

# Replaying Play In and Play Out: Synthesis of Design Models from Scenarios by Learning

Benedikt Bollig<sup>1</sup>, Joost-Pieter Katoen<sup>2</sup>, Carsten Kern<sup>2</sup>, and Martin Leucker<sup>3</sup>

<sup>1</sup> LSV, CNRS UMR 8643 & ENS de Cachan, France

<sup>2</sup> Software Modeling and Verification Group, RWTH Aachen University, Germany

<sup>3</sup> Institut für Informatik, TU München, Germany

**Abstract.** This paper is concerned with bridging the gap between requirements, provided as a set of scenarios, and conforming design models. The novel aspect of our approach is to exploit *learning* for the synthesis of design models. In particular, we present a procedure that infers a message-passing automaton (MPA) from a given set of positive and negative scenarios of the system's behavior provided as message sequence charts (MSCs). The paper investigates which classes of regular MSC languages and corresponding MPA can (not) be learned, and presents a dedicated tool based on the learning library *LearnLib* that supports our approach.

## 1 Introduction

The elicitation of requirements is the main initial phase in the typical software engineering development cycle. A plethora of elicitation techniques for requirement engineering exist. Popular requirement engineering methods, such as the Inquiry Cycle and CREWS [26], exploit use cases and scenarios to specify the system's requirements. Sequence diagrams are also at the heart of the UML. A scenario is a partial fragment of the system's behavior, describing the system components, their message exchange and concurrency. Their intuitive yet formal nature has resulted in a broad acceptance. Scenarios can be positive or negative, indicating a desired or unwanted system behavior, respectively. Different scenarios together form a more complete description of the system behavior.

The following design phase in software engineering is a major challenge as it is concerned with a paradigm shift between the *requirement* specification—a partial, overlapping and possibly inconsistent description of the system's behavior—and a conforming *design model*, a complete behavioral description of the system (at a high level of abstraction). During the synthesis of design models, usually automata-based models that are focused on intra-agent communication, conflicting requirements will be detected and need to be resolved. Typical resulting changes to requirements specifications include adding or deleting scenarios, and fixing errors that are found by a thorough analysis (e.g., model checking) of the design model. Obtaining a complete and consistent set of requirements together with a related design model is thus a highly iterative process.

This paper proposes a novel technique that is aimed to be an important stepping stone towards bridging the gap between scenario-based requirement specifications and design models. The novel aspect of our approach is to exploit *learning* algorithms for the synthesis of *distributed* design models from scenario-based specifications. Since

message-passing automata (MPA, for short) [10] are a commonly used model to realize the behavior as described by scenarios, we adopt MPA as design model. We present a procedure that interactively infers an MPA from a given set of positive and negative scenarios of the system's behavior provided as message sequence charts (MSCs). This is achieved by generalizing Angluin's learning algorithm for deterministic finite-state automata (DFA) [4] towards specific classes of bounded MPA, i.e., MPA that can be used to realize MSCs with channels of finite capacity. An important distinctive aspect of our approach is that it naturally supports the *incremental generation* of design models. Learning of initial sets of scenarios is feasible. On adding or deletion of scenarios, MPA are adapted accordingly in an automated manner. Thus, synthesis phases and analysis phases, supported by simulation or analysis tools such as *MSCan* [7], complement each other in a natural fashion. Furthermore, on establishing the inconsistency of a set of scenarios, our approach mechanically provides *diagnostic feedback* (in the form of a counterexample) that can guide the engineer to evolve his requirements. This paper investigates which classes of regular MSC languages and corresponding MPA can (not) be learned, and presents *Smyle*, a dedicated tool based on the learning library *LearnLib* [27] that supports our approach.

Generating automata-based models from scenarios has received a lot of attention. These works include algorithms to generate statechart models from MSCs [19], formalization and undecidability results for the synthesis for a simple variant of live sequence charts (LSCs) [9], and Harel's play-in, play-out approach for LSCs [11,12]. Another approach is proposed by Alur *et al.* in [2,3]. Uchitel *et al.* [29] present an algorithm for synthesizing transition systems from high-level MSCs. An executable variant of LSCs, triggered MSCs, are presented in [28]. All approaches are based on a rather complete, well-elaborated specification of the system to be, such as MSCs with loops or conditions, high-level MSCs, triggered MSCs, or LSCs, whereas for our synthesis approach only simple MSCs have to be provided as examples, simplifying the requirements specification task.

Applying learning yields an incremental approach, and facilitates the generation of diagnostic feedback. An alternative approach to using learning for inferring design models from UML sequence diagrams has been proposed in [23]. This approach focuses on learning DFA (from words) representing the global system and only considers synchronous communication. The use of learning for model-based testing in [18] has similar characteristics. Using our technique, collections of MSCs (in fact, partial orders or words with partial commutation) are learned and yield an MPA that explicitly reflects the composite structure of the system together with the asynchronous message exchange between the individual components.

After an introduction into MSCs and MPA (Sections 2 and 3), we formally define the general learning setting and describe the extension of Angluin's learning algorithm, cf. Section 4. We then consider existentially and universally bounded MPA, i.e., MPA for which some (all) possible event orderings can be realized with finite channels. It is shown (in Section 5) that universally bounded MPA and safe product MPA, as well as existentially bounded MPA with an a priori fixed channel capacity are learnable. Section 6 presents the basic functionality of our tool and some initial case study results.

## 2 Message Sequence Charts

Let  $\Sigma^*$  denote the set of finite words over a finite alphabet  $\Sigma$ . A  $\Sigma$ -labeled partial order is a triple  $\mathcal{P} = (E, \leq, \ell)$  where  $E$  is a finite set,  $\leq$  is a partial-order relation on  $E$ , i.e., it is reflexive, transitive, and antisymmetric, and  $\ell : E \rightarrow \Sigma$  is a labeling function. A linearization of  $\mathcal{P}$  is an extension  $(E, \leq', \ell)$  of  $\mathcal{P} = (E, \leq, \ell)$  such that  $\leq' \supseteq \leq$  is a total order. As we will consider partial orders up to isomorphism, the set of linearizations of  $\mathcal{P}$ , denoted  $Lin(\mathcal{P})$ , is a subset of  $\Sigma^*$ .

We fix a finite set  $Proc$  of at least two processes, which exchange messages from a finite set  $Msg$  by executing communication actions. Let  $Ch$  denote the set  $\{(p, q) \mid p, q \in Proc, p \neq q\}$  of reliable FIFO channels. For  $p \in Proc$ ,  $Act_p$  denotes the set of actions of  $p$ , i.e.,  $\{!(p, q, a) \mid (p, q) \in Ch \text{ and } a \in Msg\} \cup \{?(q, p, a) \mid (p, q) \in Ch \text{ and } a \in Msg\}$ . The action  $!(p, q, a)$  is to be read as “ $p$  sends the message  $a$  to  $q$ ”, while  $?(q, p, a)$  is the complementary action of receiving  $a$  sent from  $p$  to  $q$  (which is thus executed by  $q$ ). Moreover, let  $Act = \bigcup_{p \in Proc} Act_p$ .

**Definition 1 (Message Sequence Chart (MSC)).** An MSC (over  $Proc$  and  $Msg$ ) is a structure  $(E, \{\leq_p\}_{p \in Proc}, <_{msg}, \ell)$  with:

- $E$  is a finite set of events,
- $\ell : E \rightarrow Act$  is a labeling function,
- for any  $p \in Proc$ ,  $\leq_p$  is a total order on  $E_p = \ell^{-1}(Act_p)$ ,
- $<_{msg} \subseteq E \times E$  such that, for any  $e \in E$ ,  $e <_{msg} e'$  or  $e' <_{msg} e$  for some  $e' \in E$ , and, for any  $(e_1, e'_1) \in <_{msg}$ , there are  $p, q \in Proc$  and  $a \in Msg$  satisfying:
  - $\ell(e_1) = !(p, q, a)$  and  $\ell(e'_1) = ?(q, p, a)$ ,
  - for any  $(e_2, e'_2) \in <_{msg}$  with  $\ell(e_2) = !(p, q, b)$  for some  $b \in Msg$ :  $e_1 \leq_p e_2$  iff  $e'_1 \leq_q e'_2$  (which guarantees FIFO behavior), and
  - $\leq = (<_{msg} \cup \bigcup_{p \in Proc} \leq_p)^*$  is a partial-order relation on  $E$ .

