RWTH Aachen

Department of Computer Science
Technical Report

Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, and Martin Leucker

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2006-12

RWTH Aachen · Department of Computer Science · October 2006

The publications of the Department of Computer Science of RWTH Aachen (*Aachen University of Technology*) are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

Benedikt Bollig[1], Joost-Pieter Katoen[2], Carsten Kern[2], and Martin Leucker[3]

[1] LSV, CNRS UMR 8643 & ENS de Cachan, France
[2] Software Modeling and Verification Group, RWTH Aachen University, Germany
[3] 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 MPAs 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 [27], 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 either positive or negative, indicating a possible 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 such 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 design models from scenario-based specifications. Since message-passing automata (MPA, for short) [11] are a commonly used model to realize the behavior as described by scenarios, we adopt MPA as design model. In particular, 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.

The 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* [28], which 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 [21], formalization and undecidability results for the synthesis for a simple variant of live sequence charts (LSCs) [10], and Harel's play-in, play-out approach for LSCs [12, 13]. Another approach is proposed by Alur *et al.* in [2, 3]. Uchitel *et al.* [30] present an algorithm for synthesizing transition systems from high-level MSCs. An executable variant of LSCs, triggered MSCs, are presented in [29]. 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. The novel aspect of our technique is that we exploit learning algorithms for *synthesis* which are based on positive and *negative* scenarios. Existing approaches to synthesizing design models are based on completely different techniques and only consider positive examples. Applying learning yields an incremental approach, and facilitates the generation of diagnostic feedback.

In the setting of model-based testing, Angluin's learning algorithm has successfully been used for inferring models of system's behavior [20]. In their setting, examples are words and models are DFA, while we work with the more complicated structures of MSCs (in fact, partial orders) and MPA.

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 as well as some initial case study results.

2 Message Sequence Charts

Let Σ^* denote the set of finite words over a finite alphabet Σ. A *Σ-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 $\ell : E \to \Sigma$ 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 Σ^*.

Let $Proc$ be a finite set of at least two *processes*, which exchange messages from a finite set Msg. Communication proceeds through channels via executing communication actions. Let Ch denote the set $\{(p, q) \mid p, q \in Proc,\ p \neq q\}$ of reliable FIFO *channels*. For process $p \in Proc$, Act_p denotes the set of (communication) actions of p, i.e., $\{!(p, q, a) \mid (p, q) \in Ch$ and $a \in Msg\} \cup \{?(p, q, a) \mid (p, q) \in Ch$ 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 $(E, \{\leq_p\}_{p\in Proc}, <_{\mathrm{msg}}, \ell)$ with:*

- *E is a finite set of* events*,*
- *$\ell : E \to Act$ is a labeling function,*
- *for any $p \in Proc$, $\leq_p$ is a total order on $E_p = \ell^{-1}(Act_p)$,*
- *$<_{\mathrm{msg}} \subseteq E \times E$ such that, for any $e \in E$, $e <_{\mathrm{msg}} e'$ or $e' <_{\mathrm{msg}} e$ for some $e' \in E$, and, for any $(e_1, e_1') \in <_{\mathrm{msg}}$, 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 $(e_2, e_2') \in <_{\mathrm{msg}}$ 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*
  - *$\leq = (<_{\mathrm{msg}} \cup \bigcup_{p\in Proc} \leq_p)^*$ is a partial-order relation on E.*

Let $M = (E, \{\leq_p\}_{p\in Proc}, <_{\mathrm{msg}}, \ell)$ be an MSC. A *prefix* of M is a structure $(E', \{\leq_p'\}_{p\in Proc}, <_{\mathrm{msg}}', \ell')$ 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 $p \in Proc$, $<_{\mathrm{msg}}' = <_{\mathrm{msg}} \cap (E' \times E')$, and ℓ' is the restriction of ℓ to E'. We write $P \preceq M$ if P is a prefix of the MSC M.

The set of MSCs is denoted by $\mathbb{MSC}$.[4] 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 $\{P \mid P \preceq M$ 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 $M = (E, \{\leq_p\}_{p\in Proc}, <_{\mathrm{msg}}, \ell) \in \mathbb{MSC}$. 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 σ in v. For $B \in \mathbb{N}$, let $Lin^B(M)$ denote $\{w \in Lin(M) \mid w$ is B-bounded$\}$, and $Lin^B(\mathcal{L}) = \bigcup_{M\in\mathcal{L}} Lin^B(M)$ for $\mathcal{L} \subseteq \mathbb{MSC}$.

[4] As $Proc$ and Msg are supposed to be fixed, they are omitted.

Definition 2 (Boundedness). *Let* $M \in \mathbb{MSC}$.

1. M *is* universally B-bounded *(i.e., $\forall B$-bounded) if* $Lin(M) = Lin^B(M)$.
2. M *is* existentially B-bounded *(i.e., $\exists B$-bounded), if* $Lin(M) \cap Lin^B(M) \neq \emptyset$.

The set of $\forall B$-bounded MSCs and $\exists B$-bounded MSCs is denoted by $\mathbb{MSC}_{\forall B}$ and $\mathbb{MSC}_{\exists B}$, 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. $\mathcal{L} \subseteq \mathbb{MSC}$ is $\forall B$-bounded if $\mathcal{L} \subseteq \mathbb{MSC}_{\forall B}$, and $\exists B$-bounded if $\mathcal{L} \subseteq \mathbb{MSC}_{\exists B}$. Moreover, $\mathcal{L}$ is $\forall$-/$\exists$-bounded if it is $\forall B$-/$\exists B$-bounded for some $B \in \mathbb{N}$, respectively.

