Extending Compositional Message Sequence Graphs

Benedikt Bollig[1]*, Martin Leucker[2]**, and Philipp Lucas[1,3] ***

[1] Lehrstuhl für Informatik II, RWTH Aachen, Germany
{bollig,phlucas}@i2.informatik.rwth-aachen.de
[2] Department of Computer Systems, Uppsala University, Sweden
leucker@docs.uu.se
[3] Universität des Saarlandes, Saarbrücken, Germany
phlucas@cs.uni-sb.de

Abstract. We extend the formal developments for message sequence charts (MSCs) to support scenarios with lost and found messages. We define a notion of *extended compositional message sequence charts* (ECMSCs) which subsumes the notion of compositional message sequence charts in expressive power but additionally allows to define lost and found messages explicitly. As usual, ECMSCs can be combined by means of choice and repetition to (extended) compositional message sequence graphs. We show that—despite extended expressive power—model checking of monadic second-order logic (MSO) for this framework remains to be decidable. The key technique to achieve our results is to use an extended notion for linearizations.

1 Introduction

In the development process of a large and complex communication protocol, one usually starts with describing scenarios showing aspects of its desired behavior. The scenarios may be combined to get a more complete view of the system. This partial specification guides the realization of the resulting piece of hardware or software, which, in turn, might be checked according to these scenarios.

Message Sequence Charts (*MSCs*), which are similar to UML's sequence diagrams [2], are a prominent, standardized [12, 13], and graphical as well as textual formalism to further this approach. An MSC defines a set of processes and a set of communication actions between these processes. In the visual representation

* Part of this work was done during the author's stay at the IT department, Uppsala University, Sweden. He is grateful for the hospitality and the overall support.

** The research was mainly accomplished during the author's stay at Dept. of Computer and Information Science, University of Pennsylvania, USA, and was supported in part by NSF CCR-9988409, NSF CCR-0086147, NSF CISE-9703220, ARO DAAD19-01-1-0473, DARPA ITO MOBIES F33615-00-C-1707, and ONR N00014-97-1-0505. Additionally, the work was supported by the European Research Training Network on "Games", HPRN-CT-2002-00283.

*** supported in part by "Graduiertenkolleg *Leistungsgarantien für Rechnersysteme*" of an MSC, processes are drawn as vertical lines. A labeled arrow from one line to another corresponds to the communication of sending the labeling value from the first process to the second. Figure 1(a) gives an example of an MSC.

MSCs are combined to so-called *high-level MSCs* or *MSC graphs*[1] (*MSGs*) by means of choice and repetition. An MSG is a directed graph or finite automaton whose edges are labeled with MSCs. Every path through this graph corresponds to the sequential composition of the basic MSCs. In this way, arbitrarily long and alternative scenarios may be obtained out of the given ones.

Since MSCs and MSGs are employed in the early design phase of a system, it is essential to provide decent formal methods for analyzing them. In recent years, a lot of work was done to support this issue, however, usually for a restricted class of MSCs.

The first question which arises in this context is whether a *set* of MSCs is *realizable*. In the seminal work of [11] and [10], regular sets of MSCs were defined and studied for this reason. The main idea was to study linearizations of MSCs. A linearization of an MSC is a sequence of all its send and receive events whereby the partial order imposed by the graphical representation of the MSC is respected. It was shown that regular sets of MSCs are the ones which are definable by monadic second-order logic (MSO) over MSCs. This work was later extended in [6] to a larger class of MSCs namely one which also allows MSCs with message overtaking. Therefore, a conceptually new way to model MSCs and their linearizations was introduced.

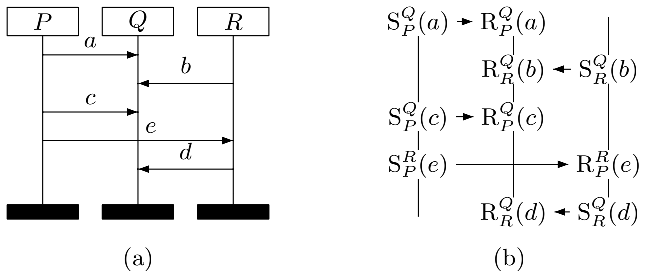

Fig. 1. An MSC and its formalization

Another approach is to analyze MSGs. Model checking of MSGs with respect to an automaton specification was studied in [1]. More specifically, the question whether the sequences of send and receive actions obtained by linearizing the MSCs established by finite paths through the MSG adhere to some MSO formula (over words) was tackled. This problem is also known as *linearization model checking*. It turned out that model checking becomes undecidable in general, unless some *boundedness* assurance is guaranteed (in the case of the more natural asynchronous composition of MSCs). On the contrary, the model-checking problem of MSGs against MSO formulas which are interpreted directly on MSCs and not on its linearizations was shown to be decidable [15].

[1] In the following, we will use the more suggestive notion of MSGs.

Compositional extensions of MSC specifications were first studied in [17]. The definition of MSCs was extended by a notion of *gates* that serve as connectors when combining MSCs. A different approach was studied in [9] and so-called high-level *compositional* MSCs (HCMSCs) were proposed. Compositional MSCs may have *open* send and receive events admitting a send event of one MSC being received by a subsequent MSC. Gates were introduced to combine MSCs *horizontally* (several MSCs representing single processes are combined to one MSC having several processes), HCMSCs suit the vertical composition of MSCs. Besides their different motivation, gates use names together with corresponding send and receive events to identify events to be connected and therefore allow a finer control over composition.

In *Life Sequence Charts* as defined by Damm and Harel in [7], messages may be labeled as *cold* or *hot*. Hot messages must be received, whereas cold messages might get lost. So, this approach combines both these possibilities in one diagram, whereas our formalization makes joining message events and lost and found messages explicit. In this way it is possible to join a lonely send event with a corresponding receive event located in an MSC further down the path of an ECMSG.

While the authors are unaware of any literature on model checking MSCs with gates, it was shown that basic properties of HCMSCs are undecidable. [16] slightly restricts the class of HCMSCs towards compositional message sequence graphs (CMSGs), still maintaining the main features of HCMSCs. They show that monadic second-order logic over these CMSGs is decidable. However, not every CMSC-labeled graph is a CMSG. While it is decidable whether such a structure is a CMSG, it might bother the user of CMSGs to understand the restrictions imposed to obtain a CMSG.

In this paper, we extend the notion of CMSGs to support aspects of the MSC standard [13], so-called *black* and *white holes*. This concept allows the modeling of lost and found messages, respectively. Thus, one is able to describe behaviors of a protocol that catches up after a message has been lost. We present scenarios for the Alternating-Bit Protocol giving evidence for the adequacy of our formalism. Furthermore, our formalism also supports message overtaking. Note that this concept was excluded in [9] as well as in [16]. Technically, we add *names* to open send and receive events, thus, we use ideas of *gates*.

Although in some definitions for compositional message sequence charts, one might end up with unmatched send or receive events, this can only happen to a limited extent. For example, for *left-closed* CMSCs defined in [9], it is possible that a send event lacks its corresponding receive event. But in that case, no further matched send event of the same type will be able to follow.

In the following, we abbreviate our extended compositional message sequence graphs by ECMSGs and call our extended version of a compositional message sequence chart ECMSC. It turns out that all scenarios definable in terms of CMSGs can also be defined by ECMSGs. Due to the fact that ECMSGs support the definition of holes and message overtaking, we easily obtain the result that ECMSGs are strictly more expressive than CMSGs.

