and quotient of two languages defined by FFA. The benefit of our encoding is that the composition of two FFA, $\mathcal{A}$ and $\mathcal{B}$, has always less than $(|\mathcal{A}| + 1) \cdot (|\mathcal{B}| + 1)$ states. Moreover the resulting automaton is still flanked. Therefore it is possible to compute the successive composition and quotient of different specifications $\mathcal{A}_1, \ldots, \mathcal{A}_n$ in time $O(|\mathcal{A}_1| \cdot \ldots \cdot |\mathcal{A}_n|)$.

Finally, we prove that FFA are strictly more concise than DFA. Indeed, on the one hand, every DFA can be easily extended into a FFA with the same set of states and transitions. On the other hand, in Sect. 5, we give an example of (a family of) regular languages that can be accepted by FFA which are exponentially more succinct than their equivalent minimal DFA.

Our main motivation for introducing a new extension of NFA is to provide an efficient way to compute the quotient of two regular languages. We believe that our work provides the first algorithm for computing the quotient of two regular languages without using determinization and without suffering from an exponential blow up of the result. Our approach can be slightly modified to support other kinds of composition operators, like for instance the synchronous product of languages, instead of simply language intersection. It can also be easily extended to take into account the addition of modalities [18]. We also believe that the notion of "flanked relation" can be easily applied to other settings, like for example tree automata. For instance, the prototype implementation of our algorithms can also handle trace languages generated by "flanked" Petri nets.

2 Notations and Definitions

A finite automaton is a tuple $\mathcal{A} = (Q, \Sigma, E, Q_{\text{in}})$ where: Q is a finite set of states; Σ is the alphabet of $\mathcal{A}$ (that is a finite set of symbols); $E \subseteq Q \times \Sigma \times Q$ is the transition relation; and $Q_{\text{in}} \subseteq Q$ is the set of initial states. In the remainder of this text, we assume that every state is final, hence we do not need a distinguished subset of accepting states. Without loss of generality, we also assume that every state in Q is reachable in $\mathcal{A}$ from Q_{in} following a sequence of transitions in E.

For every word $u \in \Sigma^*$ we denote $\mathcal{A}(u)$ the subset of states in Q that can be reached when trying to accept the word u from an initial state in the automaton. We can define the set $\mathcal{A}(u)$ by induction on the word u. We assume that ϵ is the empty word and we use the notation $u\,a$ for the word obtained from u by concatenating the symbol $a \in \Sigma$. Then $\mathcal{A}(\epsilon) = Q_{\text{in}}$ and $\mathcal{A}(u\,a) = \{q' \mid \exists q \in \mathcal{A}(u).(q, a, q') \in E\}$. By extension, we say that a word u is accepted by $\mathcal{A}$, denoted $u \in \mathcal{A}$, if the set $\mathcal{A}(u)$ is not empty.

Definition 1. *A Flanked Finite Automaton (FFA) is a pair* $(\mathcal{A}, F)$ *where* $\mathcal{A} = (Q, \Sigma, E, Q_{in})$ *is a finite automaton and* $F : Q \times \Sigma$ *is a "flanking relation" that associates symbols of* Σ *to states of* $\mathcal{A}$*. We also require the following relation between* $\mathcal{A}$ *and* F*:*

$$\forall u \in \Sigma^*, a \in \Sigma. \big((u \in \mathcal{A} \wedge u\,a \notin \mathcal{A}) \Leftrightarrow \exists q \in \mathcal{A}(u).(q, a) \in F \big) \qquad (\mathrm{F}\star)$$

We will often use the notation $q \xrightarrow{a} q'$ *when* $(q, a, q') \in E$*. Likewise, we use the notation* $q \overset{a}{\nrightarrow}$ *when* $(q, a) \in F$*.*

With our condition that every state of an automaton is final, the relation $q \xrightarrow{a} q'$ states that every word u "reaching" q in $\mathcal{A}$ can be extended by the symbol a, meaning that $u\,a$ is also accepted by $\mathcal{A}$. Conversely, the relation $q \overset{a}{\nrightarrow}$ states that the word $u\,a$ is not accepted. Therefore, in a FFA $(\mathcal{A}, F)$, when $q \in \mathcal{A}(u)$ and $(q, a) \in F$, then we know that the word u cannot be extended with a. In other words, the flanking relation gives information on the "frontier" of a prefix-closed language—the extreme limit over which words are no longer accepted by the automaton—hence the use of the noun *flank* to describe this class.

In the rest of the paper, we simply say that the pair $(\mathcal{A}, F)$ is *flanked* when condition (F$\star$) is met. We also say that the automaton $\mathcal{A}$ is *flankable* if there exist a flanking relation F such that $(\mathcal{A}, F)$ is flanked.

Testing if a Pair $(\mathcal{A}, F)$ **is Flanked.** We can use the traditional Rabin-Scott powerset construction to test whether F flanks the automaton $\mathcal{A} = (Q, \Sigma, E, Q_{\mathrm{in}})$. We build from $\mathcal{A}$ the "powerset automaton" $\wp(\mathcal{A})$, a DFA with alphabet Σ and with states in 2^Q (also called classes) that are the sets of states in Q reached after accepting a given word prefix; that is all the sets of the form $\mathcal{A}(u)$. The initial state of $\wp(\mathcal{A})$ is the class $\mathcal{A}(\epsilon) = Q_{\mathrm{in}}$. Finally, we have that $C \xrightarrow{a} C'$ in $\wp(\mathcal{A})$ if and only if there is $q \in C$ and $q' \in C'$ such that $q \xrightarrow{a} q'$.

Let $F^{-1}(a)$ be the set $\{q \mid q \overset{a}{\nrightarrow}\}$ of states that "forbids" the symbol a after a word accepted by $\mathcal{A}$. Then the pair $(\mathcal{A}, F)$ is flanked if, for every possible symbol $a \in \Sigma$ and for every reachable class $C \in \wp(\mathcal{A})$ we have: $C \cap F^{-1}(a) \neq \emptyset$ if and only if there is no class C' such that $C \xrightarrow{a} C'$.