Example 1. Let M be the MSC in Fig. 1c, where five messages are sent from 1 to 2. The word $w = !(1,2,\mathsf{req})\ (!(1,2,\mathsf{req})\ ?(2,1,\mathsf{req}))^4\ ?(2,1,\mathsf{req})$ is in $Lin(M)$, and thus is an MSC word. It is 2-bounded, but not 1-bounded. M, however, has a 1-bounded linearization, and $Lin^1(M) = \{(!(1,2,\mathsf{req})\ ?(2,1,\mathsf{req}))^5\}$. In fact, MSC M is $\exists 1$-bounded and $\forall B$-bounded for $B \geq 5$. The MSC in Fig. 1a is $\forall 4$-bounded and thus also $\exists 4$-bounded; in fact, it is even $\exists 2$-bounded. However, it is not $\exists 1$-bounded, as there is no possible schedule such that any channel always carries at most one message. The MSC in Fig. 1b is $\forall 2$-and $\exists 1$-bounded, but not $\forall 1$-bounded. Finally, we note that the set of MSCs where an arbitrary number of messages is sent from 1 to 2 is $\exists 1$-bounded, but not $\forall$-bounded.

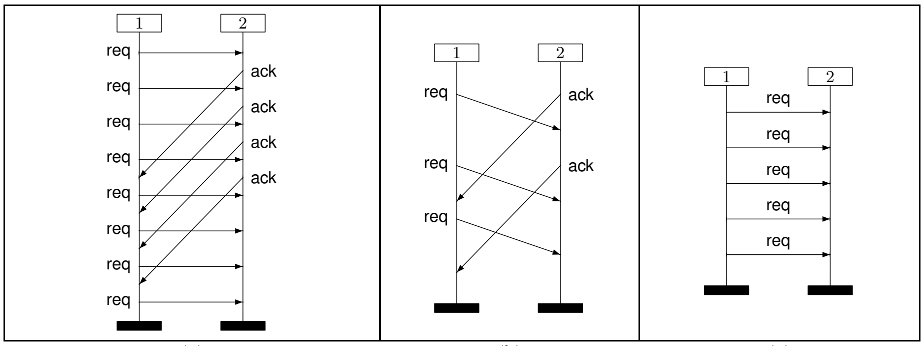
3 Message-Passing Automata

An MPA [11] is a collection of finite-state machines (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 $\mathcal{A}$ is a tuple* $((\mathcal{A}_p)_{p \in Proc}, Sync, \overline{s}^{in}, F)$ *with:*

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

As in [19, 25], we consider the linearizations of MSCs that are obtained from the global automaton induced by an MPA. For an MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in Proc}, Sync, \overline{s}^{in}, F)$, where $\mathcal{A}_p = (S_p, \Delta_p)$, this global automaton is defined as follows. The set of *configurations* of $\mathcal{A}$, denoted by $Conf_{\mathcal{A}}$, consists of pairs $(\overline{s}, \chi)$ with $\overline{s} \in S_{\mathcal{A}}$ and $\chi : Ch \to$



(a) (b) (c)

Fig. 1. Example message sequence charts

$(Msg \times Sync)^*$, indicating the channel contents. The *global transition relation* of $\mathcal{A}$, $\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Act \times Sync \times Conf_{\mathcal{A}}$, is defined by the following two inference rules ($\overline{s}[p]$ refers to the p-component of a global state $\overline{s} \in S_{\mathcal{A}}$):

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

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

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

where $\chi((q,p)) = w \cdot (a,m)$ and $\chi' = \chi[(q,p) := w]$. The initial and final configurations of the global automaton are $(\overline{s}^{in}, \chi_\varepsilon)$ and $F \times \{\chi_\varepsilon\}$, respectively, where χ_ε 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 $Lin(\mathcal{L}) = L(\mathcal{A})$. 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 $\text{MPA}_\forall$ and $\text{MPA}_{\exists B}$, respectively.

Example 2. Fig. 2a shows a not $\exists$-bounded MPA with set of synchronization messages $\{\mathrm{m}_1, \mathrm{m}_2\}$ (and simplified action alphabet). The only global final state is indicated by a dashed line. The MSC language cannot be recognized with less than two synchronization messages, which help to separate two request phases of equal length, as illustrated in Fig. 1a. 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 $\forall 2$-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.

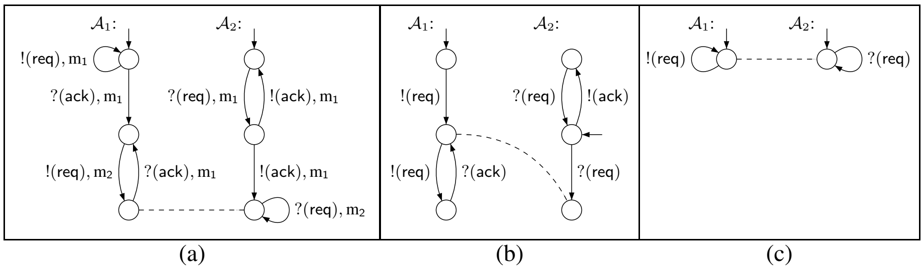


Fig. 2. Example message-passing automata

An MPA $\mathcal{A} = ((\mathcal{A}_p)_{p \in Proc}, Sync, \overline{s}^{in}, F)$, 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* [23, 25]. 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 [6, 8]. As for ordinary MPA, the notions of boundedness carry over to product MPA; let $\mathrm{MPA}^{\mathrm{p}}_{\forall}$ and $\mathrm{MPA}^{\mathrm{p}}_{\exists B}$ denote the set of $\forall$-bounded product and $\exists B$-bounded product MPA, respectively. The MPA in Fig. 2b is in $\mathrm{MPA}^{\mathrm{p}}_{\forall}$, whereas the MPA in Fig. 2c is in $\mathrm{MPA}^{\mathrm{p}}_{\exists 1}$, but not in $\mathrm{MPA}_{\forall}$.

