Specifying and Verifying Partial Order Properties using Template MSCs

Blaise Genest[1], Marius Minea[2], Anca Muscholl[1], and Doron Peled[3]

[1] LIAFA, Université Paris VII & CNRS
2, pl. Jussieu, case 7014 75251 Paris cedex 05, France
[2] Department of Computers "Politehnica" University of Timisoara
Bd. V. Pârvan nr. 2, RO-1900 Timisoara, Romania
[3] Department of Computer Science The University of Warwick
Coventry, CV4 7AL United Kingdom

Abstract. *Message sequence charts* (MSC) are a graphical language for the description of communication scenarios between asynchronous processes. Our starting point is to model systems using an assume-guarantee formalism, in the style of LSCs and Triggered MSCs. We enrich MSCs with the possibility of using gaps (template MSC), and show their expressivity. This formalism also allows to express logical formulas. We analyze the model-checking problem, whose complexity is linear in the size of the system, and ranges from PTIME to EXPSPACE in the size of the template formula.

1 Introduction

Concurrent systems are intricate and hence difficult to describe. The classical description, stemming from programming practices, is based on listing the different concurrent participants, e.g., the processes. The Message Sequence Charts (MSC) formalism allows an alternative "sequential" description of a concurrent system, where the complete behavior of all the processes involved in some given task are depicted in a visual way. The language enjoys widespread use in the specification of telecommunication protocols and has been standardized by the ITU-T [1]. In a single MSC we can describe the behavior of all the processes involved, including the local actions and the messages exchanged between them. Such a slicing of the concurrent execution provides further intuition about the behavior of the system. One of the drawbacks of this representation is that tasks are seldom executed in a sequential way, and some overlap commonly exists.

In this paper we study an MSC-related formalism that allows expressing non-contiguous tasks. This is done by adding gaps to the MSC formalism. Intended for the analysis of systems, we present the formalism and study related verification problems. We are influenced in our proposal by Live Sequence Charts [4] and Triggered MSCs [13], and include an assume-guarantee mechanism, i.e., being able to require the execution of a task provided that another task was executed.

While an individual MSC has a formally defined semantics, its relation to the system behavior is left open by the standard: the usual interpretation is that the scenario should be *possible* in the implementation. In defining *Live Sequence Charts*, Damm and Harel [4] extensively emphasize the duality of mandatory and provisional semantics, but with a much wider set of features, including abort/exit conditions and reliable or lossy transmission. The provisional semantics is used with the standardized High-Level MSCs (HMSCs for short), that are described by (hierarchical) graphs with nodes labeled by MSCs [1]. The semantics of an HMSC is the set of MSCs formed by concatenating (process by process) MSCs seen along a path. HMSCs have several drawbacks, such as the difficulty to express concurrency between two independent threads, due to the sequential control of the graph. The result is that many systems are hard to model using HMSCs. To address this problem, other kinds of specifications have been proposed, e.g. based on Petri nets with transitions labeled by MSCs, [10].

A totally different approach is taken by Triggered MSCs [13]. They replace the sequential description of HMSCs by an assume-guarantee formalism (that also exists in LSCs in form of activation messages). Causality is expressed by structuring a specification with two components: a precondition that identifies the initial behavior, and a postcondition expressing the continuation supposed to be guaranteed under this assumption. Assume-guarantee combined with the parallel operator emphasizes compositionality: a system description is most easily obtained combining MSCs for collections of directly interacting processes, and superimposing assume-guarantee patterns that further constrain interactions between individual scenarios.

We are inspired by the Triggered MSCs notation. Our suggestion attempts to improve on several points, for example, making the use of infinite assume-guarantee easier to understand. Our main contribution is to define template MSCs, and use them in the Triggered MSCs setting. We achieve conciseness by specifying only events strictly needed to identify a scenario and by using *gaps* as placeholders for other messages. With gaps, parallel composition can be simply expressed as conjunction, without the need for parallel (shuffle) operators. Using assume-guarantee template MSCs we can easily specify loops and thus infinite specifications.

A second important use of assume-guarantee template MSCs is the ability to easily specify properties that a system should satisfy (the system is given here as a set of FSMs communicating through (existentially) bounded FIFO message queues, or as an HMSC). We can express temporal properties, e.g. the fact that whenever A happens, B should eventually follow. Compared to temporal logics, MSCs have the advantage of being a visual formalism, and thus easier to use in a design and engineering environment. Moreover, template MSC formulas are a fragment of a partial-order global logics with filter, whose complexity would be much higher. We study the complexity of verifying temporal properties expressed by various classes of templates and show that it ranges from PTime to ExpSpace in the size of the formula, and is linear-time in the size of the system.

One of the main differences between template MSCs and LSCs, Triggered MSCs is the introduction of gaps inside the MSC notation, in order to express an arbitrary (but finite) amount of communication or events. The user can draw single send/receive events, with the matching event being located in a gap. Another difference is that we are using template MSCs as a specification formalism, as a visual alternative to temporal logic specification. Our specification is partial-order based, related to logics such as LTrL [15, 14] and TLC [3].

A variant of model-checking for MSCs and HMSCs is described in [12]. This uses an alternative semantics that can be seen as adding gaps between each pair of events on each process. It allows combating the undecidability of HMSC intersection. The approach in the current paper is different. Gaps are added *in the specification*, and their locations and types need to be explicitly specified. For the full version see http://www.crans.org/~genest/fossacs03_full_paper.ps

2 Message Sequence Charts and Templates

Message Sequence Charts (MSC for short) is a scenario language standardized by the ITU ([1]). They represent simple diagrams depicting the activity and communications in a distributed system. The entities participating in the interactions are called instances (or processes) and are represented by vertical lines. Message exchanges are depicted by arrows from the sender to the receiver. In addition to messages, atomic actions can also be represented.

The left part of Figure 1 gives an example of an MSC M modeling two messages sent between a *Writer* W and a *Server* S.

Definition 1 *An MSC is a tuple* $M = \langle \mathcal{P}, E, \mathcal{A}, \ell, m, < \rangle$ *where:*
- $\mathcal{P}$ *is a finite set of processes,*
- E *is a finite set of events,*
- $\mathcal{A}$ *is a finite set of names for messages and local actions,*
- $\ell : E \to \mathcal{T} = \{p!q(a), p?q(a), p(a) \mid p \neq q \in \mathcal{P}, a \in \mathcal{A}\}$ *labels an event with its* type*: in process* p*, either a send* $p!q(a)$ *of message* a *to process* q *(respectively, a receive* $p?q(a)$ *of message* a *from process* q*) or a local event* $p(a)$*. The labeling* ℓ *partitions the set of events by type (send, receive, or local),* $E = S \dot{\bigcup} R \dot{\bigcup} L$*, and by process,* $E = \dot{\bigcup}_{p \in \mathcal{P}} E_p$*.*
- $m : S \to R$ *is a bijection matching each send to the corresponding receive. If* $m(s) = r$*, then* $\ell(s) = p!q(a)$ *and* $\ell(r) = p?q(a)$ *for some* $p, q \in \mathcal{P}$ *and* $a \in \mathcal{A}$*.*
- $< \subseteq E \times E$ *is the least relation between events such that:*
  - *for any process* $p \in \mathcal{P}$*, the restriction of* $<$ *to* E_p *is a total order, and*
  - *for any* $s, r \in E$*,* $m(s) = r$ *implies* $s < r$

Moreover, the relation $<$ *is required to be acyclic.*

The event labeling ℓ implicitly defines the process $pr(e)$ for each event e: $pr(e) = p$ if $e \in E_p$, i.e., $\ell(e) \in \{p!q(a), p?q(a), p(a)\}$ for some $q \in \mathcal{P}, a \in \mathcal{A}$. We assume that channels are FIFO, i.e., there is no overtaking on messages sent on the same channel.

