A Pattern Logic for Automata with Outputs

Emmanuel Filiot, Nicolas Mazzocchi[(✉)], and Jean-François Raskin

Université libre de Bruxelles, Brussels, Belgium
nicolas.mazzocchi@ulb.ac.be

Abstract. We introduce a logic to express structural properties of automata with string inputs and, possibly, outputs in some monoid. In this logic, the set of predicates talking about the output values is parametric, and we provide sufficient conditions on the predicates under which the model-checking problem is decidable. We then consider three particular automata models (finite automata, transducers and automata weighted by integers – sum-automata –) and instantiate the generic logic for each of them. We give tight complexity results for the three logics and the model-checking problem, depending on whether the formula is fixed or not. We study the expressiveness of our logics by expressing classical structural patterns characterising for instance finite ambiguity and polynomial ambiguity in the case of finite automata, determinisability and finite-valuedness in the case of transducers and sum-automata. Consequently to our complexity results, we directly obtain that these classical properties can be decided in PTime.

1 Introduction

Motivations. An important aspect of automata theory is the definition of automata subclasses with particular properties, of algorithmic interest for instance. As an example, the inclusion problem for non-deterministic finite automata is PSpace-c but becomes PTime if the automata are k-ambiguous for a fixed k [21].

By automata theory, we mean automata in the general sense of finite state machines processing finite words. This includes what we call *automata with outputs*, which may also produce output values in a fixed monoid $\mathbb{M} = (D, \oplus, \mathbb{0})$. In

We warmly thank the anonymous reviewers for their helpful comments, and Ismaël Jecker for spotting a bug in a preliminary version of the paper. E. Filiot is a research associate of F.R.S.-FNRS. He is supported by the ARC Project Transform Fédération Wallonie-Bruxelles and the FNRS CDR project J013116F. N. Mazzocchi is a PhD student funded by a FRIA fellowship from the F.R.S.-FNRS. J.-F. Raskin is supported by an ERC Starting Grant (279499: inVEST), by the ARC project – Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond – funded by the Fédération Wallonie-Bruxelles, and by a Professeur Francqui de Recherche grant awarded by the Francqui Fondation.

M. Hoshi and S. Seki (Eds.): DLT 2018, LNCS 11088, pp. 304–317, 2018.
https://doi.org/10.1007/978-3-319-98654-8_25

such an automaton, the transitions are extended with an (output) value in D, and the value of an accepting path is the sum (for $\oplus$) of all the values occurring along its transitions. Automata over finite words in Λ^* and with outputs in $\mathbb{M}$ define subsets of $\Lambda^* \times D$ as follows: to any input word $w \in \Lambda^*$, we associate the set of values of all the accepting paths on w. For example, transducers are automata with outputs in a free monoid: they process input words and produce output words and therefore define binary relations of finite words [15].

The many decidability properties of finite automata do not carry over to transducers, and many restrictions have been defined in the literature to recover decidability, or just to define subclasses relevant to particular applications. The inclusion problem for transducer is undecidable [13], but decidable for *finite-valued* transducers [23]. Another well-known subclass is that of the *determinisable* transducers [5], defining sequential functions of words. Finite-valuedness and determinisability are two properties decidable in PTime, i.e., it is decidable in PTime, given a transducer, whether it is finite-valued (resp. determinisable). As a second example of automata with outputs, we also consider sum-automata, i.e. automata with outputs in $(\mathbb{Z}, +, 0)$, which defines relations from words to $\mathbb{Z}$. Properties such as functionality, determinisability, and k-valuedness (for a fixed k) are decidable in PTime for sum-automata [10,11].

In our experience, it is quite often the case that deciding a subclass goes in two steps: (1) define a characterisation of the subclass through a "simple" pattern, (2) show how to decide the existence of a such a pattern. For instance, the determinisable transducers have been characterised via the so called *twinning property* [4,6,24], which, said briefly, asks that the output words produced by any two different paths on input words of the form uv^n cannot differ unboundedly when n grows, with a suitable definition of "differ". Quite often, the most difficult part is step (1) and step (2) is technical but less difficult to achieve, as long as we do not seek for optimal complexity bounds (by this we mean that PTime is good enough, and obtaining the best polynomial degree is not the objective). We even noticed that in transducer theory, even though step (2) share common techniques (reduction to emptiness of reversal-bounded counter machines for instance), the algorithms are often ad-hoc to the particular subclass considered. Here is a non-exhaustive list of subclasses of transducers which are decidable in PTime: determinisable transducers [1,4–7,24], functional transducers [4,5], k-sequential transducers (for a fixed k) [8], multi-sequential transducers [7,16], k-valued transducers (for a fixed k) [14], finite-valued transducers [18,23]. Our goal in this paper is to define a common tool for step (2), i.e., define a generic way of deciding a subclass characterised through a structural pattern. More precisely, we want to define logics, tailored to particular monoids $\mathbb{M}$, able to express properties of automata with outputs in $\mathbb{M}$, such that model-checking these properties on given automata can be done in PTime.

Contributions. We define a general logic, denoted $\mathsf{PL}[\mathcal{O}]$ for "pattern logic", to express properties of automata with outputs in a fixed monoid $\mathbb{M} = (D, \oplus, \mathbb{0})$. This logic is parameterised by a set of predicates $\mathcal{O}$ interpreted on D. We first give sufficient conditions under which the problem of model-checking an automaton with outputs in $\mathbb{M}$ against a formula in this logic is decidable. Briefly, these conditions require the existence of a machine model accepting tuples of runs which satisfy the atomic predicates of the logic, is closed under union and intersection, and has decidable emptiness problem.