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 $\mathrm{MPA}^{\mathrm{sp}}_{\forall}$.

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 Σ be an alphabet. A deterministic finite automaton (DFA) over Σ 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*, $\delta : Q \times \Sigma \to Q$ 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 $L(\mathcal{A})$.

Angluin's learning algorithm is designed for learning a regular language $L(\mathcal{A}) \subseteq \Sigma^*$ in terms of a minimal DFA $\mathcal{A}$.

4.1 The Basic Algorithm

In this algorithm, a so-called $Learner$, who initially knows nothing about $\mathcal{A}$, is trying to learn $L(\mathcal{A})$ 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 $T(u) : V \to \{+, -\}$ where $+$ represents accepted and $-$ not accepted. In [4], each function $T(u)$ is called a *row*. When T is

- *closed*, meaning that, for each $u \in U$ and $a \in \Sigma$, there is a $u' \in U$ such that $T(ua) = T(u')$, and
- *consistent*, meaning that for each $u \in U$ 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 ε denoting the empty word),
- δ is defined by $\delta(T(u), a) = T(ua)$, and
- $Q^+ = \{T(u) \mid u \in U \text{ and } T(u)(\varepsilon) = +\}$,

and 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 Σ. For MPA $\mathcal{A}$, we might consider regular languages L over Act such that L corresponds to the set $Lin(\mathcal{L}(\mathcal{A}))$. Unfortunately, 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 Σ^*, i.e., we learn regular word languages that contain at most words from $\mathcal{D}$. For learning MPA, e.g., it is reasonable to set $\mathcal{D} = Lin(\mathbb{MSC})$.

It is, however, 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 in the next section) imposing similar requirements on the representing regular 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 $L \cap L_2 \neq \emptyset$. [5]

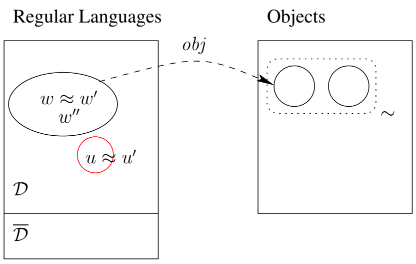


Fig. 3. Representing objects by regular languages

Naturally, $\mathcal{D}$, $\approx$, and $\vdash$ determine a particular class $\mathfrak{R}_{\text{minDFA}}(\Sigma, \mathcal{D}, \approx, \vdash) = \{L \subseteq \mathcal{D} \mid L$ is regular and closed under both $\approx$ and $\vdash\}$ of regular word languages over Σ (where any language is understood to be given by its minimal DFA). Suppose a language of this class $\mathfrak{R}_{\text{minDFA}}(\Sigma, \mathcal{D}, \approx, \vdash)$ 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 $\mathfrak{R}_{\text{minDFA}}(\Sigma, \mathcal{D}, \approx, \vdash)$. To this aim, we suppose a computable bijective mapping $obj : \mathfrak{R}_{\text{minDFA}}(\Sigma, \mathcal{D}, \approx, \vdash) \to [\mathcal{O}]_\sim = \{[o]_\sim \mid o \in \mathcal{O}\}$ (where $[o]_\sim = \{o' \in \mathcal{O} \mid o' \sim o\}$). A typical situation is depicted in Fig 3, where the larger ellipse is closed under $\approx$ ($w \approx w'$) and under $\vdash$ (assuming $\{w, w'\} \vdash \{w''\}$), whereas the smaller circle is not, as it contains u but not u'.

As Angluin's algorithm works within the class of arbitrary DFA over Σ, 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.

Let us be more precise and 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*

- *Σ 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,*
- *$obj : \mathfrak{R}_{\text{minDFA}}(\Sigma, \mathcal{D}, \approx, \vdash) \to [\mathcal{O}]_\sim$ is a bijective effective mapping in the sense that, for $L \in \mathfrak{R}_{\text{minDFA}}(\Sigma, \mathcal{D}, \approx, \vdash)$, a representative of $obj(L)$ can be computed.*

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

(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* INCLUSION$(\Sigma, \mathcal{D})$ *is* constructively decidable.

[5] Technically, $\approx$ and $\vdash$ could be encoded as a single relation. As they serve a different purpose in the next section, we separate them in the general framework, to simplify the forthcoming explanations.

(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* 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* INFCLOSURE$(\Sigma, \mathcal{D}, \approx, \vdash)$ *is* constructively decidable.