The relation $<$ is called the *visual* order on the MSC, since it corresponds to its graphical representation. It is comprised of the process ordering and the message ordering, pairwise between send and matching receive. Since $<$ is required to be acyclic, its reflexive-transitive closure $<^*$ is a partial order on the set E of events, which we will denote by $\leq$. An extension of $\leq$ to a total order is called a linearization of M. We denote by $\mathrm{Lin}(M)$ the set of all labeled linearizations of an MSC M, $\mathrm{Lin}(M) = \{\ell(e_1)\cdots\ell(e_n) \mid e_1\cdots e_n \text{ is a linearization of } M\}$.

The main objective in this paper is to model complex communication interactions using finite scenarios. We argue that HMSCs are hard to use in specification, since a designer must be able to describe the global behavior of a protocol in form of a graph. In general, finite MSCs are easier to apply, since they capture the essence of scenarios. To describe one particular aspect of the system behavior, it is frequently not needed to consider all message exchanges or indeed not even all processes. Thus, we propose to use *templates*, which are diagrams in which events/messages can alternate with *gaps*. Gaps are effectively placeholders for arbitrary many (eventually zero) events or messages between designated processes. They can be instantiated by *compositional* MSCs (CMSC for short), that is an extension of MSCs [7]. The difference between a CMSC and an MSC is that the message function m is a partial function, i.e., there can be sends or receives for which no matching event is defined. Hence by instantiating there can be messages exchanged between different gaps or messages composed by an event of the diagram and an event in a gap. The right part of Figure 3 depicts a template MSC that describes the set of all CMSCs containing a given message.

Definition 2 *(Template MSCs) A template MSC is a tuple $\langle \mathcal{P}, E, \Gamma, \mathcal{A}, \ell, m, < \rangle$, where $\langle \mathcal{P}, E, \mathcal{A}, \ell, m, < \rangle$ is a CMSC, with the following components extended:*
- *Γ is a finite set of gap markers,*
- *$\ell : E \cup \Gamma \to \mathcal{T} \cup 2^{\mathcal{T}}$, with $\ell(\gamma) \subseteq \mathcal{T}$ the message types allowed in gap γ. Let also $\Gamma_p \subseteq \Gamma$ be the set of gaps γ such that $\ell(\gamma)$ allows events on process p.*
- *$\leq\subseteq (E \cup \Gamma)^2$. For each process $p \in \mathcal{P}$ the restriction of $<$ to $E_p \cup \Gamma_p$ must be a total order.*

The order between gaps and events in the above definition ensures that template MSCs can be effectively represented as diagrams. The semantics of a template MSC is an infinite set of CMSCs, obtained by replacing the gaps by CMSCs of allowed types.

Definition 3 *(Semantics) A template MSC $M = \langle \mathcal{P}, E, \Gamma, \mathcal{A}, \ell, m, < \rangle$ defines a set of CMSCs $\mathcal{L}(M)$. A CMSC $M' = \langle \mathcal{P}, E' = E \cup \bigcup_{\gamma\in\Gamma} E^\gamma, \mathcal{A}, \ell', m', <' \rangle$ is in $\mathcal{L}(M)$ if it is obtained by replacing each gap $\gamma \in \Gamma$ with a (possibly empty) CMSC M^γ with event set E^γ such that:*

- *The type function ℓ' is the union of ℓ and the type function of each M^γ. It is required that $\ell'(e) \in \ell(\gamma)$ for every event $e \in E^\gamma$ (M^γ contains only events of allowed types).*
- *The message function m' extends m and the message function of each M^γ. It is required that m' preserves the FIFO restriction on matched events.*
- *The visual order $<'$ is the union of $<$, the visual order of each M^γ, and the set of (e, f) with $m'(e) = f$ or one of the following:*
  - *$pr(e) = pr(f)$ and $e \in E^\gamma, f \in E^\kappa$ with $\gamma < \kappa$,*
  - *$pr(e) = pr(f)$ and $e \in E^\gamma, f \in E$ with $\gamma < f$,*
  - *$pr(e) = pr(f)$ and $e \in E, f \in E^\gamma$ with $e < \gamma$.*

Remark. Although we require an ordering between any event e and a gap γ that share some process, we do not impose the order between e and *all events* of γ. That is, we may have two unordered events e, f in M', with f belonging to some gap γ where $e < \gamma$ (however, $pr(e) \neq pr(f)$ in this case). Moreover, we may have $e < \gamma < f$ in M with $e, f \in E$, $\gamma \in \Gamma$, but by replacing γ with the empty MSC, e, f are not ordered in M'.

Note also that with the same instantiation $(M^\gamma)_{\gamma \in \Gamma}$ of gaps, we can obtain different message functions m'. This is needed since we will concatenate two such CMSC instantiations $M_1 \circ M_2$, such that the result is an MSC. Thus the message function of M_2 depends on M_1. □

Notice that we would not be able to express the set of all *MSCs* containing a given message, if there is no compositionality allowed in gaps. Assume for instance that in the example in figure 1 the gaps of N are instantiated by MSCs (and not CMSCs). Then the MSC M in the left part of the figure 1 does not belong to $\mathcal{L}(N)$.

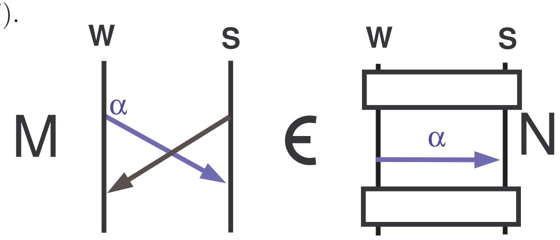


Fig. 1. Template N representing all (C)MSCs containing the message α

We define $\mathrm{Lin}(M)$ for a template MSC M as the union of the linearizations of all CMSCs from $\mathcal{L}(M)$.

Template MSCs describe only simple communication patterns. To increase their expressivity, we can use them in an assume-guarantee framework, that allows in particular to express safety and liveness-like properties (see section 4.1). For defining assume-guarantee template MSCs we first define what it means to decompose an MSC N as $N = ST$, where S, T are CMSCs. It means that there exists a linearization $x_1 \cdots x_k$ of N and some $1 \leq i \leq k$ such that $x_1 \cdots x_i$ ($x_{i+1} \cdots x_k$, resp.) is a linearization of S (T, resp.).

