Automata and Logics for Concurrent Systems: Five Models in Five Pages*

Benedikt Bollig

LSV, ENS Cachan, CNRS & Inria
bollig@lsv.ens-cachan.fr

Abstract. We survey various automata models of concurrent systems and their connection with monadic second-order logic: finite automata, class memory automata, nested-word automata, asynchronous automata, and message-passing automata.

1 Introduction

A variety of automata models have emerged over the years to provide a basis for the study of various types of concurrent systems. These models capture several communication paradigms such as shared memory or message passing, and they can deal with finite-state or infinite-state processes. In this paper, we consider several of these models in a unifying framework.

The study that we conduct here is driven by questions that arise in the area of verification. Concurrent systems are often safety-critical and come with a requirements specification to be fulfilled. In the automata-theoretic approach, which we adopt here, a system is modeled as an automaton $\mathcal{A}$, and the specification is given as a formula φ in a high-level language such as temporal logic or monadic second-order logic. In order for both, the system model and the specification, to be comparable, they should have a common domain. We associate with an automaton its language $L(\mathcal{A})$, representing the set of possible behaviors. Similarly, the specification determines a set $L(\varphi)$ of models, namely the set of behaviors that satisfy it. Then, correctness of the system can be expressed as the inclusion problem $L(\mathcal{A}) \subseteq L(\varphi)$, which is commonly referred to as the *model-checking problem*. In the different approach of *realizability*, only the specification φ is given, and we aim at an automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\varphi)$. In that case, $\mathcal{A}$ may serve as a system model that can be considered correct by construction. The models that we cover here owe much of their success to the fact that model checking and realizability have positive solutions. In particular, they all enjoy logical characterizations in terms of (an expressive fragment of) monadic second-order (MSO) logic and come, possibly under restrictions, with a decidable emptiness problem.

But what is actually a behavior? A behavior is the collection of events that we observe during an execution. There are essentially two approaches: In the

* not including introduction and references

interleaving approach, one imposes an order on a priori independent events. In the graph-based approach, a behavior reveals causal dependencies between them and explicitly records, in terms of edges/binary relations, any access to a data structure such as the current state, a channel, or a stack. We adopt here the graph-based approach. As argued convincingly in [1], it is more natural and expressive when one wants to reason about system properties beyond reachability.

Signature. One crucial parameter of a system model and its behaviors is the underlying *signature*, which consists of a nonempty finite set Σ of *actions* and a nonempty finite set R of binary relation symbols. Intuitively, a letter $a \in \Sigma$ describes an action executed by a system such as "send request to the server", or "call procedure P". Moreover, binary relations $\lhd_r$, with r ranging over R, reflect the functionality of a system. To model message passing, for example, a behavior comes with a binary relation that links a send event with the corresponding receive event. In a system involving recursive processes, there will be a binary relation connecting a procedure call with the corresponding return. Once a signature is fixed, we obtain both a canonical automata model and a canonical monadic second-order logic (see below).

Behavior. To give a meaning to the signature σ, we first have to define what a behavior is, representing one possible execution of a system. A *σ-behavior* is a tuple $B = (E, \lambda, (\lhd_r)_{r\in R})$ where E is a nonempty finite set of *events*, $\lambda : E \to \Sigma$ is a labeling function, and $\lhd_r \subseteq E \times E$ is a binary relation, for every $r \in R$. We interpret each $\lhd_r$ as edge relation and call its elements *r-edges*. As a behavior describes the progress of an execution, it is natural to require that $\bigcup_{r\in R} \lhd_r$ is acyclic (and, in particular, irreflexive). Moreover, we assume that access to a data structure is well-defined: for all $r \in R$, each event has at most one outgoing r-edge and at most one incoming r-edge. Example behaviors can be found in Figures 1–4. Later, depending on the automata model that we consider, we will restrict σ-behaviors further (e.g., to model FIFO queues or pushdown stacks).

We conclude this paragraph with a definition that will be useful later on. For $Q \subseteq R$, we say that behavior B is *disconnected* wrt. Q if, informally speaking, each event belongs to at most one Q-edge: for all $r, r' \in Q$, $(e, f) \in \lhd_r$, and $(e', f') \in \lhd_{r'}$, either $r = r' \wedge e = e' \wedge f = f'$ or $\{e, f\} \cap \{e', f'\} = \emptyset$.