Let  $M = (E, \{\leq_p\}_{p \in Proc}, <_{msg}, \ell)$  be an MSC. A prefix of  $M$  is a structure  $(E', \{\leq'_p\}_{p \in Proc}, <'_{msg}, \ell')$  such that  $E' \subseteq E$  with  $e \in E'$  and  $e' \leq e$  implies  $e' \in E'$ ,  $\leq'_p = \leq_p \cap (E' \times E')$  for any  $p \in Proc$ ,  $<'_msg = <_{msg} \cap (E' \times E')$ , and  $\ell'$  is the restriction of  $\ell$  to  $E'$ . We write  $P \preceq M$  if  $P$  is a prefix of the MSC  $M$ .

The set of MSCs is denoted by  $\mathbb{MSC}$ . A set of MSCs,  $\mathcal{L} \subseteq \mathbb{MSC}$ , is called an MSC language. For  $\mathcal{L} \subseteq \mathbb{MSC}$ , we let  $Pref(\mathcal{L})$  denote  $\{P \mid P \preceq M \text{ for some } M \in \mathcal{L}\}$  (a similar notation will be used in the context of words). Note that  $\mathbb{MSC} \subseteq Pref(\mathbb{MSC})$ .

Let  $M = (E, \{\leq_p\}_{p \in Proc}, <_{msg}, \ell) \in \mathbb{MSC}$ . We set  $Lin(M)$  to be  $Lin((E, \leq, \ell))$  (canonically extended for prefixes of  $M$ ); the linearizations of  $\mathcal{L} \subseteq \mathbb{MSC}$  are defined by  $Lin(\mathcal{L}) = \bigcup_{M \in \mathcal{L}} Lin(M)$ . Note that  $\mathcal{L} \subseteq \mathbb{MSC}$  is uniquely determined by  $Lin(\mathcal{L})$ , i.e., for any  $\mathcal{L}, \mathcal{L}' \subseteq \mathbb{MSC}$ ,  $Lin(\mathcal{L}) = Lin(\mathcal{L}')$  implies  $\mathcal{L} = \mathcal{L}'$ . A word  $w \in Act^*$  is an MSC word if  $w \in Lin(M)$  for some  $M \in \mathbb{MSC}$ ; for  $B \in \mathbb{N}$ ,  $w$  is  $B$ -bounded if, for any prefix  $v$  of  $w$  and any  $(p, q) \in Ch$ ,  $\sum_{a \in Msg} |v|_{!(p, q, a)} - \sum_{a \in Msg} |v|_{?(q, p, a)} \leq B$  where  $|v|_\sigma$  denotes the number of occurrences of  $\sigma$  in  $v$ . For  $B \in \mathbb{N}$ , let  $Lin^B(M)$  denote  $\{w \in Lin(M) \mid w \text{ is } B\text{-bounded}\}$ , and  $Lin^B(\mathcal{L}) = \bigcup_{M \in \mathcal{L}} Lin^B(M)$  for  $\mathcal{L} \subseteq \mathbb{MSC}$ .

**Definition 2 (Boundedness).** Let  $M \in \mathbb{MSC}$ . We call  $M$  universally  $B$ -bounded (i.e.,  $\forall B$ -bounded) if  $Lin(M) = Lin^B(M)$ . We call it existentially  $B$ -bounded (i.e.,  $\exists B$ -bounded) if  $Lin(M) \cap Lin^B(M) \neq \emptyset$ .

The sets of  $\forall B$ -bounded MSCs and  $\exists B$ -bounded MSCs are denoted by  $\text{MSC}_{\forall B}$  and  $\text{MSC}_{\exists B}$ , respectively. In an  $\exists B$ -bounded MSC, the events can be scheduled such that, during its execution, any channel contains at most  $B$  messages. In a  $\forall B$ -bounded MSC, any scheduling is within the channel bound  $B$ . A set  $\mathcal{L} \subseteq \text{MSC}$  is  $\forall B$ -bounded if  $\mathcal{L} \subseteq \text{MSC}_{\forall B}$ , and  $\exists B$ -bounded if  $\mathcal{L} \subseteq \text{MSC}_{\exists B}$ . It is  $\forall/\exists$ -bounded if it is  $\forall B/\exists B$ -bounded for some  $B \in \mathbb{N}$ , respectively.

*Example 1.* The MSC word  $w = !(1, 2, \text{req}) !(1, 2, \text{req}) ?(2, 1, \text{req})^4 ?(2, 1, \text{req})$  is in  $\text{Lin}(M)$  with  $M$  the MSC from Fig. 1c. Note that  $w$  is 2-bounded, but not 1-bounded. But  $M$  has a 1-bounded linearization, and  $\text{Lin}^1(M) = \{!(1, 2, \text{req}) ?(2, 1, \text{req})^5\}$ . In fact,  $M$  is  $\exists 1$ -bounded and  $\forall B$ -bounded for  $B \geq 5$ . The MSC in Fig. 1a is  $\forall 4$ -bounded and thus  $\exists 4$ -bounded. It is even  $\exists 2$ -bounded, but not  $\exists 1$ -bounded. The MSC in Fig. 1b is  $\forall 3$ - and  $\exists 1$ -bounded, but not  $\forall 2$ -bounded. Finally, we note that the set of MSCs where arbitrarily many messages are sent from 1 to 2 is  $\exists 1$ -bounded, but not  $\forall$ -bounded.

### 3 Message-Passing Automata

An MPA [10] is a collection of finite automata (called processes) that share a single global initial state and a set of global final states. Bilateral communication between the processes takes place via unbounded reliable FIFO buffers. Process transitions are labeled with send or receive actions. Action  $!(p, q, a)$  puts the message  $a$  at the end of the channel from  $p$  to  $q$ . Receive actions are enabled only if the requested message is found at the head of the channel. The expressive power of MPA is extended by allowing components to exchange *synchronization messages*.

**Definition 3 (Message-passing automaton (MPA)).** An MPA is a tuple  $((\mathcal{A}_p)_{p \in \text{Proc}}, \text{Sync}, \bar{s}^{\text{in}}, F)$  with:

- *Sync* is a nonempty finite set of synchronization messages,
- for each  $p \in \text{Proc}$ ,  $\mathcal{A}_p$  is a pair  $(S_p, \Delta_p)$  where  $S_p$  is a finite set of local states and  $\Delta_p \subseteq S_p \times \text{Act}_p \times \text{Sync} \times S_p$  is a set of local transitions,
- $\bar{s}^{\text{in}} \in S_A = \prod_{p \in \text{Proc}} S_p$  is the global initial state, and
- $F \subseteq S_A$  is a set of global final states.

As in [17,24], we consider the linearizations of MSCs that are obtained from the global automaton induced by an MPA. For an MPA  $\mathcal{A} = ((\mathcal{A}_p)_{p \in \text{Proc}}, \text{Sync}, \bar{s}^{\text{in}}, F)$ , where  $\mathcal{A}_p = (S_p, \Delta_p)$ , this global automaton is defined as follows. The set of *configurations* of  $\mathcal{A}$ , denoted by  $\text{Conf}_{\mathcal{A}}$ , consists of pairs  $(\bar{s}, \chi)$  with  $\bar{s} \in S_A$  and  $\chi : \text{Ch} \rightarrow (\text{Msg} \times \text{Sync})^*$ , indicating the channel contents. The *global transition relation* of  $\mathcal{A}$ ,  $\Longrightarrow_{\mathcal{A}} \subseteq \text{Conf}_{\mathcal{A}} \times \text{Act} \times \text{Sync} \times \text{Conf}_{\mathcal{A}}$ , is defined by the following two inference rules ( $\bar{s}[p]$  refers to the  $p$ -component of a global state  $\bar{s} \in S_A$ ):

$$\frac{(\bar{s}[p], !(p, q, a), m, \bar{s}'[p]) \in \Delta_p \quad \wedge \quad \text{for all } r \neq p, \bar{s}[r] = \bar{s}'[r]}{((\bar{s}, \chi), !(p, q, a), m, (\bar{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}}$$

where  $\chi' = \chi[(p, q) := (a, m) \cdot \chi((p, q))]$ , i.e.,  $\chi'$  maps  $(p, q)$  to the concatenation of  $(a, m)$  and  $\chi((p, q))$ ; for all other channels, it coincides with  $\chi$ .