Definition 4 *(Assume-guarantee template MSCs) Let M_a, M_g be two template MSCs. Then $M_a \rightarrowtail M_g$ and $M_a \rightarrowtail \neg M_g$ are assume-guarantee template MSCs that define sets of MSCs $\mathcal{L}(M_a \rightarrowtail M_g), \mathcal{L}(M_a \rightarrowtail \neg M_g)$:*

- $\mathcal{L}(M_a \rightarrowtail M_g) = \{N \in$ *MSC* $\mid$ *for every decomposition* $N = ST$, *either* $S \notin \mathcal{L}(M_a)$ *or* $T \in \mathcal{L}(M_g)\}$.
- $\mathcal{L}(M_a \rightarrowtail \neg M_g) = \{N \in$ *MSC* $\mid$ *for every decomposition* $N = ST$, *either* $S \notin \mathcal{L}(M_a)$ *or* $T \notin \mathcal{L}(M_g)\}$.

For an example of assume-guarantee template MSC see Figure 4. Notice that S, T can be CMSCs, but $ST = N$ is required to be an MSC. Note also that assume-guarantee template MSCs generalize MSCs, since every MSC M can be represented as $\epsilon \rightarrowtail M$, where ϵ is the empty MSC.

A **template MSC formula** is a conjunction $\bigwedge_i (M_a^i \rightarrowtail (\bigvee_j \pm M_g^{ij}))$, where $\pm$ means that guarantee MSCs may appear in either positive or negated form. That is, for each of the individual assume-guarantee specifications of the outermost conjunction, we have preconditions in form of positive scenarios, and postconditions as disjunctions of either positive or negative (forbidden) scenarios. Hence, an MSC N belongs to $\mathcal{L}(M_a \rightarrowtail \bigvee_j \pm M_g^j)$ if for every decomposition $N = ST$, whenever $S \in \mathcal{L}(M_a)$ either $T \in \mathcal{L}(M_g^j)$ for some positive M_g^j, or $T \notin \mathcal{L}(M_g^j)$ for some negative M_g^j. This conditional description allows in particular the guarantee *false*, with $\mathcal{L}(\mathit{false}) = \emptyset$. For example, an MSC N satisfies $M \rightarrowtail \mathit{false}$ iff no prefix of N is in $\mathcal{L}(M)$. The formula $\epsilon \rightarrowtail \neg M$ describes the complement of $\mathcal{L}(M)$.

3 Modeling using template MSCs

A first application of template MSCs is for modeling protocols easier than with HMSCs, [1]. The drawback of the standard notation of HMSCs is that one needs a global (graph) description combining several scenarios, resp. behaviors of the system. Using template MSCs, we model each behavior locally, that is each scenario is described on the processes that it involves. We restrict then the combination of these local behaviors by using template formulas. In the latter step, using template MSCs allows us to focus only on the relevant messages in a scenario, and avoid both repetition and the inclusion of unrelated messages.

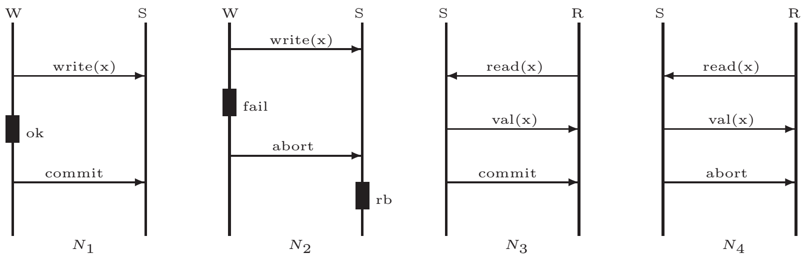


Fig. 2. Global Behavior of Writer-Server-Reader System: $(N_1 \vee N_2)^\star || (N_3 \vee N_4)^\star$

We present an example that illustrates the major features of our approach, namely the reader-writer example, taken from [13]. The system consists of three processes: a writer W and a reader R which concurrently access variables maintained by a server S. The latter has the task of maintaining atomicity and serialization of read and write operations, each of which are performed in two phases. Since triggered MSCs cannot deal easily with infinite specifications, the example from [13] involves a single read/write operation. With template formulas we do not have this problem, so we extend this example to arbitrary many write/read operations.

The writer W performs a tentative update of variable x by sending a message $w(x)$ to the server S; x is now in a "dirty" state. Then, W performs a local action ok or $fail$ which decides on the outcome of the write, and sends the corresponding message *commit* or *abort* to the server. A *commit* marks the variable as "clean". An *abort* causes the server to perform a local rollback action *rb* and potentially influences a read in progress. The reader R can send the server a request $r(x)$ for the variable x, to which the server responds with a value $val(x)$. Subsequently, the server either follows up with a *commit* message, if the sent value was clean, or has been since committed by the writer, or sends an *abort* if the sent value has to be rolled back. Although many different orders of interactions between the three processes are possible, the interaction between the pairs of directly communicating processes is simple. Our system description above contains a pair of basic scenarios for both writer-server and reader-server interaction, depicted in Figure 2.

The global behavior is a subset of that given by composing these individual scenarios. Using a notation similar to Triggered MSCs, we would write $(N_1 \vee N_2)^\star||(N_3 \vee N_4)^\star$. However, one side effect of gaps is that they make the definition of a parallel composition operator unnecessary, assuming that we compare MSC with different type sets. To express $N_1||N_3$ for instance, it suffices to extend both MSCs to all three processes, and add gaps in between all messages. The gaps in N_1 (resp. N_3) allow only events of N_3 (resp. N_1). Then, parallel composition simply becomes conjunction (language intersection): $\mathcal{L}(N_1||N_3) = \mathcal{L}(N_1) \cap \mathcal{L}(N_3)$. We need slightly more work for expressing the star of languages. First, we need an initialization step ($\epsilon \rightarrowtail M_1 \vee M_2$) meaning that every MSC in the specification should begin on W, S by a *write*, and then either *ok* and *commit*, or *fail* and *abort*. Anything can happen next, as allowed by the unrestricted gap $\gamma_\star$. The MSCs M_1, M_2 are defined in figure 3, where the gap $\gamma_\star$ has no restriction, while γ_R is restricted to events of N_3, N_4.

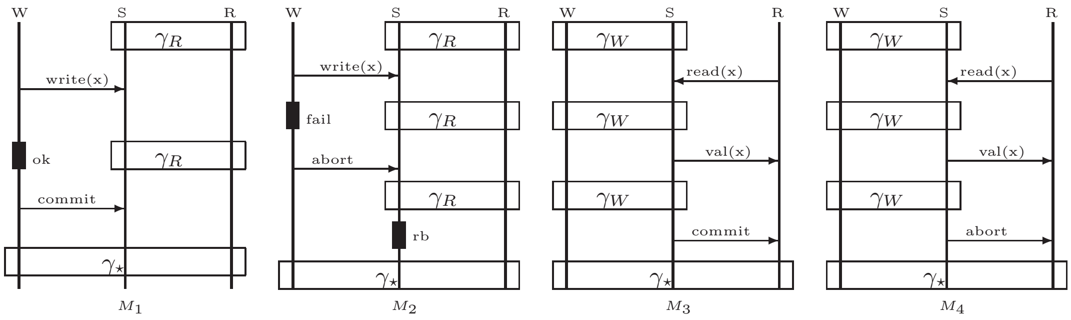
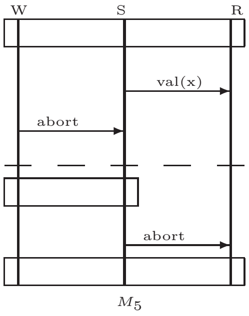