Despite the additional expressive power, we show that model checking of MSO formulas remains to be decidable in our extended setting. We combine ideas of [6] and the general approach to interpret one logic within another to obtain our results. As opposed to CMSGs defined in [16], every ECMSC-labeled graph is an ECMSG. Thus, no further test whether the setting given by a user is indeed an (E)CMSG has to be applied.

Finally, we show in which way our model can be related to so-called Mazurkiewicz-trace–closed languages. As pointed out by [16], this is the cornerstone for applying the rich theory of local temporal logics over traces such as TrPTL [22].

In the next section, we define extended compositional MSCs and ECMSGs. We give a set of scenarios for the Alternating-Bit Protocol in Section 3. Section 4 reveals the correspondence of ECMSGs and regular word languages. Model-checking procedures for ECMSGs are considered in Section 5.

An extended version of the paper appeared as [5].

2 Message Sequence Charts and Graphs

Before we present our extensions, let us start with the customary formal definition of message sequence charts, in which an MSC is understood of as a partial order of send and receive events, which can be matched subject to a bijective mapping:

Let $\mathcal{P}$ be a finite, nonempty set of *process names* (or just *processes*) and *Mess* a finite *message alphabet*. Let further $\Sigma_{\mathcal{S}} := \{\mathrm{S}_p^q(a) \mid p,q \in \mathcal{P},\ p \neq q,\ a \in \mathit{Mess}\}$ and $\Sigma_{\mathcal{R}} := \{\mathrm{R}_p^q(a) \mid p,q \in \mathcal{P},\ p \neq q,\ a \in \mathit{Mess}\}$ denote the sets of *send* and *receive actions*, respectively, and $\Sigma := \Sigma_{\mathcal{S}} \cup \Sigma_{\mathcal{R}}$ the set of *actions*. An action $\mathrm{S}_p^q(a)$ stands for sending a message a from process p to process q, and $\mathrm{R}_p^q(a)$ represents the corresponding receive action, which is then executed by process q. In this sense, $\mathit{Corr} := \{(\mathrm{S}_p^q(a), \mathrm{R}_p^q(a)) \mid p,q \in \mathcal{P},\ p \neq q,\ a \in \mathit{Mess}\}$ relates those actions that are corresponding. From now on, all premises and definitions are made with respect to a fixed set $\mathcal{P}$ of processes and a fixed message alphabet *Mess*. Thus, Σ and the derived symbols are also fixed. Throughout the paper, $\mathit{Dom}(f)$ and $\mathit{Range}(f)$ will furthermore denote the domain and the range of a function f, respectively.

A *Message Sequence Chart (MSC)* is a tuple $m = (E, \preceq, t, o, \mu, \ell)$ where

- E is the finite set of *events*.
- $t : E \to \{\mathtt{S}, \mathtt{R}\}$ is the *type function* indicating whether we deal with a *send* or *receive event* , i.e., $t(e) = \mathtt{S}$ or respectively $t(e) = \mathtt{R}$.
- $o : E \to \mathcal{P}$, the *process function*, binds each event to a process.
- $\mu : t^{-1}(\mathtt{S}) \to t^{-1}(\mathtt{R})$ is the bijective *matching function*.
- $\ell : E \to \Sigma$ is the *labeling function* such that, for each $e \in E$, $t(e) = \mathtt{S}$ implies

$$\ell(e) = \mathrm{S}_{o(e)}^{o(\mu(e))}(a) \text{ and } \ell(\mu(e)) = \mathrm{R}_{o(e)}^{o(\mu(e))}(a)$$

  for some $a \in \mathit{Mess}$. In other words, events matched by μ are labeled by ℓ with corresponding action labels.

– $\preceq \subseteq E \times E$ is a partial order that is total on each process line and orders receive events after their originating send event. Thus, we require $\preceq$ to satisfy the following:
  - $\preceq_p := \preceq \cap (o^{-1}(p) \times o^{-1}(p))$ is a total order for each $p \in \mathcal{P}$
  - $\preceq = (\bigcup_{p \in \mathcal{P}} \preceq_p \cup \{(e, \mu(e)) \mid t(e) = \mathtt{S}\})^*$

To simplify our notation, we will also use o to denote a mapping associating a label from Σ with its corresponding process, i.e., $o(\mathrm{S}_p^q(a)) = o(\mathrm{R}_q^p(a)) = p$.

The formalization of an MSC, as illustrated in Figure 1(b), is a directed graph in which the nodes, according to ℓ, are labeled with elements of Σ and where the arrows depict μ. Furthermore, a vertical line corresponds to $\preceq_p$ for a suitable $p \in \mathcal{P}$.

An MSC is just a finite object showing a scenario for an execution of an underlying system. To derive a more sophisticated description of the system, one aims at combining several MSCs. Therefore, however, it turns out to be helpful, if one allows so-called "unmatched" send events, which then might be gathered in a subsequent MSC by a corresponding "unmatched" receive event, as pointed out in [9]. Furthermore, we want to be able to describe scenarios in which, on the one hand, there has been a send event but no corresponding receive event[2], or, on the other hand, a message is received that has not been sent[3]. In terms of MSC standard [12], we want to support *black* and *white holes*, respectively. Thus, we extend our definition accordingly:

Definition 1 (Extended Compositional Message Sequence Chart). *An* Extended Compositional Message Sequence Chart (ECMSC) *is just a message sequence chart* $(E, \preceq, t, o, \mu, \ell, \lambda)$ *as above except for* t*, which is henceforth a function* $E \to \{\mathtt{S}, \mathtt{R}, \mathtt{U}, \bot\}$ *such that an event* e *with* $t(e) \in \{\mathtt{S}, \mathtt{R}\}$ *is labeled as stipulated above and* $t(e) \in \{\mathtt{U}, \bot\}$ *implies* $o(\ell(e)) = o(e)$*. In addition,* λ *is a mapping* $t^{-1}(\mathtt{U}) \to \mathbb{N}$*, called* matching information*, such that*

1. *for each* $\sigma \in \Sigma$ *and each natural number* n*,* $|\{e \in E \mid t(e) = \mathtt{U}, \ell(e) = \sigma, \text{ and } \lambda(e) = n\}| \leq 1$*, and*
2. *for all* $e \in E$ *with* $t(e) = \mathtt{U}$ *and* $\ell(e) \in \Sigma_{\mathcal{S}}$*, there is no event* e' *with* $t(e') = \mathtt{U}$*,* $(\ell(e), \ell(e')) \in Corr$*,* $e' \not\preceq e$*, and* $\lambda(e) = \lambda(e')$*.*

Let us study the previous definition in more detail. As before, if the type of an event is S or R, we deal with a send or receive event corresponding to a single message. The symbol $\bot$ can be employed to explicitly define a hole (a *lost* message or event). The symbol U denotes an unmatched message event which can possibly find a suitable communication partner in a subsequent or previous ECMSC. To give information in which way unmatched events of type U should be combined for two given ECMSCs, we employ the functions ℓ and λ. Obviously, an unmatched receive event can only be matched with an unmatched send event. We do not want to model misdirected messages, so a message sent from p to q may only be received by q—to ensure this, the labeling function ℓ provides the required information.