$$\frac{(\bar{s}[p], ?(p, q, a), m, \bar{s}'[p]) \in \Delta_p \quad \wedge \quad \text{for all } r \neq p, \bar{s}[r] = \bar{s}'[r]}{((\bar{s}, \chi), ?(p, q, a), m, (\bar{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}}$$

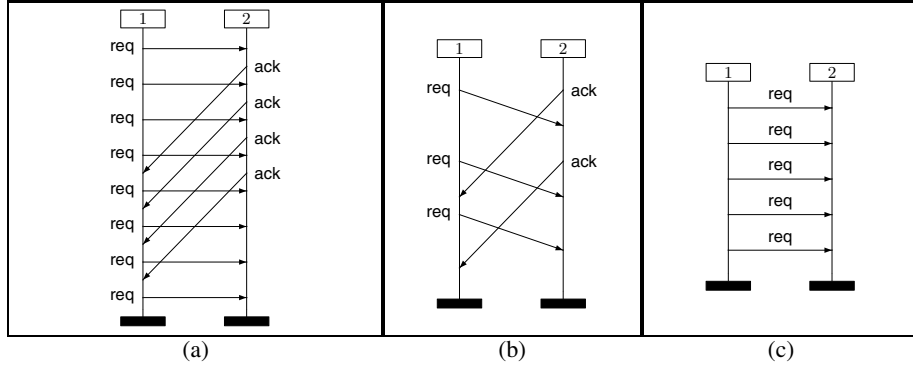


Fig. 1. Example message sequence charts

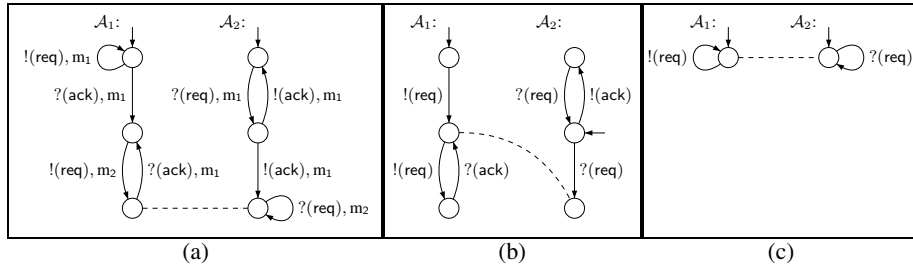


Fig. 2. Example message-passing automata

where  $\chi((q, p)) = w \cdot (a, m)$  and  $\chi' = \chi[(q, p) := w]$ . The initial and final configurations of the global automaton are  $(\bar{s}^{in}, \chi_\varepsilon)$  and  $F \times \{\chi_\varepsilon\}$ , respectively, where  $\chi_\varepsilon$  maps each channel onto the empty word.

Now MPA  $\mathcal{A}$  defines the word language  $L(\mathcal{A}) \subseteq Act^*$ , i.e., the set of words accepted by the global automaton of  $\mathcal{A}$  while ignoring synchronization messages. The MSC language of  $\mathcal{A}$ , denoted by  $\mathcal{L}(\mathcal{A})$ , is the (unique) set  $\mathcal{L}$  of MSCs such that we have  $Lin(\mathcal{L}) = L(\mathcal{A})$ . The notions of boundedness on MSCs carry over to MPA in a natural way, e.g., MPA  $\mathcal{A}$  is  $\forall$ -bounded if its MSC language is  $\forall$ -bounded. The set of  $\forall$ -bounded and  $\exists B$ -bounded MPA is denoted by  $MPA_\forall$  and  $MPA_{\exists B}$ , respectively.

*Example 2.* Fig. 2a shows a not  $\exists$ -bounded MPA with set of synchronization messages  $\{m_1, m_2\}$  (and simplified action alphabet). Its only global final state is indicated by a dashed line. Its MSC language, which contains MSCs such as in Fig. 1a, cannot be recognized with less than two synchronization messages. For the MPA in Fig. 2b, specifying a part of the alternating-bit protocol (ABP), a single synchronization message suffices (which is therefore omitted). It is  $\forall 3$ -bounded (cf. Fig. 1b). The MPA in Fig. 2c has no synchronization messages either. Its accepted MSCs are as in Fig. 1c and form an  $\exists 1$ -bounded MSC language that, however, is not  $\forall$ -bounded.

An MPA  $\mathcal{A} = ((\mathcal{A}_p)_{p \in Proc}, Sync, \bar{s}^{in}, F)$ , with  $\mathcal{A}_p = (S_p, \Delta_p)$ , is a *product MPA* if  $|Sync| = 1$  and  $F = \prod_{p \in Proc} F_p$  for some  $F_p \subseteq S_p$ ,  $p \in Proc$ . The acceptance

condition is thus *local*, i.e., any process autonomously decides to halt. Moreover, product MPA cannot distinguish between synchronization messages. MSC languages of product MPA are referred to as *realizable* [24,21]. The MPA in Figs. 2b and 2c are product MPA, whereas the MPA in Fig. 2a is not, as it employs two synchronization messages. Actually, the latter has no equivalent product MPA. As for ordinary MPA, the notions of boundedness carry over to product MPA; let  $\text{MPA}_{\forall}^p$  and  $\text{MPA}_{\exists B}^p$  denote the set of  $\forall$ -bounded product and  $\exists B$ -bounded product MPA, respectively. The MPA in Fig. 2b is in  $\text{MPA}_{\forall}^p$ , whereas the MPA in Fig. 2c is in  $\text{MPA}_{\exists 1}^p$ , but not in  $\text{MPA}_{\forall}^p$ .

An MPA is called *deadlock-free* or *safe* if, from any configuration that is reachable from the initial configuration, one can reach a final configuration. The MPA from Figs. 2b and 2c are safe, whereas the MPA depicted in Fig. 2a is not safe. The class of  $\forall$ -bounded safe product MPA is denoted by  $\text{MPA}_{\forall}^{\text{sp}}$ .

## 4 An Extension of Angluin's Algorithm

Angluin's algorithm  $L^*$  [4] is a well-known algorithm for learning deterministic finite state automata (DFA). In this section, we recall the algorithm and extend it towards learning objects that can be *represented* by DFA in a way made precise shortly. This extension allows us to learn various classes of MPA, as described below.

Let us first recall some basic definitions. Let  $\Sigma$  be an alphabet. A deterministic finite automaton (DFA) over  $\Sigma$  is a tuple  $\mathcal{A} = (Q, q_0, \delta, F)$ , where  $Q$  is its finite set of *states*,  $q_0 \in Q$  is the *initial state*,  $\delta : Q \times \Sigma \rightarrow Q$  is its *transition function*, and  $F \subseteq Q$  is the set of *final states*. The language of  $\mathcal{A}$  is defined as usual and denoted by  $L(\mathcal{A})$ .

### 4.1 The Basic Algorithm

A *Learner*, who initially knows nothing about a given DFA  $\mathcal{A}$ , is trying to learn  $\mathcal{A}$  by asking queries to a *Teacher*, who knows  $\mathcal{A}$ . There are two kinds of queries:

- A *membership query* consists in asking whether a string  $w \in \Sigma^*$  is in  $L(\mathcal{A})$ .
- An *equivalence query* consists in asking whether a *hypothesized* DFA  $\mathcal{H}$  is correct, i.e., whether  $L(\mathcal{H}) = L(\mathcal{A})$ . The *Teacher* will answer *yes* if  $\mathcal{H}$  is correct, or else supply a counterexample  $w$ , either in  $L(\mathcal{A}) \setminus L(\mathcal{H})$  or in  $L(\mathcal{H}) \setminus L(\mathcal{A})$ .