This construction suggests that checking if a pair $(\mathcal{A}, F)$ is flanked should be a costly operation, that is, it should be as complex as exploring a deterministic automaton equivalent to $\mathcal{A}$. In Sect. 3 we prove that this problem is actually PSPACE-complete.

Testing if a NFA is Flankable. It is easy to show that the class of FFA includes the class of deterministic finite state automata; meaning that every DFA is flankable. If an automaton $\mathcal{A}$ is deterministic, then it is enough to choose the "flanking relation" F such that, for every state q in Q, we have $q \overset{a}{\nrightarrow}$ if and only if there are no transitions of the form $q \xrightarrow{a} q'$ in $\mathcal{A}$. DFA are a proper subset of FFA; indeed we give examples of NFA that are flankable in Sect. 5.

On the other hand, if an automaton is not deterministic, then in some cases it is not possible to define a suitable flanking relation F. For example, consider the automaton from Fig. 1 and assume, by contradiction, that we can define a

flankable relation F for this automaton. The word b is accepted by $\mathcal{A}$ but the word $b\,b$ is not, so by definition of FFA (see eq. (F$\star$)), there must be a state $q \in \mathcal{A}(b)$ such that $q \overset{b}{\not\to}$. Hence, because q_1 is the only state in $\mathcal{A}(b)$, we should necessarily have $q_1 \overset{b}{\not\to}$. However, this contradicts the fact that the word $a\,b$ is in $\mathcal{A}$, since q_1 is also in $\mathcal{A}(a)$.

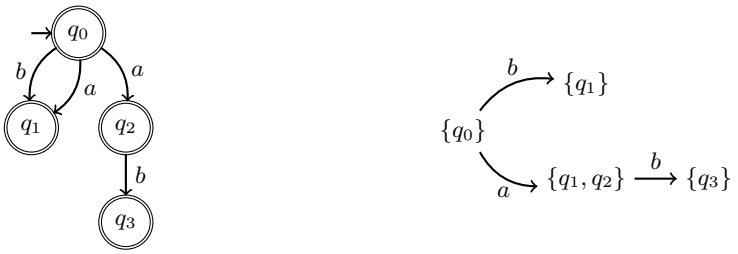


Fig. 1: A non-flankable NFA (left) and its associated Rabin-Scott powerset construction (right).

More generally, it is possible to define a necessary and sufficient condition for the existence of a flanking relation; this leads to an algorithm for testing if an automaton $\mathcal{A}$ is flankable. Let $\mathcal{A}^{-1}(a)$ denote the set of states reachable by words that can be extended by the symbol a (remember that we consider prefix-closed languages): $\mathcal{A}^{-1}(a) = \bigcup\{\mathcal{A}(u) \mid u\,a \in \mathcal{A}\}$.

It is possible to find a flanking relation F for the automaton $\mathcal{A}$ if and only if, for every word $u \in \mathcal{A}$ such that $u\,a \notin \mathcal{A}$, the set $\mathcal{A}(u) \setminus \mathcal{A}^{-1}(a)$ is not empty. Indeed, in this case, it is possible to choose F such that $(q, a) \in F$ as soon as there exists a word u with $q \in \mathcal{A}(u) \setminus \mathcal{A}^{-1}(a)$. Conversely, an automaton $\mathcal{A}$ is not flankable if we can find a word $u \in \mathcal{A}$ such that $u\,a \notin \mathcal{A}$ and $\mathcal{A}(u) \subseteq \mathcal{A}^{-1}(a)$. For example, for the automaton in Fig. 1, we have $\mathcal{A}^{-1}(b) = \{q_0, q_1, q_2\}$ while $b\,b \notin \mathcal{A}$ and $\mathcal{A}(b) = \{q_1\}$. As in the previous section, this condition can be checked directly using the powerset construction.

3 Complexity Results for Basic Problems

In this section we give some results on the complexity of basic operations over FFA. Complete proofs can be found in an extended version of this paper [2].

Theorem 1. *The universality problem for FFA is decidable in linear time.*

Proof. It is enough to prove that a FFA $(\mathcal{A}, F)$ is universal if and only if the relation F is empty; meaning that for all states $q \in Q$ it is not possible to find a symbol $a \in \Sigma$ such that $q \overset{a}{\not\to}$. As a consequence, to test whether $\mathcal{A}$ is universal, it is enough to check whether there is a state $q \in Q$ that is mapped to a non-empty set of symbols in F. Note that, given a different encoding of F, this operation could be performed in constant time. □

We can use this result to settle the complexity of testing if an automaton is flankable.

Theorem 2. *Given an automaton $\mathcal{A} = (Q, \Sigma, E, Q_{in})$ and a relation $F \in Q \times \Sigma$, the problem of testing if $(\mathcal{A}, F)$ is a flanked automaton is PSPACE-complete when there are at least two symbols in Σ.*

Proof. We can define a simple nondeterministic algorithm for testing if $(\mathcal{A}, F)$ is flanked. We recall that the relation $F^{-1}(a)$ stands for the set $\{q \mid q \overset{a}{\not\to}\}$ of states that "forbid" the symbol a. As stated in Sect. 2, to test if $(\mathcal{A}, F)$ is flanked, we need, for every symbol $a \in \Sigma$, to explore the classes C in the powerset automaton of $\mathcal{A}$ and test whether $C \overset{a}{\to} C'$ in $\wp(\mathcal{A})$ and whether $C \cap F^{-1}(a) = \emptyset$ or not. These tests can be performed using $|Q|$ bits since every class C and every set $F^{-1}(a)$ is a subset of Q. Moreover there are at most $2^{|Q|}$ classes in $\wp(\mathcal{A})$. Hence, using Savitch's theorem, the problem is in PSPACE.