[2] e.g., because the message got lost

[3] e.g., because some message from some previous session remains in the input channel

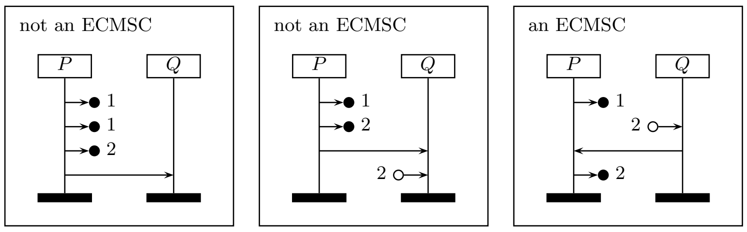

Fig. 2. Two counter-examples and an example

To allow for a more complex modeling, the matching information λ is used to identify corresponding send and receive events. As we will see below, an unmatched send event with number k usually matches the first suitable unmatched receive event that has number k, which should be one event of a subsequent ECMSC. Therefore, we require λ to be in the way that, within a given ECMSC, there is no corresponding receive event which could be matching to a send event (cf. condition 2 of Definition 1). For instance, look at Figure 2, where, in each diagram, the use of only one message $a \in \mathit{Mess}$ is assumed and a possibly unmatched message, as indicated, is sent to (received from, respectively) the neighbored process line. The diagram in the middle is no ECMSC, because condition 2 is not fulfilled. There is no reason not to match the send event with number 2 with the receive event numbered 2. However, the third diagram in this line, though it employs unmatched messages of type `U` with corresponding labelings and the same matching information, is an ECMSC. The unmatched receive event with number 2 can now only be matched with one of a previous ECMSC.

For the same reason, we do not allow to carry two unmatched events of type `U` with the same labeling and matching information (cf. condition 1 of Definition 1). Reusing the same number k for a send event, for example, means intuitively that the first message got lost and should therefore be labeled with $\perp$. Thus, the diagram on the left of Figure 2 is not an ECMSC.

Another approach to deal with lost and found messages formally, is to add one process for each channel, i.e. pair of processes. For example, for the ECMSC shown in Figure 3(a), one might add a process PQ and simulate every message from P to Q by a message from P to PQ followed by one from PQ to Q (cf. Figure 3(b)). Lost or found messages could then be represented by leaving out the second or the first message, respectively. So, one could specify and reason in the framework of CMSCs instead of ECMSCs. However, finding a similar message to the one lost could not be distinguished from passing on the original one (see lower part in Figure 3(a) and Figure 3(b)), which is undesirable especially when the messages of the scenarios are obtained by abstracting concrete message values to a finite number of messages. While the MSC on the left hand side has lost messages, the second does not. Furthermore, message overtaking cannot be modeled. Finally, explaining why a formula is not valid for the given scenario is more difficult, if the underlying scenario is changed and $n(n-1)$ processes are added to n processes. Thus, we are convinced that our direct approach is preferable to a simulating one, last but not least, for complexity reasons.

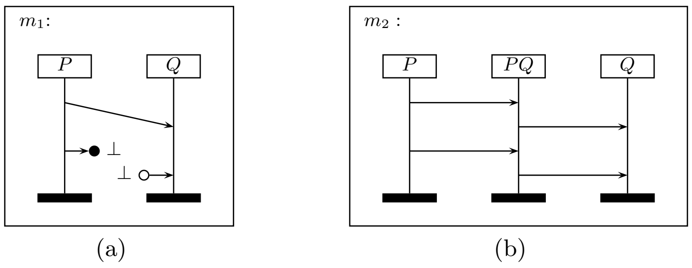

Fig. 3. Simulating lost and found messages

Before we turn towards the formal definition of the concatenation of ECMSCs, let us introduce some abbreviations: Given an ECMSC m as above, let $S_m := t^{-1}(\mathtt{S})$, $S^u_m := t^{-1}(\mathtt{U}) \cap \ell^{-1}(\Sigma_{\mathcal{S}})$, and U_m, $\perp_m$, R_m and R^u_m be defined analogously. Furthermore, we set $\lambda(m) := \mathit{Range}(\lambda)$ as well as, for a collection $M = \{m_1, \ldots, m_k\}$ of ECMSCs, $\lambda(M) := \bigcup_{i \in \{1,\ldots,k\}} \lambda(m_i)$.

Let us now consider the concatenation of ECMSCs. The main idea is that we wish to specify the behavior of a system that first follows ECMSC m_1 and then ECMSC m_2 by defining a single ECMSC $m_1 \cdot m_2$ that allows exactly the desired behaviors. In our setting, we use *asynchronous concatenation*, i.e., any process that has completed all events in m_1 may proceed to m_2. To concatenate unmatched send and receive events, we employ a function μ_{m_1,m_2}, which formally captures the previously mentioned ideas.

For ECMSCs $m_i = (E_i, \preceq_i, t_i, o_i, \mu_i, \ell_i, \lambda_i)$, $i = 1, 2$, with disjoint sets of events (if this is not the case, the events have to be renamed accordingly), let $\mu_{m_1,m_2} : S^u_{m_1} \to R^u_{m_2}$ be a partial function given by $\mu_{m_1,m_2}(e) = e'$ iff both $(\ell_1(e), \ell_2(e')) \in \mathit{Corr}$ and $\lambda_1(e) = \lambda_2(e')$. Observe that, due to conditions 1 and 2 of Definition 1, μ_{m_1,m_2} is in fact well-defined. The *product* of m_1 and m_2, denoted by $m_1 \cdot m_2$, is defined to be the ECMSC $m = (E, \preceq, t, o, \mu, \ell, \lambda)$ where

– $E = E_1 \cup E_2$
– $\preceq = (\preceq_1 \cup \preceq_2 \cup \{(e, e') \in E_1 \times E_2 \mid o_1(e) = o_2(e')\} \cup \mu_{m_1,m_2})^*$
– $S_m = S_{m_1} \cup S_{m_2} \cup \mathit{Dom}(\mu_{m_1,m_2})$
$R_m = R_{m_1} \cup R_{m_2} \cup \mathit{Range}(\mu_{m_1,m_2})$
$\perp_m = \perp_{m_1} \cup \perp_{m_2}$
$\cup \{e \in S^u_{m_1} \backslash \mathit{Dom}(\mu_{m_1,m_2}) \mid \exists e' \in S^u_{m_2} : \ell_1(e) = \ell_2(e'),\ \lambda_1(e) = \lambda_2(e')\}$
$\cup \{e \in R^u_{m_2} \backslash \mathit{Range}(\mu_{m_1,m_2}) \mid \exists e' \in R^u_{m_1} : \ell_2(e) = \ell_1(e'),\ \lambda_2(e) = \lambda_1(e')\}$
$U_m = E \backslash (S_m \cup R_m \cup \perp_m)$
– $o = o_1 \cup o_2$,
– $\mu = \mu_1 \cup \mu_2 \cup \mu_{m_1,m_2}$
– $\ell = \ell_1 \cup \ell_2$
– $\lambda = (\lambda_1 \cup \lambda_2)_{|U_m}$

In other words, the events of m comprise the ones of m_1 and m_2, and the partial order is obtained by the reflexive and transitive closure of the orders of

m_1 and m_2, ordering the events of m_2 of a process p after the events of m_1 of process p, and the order imposed by the newly matching send and receive events, denoted by (the graph of) the function μ_{m_1,m_2}. The send and receive events of m are augmented by the events related by μ_{m_1,m_2}. The lost events are the previous ones together with the unmatched send events of type `U` in m_1 for which a subsequent event with the same label and matching information (in m_2) exists, and, dually, the unmatched receive events of type `U` in m_2 which are preceded by events with the same label and number (in m_1). All remaining events remain unmatched ones of type `U` and may be used for further concatenation.