The *Learner* maintains a prefix-closed set  $U \subseteq \Sigma^*$  of prefixes, which are candidates for identifying states, and a suffix-closed set  $V \subseteq \Sigma^*$  of suffixes, which are used to distinguish such states. The sets  $U$  and  $V$  are increased when needed during the algorithm. The *Learner* makes membership queries for all words in  $(U \cup U\Sigma)V$ , and organizes the results into a *table*  $T$  which maps each  $u \in (U \cup U\Sigma)$  to a mapping  $T(u) : V \rightarrow \{+, -\}$  where  $+$  represents accepted and  $-$  not accepted. In [4], each function  $T(u)$  is called a *row*. When  $T$  is

- *closed*: for any  $u \in U$  and  $a \in \Sigma$ , there is a  $u' \in U$  with  $T(ua) = T(u')$ , and
- *consistent*: for any  $u, u' \in U$  and  $a \in \Sigma$ ,  $T(u) = T(u')$  implies  $T(ua) = T(u'a)$ ,

the *Learner* constructs a hypothesized DFA  $\mathcal{H} = (Q, q_0, \delta, Q^+)$ , where  $Q = \{T(u) \mid u \in U\}$  is the set of distinct rows,  $q_0$  is the row  $T(\varepsilon)$  (with  $\varepsilon$  denoting the empty word),

$\delta$  is defined by  $\delta(T(u), a) = T(ua)$ , and  $Q^+ = \{T(u) \mid u \in U \text{ and } T(u)(\varepsilon) = +\}$ . After that, the *Learner* submits  $\mathcal{H}$  in an equivalence query. If the answer is *yes*, the learning procedure is completed, otherwise the returned counterexample is used to extend  $U$  and  $V$ , and subsequent membership queries are performed until arriving at a new hypothesized DFA.

## 4.2 Learning Objects Represented by Subclasses of Regular Word Languages

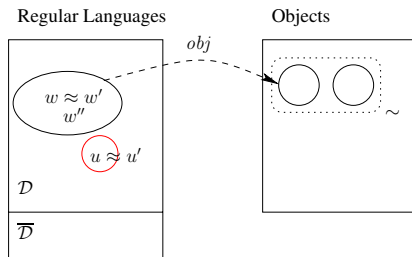
Our goal is to learn MPA from examples given as MSCs. To avail Angluin's algorithm, we need to establish a correspondence between MPA and regular word languages. As we will consider several classes of MPA with corresponding representations in the next section, let us first elaborate on general properties of representations for learning *objects* of a fixed arbitrary set of objects  $\mathcal{O}$ . These objects might be classified into equivalence classes of an equivalence relation  $\sim \subseteq \mathcal{O} \times \mathcal{O}$ . In our setting, the objects will be MPA, and two MPA are considered to be equivalent if they recognize the same MSC language.

We now have to *represent* elements from  $\mathcal{O}$  (or, rather, their equivalence classes) by regular word languages, say over an alphabet  $\Sigma$ . For MPA  $\mathcal{A}$ , we might consider regular languages  $L$  over  $Act$  such that  $L$  corresponds to the set  $Lin(\mathcal{L}(\mathcal{A}))$ . But not every regular word language over  $Act$  gives rise to an MPA. In particular, it might contain words that are not MSC words, i.e., do not correspond to some MSC. Thus, in general, it is necessary to work within a subset  $\mathcal{D}$  of  $\Sigma^*$ , i.e., we learn regular subsets of  $\mathcal{D}$ . For learning MPA, e.g., it is reasonable to set  $\mathcal{D} = Lin(MSC)$ .

It is not always sufficient to restrict to  $\mathcal{D}$  in order to obtain a precise correspondence between  $\mathcal{O}$  and regular word languages. Often, regular word languages are required to be closed under some *equivalence relation* and/or *inference rule*. E.g., an MPA always gives rise to an MSC word language that contains either any linearization of some given MSC, or none. Similarly, languages of product MPA are closed under inference (to be made precise later) imposing similar requirements on the representing language. So let us consider an equivalence relation  $\approx \subseteq \mathcal{D} \times \mathcal{D}$  and, moreover, a relation  $\vdash \subseteq 2^{\mathcal{D}} \times 2^{\Sigma^*}$  where  $L_1 \vdash L_2$  intuitively means that  $L_1$  still requires at least one element from  $L_2$ .

We say that  $L \subseteq \mathcal{D}$  is  *$\approx$ -closed* (or, closed under  $\approx$ ) if, for any  $w, w' \in \mathcal{D}$  with  $w \approx w'$ , we have  $w \in L$  iff  $w' \in L$ . Moreover,  $L$  is said to be  *$\vdash$ -closed* (or, closed under  $\vdash$ ) if, for any  $(L_1, L_2) \in \vdash$ , we have that  $L_1 \subseteq L$  implies  $L \cap L_2 \neq \emptyset$ .<sup>1</sup> Consider Fig. 3. The larger ellipse is closed under  $\approx$  ( $w \approx w'$ ) and under  $\vdash$  (assuming  $\{w, w'\} \vdash \{w''\}$ ), whereas the smaller circle is not.

Naturally,  $\mathcal{D}$ ,  $\approx$ , and  $\vdash$  determine a particular class  $\mathfrak{R}_{\min DFA}(\Sigma, \mathcal{D}, \approx, \vdash) = \{L \subseteq \mathcal{D} \mid L \text{ is regular and closed under both } \approx$



**Fig. 3.** Representing objects by regular languages

<sup>1</sup> Technically,  $\approx$  and  $\vdash$  could be encoded as a single relation. As they will serve a different purpose, we separate them in the general framework, to simplify the forthcoming explanations.

and  $\vdash$  of regular word languages over  $\Sigma$  (where any language is understood to be given by its minimal DFA). Suppose a language of this class  $\mathfrak{R}_{\min\text{DFA}}(\Sigma, \mathcal{D}, \approx, \vdash)$  can be learned in some sense that will be made precise. For learning elements of  $\mathcal{O}$ , we still need to derive an object from a language in  $\mathfrak{R}_{\min\text{DFA}}(\Sigma, \mathcal{D}, \approx, \vdash)$ . To this aim, we suppose a computable bijective mapping  $obj : \mathfrak{R}_{\min\text{DFA}}(\Sigma, \mathcal{D}, \approx, \vdash) \rightarrow [\mathcal{O}]_{\sim} = \{[o]_{\sim} \mid o \in \mathcal{O}\}$  (where  $[o]_{\sim} = \{o' \in \mathcal{O} \mid o' \sim o\}$ ). Again, Fig. 3 illustrates a typical situation.

As Angluin's algorithm works within the class of arbitrary DFA over  $\Sigma$ , its *Learner* might propose DFA whose languages are neither a subset of  $\mathcal{D}$  nor satisfy the closure properties for  $\approx$  and  $\vdash$ . To rule out and fix such hypotheses, the language inclusion problem and the closure properties in question are required to be *constructively decidable*, meaning that they are decidable and if the property fails, a *reason* of its failure can be computed. Now, let us formally define what we understand by a *learning setup*:

**Definition 4.** Let  $\mathcal{O}$  be a set of objects and  $\sim \subseteq \mathcal{O} \times \mathcal{O}$  be an equivalence relation. A learning setup for  $(\mathcal{O}, \sim)$  is a quintuple  $(\Sigma, \mathcal{D}, \approx, \vdash, obj)$  where

- $\Sigma$  is an alphabet,
- $\mathcal{D} \subseteq \Sigma^*$  is the domain,
- $\approx \subseteq \mathcal{D} \times \mathcal{D}$  is an equivalence relation such that, for any  $w \in \mathcal{D}$ ,  $[w]_{\approx}$  is finite,
- $\vdash \subseteq 2^{\mathcal{D}} \times 2^{\Sigma^*}$  such that, for any  $(L_1, L_2) \in \vdash$ ,  $L_1$  is both finite and  $\approx$ -closed, and  $L_2$  is a nonempty decidable language,
- $obj : \mathfrak{R}_{\min\text{DFA}}(\Sigma, \mathcal{D}, \approx, \vdash) \rightarrow [\mathcal{O}]_{\sim}$  is a bijective effective mapping in the sense that, for  $L \in \mathfrak{R}_{\min\text{DFA}}(\Sigma, \mathcal{D}, \approx, \vdash)$ , a representative of  $obj(L)$  can be computed.

