Conditions for synthesis of communicating automata from HMSCs

Loïc Hélouët, Claude Jard

IRISA/CNRS
Campus de Beaulieu
35042 Rennes cedex FRANCE
helouet@irisa.fr,jard@irisa.fr
http://www.irisa.fr/pampa

Abstract

Formal methods can now be used at early stages of the development process. This increases the need for consistency between two levels of formalism: a declarative level of scenario type, and an operational level of automata type. It is particularly true in the telecommunications world with the joint use of standardized languages like High-level Message Sequence Charts (HMSCs) and SDL. So it is natural to consider the transition from one level to another by automated synthesis mechanisms. Algorithms for synthesis of SDL communicating systems from HMSCs have been proposed these last years. However, the theoretical power of HMSCs is such that only a subset of HMSCs can be reasonably treated. Identification of the limits of synthesis is still in its early stages. In this article we show for the first time a necessary condition so that synthesis preserves behaviours. This condition relies on a property of generalized local choice and reconstructibility of the local sequencing. It is decidable and we present algorithms that could be implemented in practical synthesis tools.

Key words: Message Sequence Charts, synthesis, distributed systems.

1 Introduction

Specification of distributed systems starts at a high level of abstraction. An intuitive formalism must be used to capture the behaviours of communicating entities, leaving implementation details for later refinement steps. Two main approaches can be distinguished. The sequential approach (mostly based on communicating automata) emphasizes sequences of events within processes of the system. The specification is given as behaviours of single processes, that contain communication events. Languages like SDL or Estelle (5) can be used for this purpose. They are equipped with tools that allow many formal manipulations, ranging from simulation to code generation. The second approach is communication-based. It emphasizes on communications between processes. The specification is given through patterns of actions and communications performed on different processes. Use cases and scenarios are examples of such kind of specification formalisms.

The main drawback of the sequential approach is the difficulty to reason about global properties (even the simple fact that a given sending event matches a receiving event). On the other hand, sequential communicating processes can be directly implemented, as they just represent local sequences of communication primitives and internal actions.

Scenarios have the advantage to give a global view of the system activity. Causalities and concurrency are explicitly represented. Nevertheless, composing scenarios is not an easy task, and the overall meaning of a specification can be less intuitive than expected. For example, new concurrency may be introduced during sequential composition. Furthermore, scenarios are not directly implementable (in a distributed way). This may explain why they are often restrained to documentation purposes in methodologies like UML (8), and in the SDL-based tools.

Transforming a scenario model into a set of communicating finite state machines is a first step towards their implementation. This goal needs to provide answers to questions such as:

- Is a scenario implementable?
- If the answer is no, what should be changed to make it implementable?
- Is there a class of scenarios which is known to be easily implementable?

Algorithms for synthesis of SDL communicating systems from HMSCs have been proposed these last years. But the theoretical power of HMSCs is such that only a subset of HMSCs can be reasonably treated. Identification of the limits of synthesis is still in its early stages. In this article we show for the first time a necessary and sufficient condition such that synthesis preserves behaviours. This condition relies on a property of generalized local choice and reconstructibility of the local sequencing. It is decidable and we present algorithms, which could be implemented in practical tools for synthesis.

This article is organized as follows: first section describes HMSCs and defines the notion of language behaviours and of reconstructibility. Section 3 provides a state of the art of synthesis methods from HMSCs. Section 4 presents our target model of communicating finite state machines (CFSM) and formalizes the synthesis approach used by (1; 7) to synthesize SDL from HMSCs. Section 5 outlines the problems met by this approach, and proposes a condition on HMSCs that would make it valid with respect to languages equivalence.

2 Message Sequence Charts

This section introduces Message Sequence Charts, a scenario formalism standardized by the ITU (11). MSCs are defined by two levels of specification: basic Message Sequence Charts, which define simple communication scenarios, and High-level Message Sequence Charts, a kind of scenario automaton, that composes basic charts.

2.1 Basic Message Sequence Charts

Basic Message Sequence Charts (bMSCs for short) model a communication pattern between processes (called *instances*). Each instance defines a sequence of events, and is represented by a vertical axis. An event can be a message emission or reception, a timer operation, or an atomic action. As no precise meaning is associated to timer events within this article, they will be considered as atomic actions. Notice the very important fact that communications are closed: sending and receiving of a message are localized in the pattern. This explains the decidability results presented in the paper. A bMSC defines precedence relations on events: a message emission must precede the corresponding reception, and events are totally ordered along instance axis. Therefore, a bMSC can be formalized as a finite, non-autoconcurrent labeled partial order.

A bMSC is a tuple $M = (E, \leq, I, \phi)$ where:

- E is a finite set of events,
- $\leq$ is a partial order relation (antisymmetric, reflexive and transitive) called *causal order* on events,
- I is a set of names of instances that perform at least one action in M, and is called the set of *active instances* of M.
- $\phi : E \longrightarrow I$ is a labeling of events. It is required that this labeling is not auto-concurrent, which means that events belonging to the same instance are totally ordered (form chains):

$$\forall (e_1, e_2) \in E^2,\ \phi(e_1) = \phi(e_2) \implies (e_1 \leq e_2) \vee (e_2 \leq e_1)$$

Slightly abusing the notation, we will note $\phi(E) = \{i | \exists e \in E \wedge \phi(e) = i\}$ the set of instances appearing in any set of events E. For any MSC M, we will note $min(M) = \{e \in E | \nexists e' \neq e \wedge e' \leq e\}$ the set of minimal events of M, and $min_i(M) = e \in E | \phi(e) = i \wedge \forall e', \phi(e') = i, e \leq e'$ the minimal event on instance $i \in I$. For any event $e \in E$, $pred(e) = \{e' | e' \leq e\}$ will denote the set of predecessors of e. We will also denote by $em(e)$ the sending event corresponding to the receiving event e. For any labeled order

$M = (E_M, \leq_M, I_M, \phi_M)$, for any set $E' \subseteq E_M$, we will denote by $M_{/E'}$ the restriction of M to events of E'.

The events of E_M can be of three types: `send(m) to j` for the emission of the message m, `rec(m) from j` for the reception of the message m, and `action` for an internal event (atomic action, or operation on a timer). We will often note $!m$ the sending event, and $?m$ the receiving event for a message m.

Consider, for example bMSC M in Figure 1. M represents a communication pattern between two processes A and B, and can be formalized by a labeled partial order $M = (E_M, \leq_M, I_M, \phi_M)$ where :

- $\begin{array}{ll} E_M = \{ & e_1 = \texttt{send(m1) to B}, e_2 = \texttt{send(m2) to B}, \\ & e_3 = \texttt{rec(m1) from A}, e_4 = \texttt{rec(m2) from A}\} \end{array}$
- $\leq_M = \{(e_1, e_2), (e_1, e_3), (e_2, e_4), (e_3, e_4)\}$
- $I_M = \{A, B\}$
- $\phi_M = \{(e_1, A), (e_2, A), (e_3, B), (e_4, B)\}$

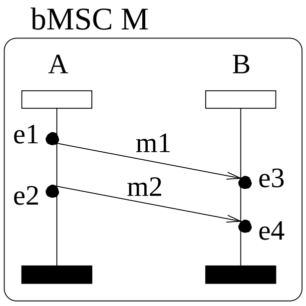

Fig. 1. An example bMSC.