Fig. 3. Initialization: $\epsilon \rightarrowtail (M_1 \vee M_2) \wedge \epsilon \rightarrowtail (M_3 \vee M_4)$

By adding an inductive step we obtain the specification. Namely, we need that either M_1 or M_2 happens after each message *commit* (or event *rb*), or there is no more event on W, S (gap γ_R), specified as:

$$\gamma_\star commit \rightarrowtail M_1 \vee M_2 \vee \gamma_R \quad \wedge \quad \gamma_\star rb \rightarrowtail M_1 \vee M_2 \vee \gamma_R$$

The same applies for M_3, M_4. These individual scenarios are interdependent, so the global system behavior is obtained by imposing additional constraints on their composition. We divide the constraints into an *assumption* part that identifies the initial behavior in a scenario, and a *guarantee* part expressing the behavior expected of the system under this assumption.



For our WSR example we identify 5 cases specified with the constraints in Figure 4. M_5 states that if a write on the Server is followed by a send of x to the Reader, and the Writer aborts (precondition), then the Server should inform the Reader about the abort (postcondition). The occurrence of the read is guaranteed by M_3 or M_4, so it needs not be specified again. Likewise, M_1, M_2 ensure that if there is no write between a send of x and an *abort* of the Writer, then the *write* has occurred in the first gap. This precondition will imply an abort for the read (postcondition). The remainder of the interaction needs not be specified, so we allow *gaps* in between these actions, corresponding to sending and receiving other messages. The other cases correspond to a commit of the value. Namely, a value is sent while no write has been produced (M_6), or a value is sent after the last write has been roll-backed (M_7)/committed (M_8), or a commit is received immediately after the value is sent (M_9).

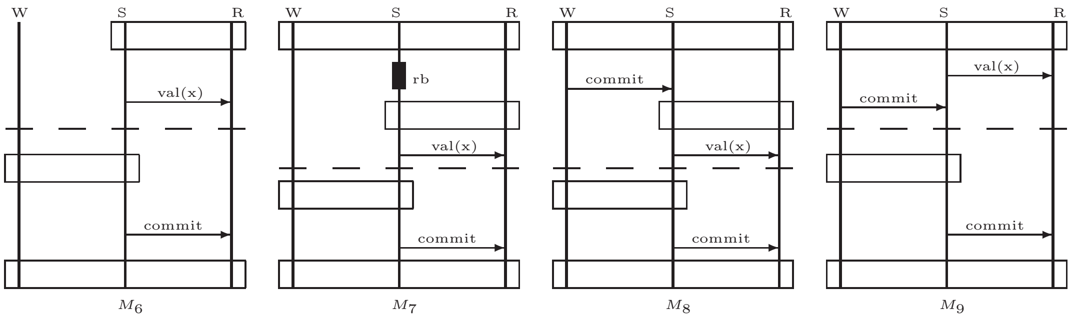


Fig. 4. Assume-Guarantee Scenarios for Writer-Server-Reader System

Hence the constraint is $M_5 \wedge M_6 \wedge M_7 \wedge M_8 \wedge M_9$. Without template MSCs, we would need to write at least every possible instantiation for gaps in our 5 cases, yielding at least 12 cases. For instance, an HMSC specifying the same model would require at least 19 states. Moreover, the size increases even more severely (exponentially) if instead of a single reader we allow several ones. With template formulas we express the constraints for each pair Writer/Reader, while an equivalent HMSC has to describe all possible combinations over all Readers. This lack of conciseness of HMSCs is a real drawback, since many algorithms involving MSCs are at least NP-hard, due to concurrency. First, HMSCs are unable to represent the parallel composition, which can lead to an exponential blow-up compared with template formulas, and to specifications that are harder to understand. Moreover, HMSCs are finitely generated, which prevents them from implementing simple protocols such as the alternating bit. Lastly, HMSCs cannot be complemented in general. Hence, since template formulas implicitly complement the assume part, they cannot be transformed into HMSCs.

4 Specifying Properties

4.1 Logical Properties

Template MSC formulas can describe easily and in a concise way some interesting properties and can be automatically verified (model-checked). We can use them for describing global properties of MSC configurations and use gaps as filters, i.e., for restricting the types of events. We denote in the examples below by γ an unrestricted gap over all processes, and $\gamma_{\neg a}$ a gap that can generate all types but a.

- $(\gamma A) \rightarrowtail \text{false} = \epsilon \rightarrowtail \neg(\gamma A \gamma)$: No execution contains the MSC A.
- $\gamma \rightarrowtail \gamma A \gamma$: Every execution contains infinitely often the MSC A.
- $\gamma A \rightarrowtail \gamma B \gamma$: Whenever A occurs, eventually B will occur.
- $(\gamma A \rightarrowtail \gamma a \gamma) \wedge [\epsilon \rightarrowtail (\gamma_{\neg a} \vee \gamma_{\neg a} A \gamma)]$: The MSC A may occur. If this is the case, then the event a must follow. Moreover, event a cannot occur before A. One can see a as an alarm event that is triggered by A.

The theorem below shows that the expressiveness of compositional gaps has a drawback, namely that the satisfiability problem for template formulas is undecidable in general. However, we can check satisfiability of a template formula S if we require that each MSC in $\mathcal{L}(S)$ has at least one linearization where communication buffers are bounded by some given value b (it is possible that other equivalent linearizations have different bounds, or no bound at all). We denote S in this case as existentially b-bounded. For instance, every HMSC (even every realizable compositional HMSC, see [7]) is existentially bounded.

Theorem 1 *1. Given a bound b and a template MSC formula S, it is decidable whether there exists an existentially b-bounded MSC in $\mathcal{L}(S)$.*
2. It is undecidable whether a template MSC formula S satisfies $\mathcal{L}(S) \neq \emptyset$.

The proof of the first statement above will follow from the results in the next section. For the second statement, we reduce from the Post correspondence problem, making use of the unbounded communication channels.

Proof.

For the undecidability part we reduce the non-emptiness problem to an extended version of Post's Correspondence Problem (PCP). Let $(v_i, w_i)_{1 \le i \le n}$ be sequences of letters. The question is to determine whether there exists a non-empty sequence of indices $(I_1, \ldots, I_k)$ such that $v_{I_1} \cdots v_{I_k} = w_{I_1} \cdots w_{I_k}$ and for all j, $|v_{I_1} \cdots v_{I_j}| \geq |w_{I_1} \cdots w_{I_j}|$.

The proof follows closely the undecidability proof of [7] for compositional HMSCs. The reduction to PCP is described within the WRL system described below. The system WRL is composed of three processes: a writer, a reader and a log. In the intended behavior, the writer sends the reader one sequence of messages among n given sequences $(V_i)_{1 \le i \le n}$. When the writer has finished sending a sequence, it sends a message m_1 to the log process, and when the reader has finished receiving, it sends a message m_2 to the log. After this, the writer can send another sequence or can stop.