On the other way, we can reduce the problem of testing the universality of a NFA $\mathcal{A}$ to the problem of testing if a pair $(\mathcal{A}, \emptyset)$ is flanked (where $\emptyset$ stands for the "empty" flanking relation over $Q \times \Sigma$). The universality problem is known to be PSPACE-hard when the alphabet Σ is of size at least 2, even if all the states of $\mathcal{A}$ are final [16]. Hence our problem is also PSPACE-hard. □

To conclude this section, we prove that the complexity of checking language inclusion between a NFA and a FFA is in polynomial time. We say that the language of $\mathcal{A}_1$ is included in $\mathcal{A}_2$, simply denoted $\mathcal{A}_1 \subseteq \mathcal{A}_2$, if all the words accepted by $\mathcal{A}_1$ are also accepted by $\mathcal{A}_2$.

Theorem 3. *Given a NFA $\mathcal{A}_1$ and a FFA $(\mathcal{A}_2, F_2)$, we can test whether $\mathcal{A}_1 \subseteq \mathcal{A}_2$ in polynomial time.*

Proof. Without loss of generality, we can assume that $\mathcal{A}_1 = (Q_1, \Sigma, E_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, E_2, I_2)$ are two NFA over the same alphabet Σ. We define a variant of the classical product construction between $\mathcal{A}_1$ and $\mathcal{A}_2$ that also takes into account the "pseudo-transitions" $q \overset{a}{\not\to}$ defined by the flanking relations.

We define the product of $\mathcal{A}_1$ and $(\mathcal{A}_2, F_2)$ as the NFA $\mathcal{A} = (Q, \Sigma, E, I)$ such that $I = I_1 \times I_2$ and $Q = (Q_1 \times Q_2) \cup \{\bot\}$. The extra state $\bot$ will be used to detect an "error condition", that is a word that is accepted by $\mathcal{A}_1$ and not by $\mathcal{A}_2$. The transition relation of $\mathcal{A}$ is such that:

- if $q_1 \overset{a}{\to} q_1'$ in $\mathcal{A}_1$ and $q_2 \overset{a}{\to} q_2'$ in $\mathcal{A}_2$ then $(q_1, q_2) \overset{a}{\to} (q_1', q_2')$ in $\mathcal{A}$;
- if $q_1 \overset{a}{\to} q_1'$ in $\mathcal{A}_1$ and $q_2 \overset{a}{\not\to}$ in $\mathcal{A}_2$ then $(q_1, q_2) \overset{a}{\to} \bot$ in $\mathcal{A}$

The result follows from the fact that $\mathcal{A}_1$ is included in $\mathcal{A}_2$ if and only if the state $\bot$ is not reachable in $\mathcal{A}$. (Actually, we can prove that any word u such that $\bot \in \mathcal{A}(u)$ is a word accepted by $\mathcal{A}_1$ and not by $\mathcal{A}_2$.) Since we cannot generate more than $|Q_1| \cdot |Q_2|$ reachable states in $\mathcal{A}$ before finding the error $\bot$, this algorithm is solvable in polynomial time. □

4 Closure Properties of Flanked Automata

In this section, we study how to compute the composition of flanked automata. We prove that the class of FFA is closed by language intersection and by the "intersection adjunct", also called quotient. On a negative side, we show that the class is not closed by non-injective relabeling.

We consider the problem of computing a flanked automaton accepting the intersection of two prefix-closed, regular languages. More precisely, given two FFA $(\mathcal{A}_1, F_1)$ and $(\mathcal{A}_2, F_2)$, we want to compute a FFA $(\mathcal{A}, F)$ that recognizes the set of words accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted simply $\mathcal{A}_1 \cap \mathcal{A}_2$.

Theorem 4. *Given two FFA $(\mathcal{A}_1, F_1)$ and $(\mathcal{A}_2, F_2)$, we can compute a FFA $(\mathcal{A}, F)$ for the language $\mathcal{A}_1 \cap \mathcal{A}_2$ in polynomial time. The NFA $\mathcal{A}$ has size less than $|\mathcal{A}_1| \cdot |\mathcal{A}_2|$.*

Proof. We define a classical product construction between $\mathcal{A}_1$ and $\mathcal{A}_2$ and show how to extend this composition on the flanking relations. We assume that $\mathcal{A}_i$ is an automaton (Q_i, Σ, E_i, I_i) for $i \in \{1, 2\}$.

The automaton $\mathcal{A} = (Q, \Sigma, E, I)$ is defined as the synchronous product of $\mathcal{A}_1$ and $\mathcal{A}_2$, that is: $Q = Q_1 \times Q_2$; $I = I_1 \times I_2$; and the transition relation is such that $(q_1, q_2) \xrightarrow{a} (q_1', q_2')$ in $\mathcal{A}$ if both $q_1 \xrightarrow{a} q_1'$ in $\mathcal{A}_1$ and $q_2 \xrightarrow{a} q_2'$ in $\mathcal{A}_2$. It is a standard result that $\mathcal{A}$ accepts the language $\mathcal{A}_1 \cap \mathcal{A}_2$.