Standard notation of bMSCs also allows for the definition of a zone on an instance axis called co-region, in which events are not ordered. According to the assumed semantics of co-regions, it can mean that events are concurrent, or that the order is not yet defined, and should be specified in further refinements of the specification. As we do not consider co-regions as a central point for our approach, we leave them for further extensions of our work. Therefore, events will be considered as totally ordered on an instance axis.

2.2 bMSC automata (HMSC)

bMSCs only allow for the specification of simple scenarios. A higher level notation called High-level Message Sequence Charts (HMSCs for short) is used to define more elaborated behaviours. Longer patterns can be constructed by sequentially composing bMSCs. HMSC H_1 in Figure 2 is a sequential composition of bMSCs M_1 and M_2. The semantics of the sequence of bMSCs defined in the standard is a weak sequential composition[1]. The result is an instance-by-instance concatenation, where, for each instance, the maximum event of the first bMSC is linked to the minimum event of the second bMSC. This gives to MSCs an interesting expressive power since communication messages can be accumulated between instances by concatenating basic patterns. For algorithmic reasons, some authors restrict the composition to a strong sequencing, forcing a synchronization barrier between each pattern. We think this dramatically decreases the modeling power of MSCs. Let us consider again the example of Figure 2. Imposing a strong sequencing between M_1 and M_2 forces instance A to wait for a synchronization with instance B before sending message m_2. Therefore, an implementation of H_1 assuming strong sequencing between M_1 and M_2 would have to introduce new communications, and the only trace defined by HMSC H_1 would be $!m_1.?m_1.!m_2.?m_2$. However, we think there is no reason to delay a process in a distributed system when no message reception is expected. Moreover, the bMSC decomposition of a specification is arbitrary, and is more the result of a need to reduce the size of charts than the expression of a synchronization. So, HMSC H_1 in Figure 2 should be considered equivalent to HMSC H_2 in Figure 3, and defines two possible traces: $!m_1.?m_1.!m_2.?m_2$, and $!m_1.!m_2.?m_1.?m_2$.

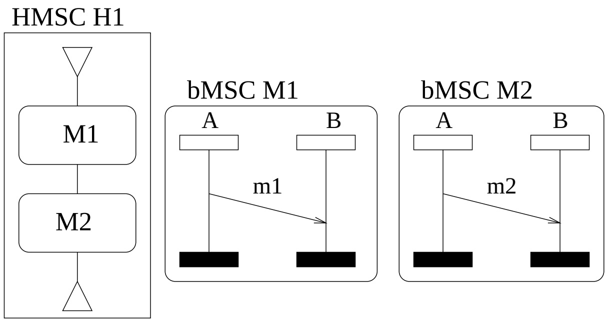

Fig. 2. HMSC H_1: sequence of bMSCs M_1 and M_2.

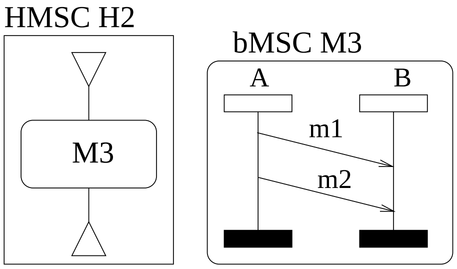

Fig. 3. HMSC H_2 equivalent to HMSC H_1 in Figure 2.

Let us define the sequencing operator $\circ$ on two MSCs $M_1 = (E_1, \leq_1, I_1, \phi_1)$ and $M_2 = (E_2, \leq_2, I_2, \phi_2)$: $M_1 \circ M_2 =< E, \leq_{M_1 \circ M_2}, I_1 \cup I_2, \phi >$, where:

[1] Weak sequential composition is close to the Pratt's local sequencing (9), where ϕ defines locality.

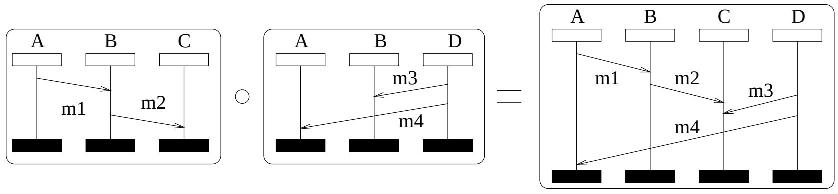

Fig. 4. Chain by chain concatenation of basic message sequence charts.

- E is the disjoint union of E_1 and E_2 : $E = \varphi_1(E_1) \cup \varphi_2(E_2)$ with $\varphi_1(E_1) \cap \varphi_2(E_2) = \emptyset$ and φ_1, φ_2 are two isomorphisms.
- $\forall e, e' \in E$, $e \leq_{M_1 \circ M_2} e'$ iff $\varphi_1^{-1}(e) \leq_1 \varphi_1^{-1}(e')$ or $\varphi_2^{-1}(e) \leq_2 \varphi_2^{-1}(e')$ or $\exists (e_1, e_2) \in \varphi_1(E_1) \times \varphi_2(E_2)$: $\phi_1(\varphi_1^{-1}(e_1)) = \phi_2(\varphi_2^{-1}(e_2)) \wedge \varphi_1^{-1}(e) \leq_1 e_1 \wedge e_2 \leq_2 \varphi_2^{-1}(e')$
- $\forall e \in E$, $\phi(e) = \phi_1(\varphi_1^{-1}(e))$ if $e \in \varphi_1(E_1)$ or $\phi(e) = \phi_2(\varphi_2^{-1}(e))$ if $e \in \varphi_2(E_2)$

More intuitively, sequential composition consists in ordering events e_1 in bMSC M_1 and e_2 in bMSC M_2 if they are situated of the same instance, and then calculating the transitive closure of the partial order obtained. An example of sequential composition is provided Figure 4.

The standard HMSC notation also contains a parallel composition operator, that will not be considered in this article. In most of the cases, HMSCs comprising parallel composition can be translated into HMSCs comprising only sequence and choice operators. Now, consider HMSC H_3 in Figure 5: HMSCs H_4 and H_5 are composed within a parallel frame (parallel composition is denoted by $\|$).

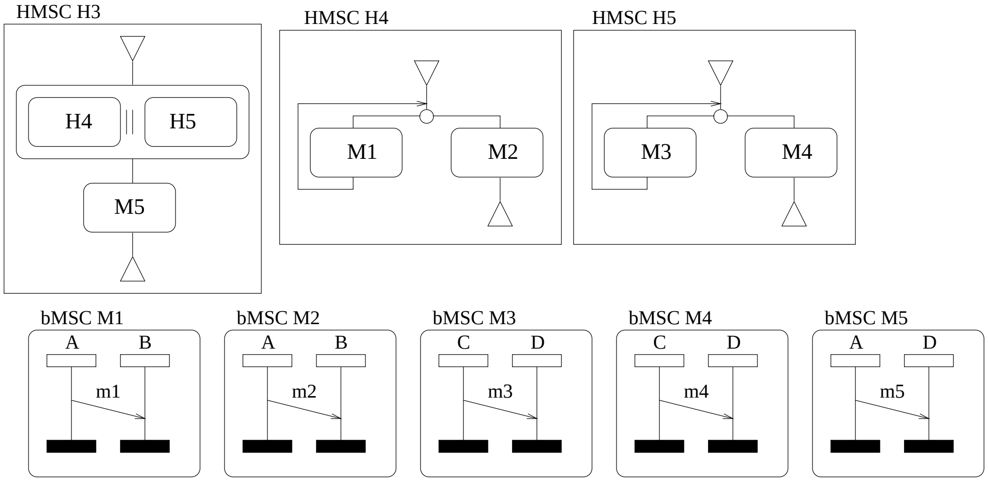

Fig. 5. HMSC H_3 comprising two parallel loops.