Then, we study three particular classes of automata with outputs: finite automata (which can be seen as automata with outputs in a trivial monoid with a single element), transducers (automata with outputs in a free monoid), and sum-automata (automata with outputs in $(\mathbb{Z}, +, 0)$). For each of them, we define particular logics, called $\mathsf{PL}_{\mathrm{NFA}}$, $\mathsf{PL}_{\mathrm{Trans}}$ and $\mathsf{PL}_{\mathrm{Sum}}$ to express properties of automata with outputs in these particular monoids. Formulas in these logics have the following form:

$$\exists\pi_1 : p_1 \xrightarrow{u_1|v_1} q_1, \ldots, \exists\pi_n : p_n \xrightarrow{u_n|v_n} q_n, \mathcal{C}$$

where the π_i are path variables, the p_i, q_i are state variables, the u_i are (input) word variables and the v_i are output value variables (interpreted in D). The subformula $\mathcal{C}$ is a quantifier free Boolean combinations of constraints talking about states, paths, input words and output values. Such a formula expresses the fact that there exists a path π_1 from some state p_1 to some state q_1, over some input word u_1, producing some value v_1, some path π_2 etc. such that they all satisfy the constraints in $\mathcal{C}$. In the three logics, paths can be tested for equality. Input words can be compared with the prefix relation, w.r.t. their length, and their membership to a regular language be tested. States can be compared for equality, and it can be expressed whether they are initial or final.

The predicates we take for the output values depends on the monoids. For transducers, output words can be compared with the non-prefix relation (and by derivation $\neq$), a predicate which cannot be negated (otherwise model-checking becomes undecidable), and can also be compared with respect to their length, and membership to a regular language can be tested. For sum-automata, the output values can be compared with $<$ (and by derivation $=, \neq, \leq$). As an example, a transducer (resp. sum-automaton) is *not* $(n-1)$-valued iff it satisfies the following $\mathsf{PL}_{\mathrm{Trans}}$-formula (resp. $\mathsf{PL}_{\mathrm{Sum}}$-formula):

$$\exists\pi_1 : p_1 \xrightarrow{u|v_1} q_1, \ldots, \exists\pi_n : p_n \xrightarrow{u|v_n} q_n, \bigwedge_{i=1}^{n} \mathsf{init}(p_i) \wedge \mathsf{final}(q_i) \wedge \bigwedge_{1\leq i<j\leq n} v_i \neq v_j.$$

For the three logics, we show that deciding whether a given automaton satisfies a given formula is PSPACE-C. When the formula is *fixed*, the model-checking problem becomes NLOGSPACE-C for $\mathsf{PL}_{\mathrm{NFA}}$ and $\mathsf{PL}_{\mathrm{Trans}}$, and NP-C for $\mathsf{PL}_{\mathrm{Sum}}$. If output values can only be compared via disequality $\neq$ (which cannot be negated), then $\mathsf{PL}_{\mathrm{Sum}}$ admits PTIME model-checking. We show that many of the properties from the literature, including all the properties mentioned before, can be expressed in these logics. As a consequence, we show that most of the PTIME upper-bounds obtained for deciding subclasses of finite automata in [2,25], of transducers in [5–8,14,16,18,22,24] and sum-automata in [3,8,10,11], can be directly obtained by expressing in our logics the structural patterns given in these papers, which characterise these subclasses.

Related works. In addition to the results already mentioned, we point out that the syntax of our logic is close to a logic, defined in [9] by Figueira and Libkin, to express path queries in graph databases (finite graphs with edges labelled by a symbol). In this work, there is no disjunction nor negation, and no distinction between input and output values. By making such a distinction, and by adding negation and disjunction, we were able to tailor our logics to particular automata models and add enough power to be able to directly express classical structural automata properties.

2 Finite Automata with Outputs

In this section, we define a general model of finite automata defining functions from the free monoid Λ^* (where Λ is a finite input alphabet) to any monoids $\mathbb{M} = (D, \oplus, \mathbb{0})$. More precisely, they are parametrised by a monoid of *output values*, read input words over some alphabet and output elements of the output monoid, obtained by summing the output values met along accepting paths.

Formally, a *monoid* $\mathbb{M}$ is a tuple $(D, \oplus_{\mathbb{M}}, \mathbb{0}_{\mathbb{M}})$ where D is a set of elements which we call here values or sometimes outputs, $\oplus_{\mathbb{M}}$ is an associative binary operation on D, for which $\mathbb{0}_{\mathbb{M}} \in D$ is neutral. Monoids of interest in this paper are the free monoid $(\Lambda^*, \cdot, \varepsilon)$ for some finite alphabet of symbols Λ (where $\cdot$ denotes the concatenation), and the monoid $(\mathbb{Z}, +, 0)$. We also let $\Lambda_\varepsilon = \Lambda \cup \{\varepsilon\}$. For $w \in \Lambda^*$, $|w|$ denotes its length, in particular $|\varepsilon| = 0$. The set of positions of w is $\{1, \ldots, |w|\}$ (and empty if $w = \epsilon$). We let $w[i]$ be the ith symbol of w. Given w_1, w_2, we write $w_1 \sqsubseteq w_2$ whenever w_1 is a prefix of w_2. *All over this paper, the input alphabet is denoted by the letter Λ.*

Definition 1 (Automata with outputs). *An* automaton A with outputs *over an (output) monoid $\mathbb{M} = (D, \oplus_{\mathbb{M}}, \mathbb{0}_{\mathbb{M}})$ is a tuple $\langle Q, I, F, \Delta, \gamma \rangle$ where Q is a non-empty finite set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, $\Delta \subseteq Q \times \Lambda_\varepsilon \times Q$ the set of transitions labelled with some element of Λ_ε, and $\gamma \colon \Delta \to D$ a mapping from transitions to output values*[1]*. The set of automata over $\mathbb{M}$ is written $\mathcal{A}_{out}(\mathbb{M})$.*

We write $\#(A)$ to refer to the number of states of A. A *path* in A is a sequence $\pi = q_0 a_1 d_1 q_1 \ldots a_n d_n q_n \in Q(\Lambda_\varepsilon D Q)^*$, for $n \geq 0$, such that for all $1 \leq i \leq n$ we have $(q_{i-1}, a_i, q_i) \in \Delta$ and $\gamma(q_{i-1}, a_i, q_i) = d_i$. The *input* of π is defined as the word $\text{in}(\pi) = a_1 \ldots a_n$ (and ε if $\pi \in Q$), the *output* of π as the element $\text{out}(\pi) = d_1 \oplus_{\mathbb{M}} \cdots \oplus_{\mathbb{M}} d_n$ (and $\mathbb{0}_{\mathbb{M}}$ if $\pi \in Q$), and the *size* of π as $|\pi| = n$. We may write $\pi : q_0 \xrightarrow{\text{in}(\pi)|\text{out}(\pi)} q_n$ to denote that π is a path from q_0 to q_n on input $\text{in}(\pi)$ and output $\text{out}(\pi)$. For convenience we write $\pi^{\triangleleft}, \pi^{\triangleright}$ to denote respectively the starting state q_0 and the ending state q_n of the path π. The set of all paths