Consider the ECMSCs shown in Figure 4 where, as before, the use of only one message $a \in \mathit{Mess}$ is assumed and a possibly unmatched message is sent to the neighbored process line. We have $m_1 \cdot m_2 = m_4$, resulting in an ECMSC with message overtaking (i.e., with equally labeled arrows which are not arranged in a FIFO manner).[4] Observe that the unmatched send and receive events numbered 2 have been matched and that the first send event numbered 1 turned into a lost one as it is followed by an equal send event of m_2 that is likewise numbered 1. Furthermore, $m_4 \cdot m_3 = m_5$, matching the events numbered 1.

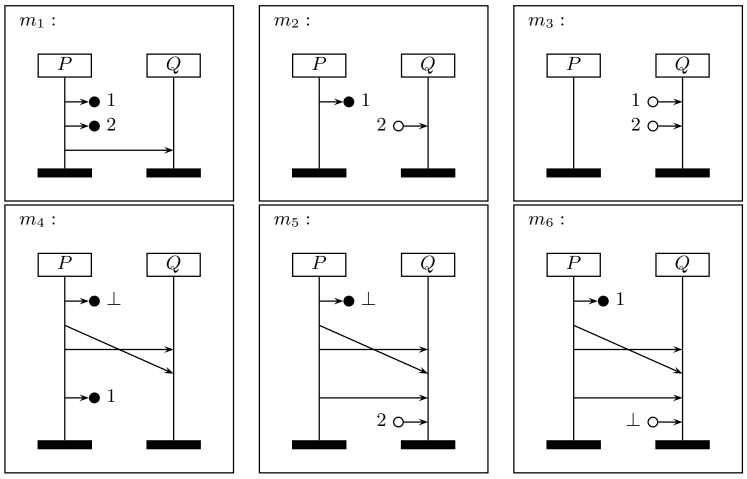

Fig. 4. Some ECMSCs

$$w_1 = \underset{1}{\mathrm{S}^Q_P(a)}\ \underset{2}{\mathrm{S}^Q_P(a)}\ \underset{3}{\mathrm{S}^Q_P(a)}\ \underset{3}{\mathrm{R}^Q_P(a)} \qquad w_2 = \underset{1}{\mathrm{S}^Q_P(a)}\ \underset{2}{\mathrm{R}^Q_P(a)}$$

$$w_1' = \underset{1}{\mathrm{S}^Q_P(a)}\ \underset{2}{\mathrm{S}^Q_P(a)}\ \underset{1}{\mathrm{S}^Q_P(a)}\ \underset{1}{\mathrm{R}^Q_P(a)} \qquad w_4 = \underset{\perp}{\mathrm{S}^Q_P(a)}\ \underset{4}{\mathrm{S}^Q_P(a)}\ \underset{3}{\mathrm{S}^Q_P(a)}\ \underset{1}{\mathrm{S}^Q_P(a)}\ \underset{3}{\mathrm{R}^Q_P(a)}\ \underset{4}{\mathrm{R}^Q_P(a)}$$

Fig. 5. Exemplary linearizations

[4] Given ECMSCs m and m', we write $m = m'$ iff m and m' are isomorphic.

We easily see that $\cdot$ is not associative. For example, while $(m_1 \cdot m_2) \cdot m_3 = m_5$, $m_1 \cdot (m_2 \cdot m_3)$ yields m_6. In the following, we let $\cdot$ associate to the left, i.e., $m_1 \cdot m_2 \cdot m_3$ denotes $(m_1 \cdot m_2) \cdot m_3$.

Another equivalence will turn out to be important, which does not distinguish between ECMSCs that only differ in the type and matching information of unmatched events: Given ECMSCs $m_i = (E_i, \preceq_i, t_i, o_i, \mu_i, \ell_i, \lambda_i)$, $i = 1, 2$, we write $m_1 \equiv m_2$ iff $(E_1, \preceq_1, t_{1|S_{m_1} \cup R_{m_1}}, o_1, \mu_1, \ell_1)$ and $(E_2, \preceq_2, t_{2|S_{m_2} \cup R_{m_2}}, o_2, \mu_2, \ell_2)$ are isomorphic. For example, referring to Figure 4, $m_5 \equiv m_6$. Furthermore, m_5 is equivalent to the ECMSC which we obtain from m_5 by replacing 2 with 3. The equivalences $=$ and $\equiv$ over ECMSCs are extended to sets as customary. In particular, $M_1 \equiv M_2$ iff $\{[m]_\equiv \mid m \in M_1\} = \{[m]_\equiv \mid m \in M_2\}$.

As we have defined the concatenation of ECMSCs, we are now ready to define ECMSGs:

Definition 2 (Extended Compositional Message Sequence Graph). *An* Extended Compositional Message Sequence Graph (ECMSG) *is a finite automaton $\mathcal{H} = (States, \Pi, \delta, s_{in}, F, \chi)$, i.e., States is a nonempty finite set of* states*, Π is a finite* alphabet*, $\delta \subseteq States \times \Pi \times States$ is the set of* transitions*, $s_{in} \in States$ is the* initial state*, and $F \subseteq States$ is the set of final states. In addition, χ maps each letter h from Π to an ECMSC $\chi(h)$.*

The word language $L(\mathcal{H}) \subseteq \Pi^*$ is defined in the expected manner. Moreover, $\mathcal{H}$ defines an ECMSC language $M(\mathcal{H}) := \{\chi(h_1 \ldots h_n) \mid h_1 \ldots h_n \in L(\mathcal{H}), h_i \in \Pi\}$ where, for $n \geq 1$, $\chi(h_1 \ldots h_n) := \chi(h_1) \cdot \ldots \cdot \chi(h_n)$, and $\chi(\varepsilon)$ is set to be the empty ECMSC. Let furthermore $\lambda(\mathcal{H}) := \lambda(Range(\chi))$. We give a large example for an ECMSG in the next section.

Note that, since FIFO matching is assumed in CMSGs, a CMSC lacks a matching information. As a CMSG in the style of [16] only traces a bounded number of unmatched events within a path—every accepting path must guarantee that the behavior is *complete*, i.e., without unmatched events—we can, involving some combinatorial considerations (cf. [5] for details), find an equivalent ECMSG for it. This justifies our notion of *extended CMSGs*.

3 A System Specification

Let us formalize scenarios for a variant of the well-known *Alternating-Bit Protocol* (ABP) [19] to exemplify how to use our framework.

The ABP was designed to guarantee a reliable data transmission through insecure channels, which may loose or duplicate a message. The general idea is that a sender process *Send* sends a message together with a control bit b to a receiver process *Receive*, which, in turn, sends the control bit back. As soon as *Send* receives the control bit b, it sends the next chunk of data, now together with $1-b$. If, however, the control bit has not been returned to the sender within a given time frame, the sender will resend the message using b, because it thinks that the data has not been delivered. The control bit, on which we concentrate in our further analysis, might be used to detect errors in the communication.