Furthermore, we require that the following hold for DFA  $\mathcal{A}$  over  $\Sigma$ :

- (D1) The problem whether  $L(\mathcal{A}) \subseteq \mathcal{D}$  is decidable. If, moreover,  $L(\mathcal{A}) \not\subseteq \mathcal{D}$ , one can compute  $w \in L(\mathcal{A}) \setminus \mathcal{D}$ . We then say that  $\text{INCLUSION}(\Sigma, \mathcal{D})$  is constructively decidable.
- (D2) If  $L(\mathcal{A}) \subseteq \mathcal{D}$ , it is decidable whether  $L(\mathcal{A})$  is  $\approx$ -closed. If not, one can compute  $w, w' \in \mathcal{D}$  such that  $w \approx w'$ ,  $w \in L(\mathcal{A})$ , and  $w' \notin L(\mathcal{A})$ . We then say that the problem  $\text{EQCLOSURE}(\Sigma, \mathcal{D}, \approx)$  is constructively decidable.
- (D3) If  $L(\mathcal{A}) \subseteq \mathcal{D}$  is closed under  $\approx$ , it is decidable whether  $L(\mathcal{A})$  is  $\vdash$ -closed. If not, we can compute  $(L_1, L_2) \in \vdash$  (hereby,  $L_2$  shall be given in terms of a decision algorithm that checks a word for membership) such that  $L_1 \subseteq L(\mathcal{A})$  and  $L(\mathcal{A}) \cap L_2 = \emptyset$ . We then say that  $\text{INFCLOSURE}(\Sigma, \mathcal{D}, \approx, \vdash)$  is constructively decidable.

Let us generalize Angluin's algorithm to cope with the extended setting, and let  $(\Sigma, \mathcal{D}, \approx, \vdash, obj)$  be a learning setup for  $(\mathcal{O}, \sim)$ . The main changes concern the processing of membership queries and the treatment of hypothesized DFA:

- Once a membership query has been processed for a word  $w \in \mathcal{D}$ , queries  $w' \in [w]_{\approx}$  must be answered equivalently. They are thus not forwarded to the *Teacher* anymore. We might think of an *Assistant* in between the *Learner* and the *Teacher* that checks if an equivalent query has already been performed. Queries for  $w \notin \mathcal{D}$  are not forwarded to the *Teacher* either but answered negatively by the *Assistant*.
- When the table  $T$  is both closed and consistent, the hypothesized DFA  $\mathcal{H}$  is computed as usual. After this, we proceed as follows:



1. If  $L(\mathcal{H}) \not\subseteq \mathcal{D}$ , compute a word  $w \in L(\mathcal{H}) \setminus \mathcal{D}$ , declare it a counterexample, and modify the table  $T$  accordingly (possibly involving further membership queries).
2. If  $L(\mathcal{H}) \subseteq \mathcal{D}$  but  $L(\mathcal{H})$  is not  $\approx$ -closed, then compute  $w, w' \in \mathcal{D}$  such that  $w \approx w'$ ,  $w \in L(\mathcal{H})$ , and  $w' \notin L(\mathcal{H})$ ; perform membership queries for  $[w]_{\approx}$ .
3. If  $L(\mathcal{H})$  is the union of  $\approx$ -equivalence classes but not  $\vdash$ -closed, then compute  $(L_1, L_2) \in \vdash$  such that  $L_1 \subseteq L(\mathcal{H})$  and  $L(\mathcal{H}) \cap L_2 = \emptyset$ ; perform membership queries for any word from  $L_1$ ; if all these membership queries are answered positively, the *Teacher* is asked to specify a word  $w$  from  $L_2$ , which will be declared “positive”.

Actually, a hypothesized DFA  $\mathcal{H}$  undergoes an equivalence test only if  $L(\mathcal{H}) \subseteq \mathcal{D}$  and  $L(\mathcal{H})$  is both  $\approx$ - and  $\vdash$ -closed. I.e., if, in the context of the extended learning algorithm, we speak of a hypothesized DFA, we actually act on the assumption that  $L(\mathcal{H})$  is the union of  $\approx$ -equivalence classes and closed under  $\vdash$ .

Let the extension of Angluin’s algorithm wrt. a learning setup as sketched above be called EXTENDEDANGLUIN (its pseudo code can be found in [6]). A careful analysis shows:

**Theorem 1.** *Let  $(\Sigma, \mathcal{D}, \approx, \vdash, obj)$  be a learning setup for  $(\mathcal{O}, \sim)$ . If  $o \in \mathcal{O}$  has to be learned, then invoking EXTENDEDANGLUIN( $(\mathcal{O}, \sim), (\Sigma, \mathcal{D}, \approx, \vdash, obj)$ ) returns, after finitely many steps<sup>2</sup>, an object  $o' \in \mathcal{O}$  such that  $o' \sim o$ .*

The theorem suggests the following definition:

**Definition 5.** *Let  $\mathcal{O}$  be a set of objects and  $\sim \subseteq \mathcal{O} \times \mathcal{O}$  be an equivalence relation. We say that  $(\mathcal{O}, \sim)$  is learnable if there is some learning setup for  $(\mathcal{O}, \sim)$ .*

## 5 Learning Message-Passing Automata

This section identifies some learnable classes of MPA, i.e, regular word languages that can be learned and generated by an MPA. It seems unlikely to find a reasonable learning approach for arbitrary MPA, which is suggested by negative results from [8]. We therefore propose to consider  $\exists$ - and  $\forall$ -regular MSC languages and study learnability for the class of MPA and product MPA.

### 5.1 Regular MSC Languages and Product MSC Languages

A word language is said to represent an MSC language  $\mathcal{L}$  whenever it contains a linearization for each  $M \in \mathcal{L}$ , and no linearizations for  $M' \notin \mathcal{L}$ . Formally:

**Definition 6 (Representative).**  *$L \subseteq Act^*$  is a representative for  $\mathcal{L} \subseteq \text{MSC}$  if  $L \subseteq \text{Lin}(\mathcal{L})$  and, for any MSC  $M$ ,  $M \in \mathcal{L}$  iff  $\text{Lin}(M) \cap L \neq \emptyset$ .*

<sup>2</sup> When learning a DFA over  $\Sigma$  with  $n$  states, the number of membership queries in the worst case is  $\mathcal{O}(|\Sigma| \cdot n^3)$ .

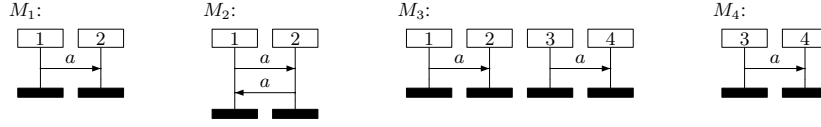


Fig. 4. Some MSCs

*Example 3.* Let  $M_1 \cdot M_2$  denote the concatenation of MSCs  $M_1$  and  $M_2$ , i.e., the unique MSC  $M$  such that  $\{w_1 w_2 \mid w_1 \in \text{Lin}(M_1), w_2 \in \text{Lin}(M_2)\} \subseteq \text{Lin}(M)$ .  $\{M\}^*$  denotes the Kleene closure of  $\cdot$ . The MSC language  $\{M_1\}^*$  for MSC  $M_1$  in Fig. 4 is not regular in the sense of [17], as  $\text{Lin}(\{M_1\}^*)$  is not a regular word language. However,  $\{M_1\}^*$  can be represented by the regular word language  $\text{Lin}^1(\{M_1\}^*) = \{!(1, 2, a) ?(2, 1, a)^n \mid n \in \mathbb{N}\}$ . Considering the MSC  $M_2$  in Fig. 4, we even have that  $\text{Lin}(\{M_2\}^*)$  is a regular representative for  $\{M_2\}^*$ .

The interesting case occurs when representatives are regular. But some MSC languages cannot be generated by MPA as their regular representatives require infinite channels.

*Example 4.* The  $\exists 1$ -bounded MSC language  $\{M_3\}^*$  for MSC  $M_3$  in Fig. 4 has the regular representative  $\{!(1, 2, a) ?(2, 1, a) !(3, 4, a) ?(4, 3, a)^n \mid n \in \mathbb{N}\}$ , but there is no  $B \in \mathbb{N}$  such that  $\text{Lin}^B(\{M_3\}^*)$  is a regular representative for  $\{M_3\}^*$ . Thus, according to results from [16], it cannot be the language of some MPA.