The *flanking relation* F is defined as follows: for each accessible state $(q_1, q_2) \in Q$, we have $(q_1, q_2) \xrightarrow{a}$ if and only if $q_1 \xrightarrow{a}$ in $\mathcal{A}_1$ or $q_2 \xrightarrow{a}$ in $\mathcal{A}_2$. What is left to prove is that $(\mathcal{A}, F)$ is flanked, that is, we show that condition (F$\star$) is correct:

- assume u is accepted by $\mathcal{A}$ and $u\,a$ is not; then there is a state $q = (q_1, q_2)$ in $\mathcal{A}$ such that $q \in \mathcal{A}(u)$ and $(q, a) \in F$. By definition of $\mathcal{A}$, we have that u is accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$, while the word $u\,a$ is not accepted by at least one of them. Assume that $u\,a$ is not accepted by $\mathcal{A}_1$. Since F_1 is a flanking relation for $\mathcal{A}_1$, we have by equation (F$\star$) that there is at least one state $q_1 \in \mathcal{A}_1$ such that $(q_1, a) \in F_1$; and therefore $(q, a) \in F$, as required.
- assume there is a reachable state $q = (q_1, q_2)$ in $\mathcal{A}$ such that $q \in \mathcal{A}(u)$ and $(q, a) \in F$; then u is accepted by $\mathcal{A}$. We show, by contradiction, that $u\,a$ cannot be accepted by $\mathcal{A}$, that is $u\,a \notin \mathcal{A}_1 \cap \mathcal{A}_2$. Indeed, if so, then $u\,a$ will be accepted both by $\mathcal{A}_1$ and $\mathcal{A}_2$ and therefore we will have $(q_1, a) \notin F_1$ and $(q_2, a) \notin F_2$, which contradicts the fact that $(q, a) \in F$. □

Next we consider the adjunct of the intersection operation, denoted $\mathcal{A}_1/\mathcal{A}_2$. This operation, also called *quotient*, is defined as the biggest prefix-closed language X such that $\mathcal{A}_2 \cap X \subseteq \mathcal{A}_1$. Informally, X is the solution to the following question: what is the biggest set of words x such that x is either accepted by $\mathcal{A}_1$ or not accepted by $\mathcal{A}_2$. Therefore the language $\mathcal{A}_1/\mathcal{A}_2$ is always defined (and not empty, since it contains at least the empty word ϵ). Actually, the quotient can be interpreted as the biggest prefix-closed language included in the set $\mathcal{L}_1 \cup \bar{\mathcal{L}}_2$, where $\mathcal{L}_1$ is the language accepted by $\mathcal{A}_1$ and $\bar{\mathcal{L}}_2$ is the complement of the language of $\mathcal{A}_2$.

The quotient operation can also be defined by the following two axioms:

$$\begin{array}{ll} \text{(Ax1)} & \mathcal{A}_2 \cap (\mathcal{A}_1/\mathcal{A}_2) \subseteq \mathcal{A}_1 \\ \text{(Ax2)} & \forall X.\, \mathcal{A}_2 \cap X \subseteq \mathcal{A}_1 \Rightarrow X \subseteq \mathcal{A}_1/\mathcal{A}_2 \end{array}$$

The quotient operation is useful when trying to solve *language equations problems* [22] and has applications in the domain of system verification and synthesis. For instance, we can find a similar operation in the contract framework of Benveniste et al. [6] or in the contract framework of Bauer et al. [4].

Our results on FFA can be used for the simplest instantiation of these frameworks that considers a simple trace-based semantics where the behavior of systems is given as a regular set of words; composition is language intersection; and implementation conformance is language inclusion. Our work was motivated by the fact that there are no known efficient methods to compute the quotient. Indeed, to the best of our knowledge, all the approaches rely on the determinization of NFA, which is very expensive in practice [18, 22].

Our definitions of quotient could be easily extended to replace language intersection by synchronous product and to take into account the addition of modalities [18].

Theorem 5. *Given two FFA $(\mathcal{A}_1, F_1)$ and $(\mathcal{A}_2, F_2)$, we can compute a FFA $(\mathcal{A}, F)$ for the quotient language $\mathcal{A}_1/\mathcal{A}_2$ in polynomial time. The NFA $\mathcal{A}$ has size less than $|\mathcal{A}_1| \cdot |\mathcal{A}_2| + 1$*

Proof. Without loss of generality, we can assume that $\mathcal{A}_1 = (Q_1, \Sigma, E_1, I_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, E_2, I_2)$ are two NFA over the same alphabet Σ. Like in the construction for testing language inclusion, we define a variant of the classical product construction between $\mathcal{A}_1$ and $\mathcal{A}_2$ that also takes into account the flanking relations.

We define the product of $(\mathcal{A}_1, F_1)$ and $(\mathcal{A}_2, F_2)$ as the NFA $\mathcal{A} = (Q, \Sigma, E, I)$ such that $I = I_1 \times I_2$ and $Q = (Q_1 \times Q_2) \cup \{\top\}$. The extra state $\top$ will be used as a sink state from which every suffix can be accepted. The transition relation of $\mathcal{A}$ is such that:

- if $q_1 \xrightarrow{a} q_1'$ in $\mathcal{A}_1$ and $q_2 \xrightarrow{a} q_2'$ in $\mathcal{A}_2$ then $(q_1, q_2) \xrightarrow{a} (q_1', q_2')$ in $\mathcal{A}$;
- if $q_2 \overset{a}{\nrightarrow}$ in $\mathcal{A}_2$ then $(q_1, q_2) \xrightarrow{a} \top$ in $\mathcal{A}$ for all states $q_1 \in Q_1$
- $\top \xrightarrow{a} \top$ for every $a \in \Sigma$

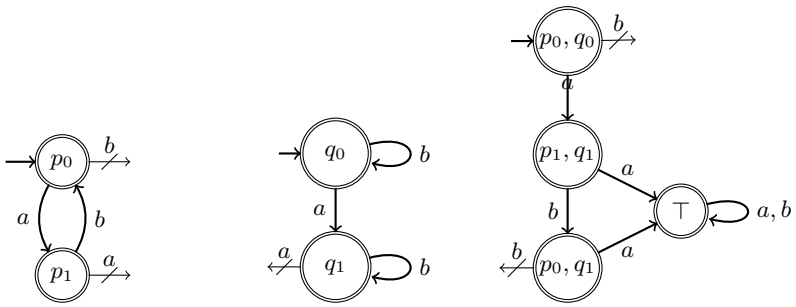
Note that we do not have a transition rule for the case where $q_1 \overset{a}{\nrightarrow}$ in $\mathcal{A}_1$ and $q_2 \xrightarrow{a} q_2'$; this models the fact that a word "that can be extended" in $\mathcal{A}_2$ but not in $\mathcal{A}_1$ cannot be in the quotient $\mathcal{A}_1/\mathcal{A}_2$. It is not difficult to show that $\mathcal{A}$ accepts the language $\mathcal{A}_1/\mathcal{A}_2$. We give an example of the construction in Fig. 2.