This example comprises two potentially infinite loops that behave in parallel. When one of the operands stops looping, the other specification also have to stop. Therefore, this specification contains an implicit synchronization, that can be compared to the non-local choice described in section 5.2. As we want to avoid that kind of specification, we limit our approach to HMSCs comprising choices, sequences, and loops. Those HMSCs can be defined using a finite automaton on bMSCs.

Definition 1 *A* HMSC *is a graph* $H = (N, \longrightarrow, \mathcal{M}, l, n_0)$*, where:*

- N *is a finite set of nodes,*
- $\longrightarrow$ *is the transition relation* ($\subseteq N^2$)*,*
- $\mathcal{M}$ *is a set of bMSCs, on disjoint sets of events. Each bMSC* $M \in \mathcal{M}$ *is a tuple* $M = < E_M, \leq_M, I_M, \phi_M >$*,*
- l *is a labeling function on transitions (* $l : N^2 \longrightarrow \mathcal{M}$ *),*
- n_o *is the starting node of the graph.*

Definition 2 *A finite* path *of a HMSC* H *is a word* $p = n_1..n_k \in N^*$ *such that* $\forall i \in 1..k-1, (n_i, n_{i+1}) \in \longrightarrow$*. Each path* $p = n_1...n_k$ *defines a unique order* $O_p = M_1 \circ .. \circ M_{k-1}$ *where* $\forall i \in 1..k-1, M_i = l(n_i, n_{i+1})$*. An* initial path *is a path starting from* n_0*.*

Definition 3 *A HMSC* H *defines a partial order family* $\mathcal{O}(H)$*, which is the set of orders* $\{O_{p_1}, O_{p_2}, ..\}$ *associated to the set of initial paths* $\{p_1, p_2, ...\}$ *of the automaton. By considering the total orderings that are compatible with at least one order of* $\mathcal{O}(H)$*, we can define the* language *accepted by a HMSC.*

Let $M = (E, \leq, I, \phi)$ *be a bMSC, and let* $w = e_1..e_n$ *be a word of* E^**.* w *is a* linearization *of* M *if and only if:* $\forall i \in 1..n-1, \forall j > i, (e_j, e_i) \notin \leq$*. Let us call* $\mathcal{L}(H)$ *the language described by a HMSC* H*.* $\mathcal{L}(H)$ *is the prefix closed set of linearizations defined by the elements of* $\mathcal{O}(H)$*. Let us note* $E_{\mathcal{M}} = \bigcup_{M \in \mathcal{M}} E_M$ *the set of events of* H*. A word* $w \in E^*_{\mathcal{M}}$ *is a word of* $\mathcal{L}(H)$ *if and only if:*

$$\exists v = w.u \wedge \exists p = n_0..n_k \text{ initial path of } H \wedge v \text{ is a linearization of } O_p$$

Definition 4 *A* choice *in a HMSC* H *is a node with more than one successor. A choice* c *defines an alternative between scenarios. Any loop-free path starting from* c *will be called a* branch *of the choice* c*.*

2.3 Reconstructibility

A difficult question raised by synthesis is how to impose locally on the events of each instance, the order globally defined by the HMSC. Knowing that the message receptions are undergone and not controlled, a key point will be the possibility of rebuilding the desired order from the order of the received messages. It is what we call reconstructibility.

Definition 5 *Let* R *be a partial order relation on a set of events* E*, and* ϕ *be a labeling of* E*. A* non-local transitive reduction *of* R *is the set of pairs*

$(e, e') \in R$ *such that* $\phi(e) \neq \phi(e')$ *and* $\nexists e'' | (e, e'') \in R \wedge (e'', e') \in R$*. It is denoted by* $e \rightarrowtail e'$*.*

In a bMSC, the non-local transitive reductions of the causal order are the pair of events associated to communications (when $e \rightarrowtail e'$, then e is a message emission and e' is the corresponding reception).

Definition 6 *The* message-transitive closure *(or mt-closure, for short) of a partial order relation* R *is written* R^{*mt}*, and is a relation* R' *such that* $(e, e') \in R'$ *if and only if:*

- *i)* $(e, e') \in R$*, or*
- *ii)* $\exists e'' \in E$ *such that* $eR'e'' \wedge e''R'e'$*, or*
- *iii)* $\exists e_1, e_2 \in E^2$ *such that* $\phi(e1) = \phi(e_2) \wedge e_1 R' e_2 \wedge e_1 \rightarrowtail e \wedge e_2 \rightarrowtail e' \wedge \phi(e) = \phi(e') \wedge (e', e) \notin R$*.*

Obviously, as ^{*mt} is a closure operation, any element of R must be in R^{*mt} (condition $i)$). $ii)$ expresses transitivity on the causality relation between events. $iii)$ e_1 and e_2 are message emissions, and e and e' are the corresponding receptions. As no ordering between e and e' exist, and as any pair of event of the same instance must be ordered, the order between message receptions is the same as the order between the corresponding emissions.

Definition 7 *A pair* (e, e') *is said to be* reconstructible *by message-transitive closure in a relation* R *if and only if* $(e, e') \in R^{*mt}$

The pair (e, e') in the first situation of Figure 6 is reconstructible. Dotted arrows symbolize the reconstructed edges. On the other hand, the pair (e, e') in the second situation is not reconstructible, as $e_1 \not\rightarrowtail e$. The pair (e', e) of the third situation is not reconstructible either, as the pair (e, e') already exists.

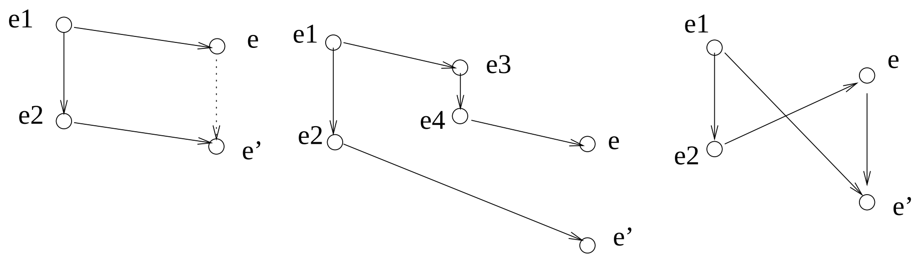

Fig. 6. Order reconstruction allowed by FIFO communications.

A synthesis method from HMSCs to communicating automata will be considered as correct if any ordering information is preserved by the transformation, or can be reconstructed using the mt-closure.

3 State of the art

Synthesizing protocols from HMSCs is not a new problem. Several approaches have been proposed. In (10), MSCs are seen as descriptions of finite state machines communicating synchronously, composed by means of regular expressions. The proposed synthesis algorithm is to project the regular expressions on each instance, to produce a protocol (i.e. a set of synchronously communicating finite state machines). Then the traces of the synthesized system are compared with the traces allowed by the HMSC. If the set of traces are equivalent, then the protocol is considered correct. This method suffers two drawbacks. First, an erroneous protocol can be constructed from a specification containing a non-local choice (defined in section 5.2), and potentially leading to deadlock. Second, concurrency between MSCs is interpreted as interleaving. When concurrency appears within an iteration, some correct protocols may be considered as incorrect during the trace comparison.

