Learning Communicating Automata from MSCs

Benedikt Bollig, Joost–Pieter Katoen, *Member, IEEE Computer Society,* Carsten Kern and Martin Leucker

***Index Terms*—D.2.1.b: Software Engineering / Requirements / Specifications / Elicitation methods; D.2.10.a: Software Engineering / Design / Design concepts; I.2.6.e: Computing Methodologies / Artificial Intelligence / Learning / Induction ; F.1.1.a: Theory of Computation / Computation by Abstract Devices / Models of Computation / Automata**

***Abstract*—This paper is concerned with bridging the gap between requirements and distributed systems. Requirements are defined as basic message sequence charts (MSCs) specifying positive and negative scenarios. Communicating finite-state machines (CFMs), i.e., finite automata that communicate via FIFO buffers, act as system realizations. The key contribution is a generalization of Angluin's learning algorithm for synthesizing CFMs from MSCs. This approach is exact—the resulting CFM precisely accepts the set of positive scenarios and rejects all negative ones—and yields fully asynchronous implementations. The paper investigates for which classes of MSC languages CFMs can be learned, presents an optimization technique for learning partial orders, and provides substantial empirical evidence indicating the practical feasibility of the approach.**

I. Introduction

The software engineering development cycle starts with elicitating requirements. Requirement capturing techniques of various nature exist. Popular requirement engineering methods, such as the Inquiry Cycle and CREWS [36], exploit use cases and scenarios for specifying system's requirements. Scenarios given as sequence diagrams are also at the heart of the UML (Unified Modeling Language). A scenario is a partial fragment of the system's behavior given as visual representation indicating the system components (vertically) and their message exchange over time (horizontally). Their intuitive yet formal nature has led to a broad acceptance by the software engineering community, both in academia as well as in industry. Scenarios can be positive or negative, indicating either a possible desired or an unwanted system behavior, respectively. Different scenarios together form a more complete description of the system behavior.

Requirements capturing is typically followed by a first design step of the system at hand. This step naturally ignores many implementation details and aims to obtain an initial system structure at a high level of abstraction. In case of a distributed system realization this, e.g., amounts to determine which processes are to be distinguished, what their high-level behaviour is, and which capacities of communication channels suffice to warrant a deadlock-free process interaction. This design phase in software engineering is highly challenging as it concerns a complex paradigm shift between the requirement specification—a partial, overlapping and possibly inconsistent description of the system's behavior that is subject to rapid change—and a conforming design model, a first complete behavioral description of the system.

The fact that target systems are often distributed complicates matters considerably as combining several individual processes may easily yield realizations that handle more than the specified scenarios, i.e., they may over-approximate the system requirements, or that suffer from deadlocks. During the synthesis of such distributed design models, conflicting requirements are detected and resolved. As a consequence, the requirements specification is adapted by adding or omitting scenarios. Besides, a thorough analysis of the design model, e.g., by means of model checking or simulation, requires fixing errors in the requirements. Obtaining a complete and consistent set of requirements together with a conforming design model is thus a complex and highly iterative process.

This paper considers requirements that are given as message sequence charts (MSCs). These MSCs are basic; high-level constructs to combine MSCs by alternative, sequential or repetitive composition are not considered. This yields a simple, yet still effective requirement specification formalism that is expressive, easy to grasp and understand. For the design models we focus on distributed systems where each process behaviour is described as a finite-state machine and processes exchange messages asynchronously via order-preserving communication channels. These communicating finite-state machines (CFMs [13]) are commonly adopted for realizing MSCs [3, 7, 23–27, 32, 35].

We exploit *learning* algorithms [4] to synthesize CFMs from requirements given as set of (positive and negative) basic MSCs. Learning fits well with the incremental generation of design models as it is feasible to infer a design model on the basis of an initial set of scenarios, CFMs are adapted in an automated manner on adding and deletion of MSCs, and diagnostic feedback is provided that may guide an amendment of the requirements when establishing an inconsistency of a set of scenarios. The use of learning for system synthesis from scenario-based requirements specifications is not new and has been proposed by several authors, see, e.g., [16, 33, 40, 41]. The main characteristics of our approach are the unique combination of:

(i) positive and *negative* MSCs are naturally supported;
(ii) realized processes interact *fully asynchronously*;
(iii) synthesized CFMs *precisely* exhibit the behaviour as specified by the MSCs;
(iv) effective optimizations tailored to *partial orders* like MSCs.

Existing learning-based synthesis techniques typically consider just possible and no undesired behaviours, yield synchronously (or partially asynchronously) interacting automata, and, most importantly, suffer from the fact that synthesized

B. Bollig is with ENS Cachan & CNRS
J.P. Katoen and Carsten Kern are with RWTH Aachen University
M. Leucker is with Technical University Munich

realizations may exhibit more behaviour than specified. That is, the obtained realizations are in fact over-approximations.

Technically speaking, this paper makes the following contributions. We extend Angluin's learning algorithm for inferring deterministic finite automata from regular languages to a learning algorithm that allows for synthesizing CFMs from MSC languages, i.e., sets of words that are described by a set of MSCs. The generalized learning algorithm is described in detail, a general learning set-up is defined, and the correctness and time-complexity of our algorithm are established. We then consider existentially, and, respectively, universally bounded CFMs, i.e., CFMs for which some, respectively all, possible event orderings can be realized with finite communication channels. It is shown that universally bounded deadlock-free CFMs, existentially bounded CFMs with an a priori fixed channel capacity, and universally-bounded weak CFMs are all learnable. We subsequently show how the memory consumption of our algorithm can be improved using so-called partial order learning. Exploiting that MSCs are in fact partial orders, this approach amounts to merging rows and columns (in the table used for learning) such that only congruent prefixes and suffixes need to be stored. The correctness of this modification is shown, and it is indicated by means of several experiments that this leads to siginificant memory savings. These experiments are carried out with the software tool *Smyle* [9], which implements several algorithms presented in this paper. Let us summarize the user tasks of our approach:

(i) *Membership queries* are posed by the learner and are mostly handled in an automated manner. A small fragment of these queries has to be answered manually. They are called *user queries*. In this case, the user has to classify the presented basic MSCs as either positive or negative.

(ii) Due to the high degree of automation, *equivalence queries* are rather rare. Here, the user has to check whether the generated automaton is correct. To facilitate this, *Smyle* supports testing and simulation of automata. In case an incorrect behaviour is detected, the user has to provide a counterexample MSC.

Organization of this paper. Section II introduces MSCs, (several classes of) CFMs, and summarizes main realizability results. Section III describes Angluin's learning algorithm in detail. Section IV constitutes the main part of this paper and extends Angluin's learning algorithm to enable inferring CFMs (rather than deterministic finite automata). To that end, we define a general learning setup and determine several CFM classes that are learnable. Section V presents an efficiency improvement of our learning algorithm by considering normal forms of equivalences classes of words generated by MSCs. Section VI describes some case studies, and shows that congruence-based learning improves the memory consumption with a factor of up to almost 75%. The paper closes with discussing related work and an epilogue. A preliminary version of this paper appeared as [8].

II. MSCs and Communicating Automata

In this section, we introduce two fundamental concepts: *message sequence charts* (MSCs) and *communicating finite-state machines* (CFMs) [13]. The former constitute an appealing and easy to understand graphical specification formalism. The latter, also known as *message passing automata* (MPA), serve as design models for the system to learn and model the communication behavior of distributed systems composed of finite-state components.

Before we start, let us recall some basic notation and terminology. An *alphabet* is a nonempty finite set whose elements are called *actions*. Finite sequences of actions are elements of Σ^* and are called *words*. Sets of words are termed *languages* and are thus subsets of Σ^*. For a word $w \in \Sigma^*$ we denote by $pref(w)$ ($suff(w)$) the set of all its prefixes (suffixes, respectively) including w itself and the empty word ϵ. We extend $pref$ and $suff$ to languages $L \subseteq \Sigma^*$ letting $pref(L) := \bigcup_{w \in L} pref(w)$ and $suff(L) := \bigcup_{w \in L} suff(w)$. We denote the powerset of a set X by 2^X.

A. Message Sequence Charts

A common design practice when developing communicating systems is to start with specifying scenarios to exemplify the intended interaction of the system to be. *Message sequence charts* (MSCs) provide a prominent notion to further this approach. They are widely used in industry, are standardized [29, 30], and resemble UML's sequence diagrams [5]. An MSC depicts a single partially ordered execution sequence of a system. It consists of a collection of processes, which, in their visual representation, are drawn as vertical lines and are interpreted as top-down time axes. Moreover, an arrow from one line to a second corresponds to the communication events of sending and receiving a message. An example MSC is illustrated in Fig. 1(a). The benefit of such a diagram is that one grasps its meaning at a glance. In the example scenario, messages m_1 and m_2 are sent from process p to process q. A further message m originates at process r and is finally received at q. However, one still has to reach an agreement on the system architecture, which does not necessarily emerge from the picture. Namely, following the MSC standard, we assume asynchronous communication: the send and receipt of a message might happen time-delayed. More precisely, there is an unbounded FIFO channel in between two processes that allows a sender process to proceed while the message is waiting for being received. Moreover, we assume a single process to be sequential: the events of one particular process are totally ordered in accordance with their appearance on its time axis. For example, regarding Fig. 1(a), we suppose that sending m_2 occurs after sending m_1. However, as the relative speed of the processes is unknown, we do not know if m_1 is received before m_2 is sent. Thus, the latter two events remain unordered.

We conclude that, in a natural manner, an MSC can be seen as a labeled partial order (labeled poset) over its events. Fig. 1(b) depicts the Hasse diagram of the labeled poset that one would associate with the diagram from Fig. 1(a). Its elements $1, \ldots, 6$ represent the endpoints of the message arrows and are called *events*. The edge relation then reflects the two constraints on the order of execution of the events: (i) events that are located on the same process line are totally

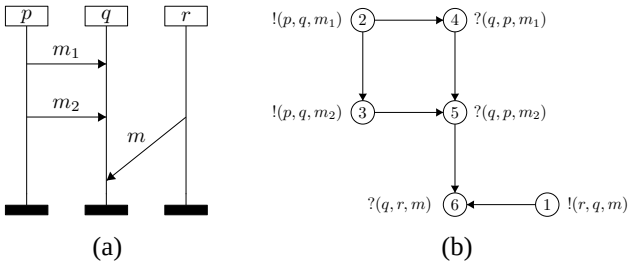


Fig. 1. MSC as a diagram (a) and as a graph (b)

ordered, and (ii) a send event has to precede the corresponding receive event. Indeed, it is reasonable to require that the transitive closure of constraints (i) and (ii) is a partial order. To keep track of the nature of an event in the poset representation, any such event is labeled with an action. Thus, a possible label is either

- a send action, which is of the form $!(p,q,m)$ meaning that p sends a message m to q, or
- a receive action, which is of the form $?(q,p,m)$ and is the complementary receive action executed by process q.

The alphabet of actions is, therefore, parametrized by nonempty and finite sets $Proc$ of *processes* and Msg of *messages*, which we suppose to be fixed in the following. We suppose $|Proc| \geq 2$. Recall that we assume an exchange of messages through channels. The set of *channels* is denoted $Ch = \{(p,q) \in Proc \times Proc \mid p \neq q\}$. The set Act_p of *actions* that may be executed by process p is given by $Act_p = \{!(p,q,m) \mid (p,q) \in Ch \text{ and } m \in Msg\} \cup \{?(q,p,m) \mid (p,q) \in Ch \text{ and } m \in Msg\}$. Moreover, let $Act = \bigcup_{p \in Proc} Act_p$ denote the set of all actions. Before we formally define what we understand by an MSC, let us first consider general Act-labeled posets, i.e., structures $(E, \preceq, \lambda)$ where E is a finite set of *events*, λ is a labeling function of the form $E \to Act$, and $\preceq$ is a partial-order relation (it is reflexive, transitive, and antisymmetric). For process $p \in Proc$, let $\preceq_p := \preceq \cap (E_p \times E_p)$ be the restriction of $\preceq$ to $E_p := \lambda^{-1}(Act_p)$ (which will later be required to give rise to a total order). Moreover, we define the relation $\prec_{\text{msg}} \subseteq E \times E$ to detect corresponding send and receive events: $i \prec_{\text{msg}} j$ if there are a channel $(p,q) \in Ch$ and a message $m \in Msg$ such that

- $\lambda(i) = !(p,q,m)$, $\lambda(j) = ?(q,p,m)$, and
- $|\{i' \preceq i \mid \lambda(i') = !(p,q,m') \text{ for some } m' \in Msg\}| = |\{j' \preceq j \mid \lambda(j') = ?(q,p,m') \text{ for some } m' \in Msg\}|$.

That is, events i and j correspond to a message exchange only if the number of messages that have been sent through channel (p,q) before i equals the number of messages that have been received before j. This ensures FIFO communication.

Definition 1 (Message Sequence Chart (MSC)) *An* MSC *is an Act-labeled poset $(E, \preceq, \lambda)$ such that*

- *for all $p \in Proc$, $\preceq_p$ is a total order on E_p,*
- *$\preceq = (\prec_{\text{msg}} \cup \bigcup_{p \in Proc} \preceq_p)^*$, and*
- *$\forall i \in E. \exists j \in E.\, i \prec_{\text{msg}} j$ or $j \prec_{\text{msg}} i$.*

See Fig. 1(a) and Fig. 2 for some example MSCs.

Stated in words, an MSC is an Act-labelled poset such that events occurring at a single process are totally ordered, and that event is either a send or a receive event. For these events the order is fixed. Independent events, though, can occur in any order. Sequential observations of labeled posets are called linearizations. A *linearization* of an Act-labeled poset $(E, \preceq, \lambda)$ is any saturation of $\preceq$ to a total order $\preceq'$, i.e., $e_{i_1} \preceq' \ldots \preceq' e_{i_n}$, where $(i_1, \ldots, i_n)$ is a permutation of $(1, \ldots, n)$ such that, for all $j, k \in \{1, \ldots, n\}$, $e_{i_j} \preceq e_{i_k}$ implies $j \leq k$. A linearization $e_{i_1} \ldots e_{i_n}$ corresponds to the word $\lambda(e_{i_1}) \ldots \lambda(e_{i_n}) \in Act^*$ and, by abuse of nomenclature, we call $\lambda(e_{i_1}) \ldots \lambda(e_{i_n})$ a linearization as well. For example,

$$!(r,q,m)\,!(p,q,m_1)\,!(p,q,m_2)\,?(q,p,m_1)\,?(q,p,m_2)\,?(q,r,m)$$

is a linearization of the MSC in Fig. 1(a). The set of linearizations of a labeled poset $\mathcal{M}$ will be denoted by $Lin(\mathcal{M})$. This mapping is canonically extended towards sets $\mathcal{L}$ of partial orders: $Lin(\mathcal{L}) = \bigcup_{\mathcal{M} \in \mathcal{L}} Lin(\mathcal{M})$.