So let us slightly 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 in Angluin's algorithm concern the processing of membership queries as well as 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. Membership 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.[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, 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 $(\mathcal{O}, \sim)$.*

[6] The pseudo code of EXTENDEDANGLUIN is included in Appendix A

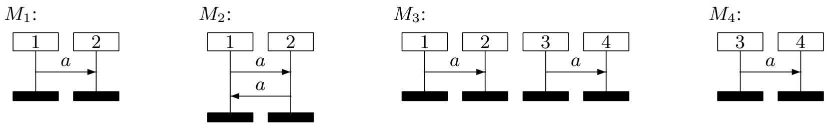


Fig. 4. Some MSCs

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 [6, 9]. 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

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 $\mathcal{L} \subseteq \mathbb{MSC}$ if $L \subseteq Lin(\mathcal{L})$ and, for any MSC M, $M \in \mathcal{L}$ iff $Lin(M) \cap L \neq \emptyset$.*

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 $\{w_1 w_2 \mid w_1 \in Lin(M_1), w_2 \in Lin(M_2)\} \subseteq Lin(M)$. $\{M\}^*$ 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 [18], as $Lin(\{M_1\}^*)$ is not a regular word language. However, $\{M_1\}^*$ can be represented by the regular word language $Lin^1(\{M_1\}^*) = \{(!(1,2,a)\ ?(2,1,a))^n \mid n \in \mathbb{N}\}$. Considering the MSC M_2 in Fig. 4, we even have that $Lin(\{M_2\}^*)$ is a regular representative for $\{M_2\}^*$.

The interesting case occurs when representatives are regular. However, some MSCs 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 $Lin^B(\{M_3\}^*)$ is regular. Thus, according to results from [17], it cannot be the language of some MPA.

Definition 7 ($\forall$- and $\exists$-regular). *$\mathcal{L} \subseteq \mathbb{MSC}$ is* $\forall$-regular *if $Lin(\mathcal{L}) \subseteq Act^*$ is regular. $\mathcal{L}$ is* $\exists$-regular *if, for some $B \in \mathbb{N}$, $Lin^B(\mathcal{L})$ 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 in the sense of Def. 7 are of interest as they are realizable by MPA.

Theorem 2 ([17, 18, 22]). *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 $\mathcal{A} \in \mathrm{MPA}_{\exists B}$, $Lin^B(\mathcal{L}(\mathcal{A}))$ is a regular representative for $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A})$ is $\exists$-regular. For $\mathcal{A} \in \mathrm{MPA}_\forall$, $Lin(\mathcal{L}(\mathcal{A}))$ is a regular representative for $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A})$ is $\forall$-regular.*

5.2 Product MSC Languages

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 $M = (E, \{\leq_p\}_{p \in Proc}, <_{\mathrm{msg}}, \ell) \in Pref(\mathbb{MSC})$, 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 $\mathcal{L} \subseteq \mathbb{MSC}$ and $M \in \mathbb{MSC}$, let $\mathcal{L} \vdash^{\mathrm{p}}_{\mathrm{MSC}} M$ 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]). *$\mathcal{L} \subseteq \mathbb{MSC}$ is a* product MSC language *if, for any $M \in \mathbb{MSC}$ and any finite $\mathcal{L}' \subseteq \mathcal{L}$, $\mathcal{L}' \vdash^{\mathrm{p}}_{\mathrm{MSC}} M$ 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 $\mathcal{L} \subseteq \mathbb{MSC}$ and $P \in Pref(\mathbb{MSC})$, we write $\mathcal{L} \vdash^{\mathrm{s}}_{\mathrm{MSC}} P$ 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 $\mathcal{L} \subseteq \mathbb{MSC}$ is called* safe *if, for any finite $\mathcal{L}' \subseteq \mathcal{L}$ and any $P \in Pref(\mathbb{MSC})$, $\mathcal{L}' \vdash^{\mathrm{s}}_{\mathrm{MSC}} P$ implies $P \preceq M$ for some $M \in \mathcal{L}$.*

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

5.3 Learning $\forall$-bounded Message-Passing Automata

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

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

To prove that $(Act, Lin(\mathbb{MSC}), \approx_{\mathrm{MW}}, \emptyset, obj_\forall)$ is indeed a learning setup for the pair $(\mathrm{MPA}_\forall, \sim_\forall)$, we recall and establish some decidability results concerning MSC languages.

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

Proof: The decidability part stems from [18, Prop. 2.4] and [26]. Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a *minimal* DFA over Act. A state $s \in Q$ is called *productive* if there is a path from s to some final state. We successively label productive states with possible channel contents. If there is such a labeling that is consistent in some sense, we can assume both that $L(\mathcal{A}) \subseteq Lin(\mathbb{MSC})$ and that $L(\mathcal{A})$ is the union of $\approx_{\text{MW}}$-equivalence classes. Let us be more precise. Any state s will be associated with a function $\chi_s : Ch \to Msg^*$ as follows:

1. The initial state and any final state are equipped with χ_ε (mapping any channel to the empty word).
2. If $s, s' \in Q$ are productive states and $\delta(s, !(p,q,a)) = s'$, then $\chi_{s'} = \chi_s[(p,q) := a \cdot \chi_s((p,q))]$.
3. If $s, s' \in Q$ are productive states and $\delta(s, ?(q,p,a)) = s'$, then we have $\chi_s = \chi_{s'}[(p,q) := \chi_{s'}((p,q)) \cdot a]$.

In fact, we have $L(\mathcal{A}) \subseteq Lin(\mathbb{MSC})$ iff a labeling of productive states with channel functions according to 1.–3. is possible. Moreover, we have that $L(\mathcal{A})$ is the union of $\approx_{\text{MW}}$-equivalence classes iff this labeling satisfies the following condition:

4. (*Diamond property*) Suppose $\delta(s, \sigma) = s_1$ and $\delta(s_1, \tau) = s_2$ with $\sigma \in Act_p$ and $\tau \in Act_q$ for some $p, q \in Proc$ satisfying $p \neq q$. If not ($\sigma = !(p,q,a)$ and $\tau = ?(q,p,a)$ for some $a \in Msg$) or $0 < |\chi_s((p,q))|$, then there exists a state $s_1' \in Q$ such that both $\delta(s, \tau) = s_1'$ and $\delta(s_1', \sigma) = s_2$.