(4) addresses the question whether a given bMSC can be the behaviour of an implementation model. This approach distinguishes five different buffering methods (ranging from one FIFO channel per message to synchronous communications). Some implementation models are equivalent (they allow for the implementation of the same bMSCs). Then, a definition of implementability for a class of architecture is provided. A bMSC M is said to be weakly implementable for a class architecture if at least one trace of M can be implemented. A bMSC M is said to be strongly implementable if all traces of M can be implemented. This article only addresses bMSCs, and implementability of two bMSCs M_1 and M_2 on a given architecture does not ensures that a composition of M_1 and M_2 is implementable using the same communication model.

(6) proposes to synthesize ROOM models from MSCs. ROOM charts are a kind of asynchronous state-charts. MSC sequential composition is considered as strong sequencing of orders, which reduces expressiveness (the language of a HMSC is the concatenation of linearizations of the scenarios, without any possibility of shuffling these words). The synthesis algorithm can be used on a subset of MSCs that do not contain non-local choices, message overtaking, or internal actions. Furthermore, MSCs are supposed to be normalized (i.e. bMSCs do not have common prefix at choice nodes), and each instance is supposed to execute at least one emission or one reception. Consequently, the non-local choice decision required by this article can be computed by considering only the immediate successors of choice nodes. We will see in section 5.2 that the standard case of weak sequential composition demands to search all reachable bMSCs to detect non-locality.

In (7), weak sequencing of HMSCs is considered. The HMSC is projected on its instances, which gives a set of skeletons of finite state machines, that are translated into SDL processes. The synthesis method assumes an SDL-like communication channel between each pair of communicating processes. However, the SDL system allows more traces than those defined by the HMSC specification. This is due to the impossibility of preserving an order between message receptions from different senders. This approach is implemented in MOST (Moscow Synthesizer Tool).

In (1) SDL processes are synthesized according to a given communication architecture. The most permissive architecture associates a SDL channel to each pair of communicating instances (the approach is then very similar to (7)). Some more restrictive architectures may prevent even simple HMSCs from being implemented. Again, the synthesized protocol may produce traces that are not specified by the HMSC. This approach is implemented in the MSC2SDL tool.

4 Automatic derivation of communicating finite state machines

Communicating finite state machines (CFSMs for short) are a commonly used representation of distributed systems (each process is described by a finite state machine that can send or receive messages). Communication channels are FIFO: reception of a message m can be performed if and only if m is the first message that can be consumed from the channel (see (3) for example). This does not allow message overtaking, which can be nevertheless specified in HMSC. Instead of considering a complex buffering mechanism for each process in that case, we prefer to slightly modify the communication semantics. We consider a message reception as possible if the message is present in the queue. This communication semantics is very similar to SDL communication without implicit message consumption.

4.1 Synthesis method

This section formalizes the approach defined in (1; 7). The target model for synthesis is finite automata communicating through queues, one queue being associated to each pair of automata.

Let $H = (N, \longrightarrow, \mathcal{M}, l, n_0)$ be a HMSC describing the behaviour of a set of communicating instances I, then:

- $E_{\mathcal{M}} = \bigcup\limits_{M \in \mathcal{M}} E_M$ is the union of events of bMSCs in $\mathcal{M}$.

- $E_{|i} = \{e|\phi(e) = i\}$ is the restriction of $E_{\mathcal{M}}$ to the set of events performed by instance i.

A set of CFSM $\mathcal{A}_H = \{\mathcal{A}_i\}_{i\in I}$ can be computed from H. Each CFSM $\mathcal{A}_i$ is a tuple $\mathcal{A}_i = (S_i, E_i, \delta_i, s_{0_i})$ such that:

- $S_i \subseteq \mathcal{P}(E_i)$ is a set of states,
- $E_i = E_{|i}$ is a set of events,
- $\delta_i \subseteq S_i \times E_i \times S_i$ is a set of transitions. $(s, e, s') \in \delta_i$ if and only if:
  - $e \in s$, and
  - $$s' = \{e' \in E_i| \quad \exists p \text{ path of } H \text{ such that } O_p = M_1 \circ .. \circ M_k \ , \\ e \in M_1 \text{ and } e' = min_i(O_{p/E_{O_p}\setminus\ pred(e)})\},$$
- $s_{0_i} = \{e \in E_i|\exists p \text{ initial path of } H, \text{ and } e = min_i(O_p)\}$ is the initial sate.

More intuitively, a state is a subset of fireable events. An event can be fired if it is contained in a state. The resulting state is composed of events located on the same path as e that are allowed after the execution of e.

4.2 Language of CFSMs

A set of CFSMs defines a possibly infinite transition system.

A **state** of a CFSM system composed of K communicating machines is a pair $S = (\{s_i\}_{i\in 1..K}, \{w_{ij}\}_{i,j\in 1..K})$, where:

- $\{s_i\}_{i\in 1..K}$ is a set of local states,
- $\{w_{ij}\}_{i,j\in 1..K}$ is a word representing the messages transiting from i to j,

The initial state for a CFSM system is $S_0 = (\{s_{0_i}\}_{i\in 1..K}, \{\epsilon, ..., \epsilon\})$. Considering any subword w of a buffer w_{ij}, we will note $|w|_m$ the number of occurrences of the message m in the word w.

A **transition** from a state of the CFSM system to another state is possible if one of the components of the system can perform it.

$$\frac{\text{-}\quad \exists j \in 1..K\ :\ e \in s_j \wedge e = \texttt{rec(m) from i} \ \wedge\ w_{ij} = v.m.v' \wedge (s_j, e, s'_j) \in \delta_j \\ \wedge \forall e' = \texttt{rec(m') from i}:\ (s_j, e', s''_j) \in \delta_j, |v|_{m'} = 0}{S \xrightarrow{e} S', \text{ where } S' = S_{\{s_j := s'_j; w_{ij} := v.v'\}}}$$

A message of type m sent by a process i can be received by an automaton $\mathcal{A}_j$ in a state s_j if the reception of m is an event that can be fired from state s_j, and if m is the first message in the buffer from i to j that can be read in sate s_j.

$$\frac{\exists j \in 1..K \ : \ e \in s_j \wedge e = \texttt{send(m) to i} \ \wedge \ w_{ji} = v \wedge (s_j, e, s'_j) \in \delta_j}{S \xrightarrow{e} S', \text{ where } S' = S_{\{s_j := s'_j ; w_{ji} := v.m\}}}$$

$$\frac{\exists j \in 1..K \ : \ e \in s_j \wedge e = \texttt{action} \ \wedge \ (s_j, e, s'_j) \in \delta_j}{S \xrightarrow{e} S', \text{ where } S' = S_{\{s_j := s'_j\}}}$$

A message emission or an atomic action can be performed by an automaton $\mathcal{A}_j$ in a state s_j if and only if this event is allowed in s_j.

This definition of communications allows us to generate automata for specification containing message crossing, which would not be possible with a strict FIFO ordering. Consider the simple bMSC of Figure 7. This specification contains a message crossing, which can however be implemented by the communicating finite state machines of Figure 8.

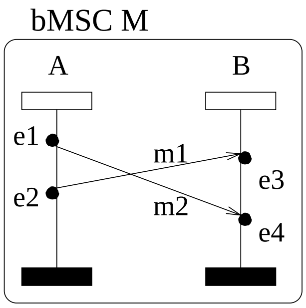

Fig. 7. A simple bMSC.