Next we show that $\mathcal{A}$ is flankable and define a suitable flanking relation. Let F be the relation in $Q \times \Sigma$ such that $(q_1, q_2) \overset{a}{\nrightarrow}$ if and only if $q_1 \overset{a}{\nrightarrow}$ in F_1 and $q_2 \xrightarrow{a} q_2'$ in $\mathcal{A}_2$. That is, the symbol a is forbidden exactly in the case that was ruled out in the transition relation of $\mathcal{A}$. What is left to prove is that $(\mathcal{A}, F)$ is flanked, that is, we show that condition (F$\star$) is correct:

– Assume u is accepted by $\mathcal{A}$ and $u\,a$ is not. Since $u\,a$ is not accepted, it must be the case that $q \neq \top$. Therefore there is a state $q = (q_1, q_2)$ in $\mathcal{A}$ such that $q_1 \in \mathcal{A}_1(u)$ and $q_2 \in \mathcal{A}_2(u)$. Also, since there is no transition with label a from q, then necessarily $q_1 \overset{a}{\not\to}$ in $\mathcal{A}_1$ and $q_2 \overset{a}{\to} q_2'$. This is exactly the case where $(q, a) \in F$, as required.
– Assume there is a reachable state q in $\mathcal{A}$ such that $q \in \mathcal{A}(u)$ and $(q, a) \in F$. Since $(q, a) \in F$, we have $q \neq \top$ and therefore $q = (q_1, q_2)$ with $q_1 \in \mathcal{A}_1(u)$, $q_1 \overset{a}{\not\to}$ in F_1, $q_2 \in \mathcal{A}_2(u)$ and $q_2 \overset{a}{\to}$ in F_2. Next, we show by contradiction that $u\,a$ cannot be accepted by $\mathcal{A}$. Indeed, if it was the case, then we would have either $q_1 \overset{a}{\to}$ in F_1 or both $u\,a \notin \mathcal{A}_1$ and $u\,a \notin \mathcal{A}_2$.

□

We give an example of the construction of the "quotient" FFA in Fig. 2. If we look more closely at the construction used in Theorem 5 that defines an automaton for the quotient of two FFA $(\mathcal{A}_1, F_1)$ and $(\mathcal{A}_2, F_2)$, we see that the flanking relation F_1 is used only to compute the flanking relation of the result. Therefore, as a corollary, it is not difficult to prove that we can use the same construction to build a quotient automaton for $\mathcal{A}_1/\mathcal{A}_2$ from an arbitrary NFA $\mathcal{A}_1$ and a FFA $(\mathcal{A}_2, F_2)$. However the resulting automaton may not be flankable.



(a) $\mathcal{A}_1$ **(b)** $\mathcal{A}_2$ **(c)** $\mathcal{A}_1/\mathcal{A}_2$

Fig. 2: Construction for the quotient of two FFA.

We can also prove that flankability is preserved by language union (see [2]): given two FFA $(\mathcal{A}_1, F_1)$ and $(\mathcal{A}_2, F_2)$, we can compute a FFA $(\mathcal{A}, F)$ that recognizes the set of words accepted by $\mathcal{A}_1$ or by $\mathcal{A}_2$, denoted $\mathcal{A}_1 \cup \mathcal{A}_2$. (Operations corresponding to the Kleene star closure or to the adjunct of the union are not interesting in our case.)

Even though the class of FFA enjoys interesting closure properties, there are operations that, when applied to a FFA, may produce a result that is not flankable. This is for example the case with "(non-injective) relabeling", that is the operation of applying a substitution over the symbols of an automaton. The same can be observed if we consider an erasure operation, in which we can replace all transitions with a given symbol by an ϵ-transition. Informally, it

appears that the property flankable can be lost when applying an operation that increases the non-determinism of the transition relation.

We can prove this result by exhibiting a simple counterexample, see the automaton in Fig. 3. This automaton with alphabet $\Sigma = \{a, b, c\}$ is deterministic, so we can easily define an associated flanking relation. For example we can choose $F = \{(q_1, a), (q_1, b), (q_1, c), (q_2, a), (q_2, c), (q_3, a), (q_3, b), (q_3, c)\}$. However, if we substitute the symbol c with a, we obtain the non-flankable automaton described in Sect. 2 (see Fig. 1).

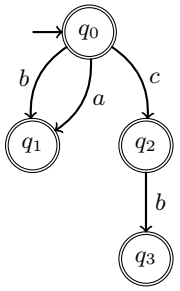


Fig. 3: Example of a FFA not flankable after relabeling c to a.

5 Succinctness of Flanked Automata

In this section we show that a flankable automaton can be exponentially more succinct than its equivalent minimal DFA. This is done by defining a language over an alphabet of size $2\,n$ that can be accepted by a linear size FFA but that corresponds to a minimal DFA with an exponential number of states. This example is due to Colcombet.

At first sight, this result may seem quite counterintuitive. Indeed, even if a flanked automata is built from a NFA, the combination of the automaton and the flanking relation contains enough information to "encode" both a language and its complement. This explains the good complexity results on testing language inclusion for example. Therefore we could expect worse results concerning the relative size of a FFA and an equivalent DFA.