Automata. A σ-automaton $\mathcal{A}$ has a finite set of *states* S. A *run* of $\mathcal{A}$ on a σ-behavior $B = (E, \lambda, (\lhd_r)_{r\in R})$ is just a mapping $\rho : E \to S$. Intuitively, for $e \in E$, $\rho(e)$ is the state taken *after* executing e. Of course, this assignment has to conform with a finite set of *transitions*. Actually, when executing e, the automaton $\mathcal{A}$ has access to some states taken previously in the run, and this access is determined by the relations associated with R. In fact, a transition is a triple (pred, a, s) where $a \in \Sigma$, $s \in S$, and $\mathit{pred} : R \rightharpoonup S$ is a *partial* mapping that allows the automaton to access the state taken at a $\lhd_r$-predecessor of e. Formally, we require that, for every $e \in E$, there is a transition (pred, a, s) such that $\lambda(e) = a$, $\rho(e) = s$, and, for all $r \in R$, the following hold:

- if e does not have a $\lhd_r$-predecessor, then $\mathit{pred}(r)$ is undefined, and

– if e has a $\lhd_r$-predecessor f (which is then unique), then $\rho(f) = pred(r)$.

Note that initial states can be implicitly modeled by functions *pred* that are partially or entirely undefined. To take into account several automata models in a unifying framework, we assume a quite general *acceptance condition*, which is a set of tuples $(O, (T_r)_{r\in R})$ where $O \subseteq \Sigma$ and $T_r \subseteq S$ for all $r \in R$. Run ρ is *accepting* if the acceptance condition contains some tuple $(O, (T_r)_{r\in R})$ such that $O = \{\lambda(e) \mid e \in E\}$ and, for all $r \in R$ and events e without $\lhd_r$-successor, we have $\rho(e) \in T_r$. In some of the settings discussed below, the *occurrence set* O can be used to guarantee that a process makes at least one move. Note that, in the simple framework of finite automata, an acceptance condition will be no more expressive than just assuming a single set of final states and requiring that a run ends in a final state. To summarize, a σ-automaton consists of a finite set of states, a finite set of transitions, and an acceptance condition. By $L(\mathcal{A})$, we denote the set of σ-behaviors that allow for an accepting run of $\mathcal{A}$.

Logic. Given a signature σ, we assume the canonical MSO logic for σ, which we denote by $\mathrm{MSO}(\sigma)$. There are infinite supplies of first-order variables $x, y, \ldots$ and second-order variables $X, Y, \ldots$ They allow us to quantify over events and sets of events, respectively, using formulas $\exists x\varphi$ and $\exists X\varphi$ (where φ is again an $\mathrm{MSO}(\sigma)$ formula). Furthermore, we can use the boolean operators negation and disjunction (and, therefore, conjunction and universal quantification). The formula $\lambda(x) = a$ expresses that event x executes $a \in \Sigma$. Finally, we have access to the binary relations in terms of formulas $x \lhd_r y$, with $r \in R$, and we include $x = y$ with the obvious meaning. The set of σ-behaviors that satisfy a given $\mathrm{MSO}(\sigma)$ sentence φ (without free variables) is defined as usual and denoted by $L(\varphi)$. We refer the reader to [21] for an introduction to MSO logic.

Later, we may have to consider fragments of MSO logic to match the expressive power of an automata model. The existential fragment of $\mathrm{MSO}(\sigma)$, denoted by $\mathrm{EMSO}(\sigma)$, contains the formulas of the form $\exists X_1 \ldots \exists X_n \varphi$ where φ does not use any second-order quantifier. When, in addition, we use only two first-order variables (which, however, can occur several times in a formula), we deal with the fragment $\mathrm{EMSO}_2(\sigma)$ of $\mathrm{EMSO}(\sigma)$. In those fragments, it is sometimes impossible, or not obvious, to encode the reflexive transitive closure of a binary predicate. We may add such predicates explicitly and write, for example, $\mathrm{EMSO}_2(\sigma + \lhd_{r_1}^* + \lhd_{r_2}^*)$ when we allow access to $\lhd_{r_1}^*$ and $\lhd_{r_2}^*$.