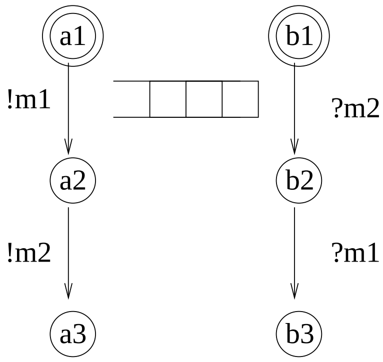

Fig. 8. A simple CFSM implementing bMSC M of Figure 7.

Let us note $\mathcal{L}(\mathcal{A}_H)$ the language described by a set of communicating automata $\mathcal{A}_H$. A word $w = e_1.e_2..e_n \in E^*_{\mathcal{M}}$ is a word of $\mathcal{L}(\mathcal{A}_H)$ if and only if $S_0 \xrightarrow{e_1}$

$S_1 \xrightarrow{e_2} ... \xrightarrow{e_n} S_n$. We will write $S \xrightarrow{w} S'$ when a state S' can be reached from state S using w. We will denote by $s(w)_i$ the state reached by the automaton $\mathcal{A}_i$ after execution of w.

5 Conditions for language equality

A protocol $\mathcal{A}_H$ synthesized from a HMSC H is considered as correct if $\mathcal{A}_H$ and H define the same language.

5.1 Soundness

The first step is to show that the synthesized CFSM is able to produce all the behaviours defined by the original HMSC.

Theorem 8 *For any HMSC H, $\mathcal{L}(H) \subseteq \mathcal{L}(\mathcal{A}_H)$*

proof:

Proving $\mathcal{L}(H) \subseteq \mathcal{L}(\mathcal{A}_H)$ is equivalent to showing:

$$\forall w \in E_{\mathcal{M}}^*, w \in \mathcal{L}(H) \Longrightarrow w \in \mathcal{L}(\mathcal{A}_H)$$

This is also equivalent to proving the property:

$$P : \forall n \in I\!N, \forall w \in E_{\mathcal{M}}^n, w \in \mathcal{L}(H) \Longrightarrow w \in \mathcal{L}(\mathcal{A}_H)$$

Obviously, P is true for $n = 0$. Let us show that P true for n implies P true for $n{+}1$. Let us suppose $P(n)$ true, and $P(n{+}1)$ false. Then, it means that there is a word $w \in E_{\mathcal{M}}^n$, and an event $e \in E_{\mathcal{M}}$ such that: $w.e \in \mathcal{L}(H) \wedge w.e \notin \mathcal{L}(\mathcal{A}_H)$. Therefore $\exists p = M_1 \circ ... \circ M_k$, path of H such that $w.e$ is a prefix of a linearization of p, and $\phi(e) = i \wedge (e \notin s(w)_i \vee (e = \text{rec(m) from } j \wedge w_{ij} \neq v.m.v')$

- $e \notin s(w)_i$ if and only if there is a predecessor e' of event e on instance i and e' has not been executed in w. Therefore, $w.e \notin \mathcal{L}(H)$, contradiction.
- $e = \text{rec(m) from } j \wedge w_{ij} \neq v.m.v'$ also leads to a contradiction, as $w.e \in \mathcal{L}(H)$ implies that any predecessor (and consequently an emission of m) appears in w.

Therefore, we have proved $\mathcal{L}(H) \subseteq \mathcal{L}(\mathcal{A}_H)$. $\square$

Unfortunately, the synthesis method defined previously does not ensures that $\mathcal{L}(H) = \mathcal{L}(\mathcal{A}_H)$. Let us consider HMSC H_6 of Figure 9: the traces defined by H_6 are $!m_1.?m_1$ and $!m_2.?m_2$. The communicating finite state machines synthesized from H_6 (see Figure 10) also describe the traces $!m_1.!m_2$ and $!m_2!m_1$. HMSC H_6 defines an implicit synchronization between instances A and B, that must agree on the scenario to perform. Clearly, this specification can not be implemented without an additionnal synchronisation mechanism. Such a situation is called non-local choice.

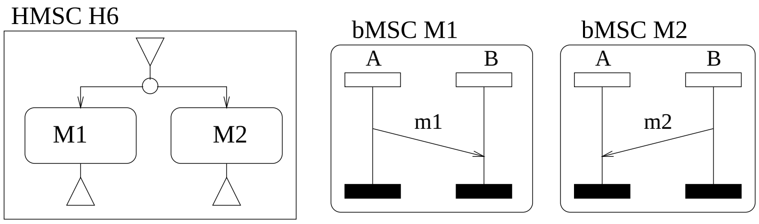

Fig. 9. A non-local HMSC.

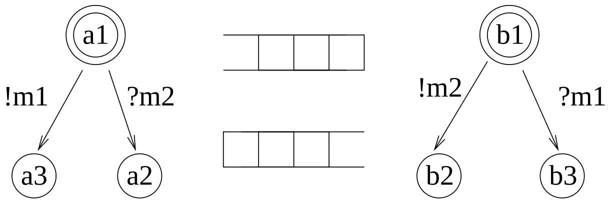

Fig. 10. CFSM synthesized from HMSC H_6 in Figure 9.

5.2 Non-local choice

The generally admitted meaning of non-local choice (2) is when more than one instance can decide to perform a scenario or another at a choice node. The intended behaviour is that the first instance able to perform the choice chooses a behaviour. The next instances reaching the same iteration of this choice have to conform to the chosen scenario. This results in a behaviour in which an instance "knows" what to do at a choice node without any communication. However, non-local choices can not be implemented without adding communications to the system. In some particular cases, however, non-local choices may express concurrency between scenarios, as in example of Figure 14.

A definition of non-local choice was previously given in (2). This definition assumes that any instance should communicate with other instances on each branch of a choice. This assumption limits the search for non-local choice to the set of outgoing edges. However, when considering weak sequential composition of bMSCs with disjoint set of instances, non-local choice is not a local property. Therefore, a global definition of non-local choice must be provided. Consider HMSC H_7 in Figure 11: choices seem to be local, but the decision to perform a scenario can be taken by A or C. The definition has thus to be extended.

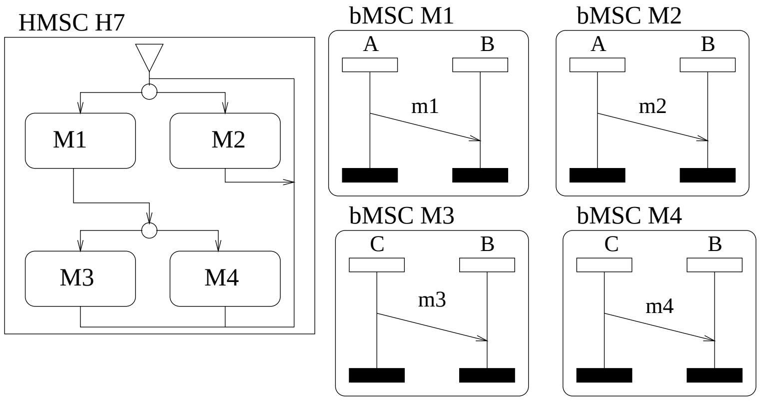

Fig. 11. Non-local choice located on more than one choice node.