Theorem 6. *For every integer n, we can find a FFA $(\mathcal{A}_n, F)$ such that $\mathcal{A}_n$ has $2\,n+2$ states and that the language of $\mathcal{A}_n$ cannot be accepted by a DFA with less than 2^n states.*

Proof. We consider two alphabets with n symbols: $\Pi_n = \{1, \ldots, n\}$ and $\Theta_n = \{\sharp_1, \sharp_2, \ldots, \sharp_n\}$. We define the language L_n over the alphabet $\Pi_n \cup \Theta_n$ as the smallest set of words such that:

- all words in Π_n^* are in L_n, that is all the words that do not contain a symbol of the kind $\sharp_i$;

– a word of the form $(u\,\sharp_i)$ is in L_n if and only if u is a word of Π_n^* that contains at least one occurrence of the symbol i. That is L_n contains all the words of the form $\Pi_n^* \cdot i \cdot \Pi_n^* \cdot \sharp_i$ for all $i \in 1..n$. We denote L_n^i the regular language consisting of the words of the form $\Pi_n^* \cdot i \cdot \Pi_n^* \cdot \sharp_i$.

Clearly the language L_n is the union of $n+1$ regular languages; $L = \Pi_n^* \cup L_n^1 \cup \cdots \cup L_n^n$. It is also easy to prove that L_n is prefix-closed, since the set of prefixes of the words in L_n^i is exactly Π_n^* for all $i \in 1..n$.

A DFA accepting the language L_n must have at least 2^n different states. Indeed it must be able to record the subset of symbols in Π_n that have already been seen before accepting $\sharp_i$ as a final symbol; to accept a word of the form $u\,\sharp_i$ the DFA must know whether i has been seen in u for all possible $i \in 1..n$.

Next we define a flankable NFA $\mathcal{A}_n = (Q_n, \Pi_n \cup \Theta_n, E_n, \{p\})$ with $2\,n+2$ states that can recognize the language L_n. We give an example of the construction in Fig. 4 for the case $n = 3$. The NFA $\mathcal{A}_n$ has a single initial state, p, and a single sink state (a state without outgoing transitions), r. The set Q_n also contains two states, p_i and q_i, for every symbol i in Π.

The transition relation E_n is the smallest relation that contains the following triplets for all $i \in 1..n$:

– the 3 transitions $p \xrightarrow{i} q_i$; $p_i \xrightarrow{i} q_i$; and $q_i \xrightarrow{i} q_i$;
– for every index $j \neq i$, the 3 transitions $p \xrightarrow{j} p_i$; $p_i \xrightarrow{j} p_i$; and $q_i \xrightarrow{j} q_i$;
– and the transition $q_i \xrightarrow{\sharp_i} r$.

Intuitively, a transition from p to p_i or q_i will select non-deterministically which final symbol $\sharp_i$ is expected at the end of the word (which sub-language L_n^i we try to recognize). Once a symbol in Θ has been seen—in one of the transition of the kind $q_i \xrightarrow{\sharp_i} r$—the automaton is stuck on the state r. It is therefore easy to prove that $\mathcal{A}_n$ accepts the union of the languages L_n^i and their prefixes.

Finally, the NFA $\mathcal{A}_n$ is flankable. It is enough to choose, for the flanking relation, the smallest relation on $Q \times \Theta_n$ such that $p_i \overset{\sharp_i}{\nrightarrow}$ and $p \overset{\sharp_i}{\nrightarrow}$ for all $i \in 1..n$; and such that $r \overset{a}{\nrightarrow}$ for all the symbols $a \in \Pi_n \cup \Theta_n$. Indeed, it is not possible to accept the symbol $\sharp_i$ from the initial state, p, or from a word that can reach p_i; that is, it is not possible to extend a word without any occurrence of the symbol i with the symbol $\sharp_i$. Also, it is not possible to extend a word that can reach the state r in $\mathcal{A}_n$. It is easy to prove that this covers all the possible words not accepted by $\mathcal{A}_n$. □

6 A Simple Use Case for FFA

In this section, we study a simple example related to controller synthesis in a component-based system. We use our approach to compute a controller, G, for a system obtained from the parallel composition of n copies of the same components: $(S_1 \parallel \cdots \parallel S_n)$. The architecture of this system is given in Fig. 5. We use this example to study the performance of our approach when compared to traditional techniques.

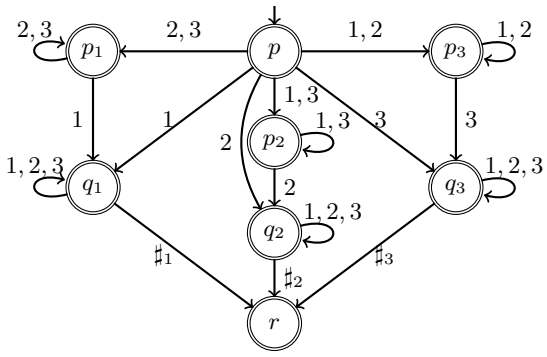


Fig. 4: Flankable NFA for the language L_3.

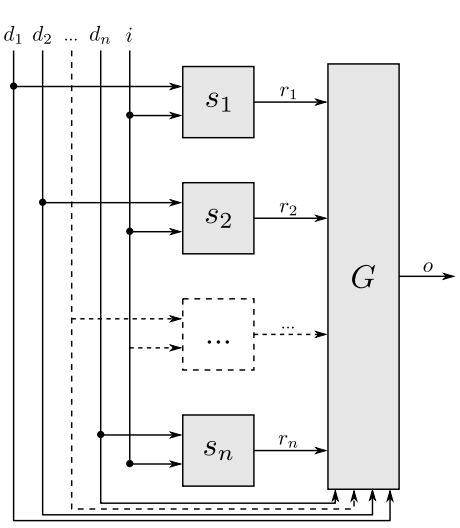
Each component S_k can receive messages from two different channels: a public channel i, shared by everyone, which represents the main input channel of the whole system; and a private channel d_k that can be used to disable the component S_k. While the component is active, it can emit a message on its output channel, r_k, after receiving the two messages i_1 and i_2, in this order, over the channel i. Once disabled, the component does not interact with its environment. The overall behavior of the system is given by the automaton in Fig. 5-(c). We expect the system to emit a message on channel o when it receives two messages on channel i. Even though this behavior is very simple, the task of the controller G is made difficult by the fact that it cannot listen on the channel i. The component G can only observe the output of the components on the channels r_i, for $i \in 1..n$, and the disabling messages.