[1] Often in the literature, output values are directly given in the transitions, i.e. the transition relation is a (finite) subset of $Q \times \Lambda_\varepsilon \times D \times Q$. Our definition is then equivalent modulo PTime transformation, and allows for a clearer distinction between input and output mechanisms.

of A is written Paths(A). A path $\pi : q_0 \xrightarrow{u|v} q_n$ is *initial* if $q_0 \in I$, *final* if $q_n \in F$ and accepting if it is both initial and final. The set of accepting paths of A is denoted by $\text{Paths}_{acc}(A)$. The *input/output relation* (or just relation) defined by A is the set of pairs $R(A) \subseteq \Lambda^* \times D$ defined by

$$R(A) = \{(u, v) \mid \exists \pi \in \text{Paths}_{acc}(A) \cdot \text{in}(\pi) = u \wedge \text{out}(\pi) = v\}$$

Finite automata, transducers and sum-automata. In this paper, we consider three instances of automata with outputs. First, *finite automata* (over Λ), are seen as automata with outputs in a trivial monoid (and which is therefore ignored). *Transducers* are automata with outputs in the free monoid Γ^*. They define relations from Λ^* to Γ^*. Finally, *sum-automata* are automata with outputs in the monoid $(\mathbb{Z}, +, 0)$.

3 A Pattern Logic for Automata with Outputs

In this section, we introduce a generic pattern logic. It is built over four kind of variables, namely path, state, input and output variables. More precisely, we let $X_P = \{\pi, \pi_1, \dots\}$, $X_Q = \{q, q_1, p \dots, \}$, $X_I = \{u, u_1, \dots\}$ and $X_O = \{v, v_1, \dots\}$ be disjoint and countable sets of resp. path, state, input and output variables. We define $Terms(X_O, \oplus, \mathbb{0})$ as the set of terms built over variables of X_O, a binary function symbol $\oplus$ (representing the monoid operation) and constant symbol $\mathbb{0}$ (neutral element).

The logic syntax is parametrised by a set of output predicates $\mathcal{O}$. Output predicates of arity 0 are called constant symbols, and we denote by $\mathcal{O}|_n$ the predicates of arity n. Predicates talking about states, paths and input words are however fixed in the logic.

Definition 2. *A* pattern formula φ *over a set of output predicates* $\mathcal{O}$ *is of the form*

$$\varphi \;=\; \exists \pi_1 : p_1 \xrightarrow{u_1|v_1} q_1, \dots, \exists \pi_n : p_n \xrightarrow{u_n|v_n} q_n, \mathcal{C}$$

where for all $1 \le i \le n$, $\pi_i \in X_P$ *and they are all pairwise different,* $p_i, q_i \in X_Q$, $u_i \in X_I$, $v_i \in X_O$, *and* $\mathcal{C}$ *is a Boolean combination of atoms amongst*

Input constraints	$: u \sqsubseteq u' \mid u \in L \mid \lvert u\rvert \le \lvert u'\rvert$	$u, u' \in X_I$
Output constraints	$: p(t_1, \dots, t_n)$	$p \in \mathcal{O}\vert_n, t_i \in Terms(X_O, \oplus, \mathbb{0})$
State constraints	$: \mathsf{init}(q) \mid \mathsf{final}(q) \mid q = q'$	$q, q' \in X_Q$
Path constraints	$: \pi = \pi'$	$\pi, \pi' \in X_P$

where L *is a regular language of words over* Λ *(assumed to be represented as an NFA). The sequence of existential quantifiers before* $\mathcal{C}$ *in* φ *is called the* prefix *of* φ. *We denote by* $\mathsf{PL}(\mathcal{O})$ *the set of pattern formulas over* $\mathcal{O}$, *and by* $\mathsf{PL}^+(\mathcal{O})$ *the fragment where output predicates does not occur under an odd number of negations.*

The size of a formula is the number of its symbols plus the number of states of all NFA representing the membership constraints. We denote by $\mathrm{Var}(\varphi)$ the variables occurring in any pattern formula φ, and by $\mathrm{Var}_P(\varphi)$ (resp. $\mathrm{Var}_Q(\varphi)$, $\mathrm{Var}_I(\varphi)$, $\mathrm{Var}_O(\varphi)$) its restriction to path (resp. state, input, output) variables. We finally let $(u = u') \overset{\text{def}}{=} u \sqsubseteq u' \wedge u' \sqsubseteq u$, $(|u| = |u'|) \overset{\text{def}}{=} (|u| \leq |u'|) \wedge (|u'| \leq |u|)$, $(|u| < |u'|) \overset{\text{def}}{=} \neg(|u'| \leq |u|)$.

Semantics. To define the semantics of a pattern formula φ, we first fix some monoid $\mathbb{M} = (D, \oplus_{\mathbb{M}}, \mathbb{0}_{\mathbb{M}})$ together with an interpretation $p^{\mathbb{M}}$ of each output predicates $p \in \mathcal{O}$ of arity $\alpha(p)$, such that $p^{\mathbb{M}} \in D$ if p is a constant and $p^{\mathbb{M}} \subseteq D^{\alpha(p)}$ otherwise. Given a valuation $\nu\colon X_O \to D$, the interpretation $.^{\mathbb{M}}$ can be inductively extended to terms t by letting $\mathbb{0}^{\nu,\mathbb{M}} = \mathbb{0}_{\mathbb{M}}$, $(t_1 \oplus t_2)^{\nu,\mathbb{M}} = t_1^{\nu,\mathbb{M}} \oplus_{\mathbb{M}} t_2^{\nu,\mathbb{M}}$ and $x^{\nu,\mathbb{M}} = \nu(x)$.