Now suppose that labeling the state space with channel functions violates 1.–3. at some point. But this immediately yields a word that is not contained in $Lin(\mathbb{MSC})$. For example, a clash in terms of productive states $s, s' \in Q$ such that $\delta(s, !(p,q,a)) = s'$ and $\chi_{s'}((p,q)) \neq a \cdot \chi_s((p,q))$ gives rise to a path from the initial state to a final state via $s \xrightarrow{!(p,q,a)} s'$ that is not labeled with an MSC word. Similarly, provided $L(\mathcal{A}) \subseteq Lin(\mathbb{MSC})$ and property 4. is violated, we specify w and w' (as required in the proposition) as words of the form $u\sigma\tau v$ and $u\tau\sigma v$, respectively. $\square$

The above decision algorithm runs in time linear in the size of the transition function of the DFA. It is easy to see that counterexamples can be computed in linear time as well. Note that the question if the $\approx_{\text{MW}}$-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 $L(\mathcal{A}) \subseteq Lin(\mathbb{MSC})$ (which are not necessarily deterministic), it was shown in [26] (for Büchi automata) that deciding if $L(\mathcal{A})$ is $\approx_{\text{MW}}$-closed is PSPACE complete. In the context of minimal DFA, however, the problem becomes much simpler.

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

Theorem 3. ($\text{MPA}_\forall, \sim_\forall$) *is learnable.*

5.4 Learning ∃-bounded Message-Passing Automata

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

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

In the following, we will show that $(\mathit{Act}, \mathit{Lin}^B(\mathbb{MSC}), \approx_{\exists B}, \emptyset, \mathit{obj}_{\exists B})$ is indeed a learning setup for $(\text{MPA}_{\exists B}, \sim_{\exists B})$. Again, we have to establish the corresponding decidability results:

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

Proof: We need to adapt the universal case accordingly (Prop. 1) and require that, associating a state s with a channel function $\chi_s : \mathit{Ch} \to \mathit{Msg}^*$, we have $|\chi_s(ch)| \leq B$ for any channel ch. Moreover, the diamond property is replaced with the following:

4. Suppose $\delta(s, \sigma) = s_1$ and $\delta(s_1, \tau) = s_2$ with $\sigma \in \mathit{Act}_p$ and $\tau \in \mathit{Act}_q$ for some $p, q \in \mathit{Proc}$ satisfying $p \neq q$. If not ($|\chi_s((q, q'))| = B$ and $\tau = !(q, q', a)$ for some $q' \in \mathit{Proc}$ and $a \in \mathit{Msg}$) and, moreover, ($\sigma = !(p, q, a)$ and $\tau = ?(q, p, a)$ for some $a \in \mathit{Msg}$) implies $0 < |\chi_s((p, q))|$, then there exists a state $s'_1 \in Q$ such that both $\delta(s, \tau) = s'_1$ and $\delta(s'_1, \sigma) = s_2$. □

Proposition 4. *For any $B \in \mathbb{N}$, $(\mathit{Act}, \mathit{Lin}^B(\mathbb{MSC}), \approx_{\exists B}, \emptyset, \mathit{obj}_{\exists B})$ 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.*

5.5 Learning ∀-bounded Safe Product Message-Passing Automata

Let us set the scene for learning ∀-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 $\vdash^{\text{p}}_{\text{MW}}$ and $\vdash^{\text{s}}_{\text{MW}}$ for word languages, which correspond to $\vdash^{\text{p}}_{\text{MSC}}$ and $\vdash^{\text{s}}_{\text{MSC}}$, respectively:

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

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

- $\sim^{\text{sp}}_{\forall} = \{(\mathcal{A}, \mathcal{A}') \in \text{MPA}^{\text{sp}}_{\forall} \times \text{MPA}^{\text{sp}}_{\forall} \mid \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')\}$,
- $\approx_{\text{MW}} = \{(w, w') \in \mathit{Lin}(M) \times \mathit{Lin}(M) \mid M \in \mathbb{MSC}\}$ (as before),
- $\vdash^{\text{sp}}_{\text{MW}} = \vdash^{\text{p}}_{\text{MW}} \cup \vdash^{\text{s}}_{\text{MW}}$,

– $obj^{\text{sp}}_{\forall} : \mathfrak{R}_{\text{minDFA}}(Act, Lin(\mathbb{MSC}), \approx_{\text{MW}}, \vdash^{\text{sp}}_{\text{MW}}) \to [\text{MPA}^{\text{sp}}_{\forall}]_{\sim^{\text{sp}}_{\forall}}$ be an effective bijective mapping, as guaranteed by Lemma 1.

We now establish that $(Act, Lin(\mathbb{MSC}), \approx_{\text{MW}}, \vdash^{\text{sp}}_{\text{MW}}, obj^{\text{sp}}_{\forall})$ is a learning setup for $(\text{MPA}^{\text{sp}}_{\forall}, \sim^{\text{sp}}_{\forall})$:

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

Proof: Decidability of $\text{INFCLOSURE}(Act, Lin(\mathbb{MSC}), \approx_{\text{MW}}, \vdash^{\text{sp}}_{\text{MW}})$ has been shown in [3, Theorem 3], where an EXPSPACE-algorithm for bounded high-level MSCs is given, which reduces the problem to a decision problem for finite automata with a $\approx_{\text{MW}}$-closed language. The first step is to construct from the given $\approx_{\text{MW}}$-closed DFA $\mathcal{H}$ a (componentwise) minimal and deterministic product MPA $\mathcal{A}$, by simply taking the projections of $\mathcal{H}$ onto Act_p for any $p \in Proc$, 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. $\vdash^{\text{s}}_{\text{MW}}$, i.e., $L(\mathcal{H})$ must actually contain a completion $uv \in Lin(\mathbb{MSC})$ 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 $p \in Proc$, compute a word $w_p \in L(\mathcal{H})$ such that the projection of u onto Act_p is a prefix of the projection of w_p onto Act_p. We set $L_1 = \bigcup_{p \in Proc}[w_p]_{\approx_{\text{MW}}}$.