Realizability and Emptiness Checking. Each of the following sections will describe a particular system model. For each setting, we proceed as follows. We will first fix a signature σ and define a class $\mathcal{B}$ of σ-behaviors. Recall that both an automata model and MSO logic are already determined by σ.

For a set $\mathcal{F} \subseteq \mathrm{MSO}(\sigma)$, we will then state two kinds of results: When we write *σ-automata and $\mathcal{F}$ are expressively equivalent over $\mathcal{B}$,* we mean that, for all $L \subseteq \mathcal{B}$, the following statements are equivalent:

– There is a σ-automaton $\mathcal{A}$ such that $L(\mathcal{A}) \cap \mathcal{B} = L$.
– There is sentence $\varphi \in \mathcal{F}$ such that $L(\varphi) \cap \mathcal{B} = L$.

In all the cases that we consider here, the transformations from formulas to automata, and back, are effective. When we write *emptiness of σ-automata over $\mathcal{B}$ is decidable,* we mean that one can decide whether, for some given σ-automaton $\mathcal{A}$, we have $L(\mathcal{A}) \cap \mathcal{B} = \emptyset$. We do not go into complexity considerations here, as this would require a more detailed treatment of $\mathcal{B}$.

Note that an effective translation of logic formulas into automata that have a decidable emptiness problem usually allows us to solve, positively, the model-checking problem (provided the logic is closed under negation and automata are closed under intersection).

2 Finite Automata

To illustrate the general framework, we first recall the basic setting of finite automata running on finite words. Apart from the finite alphabet Σ, the corresponding signature σ_{w} will just contain one single relation symbol succ to represent the direct successor relation in a word, i.e., $R = \{\mathsf{succ}\}$. Thus, a *word (structure)* is a σ_{w}-behavior $(\{1, \ldots, n\}, \lambda, \lhd_{\mathsf{succ}})$, with $n \geq 1$, such that $\lhd_{\mathsf{succ}} = \{(e, e+1) \mid e \in \{1, \ldots, n-1\}\}$. With this definition, a σ_{w}-automaton running on words is just a *finite automaton*: in terms of $\lhd_{\mathsf{succ}}$, it can only access the *current* state, i.e., the state assigned to the previous position. In our framework, the famous classical connection between finite automata and MSO logic reads as follows:

Theorem 1 (Büchi-Elgot-Trakhtenbrot [9, 11, 22]). *Finite automata (i.e., σ_{w}-automata),* $\mathrm{EMSO}(\sigma_{\mathrm{w}})$*, and* $\mathrm{MSO}(\sigma_{\mathrm{w}})$ *are expressively equivalent over words.*

3 Class Memory Automata

In this section, we consider systems that consist of an unbounded number of processes. An execution of such a system is naturally described as a *data word*. Usually, a data word is defined as a word over an infinite alphabet, where the latter is used to represent an unbounded number of process identifiers. In our framework, however, it will be more convenient to equip a word over a *finite* alphabet with an equivalence relation, where an equivalence class captures those positions that are executed by one and the same process. The signature σ_{dw} contains, apart from an arbitrary finite alphabet Σ, the relation symbols succ (with the same meaning as in words) and class (connecting *consecutive* positions in an equivalence class). Thus, $R = \{\mathsf{succ}, \mathsf{class}\}$. The idea is that a σ_{dw}-automaton can access a sort of global state (in terms of $\lhd_{\mathsf{succ}}$) and the current local state of the executing process (in terms of $\lhd_{\mathsf{class}}$). Note that the acceptance condition of σ_{dw} actually allows us to fix a set of global final states and a set of local final states.