By definition, the semantics of the controller G is the biggest solution (for G) of the language equation $(S_1 \parallel \cdots \parallel S_n \parallel G) \subseteq A$, hence: $G = A/(S_1 \parallel \cdots \parallel S_n)$.

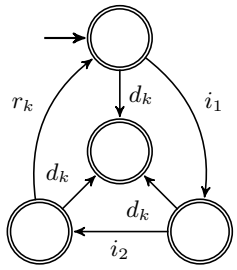
We have used this example to compare the time necessary to compute G with two approaches; first using the tool MoTraS [17], then using a prototype implementation based on FFA. MoTraS is a tool for modal transition systems that implements all the standard operations for specification theories, such as language quotient. The results are given in the table below, where we give the performance when varying the number of components in the system (the value of the parameter n). These results were obtained on a desktop computer with 8 GB of RAM.

n	7	8	9	100	200	500	1000	2000
MoTraS								
times (s)	8 s	27 s	190 s	—	—	—	—	—
(memory)	(750 MB)	(1.5 GB)	(2 GB)	—	—	—	—	—
FFA								
time (s)	<0.01 s	<0.01 s	<0.01 s	0.05 s	0.2 s	1.5 s	5.8 s	30 s
(memory)	(1.3 MB)	(1.3 MB)	(1.3 MB)	(2.4 MB)	(3.5 MB)	(6.5 MB)	(13 MB)	(23 MB)

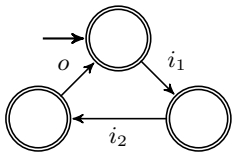
We observe that it is not possible to compute G for values of n greater than 10 using a classical approach. These results are similar to what we obtained



(a) architecture of the system



(b) specification of the component S_k



(c) global specification A

Fig. 5: Architecture and specification of a simple voting network.

using our own prototype implementation based on DFA. On the opposite, when we use flanked automata, we are able to compute the quotient for up to several thousand components.

7 Related Work

We can identify two main categories of related work. First, there is a large body of work addressing the problem of solving language equations by computing the quotient of two specifications. Then, we consider works concerned with finding classes of finite state automata with good complexity properties.

Work on equation solving and quotient. Villa et al. [23, 22] consider language equations for systems described using NFA. Actions labeling the transitions can either be inputs, if they stem from the system environment, or outputs, when they originate from the system. Composition may correspond to the synchronous product with internalization of synchronized actions. In any case, the proposed algorithms start with a determinization step, which is very expensive in practice.

In control theory [19], the plant is in most cases a DFA whose transitions can be labeled by actions that are either declared as *uncontrollable* (the controller cannot forbid them) or *unobservable* (the controller cannot *see* their occurrence). Partial observation naturally led to consider nondeterministic plants [13].

A quotient operator has also been defined for modal specifications by one of the authors [18]. In this setting, we can specify that it *may* or it *must* always

be the case that a trace can be extended with a certain action. The size of the quotient is polynomial when modal specifications are deterministic, but there is an exponential blow-up when this assumption is relaxed [5]. Quotients for extensions of modal specifications to capture timed and quantitative languages have also been recently considered [9, 3, 10].

Work on Finite State Automata. Several works have tried to find classes of finite automata that retain the same complexity as DFA on some operations while still being more succinct than the minimal DFA. One such example is the class of Unambiguous Finite Automata (UFA) [20, 21]. Informally, an UFA is a finite state automaton such that, if a word is accepted, then there is a unique run which witnesses this fact, that is a unique sequence of states visited when accepting the word. Like with DFA, the problems of universality and inclusion for UFA is in polynomial-time. Unfortunately, UFA are difficult to complement. (Actually, finding the exact complexity of complementation for UFA is still an open problem [8].) Therefore they are not a good choice for computing quotients.

Another problem lies in the use of UFA for prefix-closed languages. In this paper, we restrict our study to automata recognizing prefix-closed languages. More precisely, we assume that all the states of the automaton are final (which is equivalent). This restriction is very common when using NFA for the purpose of system verification. For instance, Kripke structures used in model-checking algorithms are often interpreted as finite state automata where all states are final. It is easy to see that, with this restriction, an UFA is necessarily deterministic.

In the context of automata on infinite words, we can also mention the *safety automata* of Isaak and Löding [14]. A safety (or looping) automaton can be viewed as a Büchi automaton in which all states are accepting, except for possibly one rejecting sink state. For unambiguous safety automata, the problems of inclusion, equivalence, and universality can be solved in polynomial time. We show similar complexity results for our class of automata (on finite words). Moreover, a FFA can also be described, superficially, as a safety automaton without the Büchi acceptance condition. Nonetheless, without the use of the flanking relation, it is not clear how to define the quotient operation for safety automata, especially if we want a compositional construction that does not involve determinization.

It should be stressed that our problem is not made simpler by the choice to restrict to prefix-closed languages. Indeed, all the classical complexity results on NFA are still valid in this context. For instance, given a NFA $\mathcal{A}$ with all its states final, checking the universality of $\mathcal{A}$ is PSPACE-hard [16]. Likewise for the minimization problem. Indeed, there are examples of NFA with n states, all final, such that the minimal equivalent DFA has 2^n states. We provide such an example in Sect. 5 of this paper. Intuitively, it is always possible to view a regular language L, over the alphabet Σ, as the prefix closed-language containing words of the form $w\sharp$, where w is in L and $\sharp$ is some new (terminal) symbol not used in Σ.