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. $\vdash^{\text{p}}_{\text{MSC}}$. 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* $(Act, Lin(\mathbb{MSC}), \approx_{\text{MW}}, \vdash^{\text{sp}}_{\text{MW}}, obj^{\text{sp}}_{\forall})$ *is a learning setup for* $(\text{MPA}^{\text{sp}}_{\forall}, \sim^{\text{sp}}_{\forall})$.

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

5.6 Learning ∀-bounded Product Message-Passing Automata

Finally, we study the problem of learning ∀-bounded product MPA. Unfortunately, we are in the situation that the canonical definition of a learning setup does not work. For $\sim^{\text{p}}_{\forall} = \{(\mathcal{A}, \mathcal{A}') \in \text{MPA}^{\text{p}}_{\forall} \times \text{MPA}^{\text{p}}_{\forall} \mid \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')\}$, we obtain:

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

Similar decision problems were considered in [2, 3, 23, 25]. Most of them, however, are concerned with the question if a *high-level MSC*, rather than a regular MSC language, can be translated into a product MPA.

6 Tool Description

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 [28], which implements *Angluin's algorithm*, and the libraries `Grappa` [5] and `JGraph` [24] for visualization purposes. For computing linearizations of MSCs we use the algorithm given in [31] 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 (∃/∀)

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 message sequence chart as positive or negative (cf. Fig. 5 (1)). The classification for validity wrt. the learning setup. Depending on the outcome *Assistant* checks the of this check, the linearizations of the current MSC

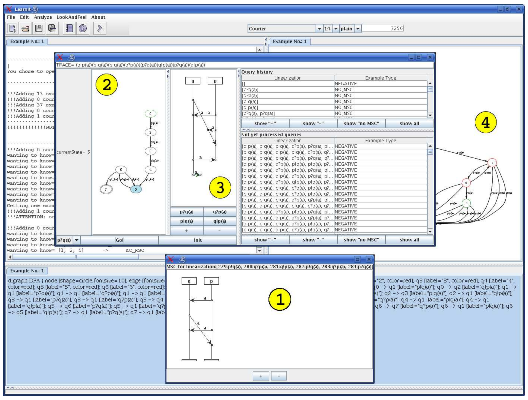


Fig. 5. *Smyle* screenshot

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 automaton (`2,4`) as well as 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.

Case studies: We applied *Smyle* to the *simple negotiation protocol* from [14], the *continuous update protocol* from [15] and a protocol being part of USB 1.1 mentioned in [16]. 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. And the last protocol was learned after 1373 membership and 12 user-queries, providing it with 4 sample MSCs. In this case, the inferred automaton was composed of 9 states. For further details we refer to Appendix B where we list two of these protocols as well as the input MSCs and the corresponding learned automata.

7 Conclusion and Future Work

This paper presented a procedure that interactively infers a message-passing automaton (MPA) from given positive and negative scenarios of the system's behavior provided as message sequence charts (MSCs). In doing so, we generalized Angluin's learning algorithm for deterministic finite-state automata (DFA) towards learning specific classes of bounded MPA.

It was shown that elements of the classes of $\forall$-regular (safe product) MPA can be learned and that elements of the class of $\exists$-regular MPA can be learned provided an a priori bound is given. A similar approach for learnability of $\forall$-regular product MPA fails. It is left open whether $(\mathrm{MPA}^{\mathrm{p}}_{\forall}, \sim^{\mathrm{p}}_{\forall})$ is learnable. Note that there are other interesting classes of learnable MPA. For example, our setting easily applies to the causal closure as defined by Adsul et al., which has the nice property that the causal closure of any regular MSC language is regular [1].