Accordingly, a *data word* is a σ_{dw}-behavior $B = (\{1, \ldots, n\}, \lambda, \lhd_{\mathsf{succ}}, \lhd_{\mathsf{class}})$, with $n \geq 1$, such that $\lhd_{\mathsf{succ}} = \{(e, e+1) \mid e \in \{1, \ldots, n-1\}\}$. Note that $\lhd_{\mathsf{class}}^{*} \cup (\lhd_{\mathsf{class}}^{-1})^{*}$ is the equivalence relation on $\{1, \ldots, n\}$ induced by $\lhd_{\mathsf{class}}$. A

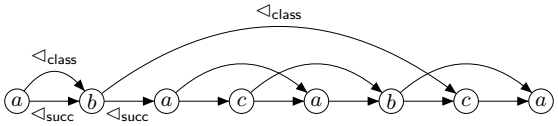

Fig. 1. A data word

data word over $\Sigma = \{a, b, c\}$ is depicted in Figure 1. The straight arrows denote succ-edges, and the curved arrows denote class-edges. Running on data words, σ_{dw}-automata actually correspond to *class memory automata* [4].

Theorem 2 (Bojanczyk et al. [5]). *Class memory automata (i.e., σ_{dw}-automata) and $\mathrm{EMSO}_2(\sigma_{\mathrm{dw}} + \lhd^*_{\mathsf{succ}} + \lhd^*_{\mathsf{class}})$ are expressively equivalent over data words. Moreover, emptiness of σ_{dw}-automata is decidable over data words.*

Theorem 2 was actually shown for the model of *data automata*, which are expressively equivalent to class memory automata [4]. Note that, over data words, the logic $\mathrm{EMSO}_2(\sigma_{\mathrm{dw}} + \lhd^*_{\mathsf{succ}} + \lhd^*_{\mathsf{class}})$ (and, therefore, σ_{dw}-automata) is not closed under negation/complementation [5]. However, for model checking, we can still use its first-order fragment.

4 Nested-Word Automata

We will now consider a setting with a fixed finite set of *recursive* processes, which are usually modeled as pushdown automata or, equivalently, *nested-word automata*. Nested-word automata have access to binary nesting (or call-return) relations, which link a function call with the corresponding return position. Since we have several processes, this gives rise to the notion of *multiply nested words*, which come with one nesting relation per process.

Formally, we assume a finite set $\mathit{Proc} = \{1, \ldots, m\}$ of *processes* with $m \geq 1$. The signature σ_{nw} consists of any finite alphabet Σ as well as the relation symbols $R = \{\mathsf{succ}, \mathsf{cr}_1, \ldots, \mathsf{cr}_m\}$. Then, a *(multiply) nested word* is a σ_{nw}-behavior $B = (\{1, \ldots, n\}, \lambda, \lhd_{\mathsf{succ}}, \lhd_{\mathsf{cr}_1}, \ldots, \lhd_{\mathsf{cr}_m})$, with $n \geq 1$, such that

- $\lhd_{\mathsf{succ}} = \{(e, e+1) \mid e \in \{1, \ldots, n-1\}\}$ (as usual),
- for each $p \in \mathit{Proc}$, the relation $\lhd_{\mathsf{cr}_p}$ is well-nested: if $e \lhd_{\mathsf{cr}_p} f$, $e' \lhd_{\mathsf{cr}_p} f'$, and $e < e' < f$, then $f' < f$ (where $<$ is the canonical strict total order on $\{1, \ldots, n\}$),
- B is disconnected wrt. $\{\mathsf{cr}_1, \ldots, \mathsf{cr}_m\}$.

A nested word with $m = 2$ and $\Sigma = \{a, b, c\}$, is depicted in Figure 2.

Theorem 3 (Alur-Madhusudan [2]). *Suppose $m = 1$, i.e., there is only one nesting relation. In that case, nested-word automata (i.e., σ_{nw}-automata) and $\mathrm{MSO}(\sigma_{\mathrm{nw}})$ are expressively equivalent over nested words. Moreover, emptiness of σ_{nw}-automata is decidable over nested words.*

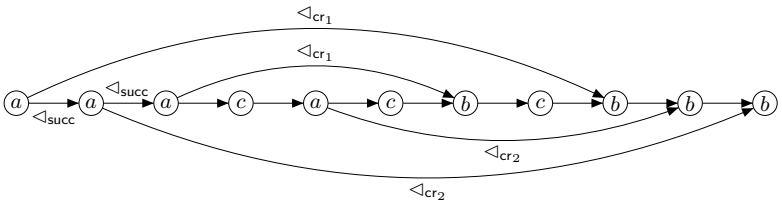

Fig. 2. A nested word

Theorem 4. *Suppose $m \geq 2$. Then,* $\mathrm{MSO}(\sigma_{\mathrm{nw}})$ *is strictly more expressive than σ_{nw}-automata over nested words [6]. Moreover, emptiness of σ_{nw}-automata is undecidable over nested words (as one can easily simulate a Minsky machine).*