**Definition 7 ( $\forall$ - and  $\exists$ -regular).**  $\mathcal{L} \subseteq \text{MSC}$  is  $\forall$ -regular if  $\text{Lin}(\mathcal{L}) \subseteq \text{Act}^*$  is regular.  $\mathcal{L}$  is  $\exists$ -regular if, for some  $B \in \mathbb{N}$ ,  $\text{Lin}^B(\mathcal{L})$  is a regular representative for  $\mathcal{L}$ .

Any  $\forall$ -regular MSC language is  $\forall$ -bounded and any  $\exists$ -regular MSC language is  $\exists$ -bounded. Moreover, any  $\forall$ -regular MSC language is  $\exists$ -regular. An MPA is called  $\forall$ -regular,  $\exists$ -regular, etc., if so is its MSC language.

*Example 5.* The MPA in Fig. 2a is not  $\exists$ -regular, whereas the MPA in Fig. 2b is  $\forall$ -regular. In particular, only finitely many global configurations are reachable from the initial configuration. The MPA in Fig. 2c is  $\exists$ -regular, but not  $\forall$ -regular.

Regular MSC languages are of interest as they are realizable by MPA.

**Theorem 2 ([16,17,20]).** *Regular MSC languages versus bounded MPA:*

- (a) For any  $\exists$ -regular MSC language  $\mathcal{L}$  (given as a regular representative), one can effectively compute an MPA  $\mathcal{A}$  such that  $\mathcal{L}(\mathcal{A}) = \mathcal{L}$ . If  $\mathcal{L}$  is  $\forall$ -regular, then  $\mathcal{A}$  can be assumed to be deterministic.
- (b) Let  $B \in \mathbb{N}$ . For  $\mathcal{A} \in \text{MPA}_{\exists B}$ ,  $\text{Lin}^B(\mathcal{L}(\mathcal{A}))$  is a regular representative for  $\mathcal{L}(\mathcal{A})$  and  $\mathcal{L}(\mathcal{A})$  is  $\exists$ -regular. For  $\mathcal{A} \in \text{MPA}_{\forall}$ ,  $\text{Lin}(\mathcal{L}(\mathcal{A}))$  is a regular representative for  $\mathcal{L}(\mathcal{A})$  and  $\mathcal{L}(\mathcal{A})$  is  $\forall$ -regular.

A realization of  $\{M_1, M_4\}$  (cf. Fig. 4) also infers  $M_3$  provided the bilateral interaction between the processes is completely independent. A set of MSCs that is closed under such an inference is a *product* MSC language (it is called *weakly realizable* in [2]). For  $M = (E, \{\leq_p\}_{p \in \text{Proc}}, <_{\text{msg}}, \ell) \in \text{Pref}(\text{MSC})$ , the behavior of  $M$  can be split into its

components  $M \upharpoonright p = (E_p, \leq_p, \ell_{|E_p})$ ,  $p \in Proc$ , each of which represents the behavior of a single agent, which can be seen as a word over  $Act_p$ . For finite set  $\mathcal{L} \subseteq MSC$  and  $M \in MSC$ , let  $\mathcal{L} \vdash_{MSC}^p M$  if, for any  $p \in Proc$ , there is  $M' \in \mathcal{L}$  such that  $M' \upharpoonright p = M \upharpoonright p$ .

**Definition 8 (Product MSC language [2]).**  $\mathcal{L} \subseteq MSC$  is a product MSC language if, for any  $M \in MSC$  and any finite  $\mathcal{L}' \subseteq \mathcal{L}$ ,  $\mathcal{L}' \vdash_{MSC}^p M$  implies  $M \in \mathcal{L}$ .

For practical applications, it is desirable to consider so-called *safe* product languages. Those languages are implementable in terms of a safe product MPA, thus one that is deadlock-free. For a finite set  $\mathcal{L} \subseteq MSC$  and  $P \in Pref(MSC)$ , we write  $\mathcal{L} \vdash_{MSC}^s P$  if, for any  $p \in Proc$ , there is  $M \in \mathcal{L}$  such that  $P \upharpoonright p$  is a prefix of  $M \upharpoonright p$ .

**Definition 9 (Safe product MSC language [2]).** A product MSC language  $\mathcal{L} \subseteq MSC$  is called *safe* if, for any finite  $\mathcal{L}' \subseteq \mathcal{L}$  and any  $P \in Pref(MSC)$ ,  $\mathcal{L}' \vdash_{MSC}^s P$  implies  $P \preceq M$  for some  $M \in \mathcal{L}$ .

**Lemma 1 ([21], cf. [2,3]).**  $\mathcal{L} \subseteq MSC$  is a  $\forall$ -regular safe product MSC language (given in terms of  $Lin(\mathcal{L})$ ) iff it is accepted by some  $\mathcal{A} \in MPA_{\forall}^{sp}$ . Both directions are effective.

## 5.2 Learning $\forall$ -Bounded Message-Passing Automata

Towards a learning setup for  $\forall$ -bounded MPA, we let

- $\sim_{\forall} = \{(\mathcal{A}, \mathcal{A}') \in MPA_{\forall} \times MPA_{\forall} \mid \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')\}$ ,
- $\approx_{MW} = \{(w, w') \in Lin(M) \times Lin(M) \mid M \in MSC\}$ , and
- $obj_{\forall} : \mathfrak{R}_{minDFA}(Act, Lin(MSC), \approx_{MW}, \emptyset) \rightarrow [MPA_{\forall}]_{\sim_{\forall}}$  be an effective bijective mapping whose existence is stated by Theorem 2 (a).

To prove that  $(Act, Lin(MSC), \approx_{MW}, \emptyset, obj_{\forall})$  is indeed a learning setup for the pair  $(MPA_{\forall}, \sim_{\forall})$ , we need to establish the corresponding decidability results.

**Proposition 1.**  $INCLUSION(Act, Lin(MSC))$  and  $EQCLOSURE(Act, Lin(MSC), \approx_{MW})$  are constructively decidable.

The decidability part stems from [17, Prop. 2.4] (see also [25]). The corresponding decision algorithm runs in time linear in the size of the transition function of the DFA. Counterexamples can be computed in linear time as well. For a detailed description, please consult [6]. Note that the question if the  $\approx_{MW}$ -closure of a regular set of MSC words is a regular language, too, is undecidable. For our learning approach, however, this problem does not play any role. For arbitrary finite automata  $\mathcal{A}$  over  $Act$  with  $L(\mathcal{A}) \subseteq Lin(MSC)$  (which are not necessarily deterministic), it was shown in [25] (for Büchi automata) that deciding if  $L(\mathcal{A})$  is  $\approx_{MW}$ -closed is PSPACE complete. In our context of minimal DFA, however, the problem becomes much simpler.

**Proposition 2.**  $(Act, Lin(MSC), \approx_{MW}, \emptyset, obj_{\forall})$  is a learning setup for  $(MPA_{\forall}, \sim_{\forall})$ .

**Theorem 3.**  $(MPA_{\forall}, \sim_{\forall})$  is learnable.

### 5.3 Learning $\exists$ -Bounded Message-Passing Automata

In this subsection, we are aiming at a learning setup for  $\exists$ -bounded MPA. As stated in Def. 7, we now have to provide a channel bound. So let  $B \in \mathbb{N}$  and set

- $\sim_{\exists B} = \{(\mathcal{A}, \mathcal{A}') \in \text{MPA}_{\exists B} \times \text{MPA}_{\exists B} \mid \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')\}$ ,
- $\approx_{\exists B} = \{(w, w') \in \text{Lin}^B(M) \times \text{Lin}^B(M) \mid M \in \text{MSC}\}$ , and
- $\text{obj}_{\exists B} : \mathfrak{A}_{\text{minDFA}}(\text{Act}, \text{Lin}^B(\text{MSC}), \approx_{\exists B}, \emptyset) \rightarrow [\text{MPA}_{\exists B}]_{\sim_{\exists B}}$  to be an effective bijective mapping whose existence is stated by Theorem 2.