B. Communicating Finite-State Machines

MSCs constitute a visual high-level specification formalism. They can be represented graphically and offer an intuitive semantics (in terms of their linearizations). On the computational side, we consider automata models that reflect the kind of communication that is depicted in an MSC. We now turn towards an automata model that, in a natural manner, generates collections of MSCs. More precisely, it generates action sequences that follow an *all-or-none* law: either all linearizations of an MSC are generated, or none of them.

A communicating finite-state machine (CFM) is a collection of finite-state machines, one for each process. According to the assumptions that we made for MSCs, we assume that communication between these machines takes place via (a priori) unbounded reliable FIFO channels. The underlying system architecture is again parametrized by the set $Proc$ of processes and the set Msg of messages. Recall that this gives rise to the set Act of actions, which will provide the transition labelings. In our automata model, the effect of executing a send action of the form $!(p,q,m)$ by process p is to put message m at the end of the channel (p,q) from process p to process q. Receive actions, written as $?(q,p,m)$, are only enabled when the requested message m is found at the head of the channel (p,q). When enabled, its execution by process q removes the corresponding message m from the channel from p to q.

It has been shown that the CFM model derived from concepts explained so far has a limited expressiveness. Certain protocols cannot be implemented without possible deadlocks by CFMs, unless the model is enriched by so-called *control* or *synchronization messages* [6, 12]. Therefore, we extend our alphabet wrt. a fixed infinite supply of control messages Λ. Let Act_p^Λ contain the symbols of the form $!(p,q,(m,\lambda))$ or $?(p,q,(m,\lambda))$ where $!(p,q,m) \in Act_p$ (respectively $?(p,q,m) \in Act_p$) and $\lambda \in \Lambda$. Intuitively, we tag messages with some control information λ to circumvent deadlocks. Finally, let $Act^\Lambda = \bigcup_{p \in Proc} Act_p^\Lambda$.

Definition 2 (Communicating Finite-State Machine (CFM)) *A* communicating finite-state machine *(CFM) is a structure*

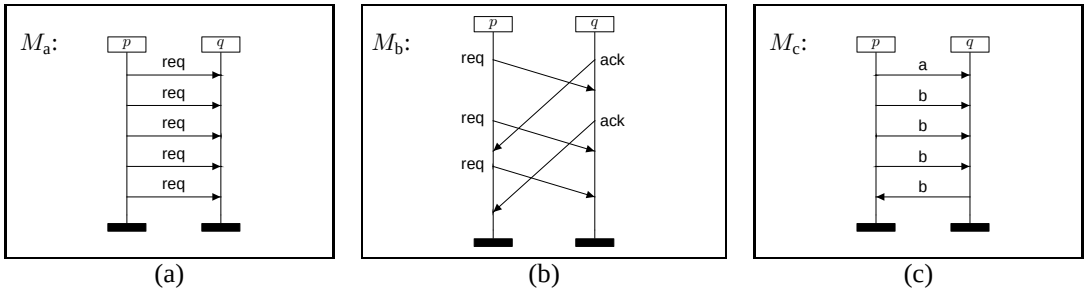


Fig. 2. Example message sequence charts

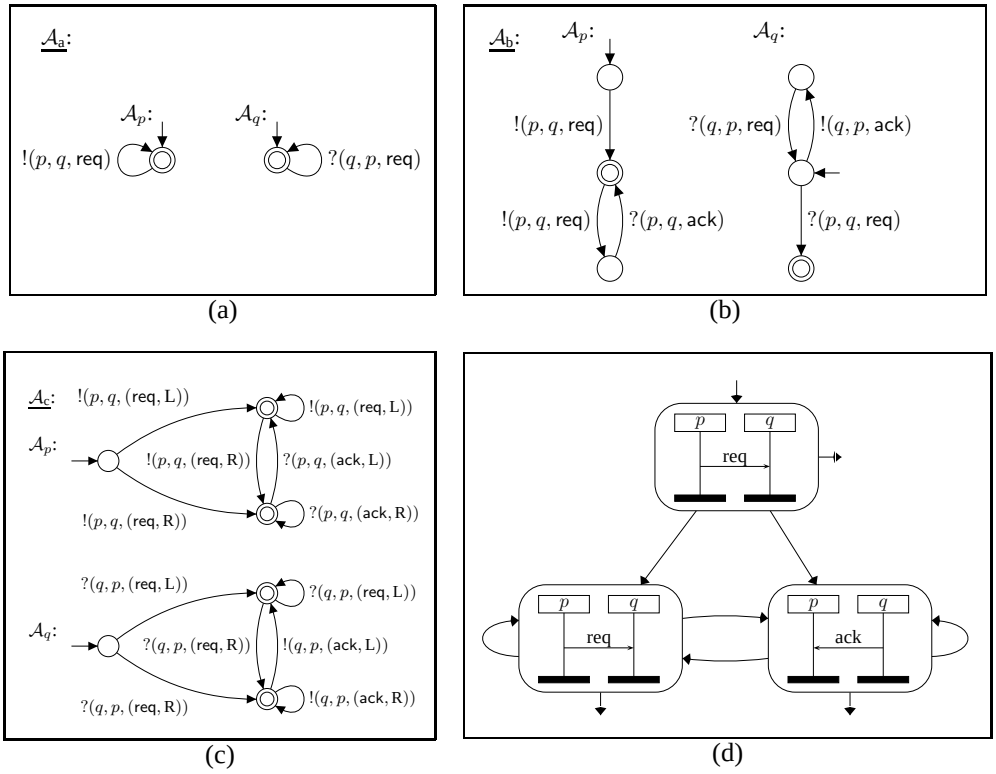


Fig. 3. Example communicating finite-state machines

$\mathcal{A} = ((\mathcal{A}_p)_{p\in Proc}, \mathcal{I})$. *For any process* $p \in Proc$, $\mathcal{A}_p = (S_p, \Delta_p, F_p)$ *constitutes the behavior of* p *where*

- S_p *is a finite set of* (local) states,
- $\Delta_p \subseteq S_p \times Act_p^{\Lambda} \times S_p$ *is the finite* transition relation, *and*
- $F_p \subseteq S_p$ *is the set of* final states.

Moreover, $\mathcal{I} \subseteq \prod_{p\in Proc} S_p$ *is the set of global initial states.*

For an example CFM, consider Fig. 3(c) where $\{\text{L}, \text{R}\} \subset \Lambda$.

Let $\mathcal{A} = ((\mathcal{A}_p)_{p\in Proc}, \mathcal{I})$ with $\mathcal{A}_p = (S_p, \Delta_p, F_p)$ be a CFM. The size of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is defined to be $\sum_{p\in Proc} |S_p|$. A *configuration* of $\mathcal{A}$ gives a snapshot of the current state of each process and the current channel contents. Thus, the set of configurations of $\mathcal{A}$, denoted by $Conf_{\mathcal{A}}$, consists of pairs $(\overline{s}, \chi)$ with $\overline{s} \in \prod_{p\in Proc} S_p$ a global state and $\chi : Ch \to (Msg \times \Lambda)^*$, determining the channel contents. The projection of a global state $\overline{s} \in \prod_{p\in Proc} S_p$ to process p is denoted by $\overline{s}_p$. An execution of a send or receive action transfers the CFM from one configuration to another, according to the *global transition relation* of $\mathcal{A}$. This transition relation $\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Act \times Conf_{\mathcal{A}}$ is given by the following two inference rules. The first rule considers the sending of a message m from p to q and is given by

$$\frac{(\overline{s}_p, !(p, q, (m, \lambda)), \overline{s}'_p) \in \Delta_p \text{ and for all } r \neq p, \overline{s}_r = \overline{s}'_r}{((\overline{s}, \chi), !(p, q, m), (\overline{s}', \chi')) \in \Longrightarrow_{\mathcal{A}}}$$

where $\chi' = \chi[(p, q) := (m, \lambda) \cdot \chi((p, q))]$, i.e., χ' maps (p, q) to the concatenation of (m, λ) and $\chi((p, q))$; for all other channels, χ' coincides with χ. This rule expresses that if the local automaton $\mathcal{A}_p$ has a transition labeled by $!(p, q, (m, \lambda))$ moving from state $\overline{s}_p$ to $\overline{s}'_p$ then the CFM $\mathcal{A}$ has a transition from $\overline{s}$ to $\overline{s}'$ where only the p component of $\mathcal{A}$ changes its state and the new message (m, λ) is appended to the end of channel (p, q). Note that the control message has been abstracted away from the action that has been encountered when taking the transition. In other words, the transition is labeled by an element from Act, which follows our intuition that elements of Λ are only used for synchronization but do not contribute to observable behavior.

The second rule is complementary and considers the receipt

of a message:

$$\frac{(\overline{s}_p, ?(p,q,(m,\lambda)), \overline{s}'_p) \in \Delta_p \text{ and for all } r \neq p, \overline{s}_r = \overline{s}'_r}{((\overline{s},\chi), ?(p,q,m), (\overline{s}',\chi')) \in \Longrightarrow_{\mathcal{A}}}$$

where $\chi((q,p)) = w \cdot (m,\lambda) \neq \epsilon$ and $\chi' = \chi[(q,p) := w]$. This rule states that if the local automaton $\mathcal{A}_p$ has a transition labeled by $?(p,q,(m,\lambda))$ moving from state $\overline{s}_p$ to $\overline{s}'_p$ then the CFM $\mathcal{A}$ has a transition from $\overline{s}$ to $\overline{s}'$, which is labeled with $?(p,q,m)$, where only the p component of $\mathcal{A}$ changes its state and the message (m,λ) is removed from the head of channel (q,p).

A *run* of CFM $\mathcal{A}$ on a word $w = a_1 \dots a_n \in Act^*$ is a sequence $c_0 \dots c_n \in Conf^*_{\mathcal{A}}$ of configurations where c_0 is an initial configuration and, for every $i \in \{1,\dots,n\}$, $(c_{i-1}, a_i, c_i) \in \Longrightarrow_{\mathcal{A}}$. The set of *initial configurations* is defined as $\mathcal{I} \times \{\chi_\epsilon\}$ where χ_ϵ maps each channel onto the empty word, representing an empty channel. The run is *accepting* if $c_n \in (\prod_{p\in Proc} F_p) \times \{\chi_\epsilon\}$, i.e., each process is in an accepting state and all messages have been received yielding empty channels. The *language* of CFM $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words $w \in Act^*$ such that there is an accepting run of $\mathcal{A}$ on w.

Closure Properties of CFM Languages

We call $w = a_1 \dots a_n \in Act^*$ with $a_i \in Act$ *proper* if

- every receive action in w is preceded by a corresponding send action, i.e., for each channel $(p,q) \in Ch$, message $m \in Msg$, and prefix u of w, we have $\sum_{m\in Msg} |u|_{!(p,q,m)} \geq \sum_{m\in Msg} |u|_{?(q,p,m)}$ where $|u|_a$ denotes the number of occurrences of action a in the word u, and
- the FIFO policy is respected, i.e., for all $1 \leq i < j \leq n$, $(p,q) \in Ch$, and $m_1, m_2 \in Msg$ with $a_i = !(p,q,m_1)$, $a_j = ?(q,p,m_2)$, and $|\{i' \leq i \mid a_{i'} = !(p,q,m) \text{ for some } m \in Msg\}| = |\{j' \leq j \mid a_{j'} = ?(q,p,m) \text{ for some } m \in Msg\}|$, we have $m_1 = m_2$.

A proper word w is called *well-formed* if it satisfies $\sum_{m\in Msg} |w|_{!(p,q,m)} = \sum_{m\in Msg} |w|_{?(q,p,m)}$.

Obviously, a run of a CFM on a word w only exists if w is proper, as a receive action is only enabled if the corresponding send message is at the head of the channel. Moreover, every word accepted by a CFM is well-formed, as acceptance implies empty channels.

In addition, as different processes interact asynchronously and, in general, independently, the language of a CFM is closed under a certain permutation rewriting. For example, consider a run of a CFM on the well-formed word

$$!(p,q,m_1)\,!(p,q,m_2)\,?(q,p,m_1)\,?(q,p,m_2)\,!(r,q,m)\,?(q,r,m)$$

i.e., process p sends a message m_1 to process q, followed by a message m_2, whereupon process q receives these messages in the correct order. We observe that process q could have received the message m_1 before sending m_2. Indeed, any CFM accepting the above action sequence will also accept the word

$$!(p,q,m_1)\,?(q,p,m_1)\,!(p,q,m_2)\,?(q,p,m_2)\,!(r,q,m)\,?(q,r,m)$$

where any message is immediately received. Moreover, the action $!(r,q,m)$ is completely independent of all the actions that are engaged in sending/receiving the messages m_1 or m_2. Thus, a CFM cannot distinguish between the above sequences and sequence

$$!(r,q,m)\,!(p,q,m_1)\,?(q,p,m_1)\,!(p,q,m_2)\,?(q,p,m_2)\,?(q,r,m).$$

Actually, $!(r,q,m)$ can be placed at any arbitrary position with the restriction that it has to occur before the complementary receipt of m. Note that the three well-formed words mentioned above all correspond to linearizations of the MSC from Fig. 1(a).

To capture the closure properties of a CFM formally, we identify labeled posets whose linearizations satisfy the *all-or-none* law, stating that either every or no linearization is accepted by a CFM. To this aim, we associate to any word $w = a_1 \dots a_n \in Act^*$ an Act-labeled poset $\mathcal{M}(w) = (E, \preceq, \lambda)$ such that w is a linearization of $\mathcal{M}(w)$ and a CFM cannot distinguish between w and all other linearizations of $\mathcal{M}(w)$. The set of events is given by the set of positions in w, i.e., $E = \{1,\dots,n\}$. Naturally, any position $i \in E$ is labeled with a_i, i.e., $\lambda(i) = a_i$. It remains to fix the partial-order relation $\preceq$, which reflects the dependencies between events. Clearly, we consider those events to be dependent that are executed by the same process or constitute the send and receipt of a message, since each process acts sequentially and a message has to be sent before it can be received. Hence, let $\preceq_p \subseteq E_p \times E_p$ with E_p as before be defined by $i \preceq_p j$ iff $i \leq j$. Moreover, let $i \prec_{\text{msg}} j$ if there is a channel $(p,q) \in Ch$ and a message $m \in Msg$ such that

- $\lambda(i) = !(p,q,m)$, $\lambda(j) = ?(q,p,m)$, and
- $|\{i' \leq i \mid \lambda(i') = !(p,q,m') \text{ for some } m' \in Msg\}| = |\{j' \leq j \mid \lambda(j') = ?(q,p,m') \text{ for some } m' \in Msg\}|$.

This is similar to the definition of $\prec_{\text{msg}}$ in the previous paragraph. Let $\preceq = (\prec_{\text{msg}} \cup \bigcup_{p\in Proc} \preceq_p)^*$. To examplify these notions, consider the well-formed word w defined as $!(r,q,m)\,!(p,q,m_1)\,!(p,q,m_2)\,?(q,p,m_1)\,?(q,p,m_2)\,?(q,r,m)$. Fig. 1(b) depicts the Hasse diagram of the Act-labeled poset $\mathcal{M}(w) = (E, \preceq, \lambda)$. Note that $\mathcal{M}(w)$ is an MSC. Indeed, we have the following two lemmas, which are considered standard in the MSC literature (see, for example, [27]).