Theorem 5. *Suppose $m = 2$. Then, σ_{nw}-automata and* $\mathrm{EMSO}(\sigma_{\mathrm{nw}})$ *are expressively equivalent over nested words [6].*

To recover a robust automata model in the presence of multiple stacks/nesting relations, a fruitful approach has been to restrict (i.e., under-approximate) the set of possible behaviors. We will present only one restriction here. However, note that a variety of other restrictions have been considered in the literature, which essentially lead to the same positive results [3, 10, 15–18].

Namely, we impose a bound on the number $k \geq 1$ of *phases* that a nested word may traverse. In a phase, only one dedicated process is allowed to perform a return/pop. Let $B = (\{1, \ldots, n\}, \lambda, \lhd_{\mathsf{succ}}, \lhd_{\mathsf{cr}_1}, \ldots, \lhd_{\mathsf{cr}_m})$ be a nested word. An *interval* of B is a set of events of the form $\{e, e+1, \ldots, f\}$ where $e \leq f$. An interval I is called a *phase* if, for all $e, e' \in \{1, \ldots, n\}$, $f, f' \in I$, and $p, p' \in \mathit{Proc}$ such that $e \lhd_{\mathsf{cr}_p} f$ and $e' \lhd_{\mathsf{cr}_{p'}} f'$, we have $p = p'$. Finally, B is *k-phase-bounded* if there are phases $I_1, \ldots, I_k$ of B such that $I_1 \cup \ldots \cup I_k = \{1, \ldots, n\}$. The nested word from Figure 2 is 2-phase-bounded, witnessed by the phases $\{1, \ldots, 9\}$ and $\{10, 11\}$.

Theorem 6 (La Torre-Madhusudan-Parlato [14]). *Let $k \geq 1$. Then, σ_{nw}-automata and* $\mathrm{MSO}(\sigma_{\mathrm{nw}})$ *are expressively equivalent over k-phase-bounded nested words. Moreover, emptiness of σ_{nw}-automata is decidable over k-phase-bounded nested words.*

5 Asynchronous Automata

In this section, we deal with *asynchronous automata* [23], whose behaviors are *Mazurkiewicz traces*. As opposed to the previous models, asynchronous automata have a rather distributed flavor, since we will no longer assume that events of a behavior are totally ordered in terms of some relation $\lhd_{\mathsf{succ}}$. We fix a finite set of processes $\mathit{Proc} = \{1, \ldots, m\}$, $m \geq 1$, and a nonempty finite set A. Let us define the signature σ_{t}. The alphabet Σ consists of all pairs (a, P) where $a \in A$

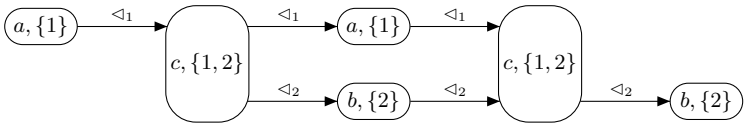

Fig. 3. A trace

and $P \subseteq \mathit{Proc}$ is a nonempty set of processes. The idea is that P contains those processes that are involved in the execution of an event. This may indeed model common access to a shared resource. Moreover, we set $R = \mathit{Proc}$. For $p \in R$, the relation $\lhd_p$ will connect two *consecutive* events that are executed by process p. Note that we may have $e \lhd_p f$ and $e \lhd_{p'} f$ for distinct processes p and p'.

Consider a σ_{t}-behavior $B = (E, \lambda, \lhd_1, \ldots, \lhd_m)$. For $p \in R$, let $E_p := \{e \in E \mid \lambda(e) = (a, P)$ for some $(a, P) \in \Sigma$ such that $p \in P\}$. Then, B is a *(Mazurkiewicz) trace* if, for all $p \in R$, $\lhd_p$ is the direct-successor relation of a total order on E_p. An example of a trace, where $m = 2$ and $A = \{a, b, c\}$, is depicted in Figure 3.