In the following, we will see that  $(\text{Act}, \text{Lin}^B(\text{MSC}), \approx_{\exists B}, \emptyset, \text{obj}_{\exists B})$  is indeed a learning setup for  $(\text{MPA}_{\exists B}, \sim_{\exists B})$ . Adapting Prop. 1, we can establish the corresponding decidability result (see [6] for the proof):

**Proposition 3.** *For any  $B \in \mathbb{N}$ , the problems  $\text{INCLUSION}(\text{Act}, \text{Lin}^B(\text{MSC}))$  and  $\text{EQCLOSURE}(\text{Act}, \text{Lin}^B(\text{MSC}), \approx_{\exists B})$  are constructively decidable.*

**Proposition 4.** *For any  $B \in \mathbb{N}$ ,  $(\text{Act}, \text{Lin}^B(\text{MSC}), \approx_{\exists B}, \emptyset, \text{obj}_{\exists B})$  is a learning setup for  $(\text{MPA}_{\exists B}, \sim_{\exists B})$ .*

**Theorem 4.** *For any  $B \in \mathbb{N}$ ,  $(\text{MPA}_{\exists B}, \sim_{\exists B})$  is learnable.*

### 5.4 Learning $\forall$ -Bounded Safe Product Message-Passing Automata

Let us set the scene for learning  $\forall$ -bounded safe product MPA. In this case, we have to create an inference rule  $\vdash \neq \emptyset$  (cf. Definitions 8 and 9). We first define relations  $\vdash_{\text{MW}}^p$  and  $\vdash_{\text{MW}}^s$  for word languages, which correspond to  $\vdash_{\text{MSC}}^p$  and  $\vdash_{\text{MSC}}^s$ , respectively:

- $\vdash_{\text{MW}}^p = \{(\text{Lin}(\mathcal{L}), \{w\}) \mid \mathcal{L} \subseteq \text{MSC} \text{ is finite and } \exists M \in \text{MSC} : \mathcal{L} \vdash_{\text{MSC}}^p M \wedge w \in \text{Lin}(M)\}$
- $\vdash_{\text{MW}}^s = \{(\text{Lin}(\mathcal{L}), L_2) \mid \mathcal{L} \subseteq \text{MSC} \text{ is finite and } \exists P \in \text{Pref}(\text{MSC}) \text{ and } u \in \text{Lin}(P) \text{ such that } \mathcal{L} \vdash_{\text{MSC}}^s P \text{ and } L_2 = \{w \in \text{Lin}(\text{MSC}) \mid w = uv \text{ for some } v \in \text{Act}^*\}\}$  (note that  $L_2$  is a decidable language).

Given these relations, we can define our learning setup as follows:

- $\sim_{\forall}^{\text{sp}} = \{(\mathcal{A}, \mathcal{A}') \in \text{MPA}_{\forall}^{\text{sp}} \times \text{MPA}_{\forall}^{\text{sp}} \mid \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')\}$ ,
- $\approx_{\text{MW}} = \{(w, w') \in \text{Lin}(M) \times \text{Lin}(M) \mid M \in \text{MSC}\}$  (as before),
- $\vdash_{\text{MW}}^{\text{sp}} = \vdash_{\text{MW}}^p \cup \vdash_{\text{MW}}^s$ ,
- $\text{obj}_{\forall}^{\text{sp}} : \mathfrak{A}_{\text{minDFA}}(\text{Act}, \text{Lin}(\text{MSC}), \approx_{\text{MW}}, \vdash_{\text{MW}}^{\text{sp}}) \rightarrow [\text{MPA}_{\forall}^{\text{sp}}]_{\sim_{\forall}^{\text{sp}}}$  be an effective bijective mapping, as guaranteed by Lemma 1.

**Proposition 5.**  *$\text{INFCLOSURE}(\text{Act}, \text{Lin}(\text{MSC}), \approx_{\text{MW}}, \vdash_{\text{MW}}^{\text{sp}})$  is constructively decidable.*

*Proof.* Decidability of  $\text{INFCLOSURE}(\text{Act}, \text{Lin}(\text{MSC}), \approx_{\text{MW}}, \vdash_{\text{MW}}^{\text{sp}})$  has been shown in [3, Theorem 3], where an EXPSPACE-algorithm for bounded high-level MSCs is given, which reduces the problem to finite automata with a  $\approx_{\text{MW}}$ -closed language. From such a  $\approx_{\text{MW}}$ -closed DFA  $\mathcal{H}$ , we compute a (componentwise) minimal, reduced (i.e., without local sink states), and deterministic product MPA  $\mathcal{A}$ , by simply taking the projections of  $\mathcal{H}$  onto  $\text{Act}_p$  for any  $p \in \text{Proc}$ , minimizing and determinizing them. Then, the MSC

language  $\mathcal{L}$  associated with  $\mathcal{H}$  is a safe product language iff  $\mathcal{A}$  is a safe product MPA realizing  $\mathcal{L}$ . From  $\mathcal{H}$ , we can moreover compute a bound  $B$  such that any run of  $\mathcal{A}$  exceeding the buffer size  $B$  cannot correspond to a prefix of some MSC word in  $L(\mathcal{H})$ . Thus, a run through  $\mathcal{A}$  (in terms of a prefix of an MSC word) that either

- exceeds the buffer size  $B$  (i.e., it is not  $B$ -bounded), or
- does not exceed the buffer size  $B$ , but results in a deadlock configuration

gives rise to a prefix  $u$  (of an MSC word) that is implied by  $\mathcal{H}$  wrt.  $\vdash_{\text{MW}}^s$ , i.e.,  $L(\mathcal{H})$  must actually contain a completion  $uv \in \text{Lin}(\text{MSC})$  of  $u$ . Obviously, one can decide if a word is such a completion of  $u$ . The completions of  $u$  form one possible  $L_2$ . It remains to specify a corresponding set  $L_1$  for  $u$ . By means of  $\mathcal{H}$ , we can, for any  $p \in \text{Proc}$ , compute a word  $w_p \in L(\mathcal{H})$  such that the projection of  $u$  onto  $\text{Act}_p$  is a prefix of the projection of  $w_p$  onto  $\text{Act}_p$ . Observe that  $w_p$  can be computed in polynomial time. We set  $L_1 = \bigcup_{p \in \text{Proc}} [w_p]_{\approx_{\text{MW}}}$ .

Finally, suppose that, in  $\mathcal{A}$ , we could neither find a prefix exceeding the buffer size  $B$  nor a reachable deadlock configuration in the  $B$ -bounded fragment. Then, we still have to check if  $\mathcal{A}$  recognizes  $\mathcal{L}$ . If not, one can compute a ( $B$ -bounded) MSC word  $w \in L(\mathcal{A}) \setminus L(\mathcal{H})$  whose MSC is implied by  $\mathcal{L}$  wrt.  $\vdash_{\text{MSC}}^p$ . Setting  $L_2 = \{w\}$ , a corresponding set  $L_1$  can be specified as the union of sets  $[w_p]_{\approx_{\text{MW}}}$ , as above.  $\square$

Together with Prop. 1, we obtain the following two results:

**Proposition 6.** *The quintuple  $(\text{Act}, \text{Lin}(\text{MSC}), \approx_{\text{MW}}, \vdash_{\text{MW}}^{\text{sp}}, \text{obj}_{\forall}^{\text{sp}})$  is a learning setup for  $(\text{MPA}_{\forall}^{\text{sp}}, \sim_{\forall}^{\text{sp}})$ .*

**Theorem 5.**  $(\text{MPA}_{\forall}^{\text{sp}}, \sim_{\forall}^{\text{sp}})$  is learnable.

### 5.5 Learning $\forall$ -Bounded Product Message-Passing Automata

Finally, we study the problem of learning  $\forall$ -bounded product MPA. Unfortunately, we are in the situation that the canonical definition of a learning setup does not work:

**Proposition 7 ([3]).**  $\text{INF CLOSURE}(\text{Act}, \text{Lin}(\text{MSC}), \approx_{\text{MW}}, \vdash_{\text{MW}}^p)$  is not constructively decidable. More specifically, it is undecidable if the language of a  $\approx_{\text{MW}}$ -closed DFA over  $\text{Act}$  is closed under  $\vdash_{\text{MW}}^p$ .

Similar decision problems were considered in [24,2,3,21]. Most of them are, however, concerned with translating a *high-level MSC* into a product MPA.