Lemma 1 *For any MSC $\mathcal{M}$, $w \in Lin(\mathcal{M})$ is well-formed.*

Lemma 2 *For any well-formed $w \in Act^*$, $\mathcal{M}(w)$ is an MSC. Moreover, $\mathcal{M}(w)$ and $\mathcal{M}(w')$ are isomorphic for all $w' \in Lin(\mathcal{M}(w))$.*

These results suggest to introduce an equivalence relation over well-formed words. The well-formed words w and w' are *equivalent*, written $w \approx w'$, if $\mathcal{M}(w)$ and $\mathcal{M}(w')$ are isomorphic. Note that this holds iff $w \in Lin(\mathcal{M}(w'))$.

Lemma 3 ([3]) *For any CFM $\mathcal{A}$,*

a) *$L(\mathcal{A})$ consists of well-formed words only.*
b) *$L(\mathcal{A})$ is closed under $\approx$.*

The last claim asserts that for all well-formed words u and v with $u \approx v$, we have $u \in L(\mathcal{A})$ iff $v \in L(\mathcal{A})$. For well-formed word w, let $[w]_\approx$ be the set of well-formed words that are equivalent to w wrt. $\approx$. For a set L of well-formed words, let $[L]_\approx := \bigcup_{w \in L}[w]_\approx$ be the *closure* of L wrt. $\approx$. The fact that $L(\mathcal{A})$ is closed under $\approx$, allows us to assign to $\mathcal{A}$ its set of MSCs $\mathcal{L}(\mathcal{A}) := \{\mathcal{M}(w) \mid w \in L(\mathcal{A})\}$. (Here, we identify isomorphic structures, i.e., we consider isomorphism classes of MSCs). This is an equivalent, visual, and more compact description of the behavior of CFM $\mathcal{A}$. Observe that $Lin(\mathcal{L}(\mathcal{A})) = L(\mathcal{A})$, i.e., the linearizations of the MSCs of CFM $\mathcal{A}$ correspond to its word language.

C. Deadlock-Free, Bounded, and Weak CFMs

In distributed computations, the notions of determinism, deadlock and bounded channels play an important role [6, 25, 26]. Roughly speaking, a CFM is deterministic if every possible execution allows for at most one run; it is deadlock-free if any run can be extended towards an accepting one.

Definition 3 (Deterministic CFM) *A CFM $\mathcal{A} = ((\mathcal{A}_p)_{p\in Proc}, \mathcal{I})$ with $\mathcal{A}_p = (S_p, \Delta_p, F_p)$ is* deterministic *if, for all $p \in Proc$, Δ_p satisfies the following two conditions: (i) If we have both $(s, !(p, q, (m, \lambda_1)), s_1) \in \Delta_p$ and $(s, !(p, q, (m, \lambda_2)), s_2) \in \Delta_p$, then $\lambda_1 = \lambda_2$ and $s_1 = s_2$. (ii) If we have both $(s, ?(p, q, (m, \lambda)), s_1) \in \Delta_p$ and $(s, ?(p, q, (m, \lambda)), s_2) \in \Delta_p$, then $s_1 = s_2$.*

The CFMs from Fig. 3(a) and (b) are deterministic whereas the CFM from Fig. 3(c) is not.

Definition 4 (Deadlock-free CFM ([26])) *A CFM $\mathcal{A}$ is* deadlock-free *if, for all $w \in Act^*$ and all runs γ of $\mathcal{A}$ on w, there exist $w' \in Act^*$ and $\gamma' \in Conf^*_{\mathcal{A}}$ such that $\gamma\gamma'$ is an accepting run of $\mathcal{A}$ on ww'.*

The CFMs from Fig. 3(a), (b) and (c) are deadlock-free. Note that, however, the CFM in Fig. 3(c) will contain a deadlock if the control messages L and R were omitted.

We obtain another essential restriction of CFMs if we require that any channel has a bounded capacity, say, $B \in \mathbb{N}$. Towards this notion, we first define when a word is B-bounded.

Definition 5 (B-bounded word) *Let $B \in \mathbb{N}$. Word $w \in Act^*$ is B-bounded if, for any prefix u of w and any $(p, q) \in Ch$, it holds*

$$0 \;\leq\; \sum_{m\in Msg} |u|_{!(p,q,m)} - \sum_{m\in Msg} |u|_{?(q,p,m)} \;\leq\; B.$$

This notion is extended to MSCs in the following way. MSC M is called *universally B-bounded* if all words in $Lin(M)$ are B-bounded. MSC M is *existentially B-bounded* if $Lin(M)$ contains at least one B-bounded word. Similar notions are adopted for CFMs, except that for existentially-boundedness, it is required that for every word u of the language, an *equivalent* word $v \approx u$ exists that is B-bounded. The intuition is that bounded channels suffice to accept representatives of the language provided the actions in an CFM are scheduled appropriately, cf. [24, 25]. Formally, the notion of bounded CFMs is defined as follows:

Definition 6 (Bounded CFM ([24, 27]))

a) *CFM $\mathcal{A}$ is* universally B-bounded, $B \in \mathbb{N}$, *if $L(\mathcal{A})$ is a set of B-bounded words. It is* universally bounded *if it is universally B-bounded for some B.*
b) *CFM $\mathcal{A}$ is* existentially B-bounded, $B \in \mathbb{N}$, *if, for every $w \in L(\mathcal{A})$, there is a B-bounded word $w' \in L(\mathcal{A})$ such that $w' \approx w$.*

A further variant of CFMs, as considered in [3, 32, 35], does not allow for sending control information with a message. Moreover, they have a single global initial state:

Definition 7 (Weak CFM) *A CFM $\mathcal{A} = ((\mathcal{A}_p)_{p\in Proc}, \mathcal{I})$ is called* weak *if*

- $|\mathcal{I}| = 1$ *and*
- *for every two transitions $(s_1, !(p, q, (m_1, \lambda_1)), s'_1)$ and $(s_2, !(p, q, (m_2, \lambda_2)), s'_2)$, we have $\lambda_1 = \lambda_2$.*

Note that the second item requires that only one message from Λ is used in the CFM. Intuitively, we could say that no synchronization message is used at all, as a weak CFM cannot distinguish between several messages.

Example 1 *Consider the weak CFMs $\mathcal{A}_\mathrm{a}$ and $\mathcal{A}_\mathrm{b}$ depicted in Fig. 3(a) and (b), respectively, which do not use control messages (recall that, formally, there is no distinction between control messages). The CFM $\mathcal{A}_\mathrm{a}$ represents a simple producer-consumer protocol, whereas $\mathcal{A}_\mathrm{b}$ specifies a part of the alternating-bit protocol. Two scenarios that demonstrate a possible behavior of these systems are given by the MSCs M_a and M_b from Fig. 2(a) and (b), respectively. Indeed, $M_\mathrm{a} \in \mathcal{L}(\mathcal{A}_\mathrm{a})$ and $M_\mathrm{b} \in \mathcal{L}(\mathcal{A}_\mathrm{b})$ (thus, $Lin(M_\mathrm{a}) \subseteq L(\mathcal{A}_\mathrm{a})$ and $Lin(M_\mathrm{b}) \subseteq L(\mathcal{A}_\mathrm{b})$). Observe that $\mathcal{A}_\mathrm{a}$ is deterministic, existentially 1-bounded, and deadlock-free. It is not universally bounded as process p can potentially send arbitrarily many messages to process q before any of these messages is received. In contrast, $\mathcal{A}_\mathrm{b}$ is universally bounded (witnessed by the bound $B = 3$) and also existentially 1-bounded. As stated before, it is also deterministic and deadlock-free.*

The CFM $\mathcal{A}_\mathrm{c}$ (cf. Fig. 3(c)), which is existentially 1-bounded, deadlock-free, and not deterministic, describes the system that is depicted informally in Fig. 3(d) in terms of a high-level *MSC: MSCs from $\mathcal{L}(\mathcal{A}_\mathrm{c})$ start with sending a request message from p to q, followed by an arbitrary sequence of further requests, which are sent from p to q, and acknowledgments, sent from q to p. Note that $\mathcal{A}_\mathrm{c}$ employs control messages to avoid deadlocks. The idea is that* L *and* R *inform the communication partner about which of the nodes at the bottom (left or right) is envisaged next.*

For weak CFMs $\mathcal{A}$, we can identify another closure property. Consider Fig. 4. If $L(\mathcal{A})$ subsumes the linearizations of the MSCs M_1 and M_2, then those of M_3 will be contained in $L(\mathcal{A})$ as well, as the bilateral interaction between the processes is completely independent. Formally, we define the inference

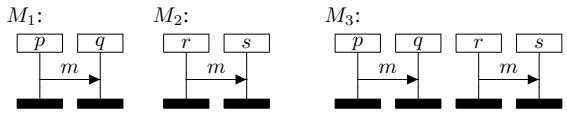


Fig. 4. Some MSCs

relation $\models \subseteq 2^{Act^*} \times Act^*$ as follows: given a set L of well-formed words and a well-formed word w, $L \models w$ if, for every $p \in Proc$, there is $u \in L$ such that $u \upharpoonright p = w \upharpoonright p$. Here, $(w \upharpoonright p) \in Act_p^*$ denotes the projection of w onto actions of process p. Indeed, $L(\mathcal{A})$ is closed under $\models$, i.e., $L(\mathcal{A}) \models w$ implies $w \in L(\mathcal{A})$.

A stronger notion, which is satisfied by any weak deadlock-free CFM is as follows. Let $L \subseteq Act^*$ be a set of well-formed words and let u be a proper word (i.e., it is the prefix of some well-formed word). We write $L \models^{df} u$ if, for every $p \in Proc$, there is $w \in L$ such that $u \upharpoonright p$ is a prefix of $w \upharpoonright p$. Language $L \subseteq Act^*$ is closed under $\models^{df}$ if $L \models^{df} w$ implies that w is a prefix of some word in L.

Lemma 4 ([3, 32]) *Let $\mathcal{A}$ be a weak CFM.*

a) *$L(\mathcal{A})$ is closed under $\models$.*
b) *If $\mathcal{A}$ is deadlock-free, then $L(\mathcal{A})$ is closed under $\models^{df}$.*

Implementability Issues

Next, we collect known results on the relationship between regular languages over Act and CFM languages.

Theorem 1 *Let $L \subseteq Act^*$ be a set of well-formed words that is closed under $\approx$, and let $B \in \mathbb{N}$. We have the following equivalences:*

a) 1) *L is regular.*
   2) *There is a universally bounded CFM $\mathcal{A}$ with $L = L(\mathcal{A})$.*
   3) *There is a deterministic universally bounded CFM $\mathcal{A}$ with $L = L(\mathcal{A})$.*
   4) *There is a universally bounded deadlock-free CFM $\mathcal{A}$ with $L = L(\mathcal{A})$.*
b) 1) *The set $\{w \in L \mid w \text{ is } B\text{-bounded}\}$ is regular, and for all $w \in L$, there is a B-bounded word w' with $w \approx w'$.*
   2) *There is an existentially B-bounded CFM $\mathcal{A}$ with $L = L(\mathcal{A})$.*
c) 1) *L is regular and closed under $\models$.*
   2) *There is a universally bounded weak CFM $\mathcal{A}$ with $L = L(\mathcal{A})$.*
d) 1) *L is regular, closed under $\models$, and closed under $\models^{df}$.*
   2) *There is a deterministic universally bounded deadlock-free weak CFM $\mathcal{A}$ with $L = L(\mathcal{A})$.*

In all four cases, all directions are effective where L is assumed to be given as a finite automaton.

The equivalences "1) $\Leftrightarrow$ 2) $\Leftrightarrow$ 3)" in Theorem 1a) go back to [27], the equivalence "1) $\Leftrightarrow$ 4)" to [7]. Theorem 1b) is due to [24]. Finally, Theorems 1c) and 1d) can be attributed to [3, 32].

TABLE I
ANGLUIN'S ALGORITHM L*

```
L*(Σ):
 1  U := {ε}; V := {ε}; T is defined nowhere;
 2  T-UPDATE();
 3  repeat
 4        while (T,U,V) is not (closed and consistent)
 5           do
 6              if (T,U,V) is not consistent then
 7                 find u,u' ∈ U, a ∈ Σ, v ∈ V s.t.:
 8                      row(u) = row(u') and
 9                      row(ua)(v) ≠ row(u'a)(v);
10                 V := V ∪ {av};
11                 T-UPDATE();
12              if (T,U,V) is not closed then
13                 find u ∈ U, a ∈ Σ such that row(ua) ≠ row(u')
14                  for all u' ∈ U;
15                 U := U ∪ {ua};
16                 T-UPDATE();
17        /* (T,U,V) is both closed and consistent, hence, H_(T,U,V)
18        can be derived */
19        perform equivalence test for H_(T,U,V);
20        if equivalence test fails then
21           get counterexample w;
22           U := U ∪ pref(w);
23           T-UPDATE();
24    until equivalence test succeeds;
25    return H_(T,U,V);
```

TABLE II
FUNCTION FOR UPDATING TABLE FUNCTION IN L*

```
T-UPDATE():
1  for w ∈ (U ∪ UΣ)V such that T(w) is not defined
2     T(w) := getClassificationFromTeacher(w);
```

III. LEARNING REGULAR LANGUAGES

In the previous sections, we formalized the notions of scenarios, or MSCs, and design models, i.e., CFMs. It remains to introduce the key concept of our synthesis approach: *learning* design models from given scenarios.

Angluin's well-known algorithm L* [4] learns a deterministic finite automaton (DFA) by querying for certain words whether they should be accepted or rejected by the automaton in question. In this section, we recall the algorithm and generalize 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 CFMs, as described in the previous section.

Let us first recall some basic definitions. Let Σ be an alphabet. A deterministic finite automaton (DFA) over Σ is a tuple $\mathcal{B} = (Q, q_0, \delta, F)$, 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 $L(\mathcal{B})$ of $\mathcal{B}$ is defined as $L(\mathcal{B}) = \{w \in \Sigma^* \mid \bar{\delta}(q_0, w) \in F\}$ where $\bar{\delta} : Q \times \Sigma^* \to Q$ is the extension of δ to words, i.e., $\bar{\delta}(q, \epsilon) = w$ and $\bar{\delta}(q, aw) = \bar{\delta}(\delta(q, a), w)$. Due to well-known automata-theoretic results, every DFA can be transformed into a unique (up to isomorphism), equivalent, minimal DFA, i.e., having a minimal number of states.