The symbol γ denotes a gap on the three processes, without any type restriction (e.g. last gap of M_5 in fig 4). The system is then described by $(\epsilon \mapsto \bigvee_{1 \leq i \leq n} N_i) \wedge (M_1 \mapsto \bigvee_{0 \leq i \leq n} N_i)$, where

- $M_1 = \gamma m_1 m_2$, meaning that eventually we see m_1 ,followed immediately by m_2.
- $N_0 = \epsilon$ means stop.
- $N_i = V_i m_1 m_2 \gamma$ means that V_i should occur, and then m_1, m_2.

Using the same patterns $m_1 m_2$ in the precondition and postcondition allows to iterate this rule.

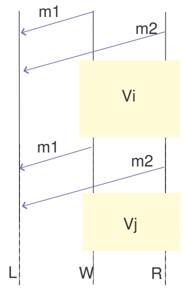


Fig. 5. MSCs generated by the WRL system

Recall that m_1 is a message from W to L, and m_2 is from R to L. We use compositional MSCs V_i, where Writer sends $|v_i|$ unmatched sends to Reader, labeled by each letter of v_i, and Reader receives $|w_i|$ unmatched sends from the Writer, labeled by each letter of w_i.

We recall that in an assume-guarantee MSC with compositional gaps, an unmatched send of a gap matches the first unmatched receive. If the matching send and receive are not labeled by the same message content, then the MSC is not legal, which ensures that $v_{I_1} \cdots v_{I_k} = w_{I_1} \cdots w_{I_k}$. Moreover, sends must precede the matching receives, which ensures that $|v_{I_1} \cdots v_{I_j}| \geq |w_{I_1} \cdots w_{I_j}|$, for all j. Then there is an MSC in the language of WRL iff there is an answer to PCP, since a legal MSC $V_{I_1} m_1 m_2 \cdots V_{I_k} m_1 m_2$ implies that $v_{I_1} \cdots v_{I_k} = w_{I_1} \cdots w_{I_k}$ and for all j, $|v_{I_1} \cdots v_{I_j}| \leq |w_{I_1} \cdots w_{I_j}|$.

However, the definition of template MSCs allows only gaps, that is we cannot specify exactly how many unmatched sends or receives will occur. We then emulate each unmatched sends with one gap, in the following way. We replace each unmatched send a by two messages m_3 from W to L, surrounding a gap allowing only type $W!R(a)$. To forbid gaps to be empty, we add the rule that two consecutive m_3 are not possible, that is $(\gamma m_3 m_3 \mapsto m_4 \gamma)$, where m_4 is a message from R to W. The message m_4 is not possible since the WRL system is not allowed to send from R to W, so two consecutive m_3 are not possible. In the same way, we can ensure that a gap stands for only one unmatched send, with the additional rule $\gamma \gamma_1 \gamma_1 \mapsto m_4 \gamma$ where γ_1 is a gap concerning only process W

with type restriction to be not a m_1 or a m_3. We can obviously do the same for unmatched receives.

□

4.2 Model-checking of Template Formulas

We consider now the problem of verifying an implementation of a communication protocol S with respect to a template MSC formula Φ. As suggested by Theorem 1, the system S needs an existential bound on buffers, denoted by b_S. This includes protocols modeled by HMSCs, communicating finite state machines with existentially bounded FIFO buffers (and even realizable compositional HMSCs, see [7]). The model for the implementation here is a finite automaton (FSM), generating linearizations of MSCs. We do *not* require that S is linearization-closed. I.e., S may generate a linearization of some MSC without generating all of them. We can obtain a linear-size FSM from any (realizable compositional) HMSC. It suffices to replace each node by a linearization of the CMSC labeling the node.

Definition 5 *For an FSM S and an assume-guarantee MSC $M_a \rightarrowtail \pm M_g$ we write $S \models (M_a \rightarrowtail \pm M_g)$ if $\mathcal{L}(S) \subseteq \mathcal{L}(M_a \rightarrowtail \pm M_g)$. The satisfaction of a template formula is defined according to the usual semantics of $\wedge, \vee$.*

In the following, we give complexity results for checking $S \models \Phi$ for various classes of template MSC formulas Φ. While S can be very large, the formula Φ and the channel bound b_S are pretty small. Hence we focus on keeping the complexity linear w.r.t. S. We will transform the formula into an automaton, so that the algorithm will be automata-based. Moreover, checking that $S \models \bigwedge_i \Phi_i$ is done for each Φ_i separately.

Proposition 1 *Given an FSM S with channel bound b_S and a template MSC M_g, we can check whether $S \models \epsilon \rightarrowtail M_g$ in space exponential in $(b_S|M_g|)$ and logarithmic in $|S|$.*

Proof. Let E be the set of events in the template MSC M_g. Let M be the MSC obtained from M_g by replacing each gap by the empty MSC. Let us fix a linearization $x = x_1 \cdots x_n$ of M. We show how to construct an NFA $\mathcal{A}_x$ accepting *every* linearization of $\mathrm{Lin}(M_g)$ whose events occur in the order given by x.

For each gap γ of M_g and each process p that is allowed in γ we use a new symbol γ^p. We first set the beginning and the end of γ on process p by choosing two positions $i \leq j$ in x and then inserting one occurrence of γ^p between each x_k, x_{k+1} for $i \leq k < j$. The choice of i, j must be consistent with the position of gap γ on process p. For instance, both the events on p before γ and the symbols κ^p with $\kappa < \gamma$ must precede x_i. Let y be a string obtained in this manner. It remains to replace each symbol γ^p by $X^*_{\gamma,p}$, with $X_{\gamma,p} \subseteq \ell(\gamma)$ the event types of γ on process p. In order to obtain all linearizations of $\mathrm{Lin}(M_g)$, between any two consecutive events x_i, x_{i+1}, the NFA will generate all possible orderings of events that preserve the sequence of gaps $X^*_{\gamma,p}$ on process p. For instance, suppose that we have $X^*_{\gamma_1,p}X^*_{\gamma_3,r}X^*_{\gamma_2,p}X^*_{\gamma_1,q}$ between two events. Then the NFA generates $(X_{\gamma_1,p} \cup X_{\gamma_3,r} \cup X_{\gamma_1,q})^*(X_{\gamma_3,r} \cup X_{\gamma_2,p} \cup X_{\gamma_1,q})^*$.

We have to ensure that messages of M are preserved. For this, we use one counter (of maximal value b_S) per message in M. Furthermore, to ensure that a message is never received before being sent, we use one counter per communication channel (of maximal value b_S). When a message m from p to q of M is sent, the counter C_m is initialized at 0. We increment it when a message is sent from p to q, and decrement when q receives from p. When the message m is received, we block the counter C_m. In the same way, we increment and decrement the counter corresponding to some channel, and block whenever it has value -1. The automaton accepts iff all counters are 0.