Then, a formula $\varphi \in \mathsf{PL}(\mathcal{O})$ is interpreted in an automaton with outputs $A \in \mathcal{A}_{out}(\mathbb{M})$ as a set of valuations $[\![\varphi]\!]_A$ of $\mathrm{Var}(\varphi)$ which we now define. Each valuation $\nu \in [\![\varphi]\!]_A$ maps state variables to states of A, path variables to paths of A, etc. Such a valuation ν satisfies an atom $u \sqsubseteq u'$ if $\nu(u)$ is a prefix of $\nu(u')$, $u \in L$ if $\nu(u) \in L$, $|u| \leq |u'|$ if $|\nu(u)| \leq |\nu(u')|$. Given a predicate $p \in \mathcal{O}$ of arity $\alpha(p)$, an atom $p(t_1, \ldots, t_{\alpha(p)})$ is satisfied by ν if $(t_1^{\nu,\mathbb{M}}, \ldots, t_{\alpha(p)}^{\nu,\mathbb{M}}) \in p^{\mathbb{M}}$. Finally, ν satisfies $\mathsf{init}(q)$ (resp. $\mathsf{final}(q)$) if $\nu(q)$ is initial (resp. $\nu(q)$ is final). The satisfiability relation is naturally extended to Boolean combinations of atoms. Finally, assume that φ is of the form $\exists \pi_1\colon p_1 \xrightarrow{u_1|v_1} q_1, \ldots, \exists \pi_n\colon p_n \xrightarrow{u_n|v_n} q_n, \mathcal{C}$, we say that A satisfies φ, denoted by $A \models \varphi$, if there exists a valuation ν of $\mathrm{Var}(\varphi)$ such that for all $i \in \{1, \ldots, n\}$, $\nu(\pi_i)\colon \nu(p_i) \xrightarrow{\nu(u_i)|\nu(v_i)} \nu(q_i)$ and ν satisfies $\mathcal{C}$ ($\nu \models \mathcal{C}$). Given a pattern formula φ and an automaton with outputs A, the *model-checking problem* consists in deciding whether A satisfies φ, i.e. $A \models \varphi$.

Example 1. Given $k \in \mathbb{N}$, the k-valuedness property has been already expressed in Introduction (assuming $= \in \mathcal{O}$). The formula $\exists \pi_0\colon p_0 \xrightarrow{u|v_0} q_0, \ldots, \exists \pi_k\colon p_k \xrightarrow{u|v_k} q_k, \mathcal{C}_0$ where $\mathcal{C}_0 = \bigwedge_{0 \leq i < j \leq k} \pi_i \neq \pi_j \wedge \bigwedge_{i=0}^{k} \mathsf{init}(p_i) \wedge \mathsf{final}(q_i)$ expresses the fact that an automaton is not $(k-1)$-ambiguous (has at least k accepting paths for some input).

4 Model-Checking Problem

In this section, we give sufficient conditions on the output monoid $\mathbb{M}$ and the set of output predicates $\mathcal{O}$ by which the model-checking of automata with outputs in $\mathbb{M}$ against pattern formulas over the output predicates $\mathcal{O}$ is decidable. In the next sections, we study the precise complexity of the model-checking problem for particular monoids $\mathbb{M}$.

Tuple acceptors. Since automata with outputs can get their output values in arbitrary monoids, to get an effective model-checking algorithm, we will assume the existence of machines, called tuple acceptors, that can recognise sets of word tuples. These machines will be required to satisfy some key properties, forming the notion of *good class* of tuple acceptors. First, what we call a *tuple acceptor* is a machine M whose semantics is a set of tuples of words $[\![M]\!] \subseteq (\Sigma^*)^n$, for some alphabet Σ and some arity $n \geq 1$. The notion of good class, formally defined later, require (i) that any regular set of tuples is recognised by some machine, for a regularity notion that we will make clear (roughly, by seeing tuples of words as words resulting from the overlapping of all components), (ii) all output predicates (and their negation) are recognised by some machine, (iii) the class is closed under union and intersection.

Regular sets of word tuples. Let Σ be some alphabet containing some symbol $\perp$, $\pi \in \Sigma^*$ and $m \geq |\pi|$. The *padding* of π with respect to m is the word $\pi' = \pi\perp^{m-|\pi|}$. Let $\pi_1, \pi_2 \in \Sigma^*$ and let $m = \max(|\pi_1|, |\pi_2|)$. For $j = 1, 2$, let π'_j the padding of π_j with respect to m. Note that $|\pi'_1| = |\pi'_2| = m$. The *convolution* $\pi_1 \otimes \pi_2$ is the word of length m defined for all $1 \leq i \leq n$ by $(\pi_1 \otimes \pi_2)[i] = (\pi'_1[i], \pi'_2[i])$. E.g. $q_1\lambda_1 d_1 q_2 \otimes p_1 = (q_1, p_1)(\lambda_1, \perp)(d_1, \perp)(q_2, \perp)$. The convolution can be naturally extended to multiple words as follows: $\bigotimes_{i=1}^n \pi_i = \pi_1 \otimes (\pi_2 \otimes \ldots \otimes \pi_n)$.

Definition 3. *A set of n-ary word tuples $P \subseteq (\Sigma^*)^n$ is* regular *if $L = \{\bigotimes_{i=1}^n \pi_i \mid (\pi_1, \ldots, \pi_n) \in P\}$ is a regular language over Σ^n. We often identify L and P.*

Good class of tuple acceptors. First, any valuation ν of a set of path variables X into paths of some automaton with values in some monoid $\mathbb{M}$ gives a way to interpret terms $t \in \text{Terms}(X, \oplus, \mathbb{0})$ as follows: for $\pi \in X$, $\pi^{\nu,\mathbb{M}} = \text{out}(\nu(\pi))$, $\mathbb{0}^{\nu,\mathbb{M}} = \mathbb{0}_\mathbb{M}$ and $(t_1 \oplus t_2)^{\nu,\mathbb{M}} = t_1^{\nu,\mathbb{M}} \oplus_\mathbb{M} t_2^{\nu,\mathbb{M}}$. Then, for a class $\mathcal{C}$ (i.e. a set) of tuple acceptors, we denote by $\mathcal{C}|_n$ its restriction to acceptors of arity n.