We developed *Smyle* as a prototype supporting the inference of design models from scenario-based specifications to validate our approach in practice. As future work, we plan to integrate *MSCan* [7] into *Smyle* to support the 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*, LNCS, pages 335–347, Hyderabad, India, 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. *Formal Models of Communicating Systems — Languages, Automata, and Monadic Second-Order Logic*. Springer, 2006.
7. B. Bollig, C. Kern, M. Schlütter, and V. Stolz. MSCan: A tool for analyzing MSC specifications. In H. Hermanns and J. Palsberg, editors, *Proceedings of the 12th International Conference on Tools and Algorithms for Construction and Analysis of Systems TACAS'06*, volume 3920 of *Lecture Notes in Computer Science*, pages 455–458, Vienna, Austria, Mar. 2006. Springer.
8. B. Bollig and M. Leucker. A hierarchy of implementable MSC languages. In *FORTE*, volume 3731 of *LNCS*, pages 53–67. Springer, 2005.
9. B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Th. Comp. Sc.*, 358(2-3):150–172, 2006.
10. 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.
11. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. of the ACM*, 30(2):323–342, 1983.
12. W. Damm and D. Harel. Lscs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:1:45–80., 2001.
13. D.Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
14. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-Based Agent Communication Protocols. In *Workshop on Agent Communication Languages*, pages 91–107, 2003.
15. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol Conformance for Logic-based Agents. In *IJCAI*, pages 679–684, 2003.
16. B. Genest. Compositional Message Sequence Charts (CMSCs) Are Better to Implement Than MSCs. In *TACAS*, pages 429–444, 2005.
17. 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.
18. 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.
19. J. G. Henriksen, M. Mukund, K. N. Kumar, and P. S. Thiagarajan. Regular collections of message sequence charts. In *MFCS*, volume 1893 of *LNCS*. Springer, 2000.
20. H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *Computer-Aided Verification*, volume 2725 of *LNCS*, pages 315–327. Springer, 2003.
21. I. Krüger, R. Grosu, P. Scholz, and M. Broy. From mscs to statecharts. In F. J. Rammig, editor, *DIPES*, volume 155 of *IFIP Conference Proceedings*, pages 61–72. Kluwer, 1998.
22. D. Kuske. Regular sets of infinite message sequence charts. *Inf. Comput.*, 187:80–109, 2003.
23. M. Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Th. Comp. Sc.*, 309(1-3):529–554, 2003.
24. J. Ltd. *JGraph - Java Graph Visualization and Layout*. http://www.jgraph.com/.
25. R. Morin. Recognizable sets of message sequence charts. In *STACS*, volume 2285 of *LNCS*, pages 523–534. Springer, 2002.
26. A. Muscholl and D. Peled. From finite state communication protocols to high-level message sequence charts. In *ICALP*, volume 2076 of *LNCS*, pages 720–731. Springer, 2001.
27. B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *Int. Conf. on Software Engineering (ICSE)*, pages 35–46. ACM, 2000.
28. H. Raffelt and B. Steffen. Learnlib: A library for automata learning and experimentation. In *FASE*, volume 3922 of *LNCS*, pages 377–380, 2006.
29. B. Sengupta and R. Cleaveland. Executable requirements specifications using triggered message sequence charts. In G. Chakraborty, editor, *ICDCIT*, volume 3816 of *Lecture Notes in Computer Science*, pages 482–493. Springer, 2005.

30. S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. Softw. Eng.*, 29(2):99–115, 2003.
31. Y. L. Varol and D. Rotem. An algorithm to generate all topological sorting arrangements. *Comput. J.*, 24(1):83–84, 1981.

A ExtendedAngluin

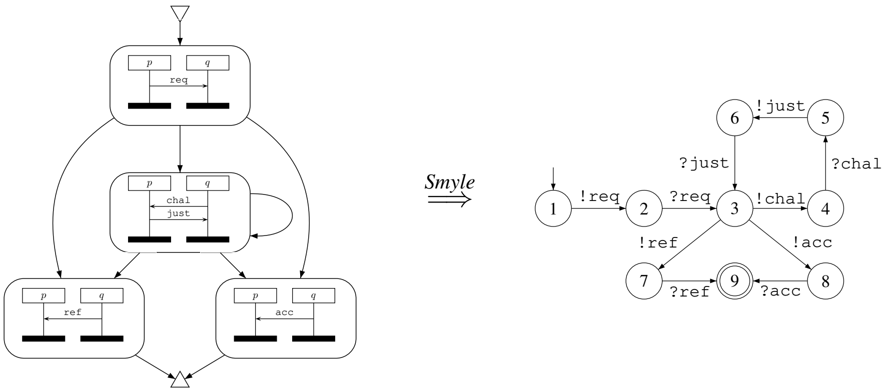
The extension of Angluin's algorithm wrt. a learning setup $(\Sigma, \mathcal{D}, \approx, \vdash, obj)$ for $(\mathcal{O}, \sim)$ is sketched in Table 1. Note that in this table we do not explicitly deal with the $Assistant$ keeping track of membership queries to avoid queries that are redundant due to $\approx$.

B Case Studies

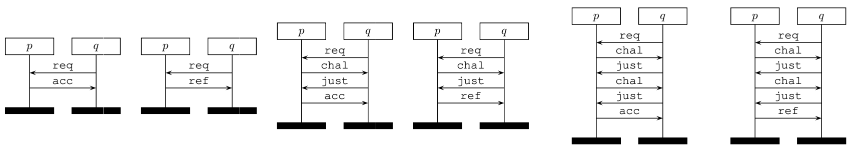
In the following we describe two of the protocols *Smyle* was applied to. In the first one (the *simple negotiation protocol* from [14]) client p sends a `req`uest to the server q. The server may either directly `acc`ept or `ref`use the client's request or enter a `chal`lenge-`just`ify phase in which he asks for more information from the client. As long as the server is not satisfied with the information provided by the client he stays in this phase. Once the server collected enough information he decides whether to `acc`ept or `ref`use the client's initial request.

The *simple negotiation protocol* (represented as high-level MSC):

$i)$



The six example MSCs *Smyle* was provided with:



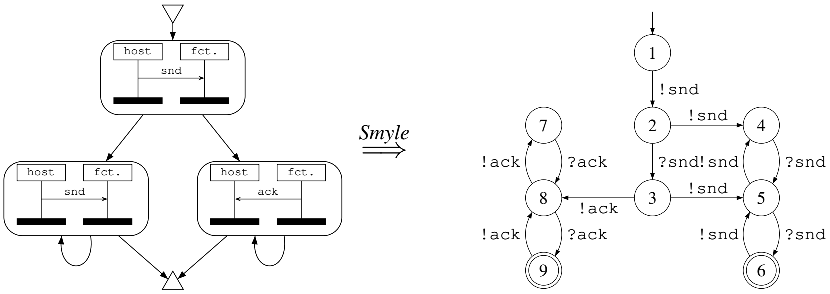
The second protocol is part of the USB 1.1 specification. The first message sent from the `host` informs the `function` that the isochronous mode (USB distinguishes between three kinds of modes: isochronous, bulk and setup) will be used and also informs whether the `function` has to play the role of the receiver or the transmitter. Depending on this decision, the protocol either turns to the left or the right node of the HMSC and stays there until the transmission is complete.