## 6 Tool Description and Future Work

We have implemented the learning approach presented in the preceding sections in the tool *Smyle* (Synthesizing Models bY Learning from Examples), which can be freely downloaded at <http://smyle.in.tum.de>. It is written in Java and makes use of the `LearnLib` library [27], which implements *Angluin's algorithm*, and the libraries `Grappa` [5] and `JGraph` [22] for visualization purposes. For computing linearizations of MSCs we use the algorithm given in [30] running in  $\mathcal{O}(n \cdot e(\mathcal{P}))$  time, where  $n$  is the number of elements of the partial order  $\mathcal{P}$  and  $e(\mathcal{P}) = |E(\mathcal{P})|$  is the number of linear extensions of  $\mathcal{P}$ . The tool is capable of learning universally regular and existentially regular MSC languages. The framework contains the following three main components:

- the *Teacher*, representing the interface between the GUI (user) and the *Assistant*
- the *Learner*, containing the LearnLib part
- the *Assistant*, keeping track of membership queries that were not yet asked, checking for  $B$ -boundedness as well as the language type ( $\exists/\forall$ )

**The learning chain:** Initially, the user is asked to specify the learning setup. After having selected a language type (existentially/universally) and a channel bound  $B$ , the user provides a set of MSCs. These MSC specifications must then be divided into *positive* (i.e., MSCs contained in the language to learn) and *negative* (i.e., MSCs not contained in the language to learn). After submitting these examples, all linearizations are checked for consistency with respect to the properties of the learning setup. Violating linearizations are stored as negative examples. Now the learning algorithm starts. The *Learner* continuously communicates with the *Assistant* in order to gain answers to membership queries. This procedure halts as soon as a query cannot be answered by the *Assistant*. In this case, the *Assistant* forwards the inquiry to the user, displaying the MSC in question on the screen. The user must classify the MSC as positive or negative (cf. Fig. 5 (1)).

The *Assistant* checks the classification for validity wrt. the learning setup. Depending on the outcome of this check, the linearizations of the current MSC are assigned to the positive or negative set of future queries. Moreover, the user's answer is passed to the *Learner*, which then continues his question-and-answer game with the *Assistant*. If the LearnLib proposes a possible automaton, the *Assistant* checks whether

the learned model is consistent with all queries that have been categorized but not yet been asked. If he encounters a counter-example, he presents it to the learning algorithm which, in turn, continues the learning procedure until the next possible solution is found. In case there is no further evidence for contradicting samples, a new frame appears (cf. Fig. 5 (2, 3)). Among others, it visualizes the currently learned DFA (2, 4) and a panel for displaying MSCs (3) of runs of the system described by the automaton. The user is then asked if he agrees with the solution and may either stop or introduce a new counter-example proceeding with the learning procedure.

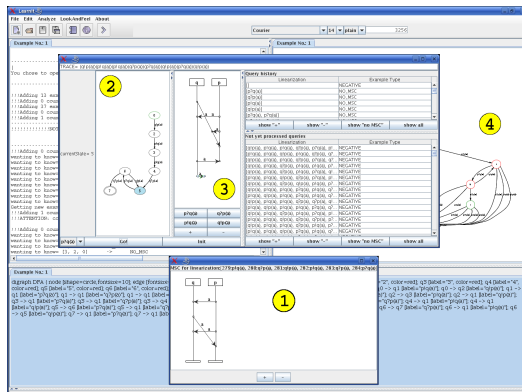


Fig. 5. *Smyle* screenshot

**Case studies:** We applied *Smyle* to the *simple negotiation protocol* from [13], the *continuous update protocol* from [14], a protocol being part of USB 1.1 mentioned in [15], and a variant of the ABP. For the first one, *Smyle* was provided with 6 positive MSCs and performed 9675 membership and 65 user-queries. It resulted in an automaton consisting of 9 states. The second protocol (giving 4 sample MSCs as input) was learned after 5235 membership and 43 user queries resulting in an automaton

containing 8 states. The third protocol was learned after 1373 membership and 12 user-queries, providing it with 4 sample MSCs. The inferred automaton was composed of 9 states. The ABP was realized by an automaton with 15 states after 19276 membership and 105 user queries, providing 4+1 positive examples. For further details such as the input MSCs and inferred automata, we refer to [6] and the webpage of our tool.

**Future work:** There are other interesting classes of learnable MPA, and our setting applies to the *causal* closure by Adsul et al. [1]. We plan to provide high-level MSCs as a means to predefine patterns of positive or negative examples. Moreover, *MSCan* [7] will be integrated into *Smyle* to support formal analysis of a suggested model.

*Smyle* is freely available for exploration at <http://smyle.in.tum.de>.

## References

1. B. Adsul, M. Mukund, K. N. Kumar, and V. Narayanan. Causal closure for MSC languages. In *FSTTCS 2005*, volume 3821 of *LNCS*, pages 335–347. Springer, 2005.
2. R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. *IEEE Trans. Softw. Eng.*, 29(7):623–633, 2003.
3. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. *Th. Comp. Sc.*, 331(1):97–114, 2005.
4. D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
5. AT&T. *Grappa - A Java Graph Package*. <http://www.research.att.com/~john/Grappa/>.
6. B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker. Replaying play in and play out: Synthesis of design models from scenarios by learning. Research Report AIB-2006-12, RWTH Aachen, 2006.
7. B. Bollig, C. Kern, M. Schlütter, and V. Stolz. MSCan: A tool for analyzing MSC specifications. In *TACAS 2006*, volume 3920 of *LNCS*, pages 455–458. Springer, 2006.
8. B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Th. Comp. Sc.*, 358(2-3):150–172, 2006.
9. Y. Bontemps, P. Heymand, and P.-Y. Schobbens. From live sequence charts to state machines and back: a guided tour. *IEEE Trans. Softw. Eng.*, 31(12):999–1014, 2005.
10. D. Brand and P. Zafropulo. On communicating finite-state machines. *J. of the ACM*, 30(2):323–342, 1983.
11. W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:1:45–80., 2001.
12. D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
13. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-based agent communication protocols. In *Workshop on Agent Communication Languages*, pages 91–107, 2003.
14. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *IJCAI 2003*, pages 679–684, 2003.
15. B. Genest. Compositional message sequence charts (CMSCs) are better to implement than MSCs. In *TACAS 2005*, volume 3440 of *LNCS*, pages 429–444. Springer, 2005.
16. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.
17. J. G. Henriksen, M. Mukund, K. N. Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Inf. and Comput.*, 202(1):1–38, 2005.

18. H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *CAV 2003*, volume 2725 of *LNCS*, pages 315–327. Springer, 2003.
19. I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to statecharts. In *DIPES 1998*, volume 155 of *IFIP Conf. Proc.*, pages 61–72. Kluwer, 1998.
20. D. Kuske. Regular sets of infinite message sequence charts. *Inf. Comput.*, 187:80–109, 2003.
21. M. Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Th. Comp. Sc.*, 309(1-3):529–554, 2003.
22. J. Ltd. *JGraph - Java Graph Visualization and Layout*. <http://www.jgraph.com/>.
23. E. Mäkinen and T. Systä. MAS – An interactive synthesizer to support behavioral modeling in UML. In *ICSE 2001*, pages 15–24. IEEE Computer Society, 2001.
24. R. Morin. Recognizable sets of message sequence charts. In *STACS 2002*, volume 2285 of *LNCS*, pages 523–534. Springer, 2002.
25. A. Muscholl and D. Peled. From finite state communication protocols to high-level message sequence charts. In *ICALP 2001*, volume 2076 of *LNCS*, pages 720–731. Springer, 2001.
26. B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *ICSE 2000*, pages 35–46. ACM, 2000.
27. H. Raffelt and B. Steffen. LearnLib: A library for automata learning and experimentation. In *FASE 2006*, volume 3922 of *LNCS*, pages 377–380, 2006.
28. B. Sengupta and R. Cleaveland. Triggered message sequence charts. *IEEE Trans. Softw. Eng.*, 32(8):587–607, 2006.
29. S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. Softw. Eng.*, 29(2):99–115, 2003.
30. Y. L. Varol and D. Rotem. An algorithm to generate all topological sorting arrangements. *Comput. J.*, 24(1):83–84, 1981.