When running on traces, σ_{t}-automata are actually *asynchronous automata* or *asynchronous cellular automata* [23].

Theorem 7 (Thomas [19]). *Asynchronous automata (i.e., σ_{t}-automata) are expressively equivalent to* $\mathrm{MSO}(\sigma_{\mathrm{t}})$ *over traces.*

Note that emptiness of σ_{t}-automata over traces is easily shown decidable, since asynchronous automata are essentially finite-state systems.

6 Message-Passing Automata

The last model that we consider adopts the message-passing paradigm and goes back to Brand and Zafiropulo [8]. It is the natural counterpart of asynchronous automata in a message-passing environment. Again, we fix a finite set of processes $\mathit{Proc} = \{1, \ldots, m\}$ (but now assuming $m \geq 2$). The set Proc determines the set $\mathit{Ch} = \{(p, q) \in \mathit{Proc} \times \mathit{Proc} \mid p \neq q\}$ of *channels*, which we assume to be reliable, FIFO, and (a priori) unbounded. The set of labels is $\Sigma = \{p!q \mid (p, q) \in \mathit{Ch}\} \cup \{q?p \mid (p, q) \in \mathit{Ch}\}$. Here, $p!q$ will label an event of process p that sends a message to q. The complementary receive event is then labeled by $q?p$. The behaviors that we consider have two kinds of edges. For $p \in \mathit{Proc}$, the relation $\lhd_p$ connects consecutive events executed by p. Moreover, for $(p, q) \in \mathit{Ch}$, the relation $\lhd_{(p,q)}$ links a send event with a receive event, which models the exchange of a message sent from p to q through the channel (p, q). Thus, our signature σ_{msc} assumes the set of relation symbols $R = \mathit{Proc} \cup \mathit{Ch}$.

Let $B = (E, \lambda, (\lhd_r)_{r \in R})$ be a σ_{msc}-behavior. For $p \in \mathit{Proc}$, we set $E_p := \{e \in E \mid$ there is $q \in \mathit{Proc} \setminus \{p\}$ such that $\lambda(e) \in \{p!q, p?q\}\}$. We call B a *message sequence chart* if

– for every $p \in \mathit{Proc}$, $\lhd_p$ is the direct-successor relation of a total order on E_p,

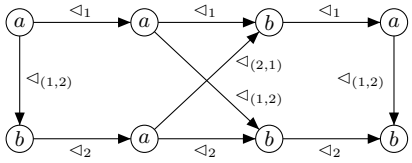

Fig. 4. A message sequence chart

- for all $(p, q) \in Ch$ and $(e, f) \in \lhd_{(p,q)}$, we have $\lambda(e) = p!q$ and $\lambda(f) = q?p$,
- for all $(p, q) \in Ch$ and $(e, f), (e', f') \in \lhd_{(p,q)}$, we have $e \lhd_p^* e'$ iff $f \lhd_q^* f'$ (which models FIFO),
- for every $e \in E$, there are $(p, q) \in Ch$ and $f \in E$ such that $e \lhd_{(p,q)} f$ or $f \lhd_{(p,q)} e$ (i.e., every event is either a send or a receive event), and
- B is disconnected wrt. Ch.

Figure 4 depicts a message sequence chart where $m = 2$.

A σ_{msc}-automaton running over message sequence charts is essentially a *message-passing automaton*, aka *communicating automaton* or *communicating finite-state machine* [8].

Theorem 8. *Message-passing automata (i.e., σ_{msc}-automata) are expressively equivalent to* $\mathrm{EMSO}(\sigma_{\mathrm{msc}})$ *over message sequence charts, but strictly less expressive than* $\mathrm{MSO}(\sigma_{\mathrm{msc}})$ *[7]. Emptiness of σ_{msc}-automata over message sequence charts is undecidable [8].*

It is an open problem whether message-passing automata are expressively equivalent to $\mathrm{EMSO}(\sigma_{\mathrm{msc}} + \lhd_1^* + \ldots + \lhd_m^*)$ over message sequence charts.