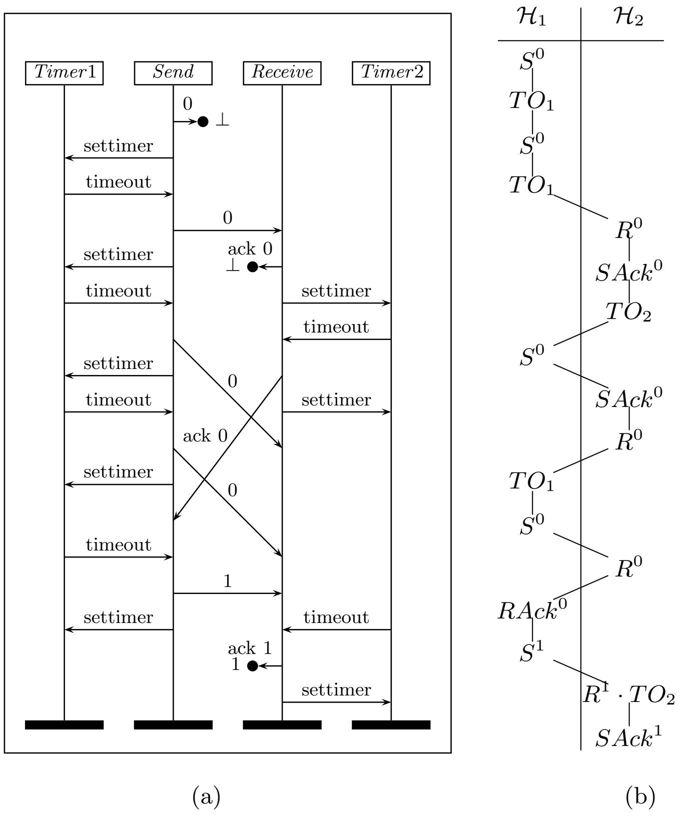

Fig. 6. A scenario of the Alternating-Bit Protocol

A typical scenario which we might expect to see, is shown in Figure 6(a). We show a sender process *Send* and a receiver process *Receive* each together with a timer process.[5] *Send* transmits a bit b towards the receiver, which is lost. Since it sets a timer, the timer expires and *Send* retransmits the data. Now, we see that the data has been transmitted but the acknowledgement is lost. As *Receive* does not get any new data, it assumes that the acknowledgement got lost and retransmits it. At the same time, *Send* retransmits the data since it did not receive an acknowledgement. *Receive* will ignore the retransmitted data since the same bit was used, indicating that we indeed had a retransmission.

Let us now consider several scenarios in a more structured way: First, a sender *Send* sends a bit b and sets a timer, which we describe in the ECMSC

[5] We will model *timers* by using separate processes because we are not interested in quantitative timing requirements. Note that we could also use timers in the sense of [13].

S^b shown in Figure 7.[6] The sending might be followed by a timeout or by an acknowledgement followed by a timeout, as shown respectively in the ECMSCs TO_1 and $RAck^b$. Thus, we may combine the scenarios as shown in Figure 7 and obtain the ECMSG $\mathcal{H}_1$. The receiver behaves dually. We identify the scenarios R^b, $SAck^b$, and TO_2, which we combine towards $\mathcal{H}_2$.

We specify the ABP simply as the parallel product $\mathcal{H}_1 \parallel \mathcal{H}_2$ of the ECMSGs $\mathcal{H}_1$ and $\mathcal{H}_2$ from Figure 7, where we define the parallel product of two ECMSGs $(States_1, \Pi, \delta_1, s^1_{in}, F_1, \chi)$ and $(States_2, \Pi, \delta_2, s^2_{in}, F_2, \chi)$ as expected by $(States_1 \times States_2, \Pi, \delta, (s^1_{in}, s^2_{in}), F_1 \times F_2, \chi)$, where δ is given by $\{((s_1, s_2), h, (s'_1, s_2)) \mid (s_1, h, s'_1) \in \delta_1, s_2 \in States_2\} \cup \{((s_1, s_2), h, (s_1, s'_2)) \mid s_1 \in States_1, (s_2, h, s'_2) \in \delta_2\}$.

A (partial) run of the product automaton is shown in Figure 6(b), whose sequence of ECMSCs yields the ECMSC aside (Figure 6(a)). Thus, the ECMSC of Figure 6(a) is a prefix of an ECMSC from $M(\mathcal{H}_1 \parallel \mathcal{H}_2)$.

Anticipating the developments to come, let us mention that we now might analyze our scenarios with respect to MSO formulas. For example, we might ask whether we can derive scenarios in which we have unmatched receive events (white holes) $(\exists y \bigvee_{p,q,a} (L_{\mathrm{R}^q_p(a)}(y) \wedge \neg \exists x (x \rightarrow y)))$. If we are faced with a setting that our channel will never produce white holes, we can simply analyze a formula φ by model checking our protocol against the formula $(\forall y \exists x ((\bigvee_{p,q,a} L_{\mathrm{R}^q_p(a)}(y)) \Rightarrow x \rightarrow y)) \Rightarrow \varphi$, i.e., ruling out those ECMSCs that do not employ an unmatched receive event.

4 ECMSCs and Regular Languages

In this section, we provide notions to handle ECMSCs by means of *linearizations.* Our goal is to achieve a characterization as in [16], that is, the language of an ECMSG can be *represented* by a regular word language. This characterization will be the basis for the model-checking procedure. The main idea of our approach is to use linearizations which are indexed by natural numbers.

A word $w = \genfrac{}{}{0pt}{}{\sigma_1}{u_1} \cdots \genfrac{}{}{0pt}{}{\sigma_n}{u_n} \in (\Sigma \times (\mathbb{N} \cup \{\perp\}))^*$ is called *ECMSC word.* Its positions can be considered to be events of an ECMSC, and together with its letters, we can obtain the matching information (cf. Figures 4 and 5). More specifically, w defines an ECMSC $m_w := (E, \preceq, t, o, \mu, \ell, \lambda)$ given as follows:

- $E = \{1, \ldots, n\}$
- $\preceq$ is the reflexive and transitive closure of $\sqsubset$ where $i \sqsubset j$ iff $i < j$ and
  - $o(\sigma_i) = o(\sigma_j)$ or
  - $(\sigma_i, \sigma_j) \in Corr$ and $u_i = u_j \neq \perp$
- $S_{m_w} = \{i \mid \sigma_i \in \Sigma_{\mathcal{S}}$ and
  $\exists j > i :\ (\sigma_i, \sigma_j) \in Corr,\ u_i = u_j \neq \perp,$
  and $(\sigma_k, u_k) \neq (\sigma_i, u_i)$ for all k with $i < k < j\}$

[6] Note that we do not show the second timer process in scenarios for the sender and the first timer process in scenarios involving the receiver for lack of space. Furthermore, note that S^b represents two scenarios: S^0 and S^1.