The resulting NFA is exponential in $b_S|M_g|$ and accepts *all* b_S-bounded linearizations of $\mathrm{Lin}(M_g)$. For a positive guarantee we can thus check $\mathcal{L}(S) \subseteq \mathrm{Lin}(M_g)$ in space exponential in $b_S, |M_g|$. For a negative guarantee we check $\mathcal{L}(S) \cap \mathrm{Lin}(M_g) = \emptyset$ in polynomial space in $b_S, |M_g|$. □

We can construct the same automaton for the precondition M_a. Then we compute in polynomial space the states of S that can be reached from an initial state by some execution corresponding to an MSC in $\mathcal{L}(M_a)$. We obtain:

Theorem 2 *Checking $S \models M_a \rightarrowtail \bigvee_i (\pm) M_g^i$ is in EXPSPACE($b_S|M_g|$).*
Checking $S \models M_a \rightarrowtail \bigvee_i \neg M_g^i$ is in PSPACE($b_S|M_g||M_a|$).

4.3 Moel-Checking in PTIME

While template specifications are quite expressive, we have seen in the previous section that model-checking is rather expensive. On the other hand, partial-order logics are in general more expensive than linear logics (e.g., LTrL [14, 15, 5] is non-elementary). For a natural fragment of LTrL that is related to our template formulas, where the until operator is replaced by the existential diamond operator, model-checking is also EXPSPACE, [3, 15]. In this section we consider a reasonable restriction of template MSCs that yields a polynomial time model-checking algorithm. Basically, we require that 1) the guarantee template has only one gap, 2) the system is linearization-complete and 3) gaps must be instantiated by simple MSCs (instead of CMSCs).

Proposition 2 *Checking $S \models \epsilon \rightarrowtail M_g$, where S is an FSM and M_g is an MSC with at most one gap can be done in time polynomial in $|M_g|$.*

Proof. For each process p, we show how to build an automaton A_p of polynomial size, recognizing linearizations where the projection on p contradicts the projection of $\mathrm{Lin}(M_g)$ onto p, denoted as $\prod_p(\mathrm{Lin}(M_g))$. We have $\prod_p(\mathrm{Lin}(M_g)) = a_1 \dots a_l X_p^* b_1 \dots b_m$ where X_p is the set of all events allowed in the unique gap of M_g on process p. For generating $\mathcal{L}(A_p) = \overline{\prod_p(\mathrm{Lin}(M_g))}$, we just need the disjunction of three regular sets, the first one testing that the number of events on p is less than $l+m$, the second one recognizing the violation of the prefix $a_1 \dots a_l$ on p, and the third one recognizing the violation of the suffix $b_1 \dots b_m$ on p. Clearly, $\mathrm{Lin}(S) \cap \mathrm{Lin}(\bigcup_p A_p) = \emptyset$ iff $S \models \epsilon \rightarrowtail M_g$. $\square$

For model-checking vs. an assume-guarantee template MSC in polynomial time we consider only *linearization-complete* systems. Recall that a system S is linearization-complete if for each execution of S that is the linearization of an MSC M, every linearization of M is an execution of S.

Proposition 3 *Given a linearization-complete system S and a template MSC M_a, we can construct a polynomial-size NFA A such that:*

1. *$\mathcal{L}(A) \cap \mathcal{L}(S) \subseteq Lin(M_a) \cap \mathcal{L}(S)$, and*
2. *for each MSC $M \in \mathcal{L}(M_a)$ with $Lin(M) \cap \mathcal{L}(S) \neq \emptyset$, A accepts at least one linearization in $Lin(M) \cap \mathcal{L}(S)$.*

In particular, the last proposition implies that we can determine in polynomial time the states of S that can be reached from an initial state by some execution corresponding to an MSC in M_a.

Proof. Consider an arbitrary linearization $x_1 x_2 \cdots x_n$ of M_a, with $x_i \in E \cup \Gamma$. Since S is complete, any MSC in $\mathcal{L}(M_a)$ that has a linearization in $\mathcal{L}(S)$ also has a linearization in $\mathcal{L}(S)$ corresponding to $x_1 \cdots x_n$. We construct A as a sequencing of NFAs for each x_i. For events, the NFA is trivial, with one transition. For gaps, we note that any event sequence in A which doesn't respect the message order is eliminated on intersection with S. Thus, it suffices to build for each gap an NFA which at each state has a transition for any event $e \in \ell(\gamma)$, augmented with a counter incremented on each send and decremented on each receive. The NFA accepts when the counter is zero, i.e. the number of total sends and receives is balanced. The intersection with S ensures acceptance of only those sequences which are balanced for each individual message. The value of the counter is bounded by the total number of outstanding messages, which is less than the size of S. Thus, A is of polynomial size. $\square$

Notice that we cannot check a template formula $M_a \rightarrowtail \bigvee_j M_g^j$ in PTIME since the non-emptiness problem of the intersection of several automata is already PSPACE-hard. Hence, we obtain the following theorem.

Theorem 3 *Checking that $S \models \bigwedge_i (M_a^i \rightarrowtail (\pm M_g^i))$ where S is a linearization-complete system, M_g has at most one gap and gaps in both M_a, M_g must be instantiated by MSCs, can be done in time polynomial in the size of M_a, M_g and S.*

4.4 Closing the gap between PTIME and EXPSPACE

The natural question arising now is to know which special cases of assume-guarantee template MSCs can be model-checked in less than exponential space. This is more than just a theoretical question, since model-checking in the general setting is very expensive, while the PTIME case given in section 4.3 can only express very simple properties.

In this section we restrict the guarantee template to have at most two gaps, which corresponds to a kind of pattern matching a finite MSC. Notice that even the complex alarm property given in section 4.1 uses just two gaps, being rather concise. The main result given in this section is that model-checking assume-guarantee template MSCs where the guarantee part has at most two gaps, can be done in PSPACE. Since the guarantee part is usually small, a polynomial space algorithm in the size of the formula and logarithmic in the size of the system is feasible in practice (remember that model checking an FSM against an LTL formula is also polynomial space in the size of the formula, but logarithmic in the size of the FSM).

Proposition 4 *Consider an FSM S (complete or not), and a template MSC M with 2 gaps. Checking that $S \models \epsilon \rightarrowtail M$ is in PSPACE$(b_S|M|)$.*

We show the result first in an easier case, namely when $M = \gamma P \gamma$, where γ is a gap on all processes without any type restriction, and P is a finite MSC (pattern). We adapt the pattern matching algorithm of [8] for Mazurkiewicz traces, improving the result stated for hierarchical MSCs in [6]. Intuitively, we do string pattern matching for each projection P_p of the pattern P on a process p and then determine occurrences that match together to an MSC pattern.

Formally, for an MSC N and a process p we denote by N_p the sequence of events of N on p. The idea is to compute for each process p the positions x_p of M_p where P_p occurs, and check that there exists some tuple of positions $(x_p)_{p \in \mathcal{P}}$ that corresponds to the pattern M. We locate a pattern P_p immediately *after* its last event. Additionally, if P has no event on process p, then we do as if there is a pattern P_p after any event of M on p.

Definition 6 *Let x_p, x_q be two occurrences of P_p and P_q, resp., in an MSC M. x_p, x_q are called compatible if there is no message (s, r) in M between p and q with $x_p < s < r < x_q$ or between q and p with $x_q < s < r < x_p$.*

Additionally, if there is a message sent by p and received by q in P, then there is no message (s, r) in M between p and q with $s < x_p$ and $x_q < r$.