Definition 4 (Good class). *A class of tuple acceptors $\mathcal{C}$ is said to be* good *for an output monoid $\mathbb{M} = (D, \oplus_\mathbb{M}, \mathbb{0}_\mathbb{M})$, a set of output predicates $\mathcal{O}$ and an interpretation $p^\mathbb{M} \subseteq D^{\alpha(p)}$ for all $p \in \mathcal{O}$ of arity $\alpha(p)$, if the following conditions are satisfied:*

1. *for all automata with outputs $A \in \mathcal{A}_{out}(\mathbb{M})$ with a set of states Q we have:*
   (a) $\forall n \geq 1, \forall R \subseteq Paths(A)^n$ regular, $R = [\![M]\!]$ for some $M \in \mathcal{C}|_n$.
   (b) all $p \in \mathcal{O}$ of arity $\alpha(p)$, all $X = \{\pi_1, \ldots, \pi_n\}$ finite sets of path variables and all $t_1, \ldots, t_{\alpha(p)} \in Terms(X, \oplus, \mathbb{0})$, there exist $M, M' \in \mathcal{C}|_n$ such that
      i. $[\![M]\!] = \{(\nu(\pi_1), \ldots, \nu(\pi_n)) \mid \nu\colon X \to Paths(A) \wedge (t_1^{\nu,\mathbb{M}}, \ldots, t_{\alpha(p)}^{\nu,\mathbb{M}}) \in p^\mathbb{M}\}$
      ii. $[\![M']\!] = Paths(A)^n \setminus [\![M]\!]$.
2. *$\forall n \geq 1$, $\forall M_1, M_2 \in \mathcal{C}|_n$, there exist $M, M' \in \mathcal{C}|_n$ such that $[\![M]\!] = [\![M_1]\!] \cap [\![M_2]\!]$ and $[\![M']\!] = [\![M_1]\!] \cup [\![M_2]\!]$.*

We say that $\mathcal{C}$ is effective *if all properties are effective and moreover it is decidable whether $[\![M]\!] \neq \varnothing$ for any (effectively represented) $M \in \mathcal{C}$. We say that $\mathcal{C}$ is* weakly good *if all properties hold except 1(b)ii.*

Effectiveness of a good class gives effective model-checking, as announced.

Theorem 1. *Let $\mathbb{M}$ be a monoid and $\mathcal{O}$ be a set of output predicates, interpreted over $\mathbb{M}$. If there exists an effective good class $\mathcal{C}$ (resp. effective weakly good class) of tuple acceptors for $\mathbb{M}$ and $\mathcal{O}$, then the model-checking problem of automata with outputs in $\mathbb{M}$ against pattern formulas $\psi \in \mathsf{PL}[\mathcal{O}]$ (resp. $\psi \in \mathsf{PL}^+[\mathcal{O}]$) is decidable.*

Proof (sketch). First, the formula is put in negation normal form: negation is pushed down to the atoms. Then, given an automaton with outputs in $\mathbb{M}$, we show that any tuple of paths which satisfy state, input and path predicates and their negations is a regular set of path tuples (this is doable even for input equality as well as input length comparison thanks to the way paths are overlapped by the definition of convolution). By condition 1a, these sets of tuples are accepted by acceptors of $\mathcal{C}$. By conditions 1(b)i and ii, tuples of paths satisfying output predicates and their negations are also accepted by acceptors of $\mathcal{C}$. Then, the closure properties (condition 2) allows us to construct an acceptor for the tuples of paths satisfying the whole formula inductively. □

5 A Pattern Logic for Finite Automata

Finite automata can be seen as automata with outputs in a trivial monoid (with a single element). As the monoid is trivial, there is no need for predicates over it and so we specialize our pattern logic into $\mathsf{PL}_{\text{NFA}} = \mathsf{PL}[\varnothing]$.

Definition 5 (Pattern logic for NFA). *The logic PL_{NFA} is the set of formulas*

$$\begin{aligned}
&\varphi := \exists\pi_1 : p_1 \xrightarrow{u_1} q_1, \ldots, \exists\pi_n : p_n \xrightarrow{u_n} q_n, \mathcal{C}\\
&\mathcal{C} := \neg\mathcal{C} \mid \mathcal{C} \vee \mathcal{C} \mid u \sqsubseteq u' \mid u \in L \mid |u| \leq |u'| \mid \mathit{init}(q) \mid \mathit{final}(q) \mid q = q' \mid \pi = \pi'
\end{aligned}$$

where for all $i \neq j$, $\pi_i \neq \pi_j$, L is a regular language over Λ (assumed to be represented as an NFA), $u, u' \in \{u_1, \ldots, u_n\}$, $q, q' \in \{q_1, \ldots, q_n\}$ and $\pi, \pi' \in \{\pi_1, \ldots, \pi_n\}$.

As a yardstick to measure the expressiveness of PL_{NFA}, we have considered the structural properties of NFA studied in two classical papers: [25] by Weber and Seidl and in [2] by Allauzen et al. The authors of these two papers give PTIME membership algorithms for k-ambiguity, finite ambiguity, polynomial ambiguity and exponential ambiguity (with as applications the approximation of the entropy of probabilistic automata for example). We refer the interested readers to these papers for the formal definitions of those classes. The solutions to these membership problems follow a recurrent schema: one defines (1) a pattern that identifies the members of the class and (2) an algorithm to decide if an automaton satisfies the pattern. The next theorem states that all these membership problems can be reduced to the model-checking problem of PL_{NFA} using a constant space reduction. The proof of this theorem is obtained by showing how the patterns identified in [25], can be succinctly and naturally encoded into (fixed) $\mathsf{PL}_{\mathrm{NFA}}$ formulas. As a corollary, we get that all the class membership problems are in NLogSpace, using a model-checking algorithm that we defined below for $\mathsf{PL}_{\mathrm{NFA}}$.