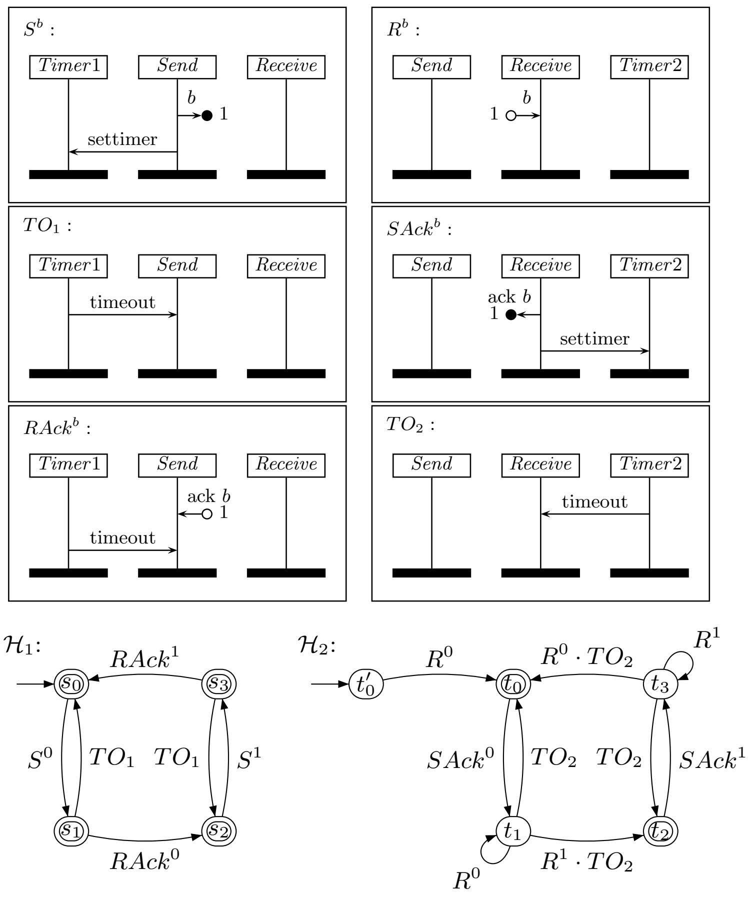

Fig. 7. The Alternating-Bit Protocol

$$
\begin{aligned}
R_{m_w} = \{i \mid\ & \sigma_i \in \Sigma_{\mathcal{R}} \text{ and} \\
& \exists j < i : (\sigma_j, \sigma_i) \in \mathit{Corr},\ u_i = u_j \neq \bot, \\
& \qquad \text{and } (\sigma_k, u_k) \neq (\sigma_i, u_i) \text{ for all } k \text{ with } j<k<i\}
\end{aligned}
$$

$$
\begin{aligned}
U_{m_w} = \quad & \{i \mid \sigma_i \in \Sigma_{\mathcal{S}},\ u_i \neq \bot, \text{ and } \nexists j > i : (\sigma_i, u_i) = (\sigma_j, u_j) \text{ or} \\
& \qquad\qquad [(\sigma_i, \sigma_j) \in \mathit{Corr} \text{ and } u_i = u_j]\} \\
\cup\ & \{i \mid \sigma_i \in \Sigma_{\mathcal{R}},\ u_i \neq \bot, \text{ and } \nexists j < i : (\sigma_i, u_i) = (\sigma_j, u_j) \text{ or} \\
& \qquad\qquad [(\sigma_j, \sigma_i) \in \mathit{Corr} \text{ and } u_i = u_j]\}
\end{aligned}
$$

$\bot_{m_w} = E \setminus (S_{m_w} \cup R_{m_w} \cup U_{m_w})$

- $o(i) = o(\sigma_i)$
- $\mu(i) = \min\{j \mid i < j \text{ and } (\sigma_i, \sigma_j) \in \mathit{Corr} \text{ and } u_i = u_j\}$
- $\ell(i) = \sigma_i$
- $\lambda(i) = u_i$ (for i with $t(i) = \mathtt{U}$)

The set of ECMSC words (recall that Σ is fixed) is denoted by $\mathcal{W}$. We call $w = \frac{\sigma_1}{u_1} \cdots \frac{\sigma_n}{u_n} \in \mathcal{W}$ a *linearization* of an ECMSC m iff $m_w = m$. Let $\mathit{Lin}(m)$

denote the set of linearizations of m. Furthermore, we will make use of $\overline{\lambda}(w) := \{u_i \mid i \in S_{m_w}\}$ describing the set of numbers used in w for matched events (positions).

To give some examples, look at Figures 4 and 5. While w_1 is a linearization of m_1, w'_1 is not [7]. In fact, any ECMSC is sufficiently described by a single linearization. The word w_2 is one of two possible linearizations of m_2, and w_4 is a linearization of m_4. Observe that $m_{w_1 w_2} = m_1 \cdot m_2 = m_4$. Generally, it holds:

Proposition 1. *For a nonempty finite set $M = \{m_1, \ldots, m_k\}$ of ECMSCs and linearizations $w_i \in Lin(m_i)$ with $\overline{\lambda}(w_i) \cap \lambda(M) = \emptyset$, $i \in \{1, \ldots, k\}$, we have $m_{w_1 \ldots w_k} \equiv m_1 \cdot \ldots \cdot m_k$.*

An important result of [16] also holds in our setting, saying—informally—that languages of ECMSGs can be represented by regular word languages. We call a language $L \subseteq \mathcal{W}$ *regular* iff there is a $B \in \mathbb{N}$ such that $L \subseteq (\Sigma \times (\{1, \ldots, B\} \cup \{\bot\}))^*$ and L is regular in the usual sense.

Theorem 1. *Let M be a set of ECMSCs.*

1. *If M is the language of an ECMSG, then there is a regular language $L \subseteq \mathcal{W}$ with $\{m_w \mid w \in L\} \equiv M$.*
2. *If there is a regular language $L \subseteq \mathcal{W}$ with $\{m_w \mid w \in L\} = M$, then M is $\equiv$-equivalent to the language of an ECMSG.*

Proof. 1. We first extend the usual notion of finite automata towards using words instead of letters in the transition function, i.e., in the following, we deal with a structure $\mathcal{A} = (States, \Pi, \delta, s_{in}, F)$ such that δ is a finite subset of $States \times \Pi^* \times States$. This is just for notational convenience and does not alter the expressive power of finite automata.

For an ECMSG $\mathcal{H} = (States, \Pi, \delta, s_{in}, F, \chi)$, we can build an (extended) finite automaton $\mathcal{A}_{\mathcal{H}} = (States', \Sigma \times (\{1, \ldots, B\} \cup \{\bot\}), \delta', s'_{in}, F')$ (with B a natural) such that $\{m_w \mid w \in L(\mathcal{A}_{\mathcal{H}})\} \equiv M(\mathcal{H})$ as follows: Initially, for each ECMSC $m \in Range(\chi)$, we pick a linearization $w_m \in Lin(m)$ with $\overline{\lambda}(w_m) \cap \lambda(\mathcal{H}) = \emptyset$. (As a general rule, for each ECMSC, one would determine a *canonical* linearization, for example, in the sense of [6].) $\mathcal{A}_{\mathcal{H}}$ is then given as follows: B is the maximal index used in a linearization. Furthermore, $States' = States$, $s'_{in} = s_{in}$, $F' = F$, and $(q, w, q') \in \delta'$ iff there is $h \in \Pi$ such that both $(q, h, q') \in \delta$ and $w = w_{\chi(h)}$. The language equivalence (with respect to $\equiv$) follows from Proposition 1.
2. We first assign to a symbol $(\sigma, u) \in \Sigma \times (\mathbb{N} \cup \{\bot\})$ the ECMSC $m_{(\sigma,u)} := (E, \preceq, t, o, \mu, \ell, \lambda)$ defined by $E = \{\mathsf{e}\}$, $\preceq = \{(\mathsf{e}, \mathsf{e})\}$, $t(\mathsf{e}) = \mathtt{U}$ if $u \in \mathbb{N}$, $t(\mathsf{e}) = \bot$ otherwise, $o(\mathsf{e}) = o(\sigma)$, $\mu = \emptyset$, $\ell(\mathsf{e}) = \sigma$, and, if $t(\mathsf{e}) = \mathtt{U}$, $\lambda(\mathsf{e}) = u$. I.e., $m_{(\sigma,u)}$ consists of exactly one unmatched event of type $\mathtt{U}$ or type $\bot$. (Note that, in fact, this definition is according to the ECMSC we obtain if (σ, u) is considered as a word.) Given a (usual) finite automaton $\mathcal{A} = (States, \Sigma \times (\{1, \ldots, B\} \cup \{\bot\}), \delta, s_{in}, F)$ whose language is L, the ECMSG $\mathcal{H}_L$ satisfying $M(\mathcal{H}_L) \equiv \{m_w \mid w \in L\}$ is given by $(States, \Sigma \times (\{1, \ldots, B\} \cup \{\bot\}), \delta, s_{in}, F, \chi)$ where $\chi((\sigma, u)) = m_{(\sigma,u)}$. $\square$