Definition 9 *Let c be a choice node. c is* local *if and only if:*

$\forall p_1 = c.n_1...n_k$, *maximal loop-free path of H ($c \notin n_1...n_k$) from c, $\forall p_2 = c.n'_1...n'_k$ maximal loop-free path of H from c, $\forall e_1 \in min(O_{p_1})$, $\forall e_2 \in min(O_{p_2}), \phi(e_1) = \phi(e_2)$.*

For a local choice c, the instance deciding the behaviour at this point of the specification will be called the *deciding instance* of choice c. From the definition of non-local choice, the algorithm is straightforward. Note that locality can be checked on loop-free paths (adding a pattern that already appears in a path does not add minimal events).

Algorithm:

for all c,`choice node in` H **do**
  $P = \{(c.n, I, J) | c \xrightarrow{M} n \wedge I = \phi(min(M)) \wedge J = \phi(M)\}$
  $MAP = \emptyset$ `/* Maximal acyclic paths */`
  while $P \neq \emptyset$ **do**

$$MAP = MAP \cup \{(w.n, I') | w = n_1...n_k \wedge n_k \xrightarrow{M} n \wedge n \in w$$
$$\wedge (w, I, J) \in P \wedge I' = I \cup (\phi(min(M)) - J)\}$$
$$P = \{(w.n, I', J') | (w, I, J) \in P, w = n_1...n_k \wedge n_k \xrightarrow{M} n \wedge$$
$$n \notin w \ \wedge J' = J \cup \phi(M) \ I' = I \cup (\phi(min(M)) - J)\}$$

  end while
  $DI = \bigcup_{(w,I) \in MAP} I$ `/* deciding instances */`
  if $|DI| > 1$ **then**
    H `contains a non-local choice`
  end if
end for

As we only consider loop-free paths of HMSC H this algorithm terminates. From now on, we will write $Loc(H)$ when a HMSC H does not contain non-local choices.

Theorem 10 $Loc(H) \not\Longrightarrow \mathcal{L}(\mathcal{A}_H) \subseteq \mathcal{L}(H)$

Ensuring that a HMSC H is local does not ensures that the synthesis produces a CFSM that is language equivalent to H

The automata produced for HMSC H_8 of Figure 12 are described in Figure 13. One can easily check that
$w = !m1.?m1.!m2.!m3.?m3.!m4.?m4....$ in a trace of $\mathcal{A}_H$, but not of H (in H, $m2$ must be received before $m4$ in that trace).

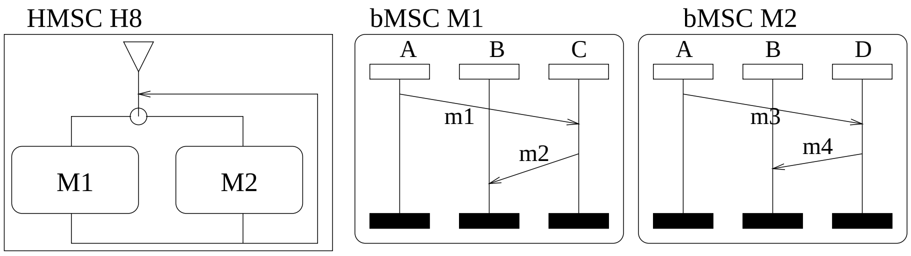

Fig. 12. HMSC H_8

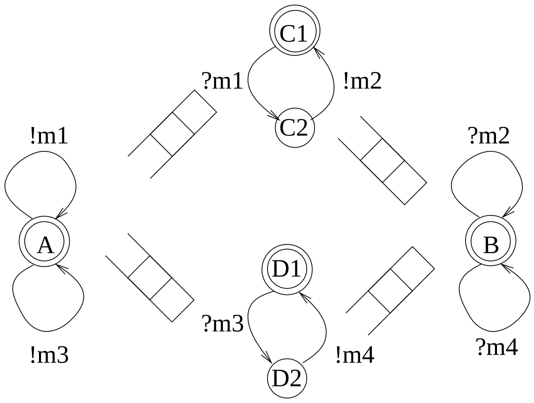

Fig. 13. CFSM synthesized from HMSC H_8 Figure 12

When synthesizing a CFSM system from a HMSC, the local ordering between events may be lost at choice nodes. Consequently, a specification may reach a state in which two messages m and m' can be received from different senders by the same automaton $\mathcal{A}_i$. In the HMSC specification, m and m' are messages from different bMSCs, and therefore their receptions are ordered (either `rec(m)` $\leq$ `rec(m')` or `rec(m')` $\leq$ `rec(m)`). If $\mathcal{A}_i$ do not have enough information to reconstruct the ordering between `rec(m)` and `rec(m')`, a wrong trace is allowed. This is what happens in the specification Figure 12: if bMSC M_1 is choosen before bMSC M_2, then m_1 is sent before m_3, and m_2 must be received before m_4. Unfortunately, $\mathcal{A}_B$ do not have enough information to prevent the reception of m_4.

Theorem 11 $\neg Loc(H) \not\Longrightarrow \mathcal{L}(\mathcal{A}_H) \not\subseteq \mathcal{L}(H)$

Even if H is a non-local Message Sequence Charts, the CFSM implementation *synthesized from H can be correct (consider HMSC H_9 in Figure 14).*

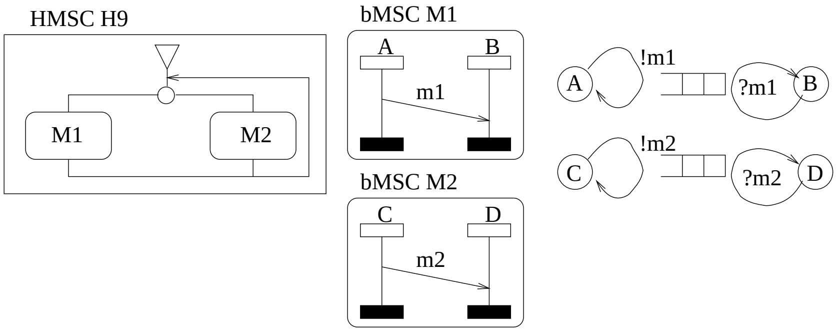

Fig. 14. Non-local HMSC with language equality.

5.3 Local sequencing reconstructibility

The locality condition is not strong enough to ensure that the CFSM generation of Section 4 produces a set of automata that is trace equivalent to the HMSC specification.

Definition 12 *A choice c of a HMSC H is said to be reconstructible if and only if:*

- *c is a local choice, and*
- *$\forall p$ path of H from c such that $O_p = B_1 \circ B_2 \circ ... \circ B_n$ with $B_i, i \in 1..n$ branch of c , $\forall x$ non-deciding instance, $\forall (e, e') \in p \ : \ e = min_x(B_i) \wedge e' = min_x(B_j) \wedge 1 \leq i < j \leq n$, (e, e') is a reconstructible pair of $\leq_{O_p}$ from $\leq_{O_p} \backslash (E_{i_{|x}} \times E_{j_{|x}})$ by mt-closure.*

More intuitively, a choice is reconstructible if removing the local ordering due to bMSC sequencing on any non-deciding instance does not affect the mt-closed ordering.

Definition 13 *A HMSC H is said to be reconstructible if and only if any choice in H is reconstructible. This property will be written $Rec(H)$.*

Imposing HMSC reconstructibility is weaker than requiring any message to be acknowledged. The example of Figure 15 shows a HMSC in which two messages are emitted by different instances and received on a single instance. However, the order between the receptions is preserved by the translation in Figure 16.