Theorem 2. *The membership problem to the subclasses of k-ambiguous, finitely ambiguous, polynomially ambiguous and exponentially ambiguous NFA can be reduced to the model-checking problem of PL_{NFA} with constant space reduction. The obtained formulas are constant (for fixed k).*

Proof. For each membership problem, our reduction copies (in constant space) the NFA and considers the model-checking for this NFA against a fixed $\mathsf{PL}_{\mathrm{NFA}}$ (one for each class). As illustration, k-ambiguity has already been expressed in Example 1. As a second example, an automaton is not polynomially ambiguous iff there exists a state p which is reachable from an initial state, and the source of two different cycles labelled identically by a word v. With $\mathsf{PL}_{\mathrm{NFA}}$ this gives: $\exists\pi_0 : q_0 \xrightarrow{u_1} p, \exists\pi_1 : p \xrightarrow{u_2} p, \exists\pi_2 : p \xrightarrow{u_2} p, \exists\pi_3 : p \xrightarrow{u_3} q, \mathsf{init}(q_0) \wedge \pi_1 \neq \pi_2 \wedge \mathsf{final}(q)$ □

The model-checking problem asks if a given NFA A satisfies a given $\mathsf{PL}_{\mathrm{NFA}}$-formula φ.

Theorem 3. *The model-checking problem of NFA against formulas in PL_{NFA} is* PSpace-C. *It is in* NLogSpace-C *if the formula is fixed.*

Proof (sketch). We use NFA as acceptors for tuples of paths. The algorithm presented in the proof of Theorem 1 yields an exponentially large NFA (and polynomial if the formula is fixed). We show that it does not need to be constructed explicitly and that a short non-emptiness witness can be searched non-deterministically on-the-fly. For PSpace-hardness, we notice that the non-emptiness of the intersection of n DFA can be easily expressed in $\mathsf{PL}_{\mathrm{NFA}}$, by seeing the n DFA as a disjoint union, and by asking for the existence of n different accepting paths over the same input in this union. □

Corollary 1 (of Theorems *2* and *3*). *The membership problem to the classes of k-ambiguous, finitely ambiguous, polynomially ambiguous and exponentially ambiguous NFA is in* NLogSpace.

6 A Pattern Logic for Transducers

Transducers are automata with outputs in a free monoid $\mathbb{M}_{Trans} = (\Gamma^*, \cdot, \varepsilon)$ and therefore define subsets of $\Lambda^* \times \Gamma^*$. Since our general pattern logic can test for output equalities (by repeating twice an output variable in the prefix), the model-checking is easily shown to be undecidable by encoding PCP:

Theorem 4. *The model-checking problem of transducers against formulas in $\mathsf{PL}[\varnothing]$ is undecidable.*

To obtain a decidable logic for transducers, we need to exclude equality tests on the output words in the logic. However, as we will see, we can instead have inequality test $\neq$ as long as it is not under an odd number of negations in the formula. We also allow to test (non) membership of output word concatenations to a regular language, as well as comparison of output word concatenations wrt their length. Formally:

Definition 6 (Pattern logic for transducers). *The logic $\mathsf{PL}_{\mathit{Trans}}$ is the set of formulas of the form*

$$\begin{aligned}
\varphi &:= \exists \pi_1 : p_1 \xrightarrow{u_1|v_1} q_1, \dots, \exists \pi_n : p_n \xrightarrow{u_n|v_n} q_n, \mathcal{C}\\
\mathcal{C} &:= \neg \mathcal{C} \mid \mathcal{C} \vee \mathcal{C} \mid u \sqsubseteq u' \mid u \in L \mid |u| \leq |u'| \mid \mathit{init}(q) \mid \mathit{final}(q) \mid q = q' \mid \pi = \pi' \mid \\
&\quad\; t \not\sqsubseteq t' \mid t \in N \mid |t| \leq |t'|
\end{aligned}$$

where for all $1 \leq i < j \leq n$, $\pi_i \neq \pi_j$ and $v_i \neq v_j$ (no implicit output equality tests), L (resp. N) is a regular language over Λ (resp. Γ), assumed to be represented as an NFA, $u, u' \in \{u_1, \dots, u_n\}$, $q, q' \in \{q_1, \dots, q_n\}$, $t, t' \in Terms(\{v_1, \dots, v_n\}, \cdot, \epsilon)$, $\pi, \pi' \in \{\pi_1, \dots, \pi_n\}$, and $t \not\sqsubseteq t'$ does not occur under an odd number of negations.

We define the macros $t \neq t' \overset{\text{def}}{=} t \not\sqsubseteq t' \vee t' \not\sqsubseteq t$, $\text{mismatch}(t, t') \overset{\text{def}}{=} t \not\sqsubseteq t' \wedge t' \not\sqsubseteq t$ and

$$\text{SDel}_{\neq}(t_1, t_1', t_2, t_2') \overset{\text{def}}{=} (|t_1'| \neq |t_2'|) \;\vee [\; t_1' t_2' \neq \epsilon \wedge \text{mismatch}(t_1, t_2)]$$

Let us explain the latter macro. Many properties of transducers are based on the notion of output delays, by which to compare output words. Formally, for any two words v_1, v_2, $\text{delay}(v_1, v_2) = (\alpha_1, \alpha_2)$ such that $v_1 = \ell\alpha_1$ and $v_2 = \ell\alpha_2$ where ℓ is the longest common prefix of v_1 and v_2. It can be seen that for any words v_1, v_1', v_2, v_2', if we have $\text{SDel}_{\neq}(v_1, v_1', v_2, v_2')$, then $\text{delay}(v_1, v_2) \neq \text{delay}(v_1 v_1', v_2 v_2')$, but the converse does not hold. But, if $\text{delay}(v_1, v_2) \neq \text{delay}(v_1 v_1', v_2 v_2')$, then $\text{SDel}_{\neq}(v_1 (v_1')^i, v_1', v_2 (v_2')^i, v_2')$ holds for some $i \geq 0$. These two facts allows us to express all the known transducer properties from the literature relying on the notion of delays. We leave however as open whether our logic can express a constraint such as $\text{delay}(v_1, v_2) \neq \text{delay}(v_3, v_4)$.