The next proposition states that the compatibility relation defined above suffices for knowing whether a pattern P occurs in M or not.

Proposition 5 *A $\wp$-tuple $(x_p)_{p \in \mathcal{P}}$ is an occurrence of P in an MSC M iff x_p, x_q are compatible for all p, q.*

Proof. The implication from left to right follows easily. For the converse assume by contradiction that $(x_p)_{p \in \mathcal{P}}$ is not a pattern because for some pair of processes p, q, the sends from p to q do not match the receives of q from p.

There are several cases to consider. Either x_p ends before the send of the last message m from p to q that hits x_q, and then this message m is after x_p and before x_q, which is not possible. Or x_q ends before the receive of the last message m from p to q that was issued in x_p, and then this message m is before x_p and after x_q, which is not possible by the additional rule.

The last case where $(x_p)_{p \in \mathcal{P}}$ is not a pattern is because of some chain of messages $(s_k, r_k)_{1 \leq k \leq m}$ with $p_k = pr(s_{k+1}) = pr(r_k)$, $r_k < s_{k+1}$ for all k, and such that $p_1 = p$, $p_{m+1} = q$, $x_p < s_1$, $r_m < x_q$. Thus, there exists some k such that $x_{p_k} < s_k$ and $r_k < x_{p_{k+1}}$. But this means that $(x_{p_k}, x_{p_{k+1}})$ are not compatible, a contradiction. □

The overall idea is to generate on-the-fly every pattern x_p for all p, and to compute the last pattern which x_p can be compatible with. If some x_p is not compatible with any x_q (for at least one q), we delete it. For instance, let (s, r) be a message between p and q, and X_q be the last pattern on q before the receive r. For all patterns $x_p < s$ on process p, the pattern X_q is the last pattern which x_p can be compatible with, since any pattern $x_q > r$ on q would satisfy $x_p < s < r < x_q$. In the case where X_q does not exist, x_p cannot be matched and should be deleted.

Let $\mathcal{A}$ be the automaton corresponding to the product of deterministic automata recognizing (word) patterns on each process, using the pattern matching algorithm of Knuth-Morris-Pratt.

We build an automaton $\mathcal{B}$ based on $\mathcal{A}$ that recognizes runs (linearizations) that do not contain pattern P. The states of $\mathcal{B}$ are of the form $(a, S, \text{Pattern}, c)$, where

- a is a state of $\mathcal{A}$.
- S is the set of unmatched sends seen so far.
- Pattern $= \bigcup_p \text{Pattern}_p$, where Pattern_p is the set of patterns on process p (not deleted) seen so far.
- c is the compatibility function. For $x \in \text{Pattern}, p \in \mathcal{P}, c(x, p) \in \text{Pattern}_p \cup \{+\infty\}$ is the last pattern that x can be compatible with. It equals $+\infty$ if x can be compatible with any pattern on p.

We then build the transitions. Let e be an event. Then $(a, S, \text{Pattern}, c) \rightarrow_e (a', S', \text{Pattern}', c')$ iff $a \rightarrow_e a'$ in $\mathcal{A}$ and

1. if $e = p!q$ then Create_new_send(e)
2. if $e = p?q$, let $s \in S$ matching the receive e. Update_dependencies(s, e)
3. if $pr(e) = p$ and $\mathcal{A}$ recognizes a pattern on p, Create_new_pattern(p).

Create_new_send(e) updates S by adding e. We describe now the procedure Update_dependencies(s, e). Let X_q be the last pattern on q before e. For each pattern x_p on p before s, $c'(x_p, q) = \min(c(x_p, q), X_q)$. If there is no such pattern X_q, then $\text{Pattern}' = \text{Pattern} \setminus \{x_p\}$ for all patterns x_p on p before s. We do the other way around (where p takes the place of q) if there is a message from p to q in P. Moreover, in case we destroy some pattern x_r, let X_r be the last pattern alive on r before x_r. We update each pattern x_t such that $c(x_t, r) = x_r$ with $c'(x_t, r) = x_r$. If such x_r does not exist, we destroy each pattern x_t such that $c(x_t, r) = x_r$.

Proposition 6 *Let $\mathcal{B}$ have final states of the form $(a, S = \emptyset, Pattern, c)$, such that $Pattern_p = \emptyset$ for at least some process p. Then $M \in \mathcal{L}(\mathcal{B})$ iff P does not appear in M.*

Proof. An easy induction proves that every pattern we delete cannot be matched. Hence if $(x_p)_{p\in\mathcal{P}}$ is an occurrence of P in M, then none of x_p will be ever destroyed, hence $\mathcal{B}$ will not accept. Notice that whenever $P = (x_p)_{p\in\mathcal{P}}$ is seen, no more patterns on any process after P will be destroyed. Hence, we will either compute the first P seen in M or we will conclude that there is no such P.

We show by induction that if there is no P in M, then the algorithm will yield a process with $\text{Pattern}_p = \emptyset$. Let Pattern be the patterns not deleted after the end of the algorithm. Assume by contradiction that there is a pattern $x_p \in \text{Pattern}_p$ and choose it minimal in Pattern_p, for all processes $p \in \mathcal{P}$. Since (x_p) is not an occurrence of P in M, applying Proposition 5, there is e.g. a message (s, r) between p and q with $x_p < s < r < x_q$. Then as soon as r is received, $c(x_p, q) < x_q$ and will never increase by definition of c. Hence when $c(x_p, q) < x_q$ will be deleted (which has to happen since x_q is the first one remaining on q), x_p will be deleted as well, which is a contradiction. □

We can easily restrict Pattern_p to have only one pattern between two consecutive unmatched sends on p. Actually, all patterns with no unmatched send between them will behave the same way in the future. For any $x_q < y_q$ patterns on q with no unmatched send between them, and for all patterns x_p on p with $c(x_p, q) = x_q$, update $c'(x_p, q) = y_q$ and for all process $r \neq q, c'(x_p, r) = \min(c(x_p, r), c(x_q, r))$. We can then destroy x_q.

That is, we have to recall at most b_S unmatched sends, $b_S + \wp$ patterns with a memory of $\wp$ patterns, and the states of the automaton $\mathcal{A}$. Then $\mathcal{B}$ is at most of size $2^{O(\wp(b_S+\wp+\log(|P|))}$.

For the general case we have to consider a guarantee MSC of the form $M_g = A\gamma_T P \gamma_{T'} B$, where T, T' are sets of event types. We deal with this case by extending the automaton $\mathcal{B}$ in a suitable way, and combining with the ideas of section 4.3.

Theorem 4 *Let S be an FSM, $M_g = \bigvee_i (\pm) M_g^i$ be a guarantee template MSC with at most two gaps and let M_a be a template MSC.*

Checking $S \models M_a \mapsto \bigvee_i (\pm) M_g^i$ is in PSPACE($b_S|M_g||M_a|$).