Table 1. The extension of Angluin's algorithm

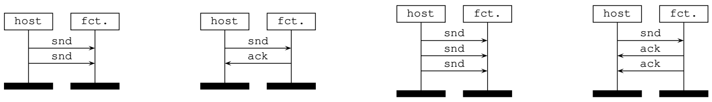
ExtendedAngluin$((\mathcal{O}, \sim), (\Sigma, \mathcal{D}, \approx, \vdash, obj))$:
initialize (U, V, T) by asking membership queries for all $w \in \{\varepsilon\} \cup \Sigma$
repeat
while T is not (closed and consistent)
do
if T is not consistent **then**
find $u, u' \in U$, $a \in \Sigma$, and $v \in V$ such that
$T(u) = T(u')$ and $T(ua)(v) \neq T(u'a)(v)$
add av to V
extend T to $(U \cup U\Sigma)V$ by membership queries
if T is not closed **then**
find $u \in U$ and $v \in V$ such that $T(ua) \neq T(u')$ for any $u' \in U$
add ua to U
extend T to $(U \cup U\Sigma)V$ by membership queries
/$*$ T is both closed and consistent $*$/
from T, construct the hypothesized DFA $\mathcal{H}$
if $L(\mathcal{H}) \not\subseteq \mathcal{D}$
then
compute $w \in L(\mathcal{H}) \setminus \mathcal{D}$
add w and all its prefixes to U
extend T to $(U \cup U\Sigma)V$ by membership queries
where the query for w is answered negatively
else
if $L(\mathcal{H})$ is not $\approx$-closed
then
compute $w, w' \in \mathcal{D}$ such that $w \approx w'$, $w \in L(\mathcal{H})$, and $w' \notin L(\mathcal{H})$
add any $u \in [w]_\approx$ and all its prefixes to U
extend T to $(U \cup U\Sigma)V$ by membership queries
else
if $L(\mathcal{H})$ is not $\vdash$-closed
then
compute $(L_1, L_2) \in \vdash$ such that $L_1 \subseteq L(\mathcal{H})$ and $L(\mathcal{H}) \cap L_2 = \emptyset$
add any $u \in L_1$ and all its prefixes to U
extend T to $(U \cup U\Sigma)V$ by membership queries
if any membership query for L_1 is answered positively **then**
ask for $w \in L_2$ (as positive example)
add w and all its prefixes to U
extend T to $(U \cup U\Sigma)V$ by membership queries
where the query for w is answered positively
else
compute $obj(\mathcal{H})$ and do equivalence test
if equivalence test fails **then**
counterexample w is provided
w and all its prefixes are added to U
extend T to $(U \cup U\Sigma)V$ by membership queries
until equivalence test succeeds
return $obj(L(\mathcal{H}))$

A protocol being part of the USB 1.1 protocol (represented as high-level MSC):

ii)



The four example MSCs *Smyle* was provided with:



Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from `http://aib.informatik.rwth-aachen.de/`. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2001-01 * Jahresbericht 2000
2001-02 Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03 Thierry Cachat: The power of one-letter rational languages
2001-04 Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05 Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07 Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08 Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09 Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11 Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 * Jahresbericht 2001
2002-02 Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03 Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04 Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05 Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07 Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09 Markus Mohnen: Interfaces with Default Implementations in Java
2002-10 Martin Leucker: Logics for Mazurkiewicz traces
2002-11 Jürgen Giesl, Hans Zantema: Liveness in Rewriting

2003-01 * Jahresbericht 2002

2003-02 Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting

2003-03 Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations

2003-04 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs

2003-05 Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard

2003-06 Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates

2003-07 Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung

2003-08 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs

2004-01 * Fachgruppe Informatik: Jahresbericht 2003

2004-02 Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic

2004-03 Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting

2004-04 Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming

2004-05 Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming

2004-06 Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming

2004-07 Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination

2004-08 Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information

2004-09 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity

2004-10 Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules

2005-01 * Fachgruppe Informatik: Jahresbericht 2004

2005-02 Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"

2005-03 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions

2005-04 Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem

2005-05 Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots

2005-06 Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information

2005-07 Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks

2005-08 Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut

2005-09 Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures

2005-10 Benedikt Bollig: Automata and Logics for Message Sequence Charts

2005-11 Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture

2005-12 Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems

2005-13 Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments

2005-14 Felix C. Freiling, Sukumar Ghosh: Code Stabilization

2005-15 Uwe Naumann: The Complexity of Derivative Computation

2005-16 Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)

2005-17 Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)

2005-18 Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises "Features"

2005-19 Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers

2005-20 Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.

2005-21 Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited

2005-22 Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins

2005-23 Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves

2005-24 Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

2006-01 * Fachgruppe Informatik: Jahresbericht 2005

2006-03 Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler

2006-04 Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05 Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F

2006-06 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color

2006-07 Thomas Colcombet, Christof Löding: Transforming structures by set interpretations

2006-08 Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs

2006-09 Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

* These reports are only available as a printed version.

Please contact biblio@informatik.rwth-aachen.de to obtain copies.