[7] unless 1 is replaced by $\bot$ in m_1

5 Model Checking for ECMSGs

We have now set out the scene to consider model checking for ECMSGs.

5.1 MSO Model Checking

Following [16], we give a decision procedure for a monadic second-order logic interpreted over ECMSCs, using the concept of a regular representative linearization.

Given a supply $\mathrm{Var} = \{x, y, \ldots\}$ of individual variables, which are interpreted over events of an ECMSC, and a supply $\mathrm{VAR} = \{X, Y, \ldots\}$ of set variables, which are interpreted over sets of events, the syntax of $\mathrm{MSO}(\mathcal{P}, \mathit{Mess})$ is defined by the following grammar:

$$\varphi ::= L_\sigma(x) \mid x \rightarrow y \mid x \in X \mid x \preceq y \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

where $\sigma \in \Sigma$, $x, y \in \mathrm{Var}$, and $X \in \mathrm{VAR}$. Moreover, we allow the usual abbreviations. The intuitive meaning of $x \rightarrow y$ is that x is a send event and y its corresponding receive event. $x \preceq y$ holds iff y follows x in the order of the ECMSC at hand. We formalize the intuitive study: Let $m = (E, \preceq, t, o, \mu, \ell, \lambda)$ be an ECMSC. Given an interpretation function $\mathcal{I}$, which assigns to an individual variable x an event $\mathcal{I}(x) \in E$ and to a set variable X a set of events $\mathcal{I}(X) \subseteq E$, the satisfaction relation $m \models_{\mathcal{I}} \varphi$ for a formula $\varphi \in \mathrm{MSO}(\mathcal{P}, \mathit{Mess})$ is inductively defined as follows:

- $m \models_{\mathcal{I}} L_\sigma(x)$ iff $\ell(\mathcal{I}(x)) = \sigma$
- $m \models_{\mathcal{I}} x \rightarrow y$ iff $\mathcal{I}(x) \in S_m$, $\mathcal{I}(y) \in R_m$, and $\mu(\mathcal{I}(x)) = \mathcal{I}(y)$

The remaining constructs are as usual.

We only consider formulas without free variables in the following and accordingly write $m \models \varphi$ instead of $m \models_{\mathcal{I}} \varphi$. For $\varphi \in \mathrm{MSO}(\mathcal{P}, \mathit{Mess})$, let $M_\varphi := \{m \mid m \models \varphi\}$. Note that M_φ is closed with respect to $\equiv$, i.e., for any ECMSCs m and m' with $m \equiv m'$, it holds $m \models \varphi$ iff $m' \models \varphi$.

Theorem 2. *Given $\varphi \in MSO(\mathcal{P}, Mess)$ and an ECMSG $\mathcal{H}$, we can decide whether $M(\mathcal{H}) \subseteq M_\varphi$.*

Proof. Let $\mathcal{A}_\mathcal{H}$ be the automaton of $\mathcal{H}$ from the first part of the proof of Theorem 1 with $L(\mathcal{A}_\mathcal{H}) \subseteq (\Sigma \times (\{1, \ldots, B\} \cup \{\bot\}))^* =: \mathcal{W}_B$. From φ, we inductively build an MSO formula $\|\varphi\|$ interpreted over elements of $\mathcal{W}_B$ such that $\{w \in \mathcal{W}_B \mid w \models \|\varphi\|\} = \{w \in \mathcal{W}_B \mid m_w \models \varphi\} =: L^B_\varphi$. The fact that $M(\mathcal{H}) \subseteq M_\varphi$ iff $L(\mathcal{A}_\mathcal{H}) \subseteq L^B_\varphi$ then leads to a decision procedure in the obvious manner.

Let us determine $\|\varphi\|$ inductively as follows: For $L_\sigma(x)$, we have to guess the right number, thus, we define $\|L_\sigma(x)\| := \bigvee_{u \in \{1,\ldots,B,\bot\}} L_{(\sigma,u)}(x)$. Element relation, negation, disjunction and existential quantification carry through, i.e., $\|x \in X\| := x \in X$, $\|\neg\psi\| := \neg\|\psi\|$, $\|\psi_1 \vee \psi_2\| := \|\psi_1\| \vee \|\psi_2\|$, $\|\exists x\psi\| := \exists x\|\psi\|$, and $\|\exists X\psi\| := \exists X\|\psi\|$. x and y are corresponding send and receive events if y follows

x, their labelings are correlated, and there is no correlated element inbetween. So we define $\|x \to y\| := \searrow(x, y)$ where the matching predicate is given by

$$\searrow(x,y) := \bigvee_{\substack{(\sigma,\tau)\in Corr\\ u\in\{1,\dots,B\}}} [x<y \wedge L_{(\sigma,u)}(x) \wedge L_{(\tau,u)}(y) \wedge \nexists z(x<z<y \wedge (L_{(\sigma,u)}(z) \vee L_{(\tau,u)}(z)))]$$

Finally, y follows x causally if there is a corresponding sequence of matching send and receive events or events on the same process line. Thus,

$$\begin{aligned}\|x \preceq y\| := \exists X[\, & x \in X \wedge y \in X \wedge \\ & \forall z(z \in X \wedge z \neq y \Rightarrow \exists z'(\, z' \in X \wedge z < z' \wedge \\ & \qquad\qquad Proc(z) = Proc(z') \vee \searrow(z, z')))]\end{aligned}$$

and $Proc(x)$ is defined as expected. □

5.2 Temporal Logics for ECMSCs

While MSO is a powerful and useful specification logic on its own, the result of the previous subsection provides simple decidability results for further (temporal) logics by encoding. For example, *TLC* [21] can easily be encoded into MSO($\mathcal{P}$, *Mess*) and model checking can be shown to be decidable in this way. However, this approach does not seem to be reasonable for practical issues, due to the non-elementary complexity of model checking MSO formulas. In the domain of Mazurkiewicz traces [8], several temporal logics have been studied and sophisticated model-checking procedures were developed.

These logics usually define *trace-closed* languages. Therefore, as pointed out by Madhusudan and Meenakshi [16], it is desirable to find the existence of temporal logics interpreted over ECMSC words that are trace-closed in some sense, i.e., whose formulas are satisfied by either all linearizations of an ECMSC or none of them. First attempts in this regard were done in [18] and [4].