Angluin's algorithm L* learns or infers a minimal DFA

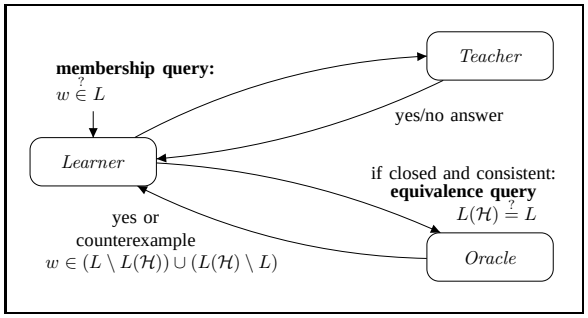


Fig. 5. Components of L* and their interaction

for a given regular language L. In the algorithm, a so-called $Learner$, who initially knows nothing about L, is trying to learn a DFA $\mathcal{B}$ such that $L(\mathcal{B}) = L$. To this end, it asks repeatedly queries of a $Teacher$ and an $Oracle$[1], who both know L. There are two kinds of queries (cf. Fig. 5):

- A *membership query* consists in asking the $Teacher$ if a word $w \in \Sigma^*$ is in L.
- An *equivalence query* consists in asking the $Oracle$ whether a *hypothesized* DFA $\mathcal{H}$ is correct, i.e., whether $L(\mathcal{H}) = L$. The $Oracle$ answers *yes* if $\mathcal{H}$ is correct, or supplies a counterexample w, drawn from the symmetric difference of L and $L(\mathcal{H})$.

The $Learner$ maintains a prefix-closed set $U \subseteq \Sigma^*$ of words that are candidates for identifying states, and a suffix-closed set $V \subseteq \Sigma^*$ of words that are used to distinguish such states. The sets U and V are increased on demand. The $Learner$ makes membership queries for all words in $(U \cup U\Sigma)V$, and organizes the results into a *table* $\mathcal{T} = (T, U, V)$ where function T maps each $w \in (U \cup U\Sigma)V$ to an element from $\{+, -\}$ where parity $+$ represents *accepted* and $-$ *not accepted*. For word $u \in U \cup U\Sigma$, let function $row(u) : V \to \{+, -\}$ be given by $row(u)(v) = T(uv)$. Such function is called a *row* of $\mathcal{T}$.

The following properties of a table are relevant. Table $\mathcal{T}$ is

- *closed*, if for all $u \in U$ and $a \in \Sigma$ there is $u' \in U$ such that $row(ua) = row(u')$, and
- *consistent*, if for all $u, u' \in U$ and $a \in \Sigma$, $row(u) = row(u')$ implies $row(ua) = row(u'a)$.

If $\mathcal{T}$ is not closed, there exists $a \in \Sigma$ such that $row(ua) \neq row(u')$ for all $u' \in U$. In this case, we move ua to U and ask membership queries for every $uabv$ with $b \in \Sigma$ and $v \in V$. Likewise, if $\mathcal{T}$ is not consistent, there exist $u, u' \in U$, $a \in \Sigma$ and $v \in V$ such that $row(u) = row(u')$ and $row(ua)(v) \neq row(u'a)(v)$. Then we add av to V and ask membership queries for every uav with $u \in U \cup U\Sigma$. If table $\mathcal{T}$ is closed and consistent, the $Learner$ constructs a hypothesized DFA $\mathcal{H}_\mathcal{T} = (Q, q_0, \delta, F)$, where

- $Q = \{row(u) \mid u \in U\}$ with q_0 the row $row(\epsilon)$,
- δ is defined by $\delta(row(u), a) = row(ua)$, and
- $F = \{r \in Q \mid r(\epsilon) = +\}$.

The $Learner$ subsequently submits $\mathcal{H}_\mathcal{T}$ as an equivalence query asking whether $L(\mathcal{H}_\mathcal{T}) = L$. If the answer is affirmative, the learning procedure is completed. Otherwise, the returned counterexample u is processed by adding every prefix of u (including u) to U, extending $U\Sigma$ accordingly, and subsequent membership queries are performed in order to make the table closed and consistent, whereupon a new hypothesized DFA is constructed, etc. (cf. Fig. 5).

The pseudo code of L* is given in Table I, supplemented by Table II, which contains the table-update function which is invoked whenever the $Teacher$ is supposed to classify a word.

Theorem 2 *([4]) Under the assumption that $Teacher$ classifies/provides words in conformance with a regular language L over Σ, invoking $\mathrm{L}^*(\Sigma)$ eventually returns the minimal DFA over Σ recognizing L. If n is the number of states of this DFA and m is the size of the largest counterexample, then the number of membership queries is in $\mathcal{O}(|\Sigma| \cdot m \cdot n^2)$ and the maximal number of equivalence queries is n. The overall running time is polynomial in m and n.*

Example 2 *Assume $\Sigma = \{a, b\}$ and let $L = \{w \in \Sigma^* \mid |w|_a = |w|_b$ and $w = uv$ implies $|u|_b \leq |u|_a \leq |u|_b + 2\}$, i.e., for any word of L, every prefix has at least as many a's as b's and at most two more a's than b's. Moreover, the number of a's in the whole word is equal to the number of b's. Clearly, L is a regular language over Σ. Let us illustrate how the minimal DFA for L is learned using L^*. Fig. 6 shows several tables that are computed while learning L. The first table is initialized for $U = \{\epsilon\}$ and $V = \{\epsilon\}$. A table entry $T(uv)$ with $u \in U \cup U\Sigma$ and $v \in V$ has parity $+$ if $uv \in L$ and $-$, otherwise. For example, consider Fig. 6i). According to the definition of L, the empty word ϵ is contained in L and, thus, $T(\epsilon) = row(\epsilon)(\epsilon) = +$. In contrast, a and b are not in L, so that $T(a) = T(b) = row(a)(\epsilon) = row(b)(\epsilon) = -$.*

This table is not closed as, e.g., $row(a) \neq row(u)$ for all $u \in U$. Hence, U has to be extended by adding a, which invokes additional membership queries. The resulting table, cf. Fig. 6ii) is closed and consistent and the learner presents the hypothesis automaton $\mathcal{H}_1$, which, however, does not conform to the target language L, as, e.g., $bb \in L(\mathcal{H}_1) \setminus L$. Therefore, bb and its prefixes are added to U.

The obtained table (Fig. 6iii)) is not consistent, as $row(a)(\epsilon) = row(b)(\epsilon) = -$ but $row(aa')(\epsilon) \neq row(a'a')(\epsilon)$. To resolve this conflict a column is added to the table, i.e., $V' := V \cup \{a'\}$ where a' was the conflicting suffix.

Some steps later, the algorithm comes up with $\mathcal{H}_3$ (cf. Fig. 6vi)), which indeed recognizes L, i.e., $L(\mathcal{H}_3) = L$, so that the learning procedure finally halts.

IV. Learning Communicating Finite-State Machines

In this section, we intend to give a learning approach to infer CFMs from example scenarios that are provided as MSCs. Let us first settle on a user profile, i.e., on some reasonable assumptions about the teacher/oracle that an inference algorithm should respect:

- The user can fix some system characteristics. For example, she might require her system to be deadlock-free, deterministic, or universally bounded.

[1]We want to explicitly distinguish between membership (i.e., easy to answer) and equivalence (i.e., more difficult to answer) queries. Conceptually, there is no reason for differentiating between them.

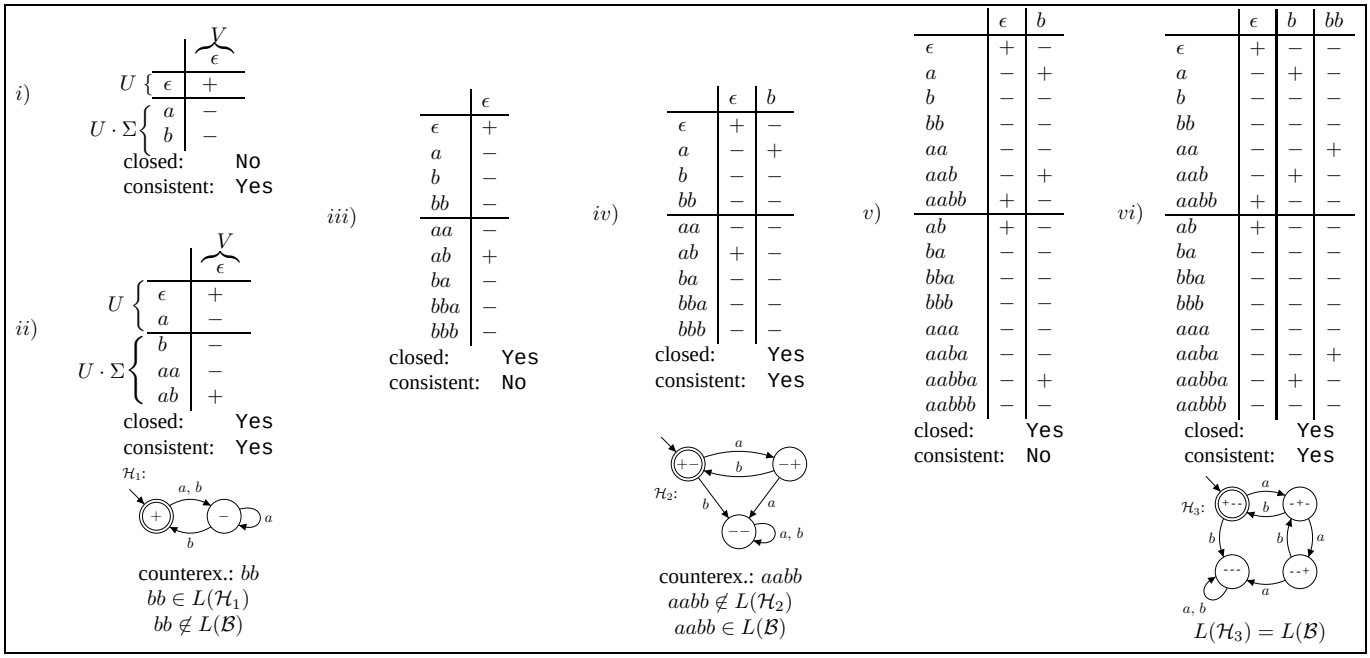


Fig. 6. An example of an L* run

- She can decide if a given scenario in terms of an MSC is desired or unwanted, thus classify it as positive or negative, respectively.
- She can accept or reject a given system and, in the latter case, come up with an MSC counterexample.

Roughly speaking, the user activity should restrict to classifying and providing MSCs. In contrast, we do not assume that the user can determine if a given system corresponds to the desired system characteristics. Apart from the fact that this would be too time consuming as a manual process, the user often lacks the necessary expertise. Moreover, the whole learning process would get stuck if the user was confronted with a hypothesis that does not match her requirements, but cannot come up with an MSC that is causal for this violation (this is particularly difficult if the system is required to be deadlock-free). So we would like to come up with some *guided* approach that "converges" against a system satisfying the requirements.

The core ingredient of an inference algorithm that matches our user profile shall be the algorithm L*, which synthesizes a minimal DFA from examples given as words. To build a bridge from regular word languages to CFMs, we make use of Theorem 1 (page 7), which reveals strong relationships between CFMs and regular word languages over the set Act of actions. More specifically, it asserts that one can synthesize

- a *deterministic universally-bounded CFM* from a regular set of well-formed words that is closed under $\approx$,
- a *universally-bounded deadlock-free CFM* from a regular set of well-formed words that is closed under $\approx$,
- an *existentially B-bounded CFM* from a regular set of well-formed B-bounded words that is closed under the restriction of $\approx$ to B-bounded words, and
- a *universally-bounded deadlock-free weak CFM* from a regular set of well-formed words that is closed under $\approx$, $\models$, and $\models^{df}$.

Towards learning CFMs, a naïve idea would now be to infer, by means of L*, a regular word language that can be translated into a CFM according to Theorem 1. The user then has to classify arbitrary words over the alphabet of actions and to deal with hypotheses that have nothing in common with MSC languages. These activities, however, do not match our user profile. Moreover, the user will be confronted with an overwhelming number of membership and equivalence queries that could actually be answered automatically. In fact, words that do not match an execution of a CFM and hypotheses that do not correspond to a CFM could be systematically rejected, without bothering the user. The main principle of our solution will, therefore, be an interface between the user and the program (i.e., the learner) that is based on MSCs only. In other words, the only objects that the user gets to see are MSCs that need to be classified, and CFMs that might already correspond to a desired design model. On the one hand, this facilitates the user activities. On the other hand, we obtain a substantial reduction of membership and equivalence queries. The latter will be underpinned, in Section VI, by a practical evaluation (cf. Table V).

Now let us turn to our adapted inference algorithm. Its core will indeed be L*. While L* does not differentiate between words over a given alphabet, however, Theorem 1 indicates that we need to consider a suitable domain $\mathcal{D} \subseteq Act^*$ containing only well-formed words. Secondly, certain restrictions have to be imposed such that any synthesized CFM recognizes a regular subset of $\mathcal{D}$. For universally-bounded (deadlock-free) CFMs, this might be the class of all well-formed words, whereas for existentially B-bounded CFMs only regular languages of B-bounded words are suitable. In other words, we have to ensure that regular word languages are learned that contain words from $\mathcal{D}$ only. As for any CFM $\mathcal{A}$,

$L(\mathcal{A})$ is closed under $\approx$, the regular subsets of $\mathcal{D}$ in addition have to be closed under $\approx$; more precisely, the restriction of $\approx$ to words from $\mathcal{D}$. Similarly, to infer a weak or deadlock-free CFM, we need a regular word language that is closed under $\models$. In our learning setup, this will be captured by a relation $\vdash \subseteq 2^{\mathcal{D}} \times 2^{\mathcal{D}}$ where $L_1 \vdash L_2$ intuitively means that L_1 requires at least one word from L_2. It is not difficult to see that this relation suffices to cover the inference relation $\models$, and as will be shown later, it can be used to capture $\models^{df}$ as well.

Let $\mathfrak{R}_{\text{minDFA}}(\mathcal{D}, \vdash)$ be the class of minimal DFA that recognize a language $L \subseteq \mathcal{D}$ satisfying

- L is closed under $\approx_{\mathcal{D}} := \approx \cap (\mathcal{D} \times \mathcal{D})$, and
- L is closed under $\vdash$, i.e., $(L_1 \vdash L_2 \wedge L_1 \subseteq L)$ implies $L \cap L_2 \neq \emptyset$.

A learning algorithm tailored to CFMs is now based on the notion of a *learning setup* for a class of CFMs, which provides instantiations of $\mathcal{D}$ and $\vdash$.