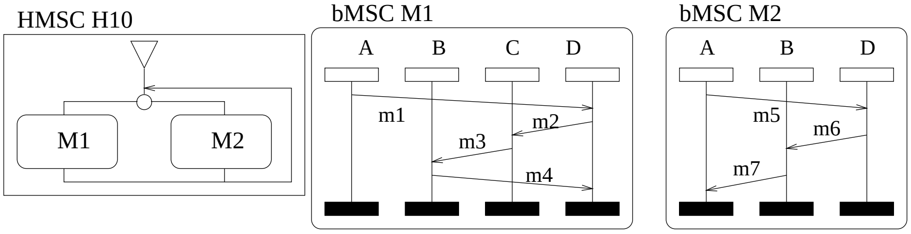

Fig. 15. Reconstructible HMSC H_10.

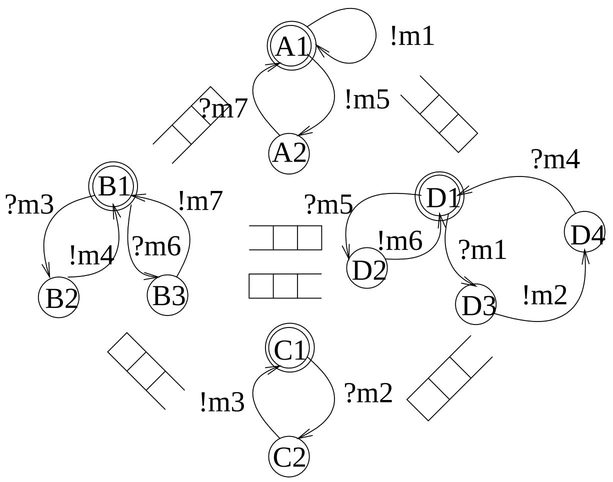

Fig. 16. CFSM synthesized from HMSC H_10 in Figure 15.

Theorem 14 $Loc(H) \wedge \mathcal{L}(\mathcal{A}_H) \subseteq \mathcal{L}(H) \Longrightarrow Rec(H)$

proof:

Let us prove that $\mathcal{L}(\mathcal{A}_H) \subseteq \mathcal{L}(H) \wedge \neg Rec(H)$ leads to a contradiction.

If $Rec(H)$ does not hold, then there is a choice node c in H such that: $B_1 =< E_1, \leq_1, A_1, I_1, \alpha_1 >$ and $B_2 =< E_2, \leq_2, A_2, I_2, \alpha_2 >$ are two branches of c, $\exists p$ path of H from c with $O_p = B_1 \circ B_2$. There exists an instance x, $\exists r_1 \in E_1$, $r_2 \in E_2$, receiving events such that: $r_1 = min_{\phi(r_1)}$, $r_2 = min_{\phi(r_2)}$, $r_1 =$rec(m_1) from i, $r_2 =$ rec(m_2) from j, $i \neq j$.

Let us call O the order $O = (\leq_{O_p} -(E_{1_x} \times E_{2_x}))^{*mt}$. From the definition of $Rec(H)$, we have $(r_1, r_2) \notin O$. As (r_1, r_2) can not be reconstructed, for any predecessor e' of r_2 in B_2, and for any successor e of r_1 in B_1, we have $(e, e') \notin O$.

According to the locality of H there are two deciding events $d_1 \in E_1$, $d_2 \in E_2$ such that $d_1 < r_1$, $d_2 < r_2$ and $\phi(d_1) = \phi(d_2)$. From the structure of the HMSC H and from the derivation method, we know that there is an automaton $\mathcal{A}_x \in \mathcal{A}_H$ containing a state s_x such that r_1 and r_2 can be fired from s_x. So, from any global state S containing s_x, r_1 and r_2 can be fired if m_1 and m_2 have been sent.

We know that there is an initial path $p_v = n_0....c$ in H leading to c. So, any linearization v of O_{p_v} is a word of $\mathcal{L}(H)$. As $\mathcal{L}(H) \subseteq \mathcal{L}(\mathcal{A}_H)$, we also have $v \in \mathcal{L}(\mathcal{A}_H)$. Consequently, $\exists S$ such that $S_0 \xrightarrow{v} S$. As we know that r_1 and r_2 are minimal events, we have $s_x \in S$.

So, we can find a word $w = v.d_1.u.em(r_1).d_2.u'.em(r_2)$ where u contains any predecessor of d_2 in O_p in $\mathcal{L}(H)$. We also know that $w \in \mathcal{L}(H)$, therefore $S_0 \xrightarrow{w} S'$ and as r_1 have not been fired, $s_x \in S'$. As m_1 and m_2 have been sent, and as $\phi(em(r_1)) \neq \phi(em(r_2))$, $w.r_2$ is a word of $\mathcal{L}(\mathcal{A}_H)$. $w.r_2$ is not a word of $\mathcal{L}(H)$, as $d_1 < d_2$ implies $r_1 < r_2$ in any word of $\mathcal{L}(H)$. This contradicts $\mathcal{L}(\mathcal{A}_H) \subseteq \mathcal{L}(H)$. $\square$

Theorem 15 $Rec(H) \Longrightarrow \mathcal{L}(\mathcal{A}_H) \subseteq \mathcal{L}(H)$

proof:
Let us suppose that $Rec(H) \wedge \mathcal{L}(\mathcal{A}_H) \not\subseteq \mathcal{L}(H)$, and let us show that it leads to a contradiction.

If $\mathcal{L}(\mathcal{A}_H) \not\subseteq \mathcal{L}(H)$, then there exists a word w such that $w \in \mathcal{L}(\mathcal{A}_H) \wedge w \notin \mathcal{L}(H)$. From the construction method, we know that $w \neq \epsilon$, so w is of the form $w = v.e$, and such that $v \in \mathcal{L}(\mathcal{A}_H) \wedge v \in \mathcal{L}(H)$. As $v \in \mathcal{L}(H)$, then there exists a path $p \in H$ such that v is a prefix of a linearization of O_p.

$v.e \notin \mathcal{L}(H)$ may hold for two reasons:

- i) $\forall p' = p.n_{k+1}....n_{k+n}$, $e \notin E_{O_{p'}}$. Therefore, $v.e \notin \mathcal{L}(\mathcal{A}_H)$. Contradiction.
- ii) $\forall p' = p.n_{k+1}....n_{k+n}$, there exists a sequence of events $e_1.e_2...e_k$ such that $\forall i \in 1..k, e_i \leq_{O_{p'}} e$, and $v.e_1...e_k.e \in \mathcal{L}(H)$.

As $v.e \in \mathcal{L}(\mathcal{A}_H)$, there exists a local state s_x such that $S_0 \xrightarrow{v} S \wedge s_x \in S \wedge (s_x, e, s'_x) \in \delta_x$. As $\mathcal{L}(H) \subseteq \mathcal{L}(\mathcal{A}_H)$, $v.e_1...e_k.e$ is also a word of $\mathcal{L}(\mathcal{A}_H)$. Three cases may appear: e is a sending event, an internal action, or a receiving event.