We review here some of the main transducer subclasses studied in the literature. We refer the reader to the mentioned references for the formal definitions. As for the NFA subclasses of the previous section, deciding them usually goes in two steps: (1) identify a structural pattern characterising the property, (2) decide whether such as pattern is satisfied by a given transducer. The class of determinisable transducers are the transducers which define sequential functions [5,6,24]. The k-sequential transducers are the transducers defining unions of (graphs) of k sequential functions [8]. The multi-sequential ones are the union of all k-sequential transducers for all k [7,16]. Finally, the k-valued transducers are the transducers for which any input word has at most k output words [14,19], and the finite-valued ones are all the k-valued transducers for all k [18,22,23]. All these classes, according to the given references, are decidable in PTime.

Theorem 5. *The membership problem of transducers to the classes of determinisable, functional, k-sequential, multi-sequential, k-valued, and finite-valued transducers can be reduced to the model-checking problem of PL_{Trans} with a constant space reduction. The obtained formulas are constant (as long as k is fixed).*

Proof. Without going through all the properties, let us remind the reader that the formula for k-valuedness has been given in the introduction. We also give the $\mathsf{PL}_{\text{Trans}}$ formulas for the class of determinisable transducer. It is known that a transducer is determinisable iff it satisfies the twinning property, which is literally the negation of:

$$\begin{aligned}\exists \pi_1 : q_1 \xrightarrow{u|v_1} p_1, \exists \pi_1' : p_1 \xrightarrow{u'|v_1'} p_1, \exists \pi_1'' : p_1 \xrightarrow{u''|v_1''} r_1,\\ \exists \pi_2 : q_2 \xrightarrow{u|v_2} p_2, \exists \pi_2' : p_2 \xrightarrow{u'|v_2'} p_2, \exists \pi_2'' : p_2 \xrightarrow{u''|v_2''} r_2,\\ \mathsf{init}(q_1) \wedge \mathsf{init}(q_2) \wedge \mathsf{final}(r_1) \wedge \mathsf{final}(r_2) \wedge \mathrm{SDel}_{\neq}(v_1, v_1', v_2, v_2')\end{aligned} \quad \square$$

Theorem 6. *The model checking of transducers against formulas in PL_{Trans} is* PSPACE-C*. It is in* NLOGSPACE-C *if the formula is fixed.*

Proof (sketch). We use Parikh automata as acceptors for tuples of paths. They extend automata with counters that can only be incremented and never tested for zero. The acceptance condition is given by a semi-linear set (represented for instance by an existential Presburger formula). The formal definition can be found e.g. in [9]. The counters allow us to compare the output length of paths, or to identify some output position of two paths with different labels (to test $v \not\preceq v'$). The counters are needed because this position may not occur at the same location in the convolution encoding of path tuples. $\square$

Corollary 2 (of Theorems 5 and 6). *The membership problem of transducers to the classes of determinisable, functional, k-sequential, multi-sequential, k-valued, and finite-valued transducers (for fixed k) is decidable in* NLOGSPACE*.*

7 A Pattern Logic for Sum-Automata

We remind the reader that sum-automata are automata with outputs in the monoid $\mathbb{M}_{\text{Sum}} = (\mathbb{Z}, +, 0)$ (assumed to be encoded in binary) and therefore define subsets of $\Lambda^* \times \mathbb{Z}$. We consider in this section two logics for expressing structural properties of sum-automata: the logic PL_{Sum} which is obtained as $\mathsf{PL}[\{\leq\}]$ where the output predicate $\leq$ is interpreted by the natural total order over integers, and a subset of this logic $\mathsf{PL}^{\neq}_{\text{Sum}}$ obtained as $\mathsf{PL}^{+}[\{\neq\}]$ where the predicate $\neq$ never appears in the scope of an odd number of negations (to avoid the expressibility of the equality predicate). We show that the fragment $\mathsf{PL}^{\neq}_{\text{Sum}}$ enjoys better complexity results. Formally, those two logics are defined as follows:

Definition 7 (Two pattern logics for sum-automata). *The logic PL_{Sum} is the set of formulas of the form*

$$\begin{aligned}\varphi &:= \exists\pi_1 : p_1 \xrightarrow{u_1|v_1} q_1, \dots, \exists\pi_n : p_n \xrightarrow{u_n|v_n} q_n, \mathcal{C}\\ \mathcal{C} &:= \neg\mathcal{C} \mid \mathcal{C}\vee\mathcal{C} \mid u \sqsubseteq u' \mid u \in L \mid |u| \leq |u'| \mid \mathit{init}(q) \mid \mathit{final}(q) \mid q = q' \mid \pi = \pi' \mid\\ &\quad\; t \leq t'\end{aligned}$$

where for all $1 \leq i < j \leq n$, $\pi_i \neq \pi_j$, L is a regular language over Λ assumed to be represented as an NFA, $u, u' \in \{u_1, \dots, u_n\}$, $q, q' \in \{q_1, \dots, q_n\}$, $t, t' \in Terms(\{v_1, \dots, v_n\}, \cdot, \epsilon)$ and $\pi, \pi' \in \{\pi_1, \dots, \pi_n\}$.

The logic $\mathsf{PL}^{\neq}_{Sum}$ is defined as above but the constraint $t \leq t'$ is replaced by $t \neq t'$ and this constraint does not occur under an odd number of negations, and moreover $v_i \neq v_j$ for all $1 \leq i < j \leq n$ (no implicit output equality tests).

We review here some of the main sum-automata subclasses decidable in PTime studied in the literature. We refer the reader to the mentioned references for the formal definitions. The class of *functional* sum-automata [11] are those such that all accepting paths associated with a given word return the same value. The classes of *k-valued* [10] and *k-sequential* sum-automata [8] are defined similarly as for transducers.