Definition 8 (Learning Setup) *Let $\mathfrak{C}$ be a class of CFMs. A* learning setup *for $\mathfrak{C}$ is a triple $(\mathcal{D}, \vdash, synth)$ where*

- *$\mathcal{D} \subseteq Act^*$, the* domain, *is a set of well-formed words,*
- *$\vdash \subseteq 2^{\mathcal{D}} \times 2^{\mathcal{D}}$ such that $L_1 \vdash L_2$ implies (L_1 is finite, $L_2 \neq \emptyset$, and L_2 is decidable),*
- *$synth : \mathfrak{R}_{\text{minDFA}}(\mathcal{D}, \vdash) \to \mathfrak{C}$ is the computable* synthesis function *such that, for each CFM $\mathcal{A} \in \mathfrak{C}$, there is $\mathcal{B} \in \mathfrak{R}_{\text{minDFA}}(\mathcal{D}, \vdash)$ with $[L(\mathcal{B})]_{\approx} = L(synth(\mathcal{B})) = L(\mathcal{A})$ (in particular, $synth$ is injective).*

The final constraint asserts that for any CFM $\mathcal{A}$ in the considered class of CFMs, a minimal DFA $\mathcal{B}$ exists (in the corresponding class of DFAs) recognizing the same word language as $\mathcal{A}$ modulo $\approx$.

Given the kind of learning setup that we will consider, we now discuss some necessary changes to the algorithm L*. As L* works within the class of arbitrary DFA over Act, conjectures may be proposed whose languages are not subsets of $\mathcal{D}$, or violate the closure properties for $\approx$ and $\vdash$ (or both). To avoid the generation of such incorrect hypothesized automata, the language inclusion problem (is the language of a given DFA included in $\mathcal{D}$?) and the closure properties in question are required to be *constructively decidable*. This means that each of these problems is decidable and that in case of a negative result, a *reason* of its failure, i.e., a counterexample, can be computed. Accordingly, we require that the following properties hold for DFA $\mathcal{B}$ over Act:

(D1) The problem whether $L(\mathcal{B}) \subseteq \mathcal{D}$ is decidable and if $L(\mathcal{B}) \not\subseteq \mathcal{D}$, one can compute some $w \in L(\mathcal{B}) \setminus \mathcal{D}$. We then say that INCLUSION$(\mathcal{D})$ is *constructively decidable*.

(D2) If $L(\mathcal{B}) \subseteq \mathcal{D}$, it is decidable whether $L(\mathcal{B})$ is closed under $\approx_{\mathcal{D}}$. If not, one can compute $w, w' \in \mathcal{D}$ such that $w \approx_{\mathcal{D}} w'$, $w \in L(\mathcal{B})$, and $w' \notin L(\mathcal{B})$. We then say that the problem EQCLOSURE$(\mathcal{D})$ is *constructively decidable*.

(D3) If $L(\mathcal{B}) \subseteq \mathcal{D}$ is closed under $\approx_{\mathcal{D}}$, it is decidable whether $L(\mathcal{B})$ is closed under $\vdash$. If not, one 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 $L_1 \subseteq L(\mathcal{B})$ and $L(\mathcal{B}) \cap L_2 = \emptyset$. We then say that INFCLOSURE$(\mathcal{D}, \vdash)$ is *constructively decidable*.

Let us now generalize Angluin's algorithm to cope with the extended setting, and let $(\mathcal{D}, \vdash, synth)$ be a learning setup for some class $\mathfrak{C}$ of CFMs. The main changes in Angluin's algorithm concern the processing of membership queries as well as the treatment of hypotheses. For the following description, we refer to Table III, depicting the pseudo code of EXTENDED-L*, our extension of L*, and Table IV which contains a modified table-update function that is invoked by this extension of L*.

The *Teacher* will provide/classify MSCs rather than words. Moreover, the equivalence test will be performed, by the *Oracle*, on the basis of a CFM rather than on the basis of a DFA. The *Oracle* will also provide counterexamples in terms of MSCs.

To undertake an equivalence test, knowledge of the target model is required as in every other learning based technique for inferring design models. Simulating and testing are possibilities to converge to a correct system implementation. In the implementation of our approach [9] we provide such means to ease the user's burden.

To realize these changes, we exploit a new table-update function EXTENDED-T-UPDATE (cf. Table IV). Therein, membership queries are filtered: a query $w \notin \mathcal{D}$ is considered immediately as negative, without presenting it to the *Teacher* (lines 2,3). Faced with a query $w \in \mathcal{D}$, the MSC $\mathcal{M}(w)$ is displayed to the *Teacher* (we call this a *user query*). His verdict will then determine the table entry for w (line 9). Once a user query has been processed for a word $w \in \mathcal{D}$, queries $w' \in [w]_{\approx_{\mathcal{D}}}$ must be answered equivalently. They are thus not forwarded to the *Teacher* (lines 6, 7). Therefore, MSCs that have already been classified are memorized in a set *Pool* (line 10).

Once table $\mathcal{T}$ is closed and consistent, a hypothesized DFA $\mathcal{H}_{\mathcal{T}}$ is determined as usual. We then proceed as follows (cf. Table III):

1) If $L(\mathcal{H}_{\mathcal{T}}) \not\subseteq \mathcal{D}$, compute a word $w \in L(\mathcal{H}_{\mathcal{T}}) \setminus \mathcal{D}$ and modify the table $\mathcal{T}$ accordingly by invoking EXTENDED-T-UPDATE (lines 18–22).
2) If $L(\mathcal{H}_{\mathcal{T}}) \subseteq \mathcal{D}$ but $L(\mathcal{H}_{\mathcal{T}})$ is not closed under $\approx_{\mathcal{D}}$, compute $w, w' \in \mathcal{D}$ such that $w \approx_{\mathcal{D}} w'$, $w \in L(\mathcal{H}_{\mathcal{T}})$, and $w' \notin L(\mathcal{H}_{\mathcal{T}})$; perform the membership queries for $[w]_{\approx}$. As these queries are asked in terms of an MSC by displaying $\mathcal{M}(w)$ to the *Teacher*, it is guaranteed that they are answered uniformly (lines 24–28).
3) If $L(\mathcal{H}_{\mathcal{T}})$ is the union of $\approx_{\mathcal{D}}$-equivalence classes but not closed under $\vdash$, compute $(L_1, L_2) \in \vdash$ such that $L_1 \subseteq L(\mathcal{H}_{\mathcal{T}})$ and $L(\mathcal{H}_{\mathcal{T}}) \cap L_2 = \emptyset$; perform membership queries for every word from L_1 (displaying the corresponding MSCs to the *Teacher*); if all these membership queries are answered positively, the *Teacher* is asked to specify an MSC that comes with a linearization w from L_2. The word w will be declared "positive". Recall that L_2 is a decidable language (and we assume that the decision algorithm is available) so that all MSCs $\mathcal{M}$ with $Lin(\mathcal{M}) \cap L_2 \neq \emptyset$ can be enumerated until a suitable MSC is selected (lines 30–41).

If, for a hypothesized DFA $\mathcal{H}_{\mathcal{T}}$, we have $L(\mathcal{H}_{\mathcal{T}}) \subseteq \mathcal{D}$, and $L(\mathcal{H}_{\mathcal{T}})$ is closed under both $\approx_{\mathcal{D}}$ and $\vdash$, then an equivalence

query is performed on $synth(\mathcal{H}_{\mathcal{T}})$, the CFM that is synthesized from the hypothesized DFA. In case a counterexample MSC $\mathcal{M}$ is provided, the table has to be complemented accordingly by a linearization of $\mathcal{M}$ (lines 43–50). Otherwise, $synth(\mathcal{H}_{\mathcal{T}})$ is returned as the desired CFM (lines 51, 52).

Theorem 3 *Let $\mathfrak{C}$ be a class of CFMs, let $(\mathcal{D}, \vdash, synth)$ be a learning setup for $\mathfrak{C}$, and let $\mathcal{A} \in \mathfrak{C}$. If the $Teacher$ classifies/provides MSCs in conformance with $\mathcal{L}(\mathcal{A})$, then invoking* EXTENDED-L$^*(\mathcal{D}, \vdash, synth)$ *eventually returns a CFM $\mathcal{A}' \in \mathfrak{C}$ such that $L(\mathcal{A}') = L(\mathcal{A})$.*

Proof: We fix a class $\mathfrak{C}$ of CFMs and a learning setup $(\mathcal{D}, \vdash, synth)$ for $\mathfrak{C}$. Moreover, let $\mathcal{A} \in \mathfrak{C}$. By the definition of a learning setup, there exists a DFA $\mathcal{B} \in \mathfrak{R}_{\text{minDFA}}(\mathcal{D}, \vdash)$ with $[L(\mathcal{B})]_{\approx} = L(synth(\mathcal{B})) = L(\mathcal{A})$. We suppose that $Teacher$ classifies/provides MSCs in accordance with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(synth(\mathcal{B}))$. On invoking EXTENDED-L$^*(\mathcal{D}, \vdash, synth)$, a word $w \in (U \cup UAct)V$ is classified by the table function T depending on whether $w \in \mathcal{D}$ and $\mathcal{M}(w) \in \mathcal{L}(\mathcal{A}) = \mathcal{L}(synth(\mathcal{B}))$. More precisely, $T(w) = +$ iff $w \in L(\mathcal{B})$, i.e., we actually perform L*, and the $Teacher$ acts in conformance with $L(\mathcal{B})$. The differences to the basic version of Angluin's algorithm are that (i) not every hypothesis $\mathcal{H}_{(T,U,V)}$ is forwarded to the $Teacher$ (in that case, counterexamples can be generated automatically), and (ii) we may add, in lines 27 and 33, several words (and its prefixes) to the table at one go. This is, however, a modification that preserves the validity of Theorem 2. Consequently, when the equivalence test succeeds (line 51), then the algorithm outputs a CFM $\mathcal{A}' = synth(\mathcal{H}_{(T,U,V)})$ with $L(\mathcal{A}') = L(\mathcal{A})$. ∎

Definition 9 (Learnability of classes of CFMs) *Class $\mathfrak{C}$ of CFMs is* learnable *if there is a learning setup for $\mathfrak{C}$.*

The sequel of this section is devoted to identify learnable classes of CFMs. To this purpose, we have to determine a learning setup for each class.

A note on the complexity. The total running time of the extended algorithm can only be considered wrt. a concrete learning setup. In particular, it heavily depends on the complexity of the synthesis of a CFM from a given minimal DFA, which tends to be very high. When studying this issue below for several learning setups, we will therefore assume that an equivalence check is performed on the basis of the minimal DFA itself rather than on a synthesized CFM (cf. line 43 in Table III). This lowers the running time of the algorithm considerably and, at the same time, is a reasonable assumption, as in all learning setups we provide below, the minimal DFA faithfully simulates all executions of the synthesized CFM (up to a channel bound when considering the case of existential bounds). So let us in the following assume the synthesis function to need constant time.

We can now state our first learnability result:

Theorem 4 *Universally-bounded CFMs are learnable.*

Proof: Let $\mathfrak{C}$ denote the class of deterministic universally-bounded CFMs. To show that $\mathfrak{C}$ is learnable, we need to determine a learning setup $(\mathcal{D}, \vdash, synth)$ for $\mathfrak{C}$. First observe that $\models$ needs not be instantiated for this class (cf. Theorem 1a). Let $\mathcal{D}$ be the set of well-formed words over Act. By Theorem 1a), there is a computable mapping $synth$ that transforms any regular set L of well-formed words that is closed under $\approx_{\mathcal{D}} = \approx$ (say, given in terms of a finite automaton) into a CFM $\mathcal{A}$ such that $L(\mathcal{A}) = L$. To show that $(\mathcal{D}, \emptyset, synth)$ is indeed a learning setup, it remains to establish that the problems INCLUSION$(\mathcal{D})$, EQCLOSURE$(\mathcal{D})$, and INFCLOSURE$(\mathcal{D}, \emptyset)$ are constructively decidable. Decidability of INCLUSION$(\mathcal{D})$ and EQCLOSURE$(\mathcal{D})$ has been shown in [27]. For DFA, these problems are actually solvable in linear time. The decidability of INFCLOSURE$(\mathcal{D}, \emptyset)$ is trivial. ∎

Together with Theorem 1a), we immediately obtain the learnability of two subclasses:

Corollary 1 *Deterministic universally-bounded CFMs and universally-bounded deadlock-free CFMs are learnable.*

Now let us have a closer look at the complexity of our algorithm, when it is instantiated with the learning setup that we developed in the proof of Theorem 4. In the best case, we start with a deterministic CFM. In the following, let m denote the maximal number of events of an MSC that is either provided or to be classified by the user ($Teacher$ or $Oracle$).

Theorem 5 *Let $\mathfrak{C}$ be the class of deterministic universally-bounded CFMs and let $\mathcal{A} \in \mathfrak{C}$ be universally B-bounded. The number of equivalence queries needed to infer a CFM $\mathcal{A}' \in \mathfrak{C}$ with $L(\mathcal{A}) = L(\mathcal{A}')$ is at most $(|\mathcal{A}| \cdot |Msg| + 1)^{B \cdot |Proc|^2 + |Proc|}$. Moreover, the number of membership queries and the overall running time is polynomial in $|\mathcal{A}|$, $|Msg|$, and m, and it is exponential in $|Proc|$ and B.*

Proof: Suppose $\mathcal{A} \in \mathfrak{C}$ is the input CFM. Without loss of generality, we assume that the synchronization messages from Λ that are used in $\mathcal{A}$ are precisely the local states of $\mathcal{A}$. Then, the number of states of the unique minimal DFA $\mathcal{B}$ satisfying $L(synth(\mathcal{B})) = L(\mathcal{A})$ is bounded by $C = |\mathcal{A}|^{|Proc|} \cdot (|Msg| \cdot |\mathcal{A}| + 1)^{B \cdot |Proc|^2}$. The first factor is the number of global states of $\mathcal{A}$, whereas the second factor contributes the number of possible channel contents ($|Msg| \cdot |\mathcal{A}|$ being the number of messages). Hence, C constitutes an upper bound for the number of equivalence queries. We will now calculate the number of membership queries, which is bounded by the size of the table that we obtain when the algorithm terminates. Note first that the size of Act is bounded by $2|Proc|^2 \cdot |Msg|$. During a run of the algorithm, the size of V is bounded by C, as the execution of program line 9 always comes with creating a new state. The set U can increase at most C-times, too. The number of words that are added to U in line 21 can be bounded by $2C$. The length of words w and w', as added in line 27 can likewise be bounded by $2C$. The number of words added in line 47 depends on the size of a counterexample that is provided by the $Oracle$. Note that lines 33 and 38 are of no importance here because, as mentioned before, $\vdash$ was not