Again, restricting the set of message sequence charts further, one obtains positive results wrt. MSO logic and model-/emptiness-checking. The restrictions we consider here rely on the notion of a *linearization* of a message sequence chart $B = (E, \lambda, (\lhd_r)_{r \in R})$, which is any total order $\preceq$ on E such that $(\bigcup_{r \in R} \lhd_r)^* \subseteq \preceq$. For $k \geq 1$, linearization $\preceq$ is called *k-bounded*, if, for all $(p, q) \in Ch$,

$$\max_{g \in E} |\{(e, f) \in \lhd_{(p,q)} \mid e \preceq g \prec f\}| \leq k\,.$$

Intuitively, along the linearization, there are, at any time and in any channel, no more than k pending messages. We call B *universally k-bounded* if every of its linearizations is k-bounded. Accordingly, we call B *existentially k-bounded* if at least one of its linearizations is k-bounded. The message sequence chart from Figure 4 is universally 2-bounded (but not universally 1-bounded), and it is existentially 1-bounded.

Theorem 9 (Henriksen-Mukund-Narayan Kumar-Sohoni-Thiagarajan [13]). *Let $k \geq 1$. Message-passing automata (i.e., σ_{msc}-automata) are expressively equivalent to* $\mathrm{MSO}(\sigma_{\mathrm{msc}})$ *over universally k-bounded message sequence charts. Moreover, emptiness of σ_{msc}-automata over universally k-bounded message sequence charts is decidable.*

Theorem 10 (Genest-Kuske-Muscholl [12]). *Let $k \geq 1$. Message-passing automata (i.e., σ_{msc}-automata) are expressively equivalent to* $\mathrm{MSO}(\sigma_{\mathrm{msc}})$ *over existentially k-bounded message sequence charts. Moreover, emptiness of σ_{msc}-automata over existentially k-bounded message sequence charts is decidable.*

7 Conclusion

Note that many more automata models enjoy logical characterizations in terms of (fragments of) MSO logic, such as tree automata or certain automata running on graphs [20]. In this paper, however, we focused on automata that may serve as models of concurrent systems and whose semantics is a set of possible executions. The apparent similarities between these models suggest that more general meta results may be possible. It would be interesting to identify signatures σ and general classes of σ-behaviors for which σ-automata and $\mathrm{MSO}(\sigma)$ (or suitable fragments) are expressively equivalent.

References

1. C. Aiswarya and P. Gastin. Reasoning about distributed systems: WYSIWYG. In *FSTTCS'14*, volume 29 of *Leibniz International Proceedings in Informatics*, pages 11–30. Leibniz-Zentrum für Informatik, 2014.
2. R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):1–43, 2009.
3. M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In *DLT'08*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.
4. H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.
5. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.
6. B. Bollig. On the expressive power of 2-stack visibly pushdown automata. *Logical Methods in Computer Science*, 4(4:16), 2008.
7. B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, 358(2-3):150–172, 2006.
8. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.
9. J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.
10. A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR'12*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.
11. C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–52, 1961.
12. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6):920–956, 2006.

13. J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005.
14. S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE Computer Society Press, 2007.
15. S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. In *LATIN'08*, volume 6034 of *LNCS*, pages 96–107. Springer, 2010.
16. S. La Torre, M. Napoli, and G. Parlato. Scope-bounded pushdown languages. In *DLT'14*, volume 8633 of *LNCS*, pages 116–128. Springer, 2014.
17. S. La Torre, M. Napoli, and G. Parlato. A unifying approach for multistack pushdown automata. In *MFCS'14*, volume 8634 of *LNCS*, pages 377–389. Springer, 2014.
18. P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL'11*, pages 283–294. ACM, 2011.
19. W. Thomas. On logical definability of trace languages. In *Proceedings of Algebraic and Syntactic Methods in Computer Science (ASMICS)*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.
20. W. Thomas. Elements of an automata theory over partial orders. In *POMIV'96*, volume 29 of *DIMACS*. AMS, 1996.
21. W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.
22. B. A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J*, 3:103–131, 1962. In Russian; English translation in *Amer. Math. Soc. Transl.* 59, 1966, 23–55.
23. W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.