Theorem 7. *The membership problem of sum-automata in the class of functional, k-valued, and k-sequential automata can be reduced to the model-checking problem of $\mathsf{PL}^{\neq}_{Sum}$. Moreover, the obtained $\mathsf{PL}^{\neq}_{Sum}$ formulas are constant (as long as k is fixed).*

Proof. We have already shown in the introduction that functionality [11] and more generally k-valuedness [10] are expressible in $\mathsf{PL}^{\neq}_{\mathrm{Sum}}$. The twinning property [1,11] is as well expressible in $\mathsf{PL}^{\neq}_{\mathrm{Sum}}$, just by replacing in the formula expressing it for transducers (proof of Theorem 5) the atom $\mathrm{SDel}_{\neq}(v_1, v_1', v_2, v_2')$ by $v_1' \neq v_2'$. In [8], a generalization of the twinning property is shown to be complete for testing k-sequentiality. □

The proof of the results below for $\mathsf{PL}_{\mathrm{Sum}}$ follows arguments that are similar to those developed for transducers in the proof of Theorem 6, and for the PTime result for $\mathsf{PL}^{\neq}_{\mathrm{Sum}}$, we use a reduction to the k-valuedness problem of sum-automata [10].

Theorem 8. *The model checking of sum-automata against formulas in PL_{Sum} is* PSpace-C, NP-C *when the formula is fixed, and* NLogSpace-C *if in addition the values of the automaton are encoded in unary. The model checking of sum-automata against formulas in $\mathsf{PL}^{\neq}_{Sum}$ is* PSpace-C, *and in* PTime *when the formula is fixed (even if the values of the automaton are encoded in binary).*

Corollary 3 (of Theorems 7 and 8). *The membership problem of sum-automata in the class of functional, k-valued, and k-sequential automata is decidable in* PTime.

Note that we have shown that the k-valuedness property is expressible in $\mathsf{PL}^{\neq}_{\mathrm{Sum}}$, and so the k-valuedness property is reducible to the model-checking problem of $\mathsf{PL}^{\neq}_{\mathrm{Sum}}$. Nevertheless, this result does not provide a new algorithm for k-valuedness as our model-checking algorithm is based on a reduction to k-valuedness [10].

8 Extensions and Future Work

The logics we have presented can be extended in two ways by keeping the same complexity results, no matter what the output monoid is. The first extension allows to express properties of automata whose states can be coloured by an arbitrary (but fixed) set of colours. This is useful for instance to express properties of disjoint unions of automata, the colours allowing to identify the subautomata. The second extension is adding a bunch of universal state quantifiers before the formula. This does not change the complexity, and allow for instance to express properties such as whether an automaton is trim (all its states are accessible and co-accessible). As future work, we would like to investigate other monoids (discounted sum group for instance [11]), and other data structures for which transducers and weighted automata have been defined: nested words, infinite words and trees are the main structures we want to work on.

References

1. Allauzen, C., Mohri, M.: Efficient algorithms for testing the twins property. J. Autom. Lang. Comb. **8**(2), 117–144 (2003)
2. Allauzen, C., Mohri, M., Rastogi, A.: General algorithms for testing the ambiguity of finite automata and the double-tape ambiguity of finite-state transducers. Int. J. Found. Comput. Sci. **22**(4), 883–904 (2011)
3. Bala, S., Koniński, A.: Unambiguous automata denoting finitely sequential functions. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 104–115. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37064-9_11
4. Béal, M.-P., Carton, O.: Determinization of transducers over finite and infinite words. Theor. Comput. Sci. **289**(1), 225–251 (2002)
5. Béal, M.-P., Carton, O., Prieur, C., Sakarovitch, J.: Squaring transducers: an efficient procedure for deciding functionality and sequentiality. TCS **292**(1), 45–63 (2003)
6. Choffrut, C.: Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. Theor. Comput. Sci. **5**(3), 325–337 (1977)
7. Choffrut, C., Schutzenberger, M.P.: Decomposition de fonctions rationnelles. In: Monien, B., Vidal-Naquet, G. (eds.) STACS 1986. LNCS, vol. 210, pp. 213–226. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-16078-7_78
8. Daviaud, L., Jecker, I., Reynier, P.-A., Villevalois, D.: Degree of sequentiality of weighted automata. In: Esparza, J., Murawski, A.S. (eds.) FoSSaCS 2017. LNCS, vol. 10203, pp. 215–230. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54458-7_13

9. Figueira, D. Libkin, L.: Path logics for querying graphs: combining expressiveness and efficiency. In: LICS, pp. 329–340 (2015)
10. Filiot, E., Gentilini, R., Raskin, J.-F.: Finite-valued weighted automata. In: FSTTCS, pp. 133–145 (2014)
11. Filiot, E., Gentilini, R., Raskin, J.-F.: Quantitative languages defined by functional automata. LMCS **11**(3) (2015)
12. Galil, Z.: Hierarchies of complete problems. Acta Inform. **6**(1), 77–88 (1976)
13. Griffiths, T.V.: The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. J. ACM **15**(3), 409–413 (1968)
14. Gurari, E.M., Ibarra, O.H.: A note on finite-valued and finitely ambiguous transducers. Theor. Comput. Syst. **16**(1), 61–66 (1983)
15. Berstel, J.: Transductions and Context-Free Languages. Teubner, Stuttgart (1979)
16. Jecker, I. Filiot, E.: Multi-sequential word relations. IJFCS 29(2), 271–295 (2018)
17. Klaedtke, F., Rueß, H.: Monadic second-order logics with cardinalities. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 681–696. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45061-0_54
18. Sakarovitch, J., de Souza, R.: On the decidability of bounded valuedness for transducers. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 588–600. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85238-4_48
19. Sakarovitch, J., de Souza, R.: Lexicographic decomposition of k -valued transducers. Theor. Comput. Syst. **47**(3) (2010)
20. Scarpellini, B.: Complexity of subcases of presburger arithmetic. Trans. Am. Math. Soc. **284**(1), 203–218 (1984)
21. Stearns, R.E., Hunt III, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM J. Comput. **14**(3), 598–611 (1985)
22. Weber, A.: On the valuedness of finite transducers. Acta Inform. **27**(8), 749–780 (1990)
23. Weber, A.: Decomposing finite-valued transducers and deciding their equivalence. SIAM J. Comput. **22**(1), 175–202 (1993)
24. Weber, A., Klemm, R.: Economy of description for single-valued transducers. Inf. Comput. **118**(2), 327–340 (1995)
25. Weber, A., Seidl, H.: On the degree of ambiguity of finite automata. Theor. Comput. Sci. **88**(2), 325–349 (1991)