TABLE III
THE EXTENSION OF ANGLUIN'S ALGORITHM

```
EXTENDED-L*(𝒟, ⊢, synth):
 1  U := {ε}; V := {ε}; T is defined nowhere;
 2  Pool := ∅;
 3  EXTENDED-T-UPDATE();
 4  repeat
 5        while (T, U, V) is not (closed and consistent)
 6           do
 7              if (T, U, V) is not consistent then
 8                 find u, u' ∈ U, a ∈ Act, and v ∈ V such that row(u) = row(u') and row(ua)(v) ≠ row(u'a)(v);
 9                 V := V ∪ {av};
10                 EXTENDED-T-UPDATE();
11              if (T, U, V) is not closed then
12                 find u ∈ U and a ∈ Act such that row(ua) ≠ row(u') for all u' ∈ U;
13                 U := U ∪ {ua};
14                 EXTENDED-T-UPDATE();
15        /* (T, U, V) is both closed and consistent */
16        ℋ := ℋ_(T,U,V);
17        /* check closedness properties for ≈_𝒟 and ⊢ */
18        if L(ℋ) ⊈ 𝒟
19          then
20                compute w ∈ L(ℋ) \ 𝒟;
21                U := U ∪ pref(w);
22                EXTENDED-T-UPDATE();
23          else
24                if L(ℋ) is not ≈_𝒟-closed
25                  then
26                     compute w, w' ∈ 𝒟 such that w ≈_𝒟 w', w ∈ L(ℋ), and w' ∉ L(ℋ);
27                     U := U ∪ pref(w) ∪ pref(w');
28                     EXTENDED-T-UPDATE();
29                  else
30                     if L(ℋ) is not ⊢-closed
31                       then
32                          compute (L1, L2) ∈ ⊢ such that L1 ⊆ L(ℋ) and L(ℋ) ∩ L2 = ∅;
33                          U := U ∪ pref(L1);
34                          EXTENDED-T-UPDATE();
35                          if T(w) = + for all w ∈ L1 then
36                             ℳ := getMSCFromTeacher(L2);
37                             choose w ∈ Lin(ℳ) ∩ L2;
38                             U := U ∪ pref(w);
39                             T(w) := +;
40                             Pool := Pool ∪ {ℳ};
41                             EXTENDED-T-UPDATE();
42                       else
43                          do equivalence test for synth(ℋ_(T,U,V));
44                          if equivalence test fails then
45                             counterexample ℳ is provided, classified as parity ∈ {+, −};
46                             choose w ∈ Lin(ℳ) ∩ 𝒟;
47                             U := U ∪ pref(w);
48                             T(w) := parity;
49                             Pool := Pool ∪ {ℳ};
50                             EXTENDED-T-UPDATE();
51  until equivalence test succeeds;
52  return synth(ℋ);
```

instantiated for this learning setup. Summarizing, the number of membership queries is in $\mathcal{O}((C^3 + mC^2) \cdot |Act|)$. As for a given minimal DFA $\mathcal{H}$, one can detect in polynomial time if $L(\mathcal{H}) \subseteq \mathcal{D}$ and if $L(\mathcal{H})$ is $\approx_{\mathcal{D}}$-closed, the overall running time of the algorithm is polynomial in $|Msg|$, m, and $|\mathcal{A}|$, and it is exponential in $|Proc|$ and B. ■

The following theorem states that the complexity is higher when we act on the assumption that the CFM to learn is non-deterministic.

Theorem 6 *Let $\mathfrak{C}$ be the class of universally-bounded (deadlock-free) CFMs and let $\mathcal{A} \in \mathfrak{C}$ be universally B-bounded. The number of equivalence queries needed to infer a CFM $\mathcal{A}' \in \mathfrak{C}$ with $L(\mathcal{A}) = L(\mathcal{A}')$ is at most $2^{(|\mathcal{A}| \cdot |Msg|+1)^{B \cdot |Proc|^2 + |Proc|}}$. Moreover, the number of membership queries and the overall running time is polynomial in m, exponential in $|\mathcal{A}|$ and $|Msg|$, and doubly exponential in $|Proc|$ and B.*

Proof: We follow the proof of Theorem 5. As $\mathcal{A}$ can be non-deterministic, however, we have to start from the assumption that the number of states of the unique minimal

TABLE IV
UPDATE THE TABLE IN EXTENDED-L*

```
EXTENDED-T-UPDATE():
 1 for w ∈ (U ∪ UAct)V such that T(w) is not defined
 2     if w ∉ D
 3         then T(w) := −;
 4         else if M(w) ∈ Pool
 5                 then
 6                     choose w' ∈ [w]≈D such that T(w') is defined;
 7                     T(w) := T(w');
 8                 else
 9                     T(w) := getClassificationFromTeacher(M(w));
10                     Pool := Pool ∪ {M(w)};
```

DFA $\mathcal{B}$ satisfying $L(synth(\mathcal{B})) = L(\mathcal{A})$ is bounded by $2^{(|\mathcal{A}|\cdot|Msg|+1)^{B\cdot|Proc|^2+|Proc|}}$. ■

Theorem 7 *For $B \in \mathbb{N}$, existentially B-bounded CFMs are learnable.*

Let $\mathfrak{C}$ be the class of existentially B-bounded CFMs. We can provide a learning setup such that, for all $\mathcal{A} \in \mathfrak{C}$, the number of equivalence queries needed to infer an existentially B-bounded CFM $\mathcal{A}' \in \mathfrak{C}$ with $L(\mathcal{A}) = L(\mathcal{A}')$ is at most $2^{(|\mathcal{A}|\cdot|Msg|+1)^{B\cdot|Proc|^2+|Proc|}}$. Moreover, the number of membership queries and the overall running time are polynomial in m, exponential in $|Msg|$ and $|\mathcal{A}|$, and doubly exponential in B and $|Proc|$.

Proof: To obtain a learning setup $(\mathcal{D}, \vdash, synth)$ for $\mathfrak{C}$, let $\mathcal{D}$ be the set of B-bounded well-formed words over Act. As in the previous proof, $\vdash$ is not needed, i.e., we set $\vdash$ to be $\emptyset$. By Theorem 1c), there is a computable mapping $synth$ that transforms any regular set L of B-bounded well-formed words that is closed under $\approx_{\mathcal{D}}$ into a CFM $\mathcal{A}$ with $L(\mathcal{A}) = [L]_{\approx}$. In order to show that $(\mathcal{D}, \emptyset, synth)$ is a learning setup it remains to show that the problems INCLUSION($\mathcal{D}$) and EQCLOSURE($\mathcal{D}$) are constructively decidable. This is shown by a slight modification of the algorithm in [27] for universally bounded languages. This goes as follows.

Let $\mathcal{B} = (Q, q_0, \delta, F)$ 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 any productive state with a channel content, i.e., a function $\chi_s : Ch \to Msg^*$ will be associated to any state $s \in Q$ such that:

1) The initial state q_0 and any final state $q \in F$ are equipped with χ_ϵ, mapping any channel to the empty word.
2) If $s, s' \in Q$ are productive states and $\delta(s, !(p,q,m)) = s'$, then $\chi_{s'} = \chi_s[(p,q) := m \cdot \chi_s((p,q))]$, i.e., m is appended to channel (p,q).
3) If $s, s' \in Q$ are productive states and $\delta(s, ?(q,p,m)) = s'$, then $\chi_s = \chi_{s'}[(p,q) := \chi_{s'}((p,q)) \cdot m]$, i.e., m is removed from the channel (p,q).

$L(\mathcal{B})$ is a set of well-formed words iff there exists a labeling of productive states with channel functions satisfying 1)–3). If a state-labeling violates one of the conditions 1)–3), then this is due to a word that is not well-formed. This word acts as a counterexample for the INCLUSION($\mathcal{D}$) problem. For example, a clash in terms of productive states $s, s' \in Q$ such that $\delta(s, !(p,q,m)) = s'$ and $\chi_{s'}((p,q)) \neq m \cdot \chi_s((p,q))$ gives rise to a path from the initial state to a final state via the transition $(s, !(p,q,m), s')$ that is labeled with a non-well-formed word. This word then acts as a counterexample. Thus, INCLUSION($\mathcal{D}$) is constructively decidable.

To show decidability of EQCLOSURE($\mathcal{D}$), consider a further (decidable) property:

4) Suppose $\delta(s,a) = s_1$ and $\delta(s_1, b) = s_2$ with $a \in Act_p$ and $b \in Act_q$ for some $p, q \in Proc$ satisfying $p \neq q$. If not $(|\chi_s((q,q'))| = B$ and $b = !(q,q',m)$ for some $q' \in Proc$ and $m \in Msg)$ and, moreover, $(a = !(p,q,m)$ and $b = ?(q,p,m)$ for some $m \in Msg)$ implies $0 < |\chi_s((p,q))|$, then there exists a state $s_1' \in Q$ such that $\delta(s,b) = s_1'$ and $\delta(s_1', a) = s_2$.

This *diamond* property describes in which case two successive actions a and b may be permuted. It follows that the set $L(\mathcal{B})$ of well-formed words is closed under $\approx_{\mathcal{D}}$ iff condition 4) holds. This is thanks to the fact that we deal with a deterministic automaton. In case 4) is violated, let w and w' be words of the form $uabv$ and $ubav$, respectively. These words prove that $L(\mathcal{B})$ is not closed under $\approx_{\mathcal{D}}$. Thus, EQCLOSURE($\mathcal{D}$) is constructively decidable. Note that both INCLUSION($\mathcal{D}$) and EQCLOSURE($\mathcal{D}$) are actually solvable in linear time.

To establish the bounds on the overall running time and on the number of equivalence and membership queries, we refer to the considerations in the proofs of Theorems 4 and 1. ■

Theorem 8 *Deterministic universally bounded deadlock-free weak CFMs are learnable.*

Let $\mathfrak{C}$ be the class of deterministic universally-bounded deadlock-free weak CFMs. We can provide a learning setup such that, for all universally B-bounded CFMs $\mathcal{A} \in \mathfrak{C}$, the number of equivalence queries needed to infer an equivalent CFM $\mathcal{A}' \in \mathfrak{C}$ is at most $(|\mathcal{A}| \cdot |Msg| + 1)^{B\cdot|Proc|^2}$. Moreover, the number of membership queries and the overall running time are polynomial in $|Msg|$ and m, exponential in $|\mathcal{A}|$, and doubly exponential in B and $|Proc|$.

Proof: Let $\mathcal{D}$ be the set of all well-formed words. Unlike the previous proofs, we need an inference relation $\vdash \neq \emptyset$ that respects both $\models$ and $\models^{df}$. Let $\vdash$ be the union of

$$\{(L, \{w\}) \mid L \models w \text{ and } L \subseteq \mathcal{D} \text{ is finite}\}$$

(which reflects $\models$) and

$$\{(L_1, L_2) \mid L_1 \subseteq \mathcal{D} \text{ is finite and } L_2 = \{uv \in \mathcal{D} \mid L_1 \models^{df} u\} \neq \emptyset\}$$

(which reflects $\models^{df}$). Theorem 1e) provides the required synthesis function.

Decidability of INFCLOSURE($\mathcal{D}, \vdash$) has been shown in [3, Theorem 3]. Alur et al. provide an EXPSPACE-algorithm for bounded high-level MSCs, which reduces the problem at hand to a decision problem for finite automata with an $\approx$-closed language. The latter is actually in PSPACE. The first step is to construct from the given $\approx$-closed DFA $\mathcal{H}$ a (componentwise) minimal and deterministic weak CFM $\mathcal{A}'$, by simply

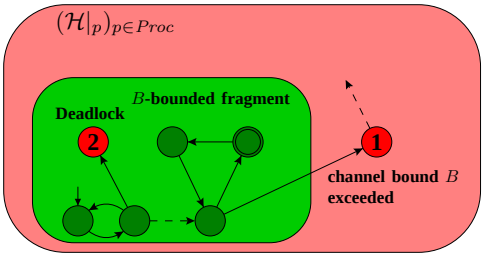


Fig. 7. Schematic view of error cases for proof of Theorem 8

taking the projections $\mathcal{H}|_p$ of $\mathcal{H}$ onto Act_p for any $p \in Proc$, determinizing and minimizing them. Then, $L(\mathcal{H})$ is closed under both $\models$ and $\models^{df}$ iff $\mathcal{A}'$ is a deadlock-free CFM such that $L(\mathcal{A}') = L(\mathcal{H})$. 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 word in $L(\mathcal{H})$.

Thus, a partial run of $\mathcal{A}'$ that either

- exceeds the buffer size B (i.e., it is not B-bounded; cf. Fig. 7, node 1), or
- respects the buffer size B, but results in a deadlock configuration (cf. Fig. 7, node 2),

gives rise to a proper word $u \in Act^*$ that is implied by $\mathcal{H}$ wrt. $\models^{df}$, i.e., $L(\mathcal{H})$ must actually contain a well-formed completion uv 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 $u \restriction p$ is a prefix of $w_p \restriction p$. We set $L_1 = \{w_p \mid p \in Proc\}$.

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 $L(\mathcal{H})$. If this is not the case, one can compute a (B-bounded) word $w \in L(\mathcal{A}') \setminus L(\mathcal{H})$ such that $L(\mathcal{A}') \models w$. Setting $L_2 = \{w\}$, a corresponding set L_1 can be specified as $\{w_p \mid p \in Proc\}$, as above.

Let us turn to the complexity of this particular learning setup. We can partly follow the proof of Theorem 4. As lines 30–41 come into play, however, the complexity estimation is more complicated. The number of equivalence queries is bounded by $C = (|\mathcal{A}| \cdot |Msg| + 1)^{B\cdot|Proc|^2+|Proc|}$ where $\mathcal{A}$ is the CFM at hand. To compute the number of membership queries, we have to take into account the number of words that are added to U in lines 33 and 38 in the pseudo code of the algorithm. To this aim, note that the number of global states of the deterministic weak CFM $\mathcal{A}'$ that we compute above is bounded by $2^{C\cdot|Proc|}$. Moreover, the number of possible channel contents is bounded by $(|Msg| + 1)^{B'\cdot|Proc|^2}$ where $B' = C$ is the maximal number of states of $\mathcal{H}$. Hence,

$$N := 2^{C\cdot|Proc|} \cdot (|Msg| + 1)^{C\cdot|Proc|^2}$$

is an upper bound for the number of configurations of $\mathcal{A}'$ that we have to consider. Moreover, N constitutes a bound on the length of words from L_1 as far as it concerns $\models^{df}$. In turn, L_1 contains $|Proc|$ many words. Now let us turn towards $\models$. To obtain a word w from $L(\mathcal{A}') \setminus L(\mathcal{H})$, we build the product of the complement automaton of $\mathcal{H}$, which is of the same size as $\mathcal{H}$, and the configuration automaton of $\mathcal{A}'$. Thus, the length of w, which consititutes one possible L_2, can be bounded by $C \cdot N$ so that $pref(L_1)$ contains at most $|Proc| \cdot C \cdot N$ words. Therefore, the number of membership queries is in $\mathcal{O}((|Proc| \cdot N \cdot C^3 + mC^2) \cdot |Act|)$. Furthermore, we deduce that the overall running time of the algorithm is polynomial in $|Msg|$ and m, exponential in $|\mathcal{A}|$, and doubly exponential in B and $|Proc|$. ■

Theorem 1d) provides a characterization of (deterministic) universally bounded weak CFMs in terms of regular word languages. Let $\mathcal{D}$ be the set of all well-formed words and let $\vdash$ be given by $\{(L, \{w\}) \mid L \models w \text{ and } L \subseteq \mathcal{D} \text{ is finite}\}$ reflecting $\models$. Unfortunately, the problem INFCLOSURE$(\mathcal{D}, \vdash)$ is undecidable [3] so that the above approach does not work for this particular class of CFMs. One might argue that universally bounded weak CFMs are still learnable, as their regular word languages can be inferred with L*. But an approach that relies solely on L* requires additional expertise from a user. The latter has to make sure by herself that the final hypothesis corresponds to a universally bounded weak CFM. But if we assume that the user needs some guidance and, at the beginning, has an incomplete idea of her system, then we have, for the moment, no means to infer universally bounded weak CFMs.