We will now establish a strong connection between ECMSCs and the theory of Mazurkiewicz traces, which subsequently yields trace-closed logics as well as corresponding decision procedures in a natural manner. With respect to Σ, the *dependence relation* $D(\Sigma) \subseteq (\Sigma \times (\mathbb{N} \cup \{\bot\}))^2$ is given by $(\sigma, u)D(\Sigma)(\sigma', u')$ iff

- $o(\sigma) = o(\sigma')$ or
- $(\sigma, \sigma') \in Corr$ and $u = u' \neq \bot$ or
- $(\sigma', \sigma) \in Corr$ and $u = u' \neq \bot$.

For each natural B, $\tilde{\Sigma} := (\Sigma \times \{1, \dots, B, \bot\}, D(\Sigma) \cap (\Sigma \times \{1, \dots, B, \bot\})^2)$ turns out to be a *Mazurkiewicz trace alphabet* [8]. Kuske and Morin investigate relations like this in detail [14, 20].

A relation $\sim_{\tilde{\Sigma}} \subseteq \mathcal{W}_B \times \mathcal{W}_B$ provides information about which ECMSC words are seen to be equivalent with respect to $D(\Sigma)$. So let $\sim_{\tilde{\Sigma}}$ be the least equivalence relation satisfying the following: If $w = w_1(\sigma, u)(\sigma', u')w_2$ and $w' = w_1(\sigma', u')(\sigma, u)w_2$ for suitable w_1, w_2 and not $(\sigma, u)D(\Sigma)(\sigma', u')$, then $w \sim_{\tilde{\Sigma}} w'$.

Thus, within an equivalence class of $\sim_{\tilde{\Sigma}}$, we are allowed to permute two neighbored positions in an ECMSC word that are labeled with independent actions (actions which are not dependent).

The above connection between ECMSCs and Mazurkiewicz traces is useful due to the following fact: For each ECMSC m, $Lin(m) \cap \mathcal{W}_B$ is the finite union of equivalence classes of $\sim_{\tilde{\Sigma}} \cap (\mathcal{W}_B \times \mathcal{W}_B)$. It is then effortlessly possible to suit temporal logics for Mazurkiewicz traces like TrPTL [22] or LTrL [23] to ECMSCs and to employ corresponding decision procedures, for example in the style of [3].

6 Conclusion

In this paper, we presented a formal definition of extended compositional message sequence charts (ECMSCs) with *black* and *white holes*, which supports methods of formal analysis. Furthermore, we allow message overtaking as defined in the MSC standard. The ECMSCs can be combined by means of choice and repetition towards (extended) compositional message sequence graphs (ECMSGs). Giving scenarios and ECMSGs for the Alternating-Bit Protocol, we validated that our formalism is useful for practical applications.

Despite its extended expressive power, we have shown that model checking monadic second-order formulas is decidable. Similar as in [16], our decision procedure uses the idea to *represent* the language of an ECMSG by a regular language. However, the standard encoding of MSCs by words as used in [1, 16, 9] cannot be applied. We introduce *indexed* linearizations to obtain the result that languages of ECMSGs indeed can be represented by regular word languages.

These indexed linearizations give furthermore a link between MSCs and Mazurkiewicz traces in a natural way, as requested by [16]. This link is the cornerstone for easily carrying over the sophisticated work on temporal logics for Mazurkiewicz traces to the domain of MSCs.

References

1. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of the 10th International Conference on Concurrency Theory*, LNCS1664, pp. 114–129. Springer, 1999.
2. J. Araújo. Formalizing sequence diagrams. In L. Andrade, A. Moreira, A. Deshpande, and S. Kent, editors, *Proc. of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, volume 33, 10 of *ACM SIGPLAN Notices*, New York, 1998. ACM Press.
3. B. Bollig and M. Leucker. Deciding LTL over Mazurkiewicz traces. In C. Bettini and A. Montanari, editors, *Proc. of the Symposium on Temporal Representation and Reasoning (TIME'01)*, pp. 189–197. IEEE Computer Society Press, 2001.
4. B. Bollig and M. Leucker. Modelling, Specifying, and Verifying Message Passing Systems. In C. Bettini and A. Montanari, editors, *Proc. of the Symposium on Temporal Representation and Reasoning (TIME'01)*, pp. 240–248. IEEE Computer Society Press, 2001.
5. B. Bollig, M. Leucker, and P. Lucas. Extending compositional message sequence graphs. Technical Report MS-CIS-02-09, University of Pennsylvania, Apr. 2002.

6. B. Bollig, M. Leucker, and T. Noll. Generalised regular MSC languages. In *Proc. of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS '02)*, LNCS 2303, pp. 52–66. Springer, 2002.
7. W. Damm and D. Harel. LSC's: Breathing life into message sequence charts. Technical Report CS98-09, Weizmann Institute of Science, Apr. 1998.
8. V. Diekert and Y. Métivier. Partial commutation and traces. In G. Rozenberg and A. Salomaa, editors, *Handbook on Formal Languages*, volume III. Springer, 1997.
9. E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In T. Margaria and W. Yi, editors, *Proc. of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, LNCS 2031. Springer, Apr. 2001.
10. J. G. Henriksen, M. Mukund, K. N. Kumar, and P. S. Thiagarajan. On message sequence graphs and finitely generated regular msc languages. In *Proc. of 27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, LNCS 1853, pp. 675–686. Springer, 2000.
11. J. G. Henriksen, M. Mukund, K. N. Kumar, and P. S. Thiagarajan. Regular collections of message sequence charts. In *Proc. of 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'2000)*, LNCS 1893, pp. 405–414. Springer, 2000.
12. ITU-TS. ITU-TS Recommendation Z.120anb: Formal Semantics of Message Sequence Charts. Technical report, ITU-TS, Geneva, 1998.
13. ITU-TS. ITU-TS Recommendation Z.120: Message Sequence Chart 1999 (MSC99). Technical report, ITU-TS, Geneva, 1999.
14. D. Kuske. Another step towards a theory of regular MSC languages. In *Proc. of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, LNCS 2285. Springer, 2002.
15. P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proc. of 27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, LNCS 2076. Springer, 2001.
16. P. Madhusudan and B. Meenakshi. Beyond message sequence graphs. In *Proc. of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 2245, pp. 256–267. Springer, 2001.
17. S. Mauw and M. A. Reniers. High-level message sequence charts. In *Proc. of the Eighth SDL Forum (SDL'97)*, pp. 291–306, 1997.
18. B. Meenakshi and R. Ramanujam. Reasoning about message passing in finite state environments. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Proc. of 27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, LNCS 1853. Springer, 2000.
19. R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.
20. R. Morin. Recognizable sets of message sequence charts. In *Proc. of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02), 2002*, LNCS 2285. Springer, 2002.
21. D. Peled. Specification and verification of message sequence charts. In *Proc. IFIP FORTE/PSTV*, pp. 139–154, 2000.
22. P. S. Thiagarajan. A trace consistent subset of PTL. In I. Lee and S. A. Smolka, editors, *Proc. of the 6th International Conference on Concurrency Theory (CONCUR'95)*, LNCS 962, pp. 438–452, Philadelphia, Pennsylvania, 1995. Springer.
23. P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. In *Proc. 12th Annual IEEE Symposium on Logic in Computer Science*, Warsaw, Poland, 1997. IEEE Computer Society Press.