Automating scenario merging

Loïc Hélouët[1], Thibaut Hénin[2], Christophe Chevrier[3]

[1]IRISA/INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France
loic.helouet@irisa.fr
[2]ENS Cachan, Campus de Ker Lann, 35170 Bruz
thibaut.henin@irisa.fr
[3]France Télécom R&D, 2. av Pierre Marzin,22300 Lannion, France
christophe.chevrier@francetelecom.com

abstract: The requirement capture for the design of a distributed system often ends with a collection of use cases or scenarios, each of which illustrating a peculiar functionality or a typical execution of the system. The actual behavior of the system under design can be considered as a superposition of all use cases. However, current scenario languages do not propose such superposition mechanisms. An operator for Message Sequence Charts defined as a sum of MSCs has already been proposed. However, to compute such sum, the designer must provide explicitly a description of common parts (called the interface) in operands. This paper proposes a automatic construction of this interface. Many interfaces can be considered as candidates for a sum . However, we propose an heuristic that allows searching efficiently the best interfaces among possible solutions.

1 Introduction

Scenarios are a popular formalism describing runs of distributed systems. They often appear in protocol descriptions as illustrations of a system's use. They are also proposed to design use cases in the UML standard. Several scenario languages have been proposed: Message Sequence charts [4], Live Sequence Charts [2], UML's sequence diagrams [8]. MSCs, LSCs and sequence diagrams represent scenarios as compositions of basic diagrams which are more or less finite chronograms. A natural formal representation of these basic diagrams is labeled partial orders. In addition to the basic diagrams, all notations propose several composition operators such as sequence, iteration, alternative, etc.

These extensions are available in MSCs since 1996 [10], and came later for sequence diagrams in UML 2.0 [8]. However, even a high-level description is not sufficient to capture all the behaviors of a system. Of course scenarios can not be considered as a programming language, and their role is not to be exhaustive. Scenarios should rather be considered as an abstraction of a system's behavior, attached to a particular functionality, or to a particular point of view (the user's view of the system, for example). As a consequence, the requirement phase in design often ends with a collection of different scenarios describing the system from different points of view, hence with some similarities (see [3] for an example of scenario collection for the same system). The behavior of the system under study is not a parallel composition of these description. It can not be either defined as an alternative or a sequence, but rather as a superposition of scenarios. Such superposition operators have already been proposed for parallel programs by [1]. Figure 1 illustrates this situation: it depicts two bMSCs, M_1 and M_2. M_1 describes an exchange of three messages $m1$,$m2$ and $m3$, in this order. M_2 depicts only an exchange of two messages $m1$ and $m2$, but specifies that an internal action a must take place on instance A between the emissions of $m1$ and $m2$. The desired result is the bMSC M_3, that can be considered as a "sum" of M_1 and M_2. Note that as M_1 and M_2 do not specify any ordering between the emission of $m2$ and action a, these two events are kept unordered in the sum, and appear in a coregion, symbolized by the dashed part of instance A.

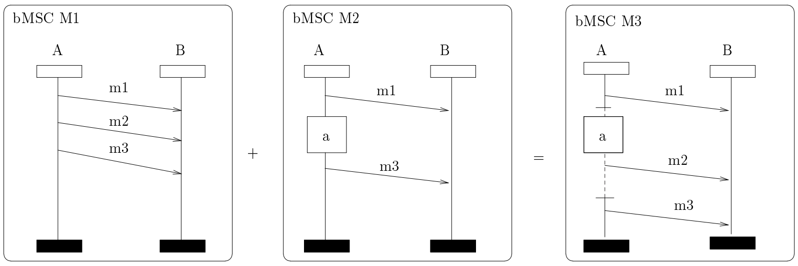


Fig. 1. Superposition of two scenarios

If all scenarios obtained after the requirement phase are seen as a partial view of an actual behavior, then a composition of two abstractions should be a more concrete scenario that refine both views. An operator called *amalgamated sum* has been proposed in [6]. To sum two bMSCs, the common part (also called *interface*) of the operands have to be defined. The result of the sum is an union of both scenarios where the common part is not duplicated. The main drawback so far is that the identical parts in operands have to be explicitly given before composition. Automating the product is not always possible, as there is more than one possible interface and amalgamated sum hence requires some directives from an user. In this work, we investigate how to compute efficiently the best possible interfaces between two bMSCs to reduce the work of end users. Note that [6] also defines a superposition operator for HMSCs, but this is out of the scope of this work. The paper is organized as follows: Section 2 recalls some usual notations on scenarios and defines the notion of amalgamated sum of bMSCs. Section 3 defines the notion of common part of two bMSCs and highlights some of their properties. Section 4 shows how the search for an interface can be brought back to an heuristic search algorithm. Section 5 gives the algorithms to compute common parts, Section 6 shows an example of automatic composition of bMSCs, and section 7 concludes this work. Proofs of propositions are provided as an appendix at the end of the paper.

2 Scenarios

Message Sequence Charts is a scenario formalism standardized by ITU [4]. It composed of two description levels. At the lowest level, basic Message Sequence Charts (bMSCs for short) define interactions among objects called *instances.* BMSCs can be considered as a formal definition of chronograms. In a bMSCs, interactions among instances are performed using asynchronous communications.