Note that the complexity of our algorithms is, in most cases, not worse than that of L* if we refer to the size of the underlying minimal DFA. The only instance where the complexity is exponentially higher compared to L* (wrt. the size of the minimal DFA) is reported in Theorem 8. This explosion, however, accounts for the automatic test of hypotheses for deadlocks, which, otherwise, would have to be carried out manually by the user.

V. PARTIAL-ORDER LEARNING

We now derive an improved algorithm that groups words into equivalence classes so that they can be stored efficiently without the need of memorizing all linearizations of MSCs, their prefixes, and their suffixes explicitly. The intention is to reduce the amount of memory necessary for storing Angluin's table. Instead of storing all elements of a class of congruent words, only one representative of each class, a normal form, will be recorded in the table. This approach amounts to merging rows and columns for congruent prefixes and suffixes, respectively, and leads to a substantial reduction of the table size as will be shown experimentally in the next section.

Let $(\mathcal{D}, \approx, \vdash)$ be a learning setup for class $\mathfrak{C}$ of CFMs. In order to represent congruence classes, we introduce a normal form for both prefixes and suffixes of well-formed words. Consider a lexicographic (i.e., strict total) ordering $<_{\text{lex}}$ on Act, which is extended to words over Act in the usual way. Let $pnf, snf : Act^* \to Act^*$. The function pnf assigns to a word $w \in pref(\mathcal{D})$ the minimal word wrt. $<_{\text{lex}}$ that is equivalent to w (to be made precise below). To words that are not in $pref(\mathcal{D})$, pnf assigns an arbitrary receive action from Act. The mapping snf assigns to a word $w \in suff(\mathcal{D})$ its normal form, i.e., the minimum (wrt. $<_{\text{lex}}$) among all

equivalent words, and it associates with every other word an arbitrary send action. The precise definition goes as follows. Let $w \in Act^*$.

- If $w \in \mathit{pref}(\mathcal{D})$, then we set $\mathit{pnf}(w) := \min_{<_{\text{lex}}}\{w' \in \mathit{pref}(\mathcal{D}) \mid \exists v \in Act^*\colon wv \approx_{\mathcal{D}} w'v\}$ where $\min_{<_{\text{lex}}}$ returns the minimum of a given set wrt. $<_{\text{lex}}$. Otherwise, let $\mathit{pnf}(w)$ be any arbitrary receive action.
- If $w \in \mathit{suff}(\mathcal{D})$, then we set $\mathit{snf}(w) := \min_{<_{\text{lex}}}\{w' \in \mathit{suff}(\mathcal{D}) \mid \exists u \in Act^*\colon uw \approx_{\mathcal{D}} uw'\}$. Otherwise, $\mathit{snf}(w)$ is an arbitrary send action.

Note that $\mathit{pnf}(\epsilon) = \mathit{snf}(\epsilon) = \epsilon$ and, moreover, $\mathit{pnf}(w) = \mathit{snf}(w)$ iff w is well-formed. The mappings pnf and snf are canonically extended to sets $L \subseteq Act^*$, i.e., $\mathit{pnf}(L) = \bigcup_{w\in L} \mathit{pnf}(w)$ and $\mathit{snf}(L) = \bigcup_{w\in L} \mathit{snf}(w)$. We assume in the following that both pnf and snf are computable.

It is crucial for the application of normal forms that a given domain $\mathcal{D}$ satisfies, for all $u, v, u', v' \in Act^*$, the following properties:

If $uv \notin \mathcal{D}$, then $u \notin \mathit{pref}(\mathcal{D})$ or $v \notin \mathit{suff}(\mathcal{D})$. (*)

If $uv \in \mathcal{D}$ and $u'v' \approx_{\mathcal{D}} uv'$, then $u'v \in \mathcal{D}$. (**)

If $uv \in \mathcal{D}$ and $u'v' \approx_{\mathcal{D}} u'v$, then $uv' \in \mathcal{D}$. (***)

Under these assumptions, which are satisfied by all the concrete learning setups presented so far, it will indeed be sufficient to look at normal forms when constructing a table in the extension of Angluin's algorithm, which may result in significantly smaller tables. We obtain the extension of EXTENDED-L*, which we call PO-EXTENDED-L* simply by replacing every command of the form $U := U \cup L$ (where L is an arbitrary set of words) by $U := U \cup \mathit{pnf}(L)$, and every command of the form $V := V \cup L$ by $V := V \cup \mathit{snf}(L)$. In particular, EXTENDED-T-UPDATE remains unchanged and is taken from Table IV.

The correctness of our improved algorithm is stated in the following theorem.

Theorem 9 *Let $(\mathcal{D}, \vdash, \mathit{synth})$ be a learning setup for class $\mathfrak{C}$ of CFMs such that $\mathcal{D}$ satisfies* (*)–(***)*. Moreover, let $\mathcal{A} \in \mathfrak{C}$. If the $\mathit{Teacher}$ classifies/provides MSCs in conformance with $\mathcal{L}(\mathcal{A})$, then invoking* PO-EXTENDED-L$^*(\mathcal{D}, \vdash, \mathit{synth})$ *returns, after finitely many steps, a CFM $\mathcal{A}' \in \mathfrak{C}$ such that $L(\mathcal{A}') = L(\mathcal{A})$.*

Proof: Consider an instance of (T, U, V) during a run of EXTENDED-L*. For $w \in (U \cup UAct)V$, the value of $T(w)$ is $-$ if $w \notin \mathcal{D}$. If, on the other hand, $w \in \mathcal{D}$, then $T(w)$ only depends on the classification of $\mathcal{M}(w)$ by the $\mathit{Teacher}$. So let $u, v \in Act^*$. We consider the two abovementioned cases.

- Suppose $uv \notin \mathcal{D}$. Then, by (*), $u \notin \mathit{pref}(\mathcal{D})$ or $v \notin \mathit{suff}(\mathcal{D})$. Thus, $\mathit{pnf}(u)$ is a receive action or $\mathit{snf}(v)$ is a send action so that $\mathit{pnf}(u) \cdot \mathit{snf}(v) \notin \mathcal{D}$.
- Suppose $uv \in \mathcal{D}$. Then, $u \in \mathit{pref}(\mathcal{D})$ and $v \in \mathit{suff}(\mathcal{D})$. By the definition of the mappings pnf and snf, there are u' and v' such that $\mathit{pnf}(u) \cdot v' \approx_{\mathcal{D}} uv'$ and $u' \cdot \mathit{snf}(v) \approx_{\mathcal{D}} u'v$. By (**) and (***), $\{\mathit{pnf}(u) \cdot v, u \cdot \mathit{snf}(v)\} \subseteq \mathcal{D}$ so that $\mathit{pnf}(u) \cdot v \approx_{\mathcal{D}} uv$ and $u \cdot \mathit{snf}(v) \approx_{\mathcal{D}} uv$. Applying (**) (or (***)) a second time, we obtain $\mathit{pnf}(u) \cdot \mathit{snf}(v) \in \mathcal{D}$. We deduce $\mathit{pnf}(u) \cdot \mathit{snf}(v) \approx_{\mathcal{D}} uv$, which implies $\mathcal{M}(uv) = \mathcal{M}(\mathit{pnf}(u) \cdot \mathit{snf}(v))$.

Thus, it does not matter if an entry $T(w)$ in the table is made on the basis of w or on $\mathit{pnf}(u) \cdot \mathit{snf}(v)$, regardless of the partitioning uv of w. In particular, if we replace, in U and V, every word with its respective normal form, then the resulting table preserves consistency and closure properties. Moreover, the DFA that we can construct given the new table is closed and consistent is isomorphic to that of the original table.

As this replacement is precisely what is systematically done in PO-EXTENDED-L*, the theorem follows. ■

Again, the complexity of the modified algorithm depends on the concrete learning setup. Actually, the theoretical time complexity can in general not be improved compared to EXTENDED-L*. However, as the next section will illustrate, the space complexity can be considerably reduced. In the following, we report on positive practical experiences with EXTENDED-L* and PO-EXTENDED-L*.

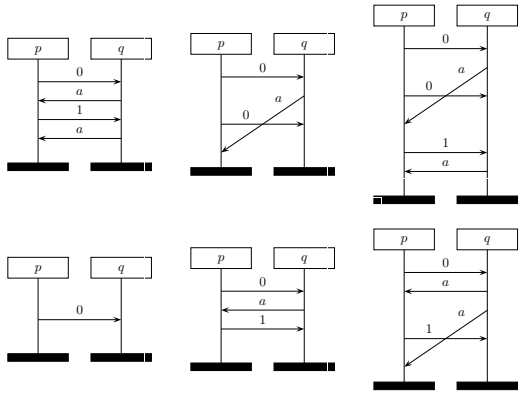
Note that the idea of exploiting an independece relation for learning is not new and appears already in [20] in the context of grey-box checking.

VI. CASE STUDIES

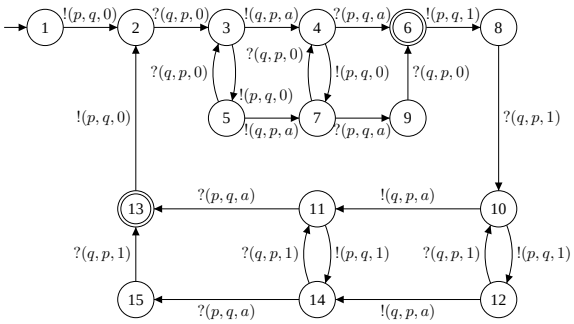
We applied our learning tool *Smyle* [9] to several small and moderately-sized case studies such as a part of the USB 1.1 protocol [23], the *continuous update protocol* [22], the *simple negotiation protocol* [21], the well-known *alternating bit protocol* (ABP) [39] and two variants of a *leader election protocol* [14]. Note that protocols such as the ABP and leader election are error-prone and not straightforward to design correctly from scratch. In the following, we show the learning process for these protocols in more detail and provide statistics for all mentioned protocols indicating the required user effort and the reductions obtained using partial-order learning. We like to emphasize that the protocol designs generated by *Smyle* are guaranteed to be correct by construction, provided (of course) the user-specified MSCs are correct.

Alternating bit protocol (ABP)

The main goal of the ABP [39] is to ensure the reliability of data transmission through an unreliable FIFO channel, i.e., data loss as well as data duplication are possible. Two processes participate in the communication, the *producer* p and the *consumer* q. The channel from p to q is lossy whereas the channel in the other direction is reliable. The protocol works as follows: initially, a bit b is set to 0. Process p keeps sending the value of b until it receives an acknowledgment a from q. After receiving a, process p inverts the value of b and sends the new value until the next a-message is received from q. The communication may terminate after receiving any a (but at least one). For some example MSCs fulfilling this specification see Figure 8i). Note that there is no reason to distinguish between the acknowledgment messages because the channel (q, p) is assumed to be faultless.



i) Three *positive* and three *negative* example MSCs for the ABP



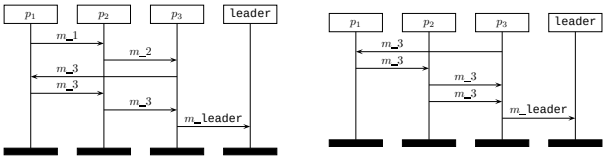
ii) The inferred hypothesis DFA $\mathcal{H}$ after 64 user queries (and 697 membership queries)

Fig. 8. i) Input MSCs and ii) correct and complete hypothesis DFA $\mathcal{H}$

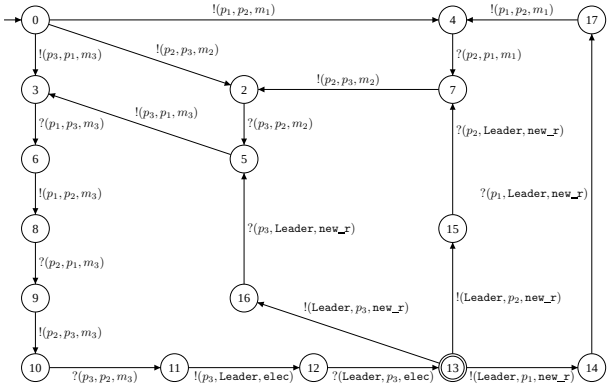
The CFM to be learned was specified as $\exists$1-bounded. Feeding *Smyle* with five positive MSCs of which three are listed in Figure 8 i) yields the correct hypothesis as depicted in Figure 8 ii). To that end, the learning algorithm internally dealt with 2,286 membership queries of which 64 were user queries. I.e., the 2,222 remaining queries are answered automatically by our approach whereas in the original L* algorithm all of them had to be answered manually. Note that the number of memberhsip queries is reduced to 697 (i.e., by 69.5%) using partial-order learning. All results on learning the ABP, also with higher channel bounds, are given in Table V.

Leader election protocol

Leader election plays an important role in many distributed applications. In a network of identical (up to their unique process id called `pid` for short) communicating units, one often uses a leading entity (owning a unique *leader token*) to control the behavior of the others. However, a problem arises if, due to communication failures or other possible problems, the leader token is lost. The goal of leader election protocols is then to select a unique leader among the processes. We consider a leader election protocol in a unidirectional ring [14]. The protocol works as follows: one process starts sending its `pid` to its clockwise neighbor who compares the value of the received `pid` with its own. In case both values are equal, a leader has been found and the current process declares



i) One positive and one negative scenario for the leader election protocol



ii) The inferred hypothesis DFA $\mathcal{H}$ after 196 user queries (and 6,864 membership queries)

Fig. 9. i) Some input MSCs, ii) hypothesis $\mathcal{H}$

itself the leader by sending message m`_leader`. If its own `pid` has a higher value, it forwards its `pid` to its clockwise neighbor; otherwise, the received `pid`. Due to the high amount of concurrency, we consider three processes p_1, p_2 and p_3 and allow to send only one message at each point in time.

We learned two versions of the protocol: one with only a single election round, and one with arbitrarily many consecutive election rounds. *Smyle* learned the first variant by starting with three input MSCs. It displayed the correct hypothesis after 43 user queries and 900 membership queries using partial-order learning, a reduction of 75.1% compared to the case without this optimization. For the second vartiant we used six input MSCs (e.g., the first one of Figure 9 i)). The correct hypothesis depicted in Figure 9 ii) was obtained after 196 user queries. Figure 9 i), for example, indicates a negative scenario. In this setting, partial-order learning yielded a reduction of about 53% resulting in 6,864 membership queries.