We show now that this result is optimal:

Theorem 5 *Consider an FSM S and template MSCs M_g^i with at most two gaps. Checking that $S \models \epsilon \mapsto \bigvee_i M_g^i$ is PSPACE-hard. The result holds even if gaps are instantiated by MSCs (instead of cMSCs), the FSM is complete and has a fixed buffer bound b_S.*

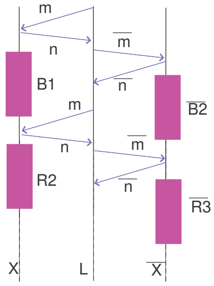
Proof. We reduce $S \models \epsilon \mapsto \bigvee_i M_g^i$ to the acceptance problem for a Turing machine T of linear space on word w. Let N be the space bound of T on w.

The proof uses classical ideas of encoding space-bounded Turing machine computations, as used e.g. in [11, 15]. In our case, we need logarithmic size counters for tape positions, since our assume-guarantee template MSCs cannot test (with polynomial size) the equality of two polynomially long strings.

Let Γ be the tape alphabet of T. We define a system S on three processes $X, L, \overline{X}$. There are local events $\Gamma_X = \Gamma \cup \{\underline{(1,b)}, \underline{(1,r)}, \ldots, (N,b), (N,r)\}$ on X and local events $\Gamma_{\overline{X}} = \overline{\Gamma} \cup \{\overline{(1,b)}, \overline{(1,r)}, \ldots, \overline{(N,b)}, \overline{(N,r)}\}$ on $\overline{X}$. Moreover, there is a message m from L to X and a message n from X to L. We define also $\overline{m}$ from L to $\overline{X}$ and $\overline{n}$ from $\overline{X}$ back to L.

The system generates $(mn(\Gamma_X \cup \Gamma_{\overline{X}})^* \overline{mn}(\Gamma_X \Gamma_{\overline{X}})^*)^*$. It is easy to generate a complete system. Notice that the system has a buffer bound of 4.

Consider a computation $\rho = \alpha_0, \alpha_1 \cdots \alpha_k$ of T, where each α_i is a configuration of size N. We encode ρ as $mn\overline{mn}B_1\overline{B_2}mn\overline{mn}R_2\overline{R_3}mn\overline{mn}B_3\overline{B_4}\cdots$ as in the figure below. Notice that $\overline{B_i}$ and R_i are concurrent.



We define $B_i = (0,b)\alpha_{i,0}(1,b)\cdots(N,b)\alpha_{i,N}$. We define R_j the same way as B_j with r replacing b. We define $\overline{B_j}$ the same way as B_j with $\overline{x}$ replacing x for all letters x.

We define Φ such that $S \models \Phi$ iff there is no accepting run of T on w. That is, we want that $\mathcal{L}(\Phi)$ is exactly the set of strings that do not encode a run, plus the non-accepting runs.

An MSC M is not an accepting run on w iff it belongs to one of the following sets

1. S_{shape} is the set of MSCs that do not have the required form for $B_i, \overline{B_i}, R_i, \overline{R_i}$.
2. S_{eq} is the set of MSCs where there exists some i with $R_i \neq \overline{B_i}$.
3. S_{succ} is the set of MSCs where there exists some i with $\overline{R_{i+1}}$ not a successor configuration of R_i.
4. S_{init} where B_1 is not the initial configuration.
5. S_{fin} where B_k is not a final configuration.

Let us sketch how to define template formulas that describe the above sets. It is easy to express that M does not have the right shape, since we can do it locally. For the sets described by S_{eq} and S_{succ}, we use the counters for identifying the corresponding positions. For example, for S_{eq}, we define:

– γ as a (possibly closed) gap without restriction.

– $M_{eq} = \vee_{i \leq n} \vee_{\gamma' \neq \gamma} \gamma\,(i,b)\gamma(\bar{i},r)\overline{\gamma'}\,\gamma$.

It is easy to show that $\mathcal{L}(\epsilon \mapsto (M_{eq})) = S_{eq}$. We then define $\Phi = \epsilon \mapsto (M_{shape} \vee M_{eq} \vee M_{succ} \vee M_{init} \vee M_{final})$. □

5 Conclusion

We have proposed template MSC formulas as an extension of Triggered MSCs by adding gaps and showed how to use them as a visual specification formalism. The two main components are the use of assume-guarantee CMSC as in Triggered MSCs and LSCs, together with gaps. The formalism is quite expressive and allows to specify safety and liveness-like properties. A drawback of the expressivity is that satisfiability is undecidable, unless there is an existential bound on communication channels. Notice that for LSCs, satisfiability (consistency) is shown to be decidable in [9] with a synchronous semantics of communication. However, synchronous communication is a severe restriction for specifying protocols.

We study the use of template MSCs as a specification formalism, given as a visual alternative to linear temporal logic, and analyse the complexity of checking various template MSC properties. Model-checking a realizable compositional HMSC S gives the following complexities for the restrictions below:

$S \models M_a \rightarrowtail M_g$	M_a	M_g
PTIME	closed gaps and S complete or one closed gap	$\pm$ one closed gap, no disjunction
PSPACE	no restriction	negative templates or $\pm$ two gaps
EXPSPACE	no restriction	no restriction

Fig. 6. Complexity of the Model-Checking

References

1. ITU-TS recommendation Z.120, Message Sequence Charts, Geneva, 1999.
2. R. Alur and K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *ICALP'01*, LNCS 2076, pp. 797–808, 2001.
3. R. Alur and K. McMillan and D. Peled. Deciding global partial-order properties In *ICALP'98*, LNCS 1443, pp. 41–52, 1998.
4. W. Damm and D. Harel. LSCs: Breathing life into Message Sequence Charts. Formal Methods in System Design 19(1): 45-80 (2001).
5. V. Diekert, P. Gastin. LTL is expressively complete for Mazurkiewicz traces. In *JCSS* 64(2): 396-418, 2002
6. B. Genest, A. Muscholl. Pattern Matching and Membership for Hierarchical Message Sequence Charts. In *LATIN'02*, LNCS 2286, pp. 326-340, 2002.
7. E. Gunter, A. Muscholl, and D. Peled. Compositional Message Sequence Charts. In *TACAS'01*, pp. 496–511. LNCS 2031, 2001.
8. H. Liu, C. Wrathall and K. Zeger. Efficient Solution of Some Problems in Free Partially Commutative Monoids. *Inf. and Comp.*, 89:180–198, 1990.

9. D. Harel and H. Kugler. Synthesizing State-Based Object Systems from LSC Specifications. Netcharts: bridging the gap between HMSCs and executable specifications. In *International Journal of Foundations of Computer Science* 13(1): 5-51, 2002.
10. M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. Netcharts: bridging the gap between HMSCs and executable specifications. In *CONCUR'03*, LNCS 2761 ,pp 296–310, 2003.
11. A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *MFCS'99*, LNCS 1672, pp. 81–91, 1999.
12. A. Muscholl, D. Peled, and Z. Su. Deciding properties for Message Sequence Charts. In *FoSSaCS'98*, LNCS 1378, pp. 226-242, 1998.
13. B. Sengupta and R. Cleaveland. Triggered Message Sequence Charts. In *SIGSOFT 2002/FSE-10*, ACM Press, 2002.
14. P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic. In *Information and Computation*, 179(2): 230-249, 2002.
15. I. Walukiewicz. Difficult configurations – On the complexity of LTrL. In *ICALP '98*, LNCS 1443, pp. 140-151, 1998.