8 Conclusion

We define a new subclass of NFA for prefix-closed languages called flanked automata. Intuitively, a FFA $(\mathcal{A}, F)$ is a simple extension of NFA where we add in the relation F extra information that can be used to check (non-deterministically) whether a word is not accepted by $\mathcal{A}$. Hence a FFA can be used both to test whether a word is in the language associated with $\mathcal{A}$ or in its complement. As a consequence, we obtain good complexity results for several interesting problems such as universality and language inclusion. This idea of adding extra information to encode both a language and its complement seems to be new. It is also quite different from existing approaches used to define subclasses of NFA with good complexity properties, like unambiguity for example.

Our work could be extended in several ways. First, we have implemented all our proposed algorithms and constructions and have found that—for several examples coming from the system verification domain—it was often easy to define a flanking relation for a given NFA (even though we showed in Sect. 2 that it is not always possible). More experimental work is still needed, and in particular the definition of a good set of benchmarks.

Next, we have used the powerset construction multiple times in our definitions. Most particularly as a way to test if a FFA is flanked or if a NFA is flankable. Other constructions used to check language inclusion or simulation between NFA could be useful in this context like, for example, the antichain-based method [1].

Finally, we still do not know how to compute a "succinct" flanked automaton from a NFA that is not flankable. At the moment, our only solution is to compute a minimal equivalent DFA (since DFA are always flankable). While it could be possible to subsequently simplify the DFA—which is known to be computationally hard [15]—it would be interesting to have a more direct construction.

Acknowledgments. We thank Denis Kuperberg, Thomas Colcombet, and Jean-Eric Pin for providing their expertise and insight and for suggesting the example that led to the proof of Theorem 6.

References

1. Abdulla, P.A., Chen, Y.F., Holik, L., Mayr, R., Vojnar, T.: When simulation meets antichains. In: TACAS. LNCS, vol. 6015. Springer (2010)
2. Avellaneda, F., Dal Zilio, S., Raclet, J.: On the complexity of flanked finite state automata. CoRR abs/1509.06501 (2015)
3. Bauer, S.S., Fahrenberg, U., Juhl, L., Larsen, K.G., Legay, A., Thrane, C.R.: Weighted modal transition systems. Formal Methods in System Design 42(2), 193–220 (2013)
4. Bauer, S.S., David, A., Hennicker, R., Guldstrand Larsen, K., Legay, A., Nyman, U., Wasowski, A.: Moving from specifications to contracts in component-based design. In: FASE. LNCS, vol. 7212, pp. 43–58. Springer (2012)

5. Beneš, N., Delahaye, B., Fahrenberg, U., Křetínskỳ, J., Legay, A.: Hennessy-milner logic with greatest fixed points as a complete behavioural specification theory. In: CONCUR 2013–Concurrency Theory, pp. 76–90. Springer (2013)
6. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: Formal Methods for Components and Objects. LNCS, vol. 5382, pp. 200–225. Springer (2008)
7. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T., Larsen, K.G.: Contracts for system design (2012)
8. Colcombet, T.: Forms of Determinism for Automata. In: Symposium on Theoretical Aspects of Computer Science (STACS). vol. 14, pp. 1–23 (2012)
9. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: A complete specification theory for real-time systems. In: HSCC. pp. 91–100. ACM (2010)
10. Fahrenberg, U., Ketnsk, J., Legay, A., Traonouez, L.M.: Compositionality for quantitative specifications. In: Formal Aspects of Component Software. LNCS, vol. 8997, pp. 306–324. Springer International Publishing (2015)
11. Gierds, C., Mooij, A.J., Wolf, K.: Reducing adapter synthesis to controller synthesis. Services Computing, IEEE Transactions on 5(1), 72–85 (2012)
12. Henzinger, T.A., Qadeer, S., Rajamani, S.K.: You assume, we guarantee: Methodology and case studies. In: Computer Aided Verification, LNCS, vol. 1427, pp. 440–451. Springer (1998)
13. Heymann, M., Lin, F.: Discrete-event control of nondeterministic systems. Automatic Control, IEEE Transactions on 43(1), 3–17 (1998)
14. Isaak, D., Löding, C.: Efficient inclusion testing for simple classes of unambiguous ω-automata. Inf. Process. Lett. 112(14-15) (2012)
15. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM Journal on Computing 22(6), 1117–1141 (1993)
16. Kao, J.Y., Rampersad, N., Shallit, J.: On NFAs where all states are final, initial, or both. Theoretical Computer Science 410(4749), 5010–5021 (2009)
17. Kretínský, J., Sickert, S.: Motras: A tool for modal transition systems and their extensions. In: 11th International Symposium Automated Technology for Verification and Analysis (ATVA) (2013)
18. Raclet, J.B.: Residual for component specifications. ENTCS 215, 93–110 (2008), workshop on Formal Aspects of Component Software (FACS)
19. Ramadge, P., Wonham, W.: The control of discrete event systems. Proceedings of the IEEE 77(1), 81–98 (1989)
20. Schmidt, E.M.: Succinctness of Description of Context-Free, Regular and Unambiguous Languages. Ph.D. thesis, Cornell University (1978)
21. Stearns, R.E., Hunt III, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM Journal on Computing 14(3), 598–611 (1985)
22. Villa, T., Petrenko, A., Yevtushenko, N., Mishchenko, A., Brayton, R.: Component-based design by solving language equations. IEEE PP(99), 1–16 (2015)
23. Villa, T., Yevtushenko, N., Brayton, R.K., Mishchenko, A., Petrenko, A., Sangiovanni-Vincentelli, A.: The unknown component problem: theory and applications. Springer (2011)
24. Yellin, D., Strom, R.: Protocol specifications and component adaptors. ACM Transactions on Programming Languages and Systems (TOPLAS) 19(2), 292–333 (1997)