Results

The learning statistics for the considered case studies are summarized in Table V. The first three columns detail the total number of membership queries that were needed, and indicate the savings obtained when learning the protocols with partial-order learning (column w. POL) compared to the case without (column w.o. POL). In all cases, a substantial reduction is obtained. This also applies to the size of the tables needed during learning. This indicates that besides a significant reduction in the number of membership queries, significant memory savings are obtained. The numbers clearly indicate that on increasing channel bounds (for the ABP), the number of membership queries quickly becomes quite high. The fourth column indicates the number of membership queries that had

TABLE V
STATISTICAL RESULTS OF CASE STUDIES

Protocol	#membership queries w.o. POL	w. POL	savings	#user queries	#equivalence queries	$\|\mathcal{H}\|$	#rows in table w.o. POL	w. POL	reduction	learning setup
part of *USB 1.1*	488	200	59.0%	14	1 (5)	9	61	26	57.4%	$\exists 2$
continuous update	712	264	62.9%	21	1 (3)	8	89	34	61.8%	$\exists 1$
negotiation	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	$\exists 1$
ABP	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	$\exists 1$
ABP	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	$\exists 2$
ABP	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	$\exists 3$
leader election (1 round)	3,612	900	75.1%	43	1 (2)	13	301	76	74.8%	$\forall$
leader election (≥ 1 rounds)	14,704	6,864	53.3%	196	2 (5)	17	919	430	53.2%	$\forall$

to be dealt with by the user. These queries amount to classify generated MSCs as either positive or negative scenarios and are usually not very difficult to handle. The number of user queries can be reduced significantly by providing mechanisms (such as the logic PDL [10, 11]) to classify an entire set of MSCs that feature a certain pattern.

Equivalence queries are harder to handle but their number for the example protocols is rather low (cf. fifth column). The numbers in brackets in this column indicate the number of required equivalence queries when using the standard Angluin's L^* algorithm. It clearly shows that with our approach this number is lowered significantly. As these queries require a user-driven simulation (and/or testing) of an automaton, this reduction is crucial and yields a substantial reduction in the development time. With an experienced human teacher, all case studies from Table V could be performed in a single working day. Using PDL formulae to filter user queries [10, 11] this could be lowered to a few hours.

VII. RELATED WORK

Synthesizing design models or programs from scenarios has received a lot of attention. Let us distinguish *basic MSCs* and *high-level MSCs* (and *live sequence charts* (LSCs)).

Synthesis from basic MSCs

A basic MSC, as used in our paper, does neither contain loops nor alternatives, and describes a finite set of behaviors. Thus, a finite set of basic MSCs also describes a finite set of behaviors. Typically, a system under development has infinitely many behaviors, so that a finite set of scenarios in terms of basic MSCs can only be an approximation of all these behaviors. In fact, the learning algorithm generalizes the finite set of given scenarios to a typically infinite set represented by the design model. In simple words, we synthesize design models from finitely many *examples*. Note that two different basic MSCs describe distinct behaviors. Thus, classifying one MSC as desired and one (different) as undesired cannot lead to an *inconsistent* set of desired and unwanted behaviors.

One of the first attempts to exploit learning for interactively synthesizing models from examples was proposed in [33] where for each process in the system an automaton is inferred using Angluin's learning technique. A significant drawback of this approach is that composing the resulting automata in parallel yields a system that may exhibit undesired behavior and may deadlock.

Damas et al. [16] use an interactive procedure of classifying positive and negative scenarios for deriving an LTS for each process. To this end, they first employ passive learning algorithms [19, 37] to infer a global intermediate model that exactly conforms to the given sample but which—as long as the sample does not fulfill certain completeness properties—does not necessarily yield a minimal system model. The global model is subsequently transformed into a distributed system. Then, usually additional effort is required as this projection onto the system's components may entail implied behavior such that manually unwanted "[...] implied scenarios have to be detected and excluded". While [16] consider synchronous communication, our approach is based on asynchronous communication. This makes the systems larger but, more importantly, closer to distributed implementations. Finally, [16] is a *passive* learning approach which does not support incremental model generation. Our approach naturally supports extensions and amendments of requirements.

Another passive learning approach [15] uses grammatical inference to derive a formal model of a process from a given stream of system events. In contrast to our and the aforementioned approaches, this method only works with positive data. Though this procedure may require less user effort than ours, it builds on the restrictive assumption that the events of the process are monitorable by the learner. The authors also shortly comment on detecting concurrency by searching for unrelated events but leave it for future work to improve their approach.

Similar to [28], [34] propose to use filters, i.e., automated replies to queries, for reducing the number of membership queries that—due to the high number of questions—is usually infeasible for human teachers to answer. The general idea is to exploit additional knowledge of an expert teacher which for each negatively answered membership query specifies prefixes or suffixes for which negative membership is known. In their approach they employ these filters for membership as well as equivalence queries, but conclude that for equivalence queries no substantial improvements are obtained. This result is in contrast to our approach which considerably decreases this number.

Synthesis from (high-level) MSCs

The synthesis from scenarios given using richer formalisms, e.g., MSCs with loops and alternatives, high-level MSCs, or live sequence charts has received quite some attention. The underlying assumption is that not only several examples of the expected behavior are given but that the given behavior (mostly) corresponds to the system behavior. Then, the technical question arising is how to translate from a scenario-based formalism to a state-based formalism. One of the initial works along this line is [31], which sketches the translation from (high-level) MSCs to statechart models. Similarly, [41] presents a rigorous approach for synthesizing transition systems from high-level MSCs.

The question whether the behavior given by a finite set of MSCs or high-level MSC can in fact be realized by weak CFMs or CFMs is studied, respectively, in [2], [3], and [26]. In simple words, it turns out that the set of scenarios has to meet certain restrictions to be realizable and that the question, whether it is realizable or not is often undecidable.

Note that describing desired and unwanted behavior in terms of high-level MSCs would allow for inconsistent sets of scenarios as also different high-level MSC may describe a common subset of behaviors.

The works [3, 7, 23–27, 32, 35] synthesize CFMs from particular classes of finite automata, which can be seen as generalizations of high-level MSCs. Recall that results from [3, 7, 24, 27, 32] together constitute Theorem 1, which, however, is only an ingredient of our algorithms. Our objective is actually to synthesize these particular finite automata that, in turn, allow for constructing a CFM.

In [40], a synthesis technique is proposed that constructs behavior models in the form of modal transition systems (MTS) and is based on a combination of safety properties and scenarios. MTSs distinguish required, possible and proscribed behavior and can thus be seen as design models that are more abstract compared to the CFMs that are synthesized using our approach.

Damm and Harel [17] pointed out that the expressiveness of (high-level) MSCs is often inappropriate to specify complete system behavior and introduced the richer notion of live sequence charts (LSCs). Harel's play-in, play-out approach for LSCs [18] allows us to execute the possible behavior defined in terms of LSCs, which essentially results in a programming methodology based on LSCs. A similar, executable variant of LSCs, triggered MSCs, is presented in [38].

All the previously mentioned 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 basic MSCs have to be provided as examples, simplifying the requirements specification task.

VIII. Conclusion and Future Work

This paper generalized Angluin's learning algorithm for synthesizing a DFA from positive and negative samples to a procedure that infers a CFM from a set of negative and positive basic MSCs. This yields an exact approach —the resulting CFM precisely accepts the positive MSCs and rejects the negative ones— and is applicable to various classes of CFMs such as various types of deterministic universally-bounded (weak) CFMs and existentially bounded CFMs. Note that our learning setting is also applicable to other classes such as, e.g., the causal closure as defined by Adsul et al. [1]. It remains to study whether the class of existentially B-bounded (deadlock-free) weak CFMs is learnable. For the deadlock-free case, one needs to generalize a result by Lohrey, who shows that one can decide whether a globally-cooperative HMSC is implementable as a deadlock-free CFM [32, Theorem 3.5].

We have shown the feasibility of our approach by reporting on some experiments that we carried out with our tool *Smyle* [2]. By exploiting the properties of partial orders (as MSCs) a significant reduction of the memory consumption could be achieved. Alternative improvements are, e.g., to reduce the number of user queries by using a logic to specify a priori undesired partial behaviour, e.g., in case of the ABP "no bit change without prior acknowledgement". First results towards using PDL [11] to this purpose have been recently reported in [10]. The latter paper also describes how to embed *Smyle* into an incremental software engineering process.

Possible directions for future work are to support co-regions in basic MSCs, to realize the algorithms for weak CFMs, and to investigate further classes of learnable CFMs.

IX. Acknowledgment

This work is partially supported by EGIDE/DAAD-PROCOPE (Procope 2008/2009) and the DFG Research Training Group AlgoSyn. We would thank our student assistants David Piegdon and Stefan Schulz (both RWTH Aachen University) for their support in tool development. Moreover, we thank the anonymous reviewers for their valuable suggestions.

References

[1] B. Adsul, M. Mukund, K. N. Kumar, and V. Narayanan. Causal closure for MSC languages. In *FSTTCS*, volume 3821 of *LNCS*, pages 335–347. Springer, 2005.

[2] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. *IEEE TSE*, 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] J. Araújo. Formalizing sequence diagrams. In *Proceedings of the OOPSLA Workshop on Formalizing UML. Why? How?*, 1998.

[6] N. Baudru and R. Morin. Safe implementability of regular message sequence chart specifications. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, volume 2380 of *LNCS*, pages 210–217. Springer, 2003.

[7] N. Baudru and R. Morin. Synthesis of safe message-passing systems. In *FSTTCS*, volume 1664 of *LNCS*, pages 277–289. Springer, 2007.

[8] B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker. Replaying play in and play out: Synthesis of design models from scenarios by learning. In *TACAS*, volume 4424 of *LNCS*, pages 435–450. Springer, 2007.

[9] B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker. *Smyle*: A tool for synthesizing distributed models from scenarios by learning. In *CONCUR*, volume 5201 of *LNCS*, pages 162–166. Springer, 2008.

[10] B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker. SMA—The Smyle Modeling Approach. In *CEE-SET 08 (IFIP)*, LNCS. Springer, 2009. To appear.

[2] The *Smyle* tool is freely available for exploration at: `http://www.smyle-tool.org/`.

[11] B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. In *FSTTCS*, volume 4855 of *LNCS*, pages 303–315. Springer, 2007.
[12] B. Bollig and M. Leucker. A hierarchy of implementable MSC languages. In *FORTE*, volume 3731 of *LNCS*, pages 53–67. Springer, 2005.
[13] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. of the ACM*, 30(2):323–342, 1983.
[14] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283, 1979.
[15] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM TOSEM*, 7(3):215–249, 1998.
[16] C. Damas, B. Lambeau, and P. Dupont. Generating annotated behavior models from end-user scenarios. *IEEE TSE*, 31(12):1056–1073, 2005.
[17] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:1:45–80., 2001.
[18] D.Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
[19] P. Dupont. Incremental regular inference. In L. Miclet and C. de la Higuera, editors, *Int. Coll. on Grammatical Inference: Learning Syntax from Sentences (ICGI)*, volume 1147 of *LNCS*, pages 222–237. Springer, 1996.
[20] E. Elkind, B. Genest, D. Peled, and H. Qu. Grey-box checking. In *FORTE*, volume 4229 of *LNCS*, pages 420–435. Springer, 2006.
[21] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-based agent communication protocols. In *Workshop on Agent Communication Languages*, pages 91–107, 2003.
[22] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *IJCAI*, pages 679–684, 2003.
[23] B. Genest. Compositional message sequence charts (CMSCs) are better to implement than MSCs. In *TACAS*, volume 3440 of *LNCS*, pages 429–444. Springer, 2005.
[24] 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.
[25] B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fund. Inf.*, 80(2):147–167, 2007.
[26] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state high-level MSCs: Model-checking and realizability. *Journal on Computing and System Sciences*, 72(4):617–647, 2006.
[27] J. G. Henriksen, M. Mukund, K. N. Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Inf. Comput.*, 202(1):1–38, 2005.
[28] H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *CAV*, volume 2725 of *LNCS*, pages 315–327. Springer, 2003.
[29] ITU-TS Recommendation Z.120anb: Formal Semantics of Message Sequence Charts, 1998.
[30] ITU-TS Recommendation Z.120 (04/04): Message Sequence Chart, 2004.
[31] I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to Statecharts. In *DIPES*, volume 155 of *IFIP Conf. Proc.*, pages 61–72. Kluwer, 1998.
[32] M. Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Th. Comp. Sc.*, 309(1-3):529–554, 2003.
[33] E. Mäkinen and T. Systä. MAS – An interactive synthesizer to support behavioral modeling in UML. In *ICSE*, pages 15–24. IEEE CS Press, 2001.
[34] A. L. Martins, H. S. Pinto, and A. L. Oliveira. Using a more powerful teacher to reduce the number of queries of the L* algorithm in practical applications. In *EPIA*, LNCS, pages 325–336. Springer, 2005.
[35] R. Morin. Recognizable sets of message sequence charts. In *STACS*, volume 2285 of *LNCS*, pages 523–534. Springer, 2002.
[36] B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *ICSE*, pages 35–46. ACM, 2000.
[37] J. Oncina and P. García. Inferring regular languages in polynomial updated time. In *4th Spanish Symp. on Pattern Recognition and Image Analysis*, volume 1 of *Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, 1992.
[38] B. Sengupta and R. Cleaveland. Triggered message sequence charts. *IEEE TSE*, 32(8):587–607, 2006.
[39] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 2002.
[40] S. Uchitel, G. Brunet, and M. Chechik. Behaviour model synthesis from properties and scenarios. In *ICSE*, pages 34–43. IEEE CS Press, 2007.
[41] S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE TSE*, 29(2):99–115, 2003.

Benedikt Bollig received his PhD degree from Aachen University of Technology (RWTH Aachen) in 2005. A revised version of his thesis on automata models and logics for message sequence charts has been published by Springer. In 2003/2004, the German Academic Exchange Service (DAAD) funded his six-month research stay at Birmingham University. Since 2005, Benedikt is a CNRS full-time researcher. His research interests are centered around logics for specification, formal languages and automata, with a focus on applications in the synthesis and verification of concurrent and timed systems.

Joost-Pieter Katoen is a full professor at the RWTH Aachen University and is associated to the University of Twente. His research interests are concurrency theory, model checking, timed and probabilistic systems, and semantics. He co-authored more than 100 journal and conference papers, and recently published a comprehensive book (with Christel Baier) on "Principles of Model Checking".

Carsten Kern received his Diploma in computer science in 2005 from RWTH Aachen University. In August 2009 he successfully defended his dissertation at the chair of Software Modeling and Verification at the RWTH Aachen University where he is employed for another two months. His current research interests cover learning of nondeterministic and communicating automata.

Martin Leucker is currently a professor at the Technische Universität München for Theoretical Computer Science and Software Reliability. He obtained his Ph.D. at RWTH Aachen, Germany, and worked afterwards as a Postdoc at the University of Philadelphia, USA, and, within the European Research and Training Network on Games, at Uppsala University, Sweden. He pursued his Habilitation at TU München while being a member of Manfred Broy's group on Software and Systems Engineering. He is the author of more than 60 reviewed conference and journal papers ranging over software engineering, formal methods, and theoretical computer science.