- Suppose e is a sending event, or an internal action. Then there exists an event $e_j, j \in 1..k$ such that $\phi(e_j) = \phi(e) \wedge \forall i < j, \phi(e_i) \neq \phi(e)$. Therefore, $\exists (s_x, e_j, s''_x) \in \delta_x$. e and e_j are minimal event on instance $x = \phi(e)$ for branches of a choice c. According to the locality property, e_j and e are deciding events for their respective branch. So, $O_{p'}$ is of the form $O_{p'} = M_1 \circ ... \circ M_l \circ B_1 \circ B_2$, where B_1 contains e_j and B_2 contains e. Therefore, $\exists p''$ path of H, such that $O_{p''} = M_1 \circ ... \circ M_l \circ B_2$. So $v.e \in \mathcal{L}(H)$. Contradiction.
- Suppose e is a receiving event. As $v.e \in \mathcal{L}(\mathcal{A}_H)$, any emission needed for executing e is performed in v. Again, $v.e \notin \mathcal{L}(H)$ implies that there is an event $e_j, j \in 1..k$ such that $\phi(e_j) = \phi(e) \wedge \forall i < j, \phi(e_i) \neq \phi(e)$, and $\exists (s_x, e_j, s''_x) \in \delta_x$. Still considering the locality of H, e_j is also a receiving event, and there is a choice between two branches B_1 and B_2,

$e_j = min_x(B_1)$, and $e = min(B_2)$. If $\phi(em(e)) = \phi(em(e_j))$, then according to the semantics of reception, e cannot be executed before e_j by $\mathcal{A}_H$. So, $\phi(em(e)) = \phi(em(e_j))$. Of course, $\phi(e)$ is not a deciding instance. So there is a choice c between B_1 and B_2, and a path $B_1 \circ B_2$, such that (e_j, e) can not be reconstructed. Contradiction with $Rec(H)$.

□

From theorem 14 and theorem 15, the following property holds true:

$$Loc(H) \Longrightarrow \Big(Rec(H) \Longleftrightarrow \mathcal{L}(\mathcal{A}_H) \subseteq \mathcal{L}(H)\Big)$$

Using theorem 8, this property becomes:

$$Loc(H) \Longrightarrow \Big(Rec(H) \Longleftrightarrow \mathcal{L}(\mathcal{A}_H) = \mathcal{L}(H)\Big)$$

Note that the proposition $Rec(H) \Longleftrightarrow \mathcal{L}(\mathcal{A}_H) = \mathcal{L}(H)$ is false (due to theorem 11). Example of Figure 14 exhibits a non-local specification where $\mathcal{L}(\mathcal{A}_H) = \mathcal{L}(H)$. Such a kind of specification contains hidden parallelism within choices (instances A, B and C, D never synchronize).

5.4 Decision of reconstructibility

This section provides an algorithm for deciding the reconstructibility of a HMSC H. The reconstructibility property can be decided on prefix of paths originating from a choice.

Proposition 16 *$\forall c$, choice of H, c is reconstructible if for any pair of branch B_i, B_j such that $\exists p$, path from c and $O_p = B_i \circ B_j$, for any non-deciding instance x, $(min_x(B_i), min_x(B_j))$ is reconstructible from $\leq_{B_i \circ B_j} - (E_{i_{|x}} \times E_{j_{|x}})$ by mt-closure.*

Algorithm:

for all c, `choice node of` H **do**
  $P = \{(c.n, M) | c \xrightarrow{M} n\}$
  $C = \emptyset$ `/* Cycles */`
  $MAP = \emptyset$ `/* Maximal loop-free paths */`
  while $P \neq \emptyset$ **do**
    $C = C \cup \{M \circ M' | (w = c.n_1..n_k, M) \in P \wedge n_k \xrightarrow{M'} c\}$
    $MAP = MAP \cup \{M \circ M' | (w = c.n_1..n_k, M) \in P \wedge n_k \xrightarrow{M'} n \wedge n \in w\}$
    $P = \{(w.n, M \circ M') | (w = c.n_1..n_k, M) \in P \wedge n_k \xrightarrow{M'} n \wedge n \notin w\}$
  end while
  for all $(B_i, B_j) \in C \times (MAP \cup C)$ **do**
    if $\exists x \in I | (min_x(B_i), min_x(B_j)) \notin (B_i \circ B_j - (B_{i|x} \times B_{j|x}))^{*mt}$ **then**
      Order can not be reconstructed
    end if
  end for
end for

6 Conclusion

This paper proposed a formal definition of non-local choice for HMSCs with weak sequential composition. Non-local choices can be detected on loop-free (and therefore finite) paths of the HMSC. Then a procedure for synthesizing CFSMs was proposed. The absence of non-local choices in a HMSC is not a sufficient condition for ensuring the synthesis produces a correct protocol: order reconstructibility is also required. Reconstructibility of an HMSC is also decidable on loop-free paths, and could be implemented as a front-end for protocol synthesis tools.

When a specification is non-reconstructible, some acknowledgment messages can be added to transform it into a reconstructible one. A possible extension of this work would be to consider an automation of these transformations. Adding stamps to messages when necessary could also avoid executing non-specified traces. Considering stamps as well as introducing co-regions (a weaker ordering on instances than total ordering) would probably need to modify the CFSM target model.

7 Acknowledgments

The authors are very grateful to professor Ferhat Khendek at Concordia University, for helpful discussions on formal aspects of synthesis from MSCs, and on his tool MSC2SDL. We are also very grateful to Benoît Caillaud for his comments on this work and suggestions for future extensions.

References

[1] M.Abdalla, F.Khendek,G.Butler, New Results on Deriving SDL Specifications from MSCs, in Proceedings of 9th SDL forum, R.Dssouli, G.Bochman & Y.Lahav editors, pp 51-66.

[2] H. Ben-Abdallah and S. Leue, Syntactic Detection of Process Divergence and non-Local Choice in Message Sequence Charts, in: E. Brinksma (ed.), Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'97, Enschede, The Netherlands, April 1997, Lecture Notes in Computer Science, Volume 1217, p. 259 - 274 Springer-Verlag, 1997

[3] D.Brand, P.Zafiropulo, On communicating Finite-State Machines. Journal of the ACM, vol 30, No 2, April 1983, pp 323-342.

[4] A. Engels, S. Mauw, M.A. Reniers: A Hierarchy of Communication Models for Message Sequence Charts. In: T. Mizuno, N. Shiratori, T. Higashino and A. Togashi (ed.): Formal Description Tecniques and Protocol Specification, Testing and Verification, Proceedings of FORTE X and PSTV XVII '97, pages 75-90, Osaka, Japan, 18-21 November 1997. Chapman & Hall.

[5] International Standard ISO 9074, Information Processing Systems - Open Systems Interconnection - Estelle: A Formal Description Technique Based on an Extended State Transition Model, Geneve, 1989.

[6] S.Leue, L.Mehrmann, M.Rezai, Synthesizing ROOM Models from Message Sequence Chart Specifications, 13th IEEE Conference on Automated Software Engineering, Honolulu, Hawaii, October 1998.

[7] N.Mansurov, D.Zhukov, Automatic Synthesis of SDL models in use case Methodology, in Proceedings of 9th SDL forum, R.Dssouli, G.Bochman & Y.Lahav editors, pp 225-240.

[8] OMG, Unified Modelling Language 1.1, September 1997.

[9] V.Pratt, Modeling Concurrency with Partial Orders, International Journal of Parallel Programming, Vol 15, No 1, 1986, pp 33-71

[10] K.Yamanaka, S.Komura, J.Kato, H.Ichikawa, Deriving Protocols from Message Sequence Charts in a Communicating Processes Model.

[11] TU-TS Recomendation Z.120: Message Sequence Chart 1996 (MSC96) Technical Report, ITU-TS, Geneva, 1996.