The second level of this formalism is called High-level Message Sequence Charts [10], and allows for the definition of more elaborated descriptions containing iterations, and alternatives. In this section, we will only consider bMSCs and their amalgamated sum. Formally, bMSCs can be considered as labeled partial orders [5] and can be defined as follows:

Definition 1. *A* bMSC *is a tuple $B = (E, \leq, A, I, \alpha, \phi, m)$, where:*

- *$E = E_S \cup E_R \cup E_A$ is a set of events that can be partitioned into a set of message emissions, a set of message receptions, and a set of atomic actions.*
- *$\leq$ is a preorder on E, i.e a transitive and reflexive relation*
- *A is a set of action names,*
- *I is a set of instances,*
- *$\alpha : E \longrightarrow A$ is a mapping that associate a label to each event.*
- *$\phi : E \longrightarrow I$ is a mapping that associates an instance name to each event. $\phi(e)$ will sometimes be called the locality of e.*
- *$m : E_S \longrightarrow E_R$ is a mapping that associates a message reception to each message emission.*

Note that $\leq$ is not necessarily a partial order, i.e. from our definition, a bMSC may not describe a valid execution of a system. When $\leq$ is a partial order, we will say that B is well-formed*. Let us also define $B_\epsilon = (E_\epsilon, \leq_\epsilon, A_\epsilon, I_\epsilon, \alpha_\epsilon, \phi_\epsilon, m_\epsilon)$, the empty bMSC, i.e. such that $E_\epsilon = \emptyset$.*

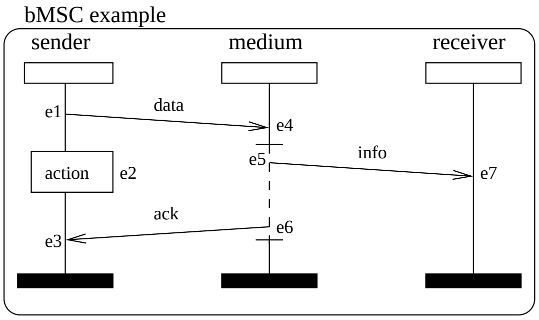


Fig. 2. An example of bMSC

Figure 2 shows an example of bMSC. Three instances $Sender$, $Medium$, and $Receiver$ exchange messages $Data$, ack, and $info$. An atomic action action is performed by instance Sender. Note that the ordering on an instance lifeline is not necessarily a total order. This is symbolized by a dashed part on the lifeline called a *coregion*. In our example, the emission of messages $info$ and ack are unordered.

For a given bMSC B, we will denote by $\downarrow (e) = \{e' \in E \mid e' \leq e\}$ the *causal past* of event e in B. Similarly, we will define the *future* of an event by $\uparrow (e) = \{e' \in E \mid e \leq e'\}$. A bMSC describes a set of *executions*, each of which is a linearization of its partial ordering (i.e an execution of a bMSC $B = (E, \leq, A, I, \alpha, \phi, m)$ is a word $w = e_1 \ldots e_{|E|}$ such that any event of E

appears exactly once in w, and furthermore, for all e_i, e_{i+k} in w, $e_{i+k} \not\leq e_i$). The set of linearizations of a bMSC will be denoted by $Lin(B)$. For a given linearization $w \in Lin(B)$, $w_{[i]}$ will denote the i^{th} event of word w.

The usual operators proposed for Message Sequence Charts are sequential composition, parallel composition, alternative and iteration. As already mentioned, these operators do not allow for a composition that would consider some redundancy between two bMSCs. In a recent paper, [6] has proposed an operator called *amalgamated sum*, that allows for the kind of composition we are looking for. In this composition, the common part between two bMSCs B_1 and B_2 is defined by another bMSC called the *interface*, that is related to B_1 and B_2 by bMSC morphisms. Roughly speaking, a bMSC morphism $f : B_I \longrightarrow B$ from an interface B_I to a bMSC B exists if and only if B_I is "contained" in B, and describes how B_I is injected in B.

Definition 2. *Let* $B_1 = (E_1, \leq_1, A_1, I_1, \alpha_1, \phi_1, m_1)$, $B_2 = (E_2, \leq_2, A_2, I_2, \alpha_2, \phi_2, m_2)$ *be two bMSCs. A* morphism *from* B_1 *to* B_2 *is a triple* $f = (f_1, f_2, f_3)$*, where:*

- $f_1 : E_1 \longrightarrow E_2$ *is an injective function from* E_1 *to* E_2.
- $f_2 : A_1 \longrightarrow A_2$ *is an injective mapping from* A_1 *to* A_2.
- $f_3 : I_1 \longrightarrow I_2$ *in an injective mapping from* I_1 *to* I_2.

Furthermore, the morphisms should satisfy the following properties:

i) $\forall e, e' \in E_1, e \leq_1 e' \Rightarrow f_1(e) \leq_2 f_1(e')$ *: the ordering among events is preserved by the morphism.*
ii) $\forall e, e' \in E_1, m(e) = e' \Rightarrow m(f_1(e)) = f_1(e')$ *: messages are preserved by the morphisms*
iii) $\forall e_1, \in E_1, e_2 \in E_2$ *such that* $e_2 = f(e_1)$, $\exists e'_2 \in E_2, m(e_2) = e'_2 \Longrightarrow \exists e'_1 \in E_1$ *such that* $m(e_1) = e'_1 \wedge f(e'_1) = e'_2$ *and* $\exists e'_2 \in E_2, m(e'_2) = e_2 \Longrightarrow \exists e'_1 \in E_1$ *such that* $m(e'_1) = e_1 \wedge f(e'_1) = e'_2$*. This condition means that messages are mapped integrally by bMSC morphisms, and that we can not associate a single event with a message emission or reception.*
iv) $f_3 \circ \phi_1 = \phi_2 \circ f_1$*: This property means that the locality of events is coherent: two events located on the same instance in* B_1 *will be located on the same instance in* B_2.
v) $f_2 \circ \alpha = \alpha' \circ f_1$ *: This property means that the labeling of events remains coherent through morphisms. Two events with identical label will also have images with identical labels.*

Note that in this definition, nothing forces labeling or locality to be similar in B_1 and B_2. Figure 3 shows two examples of bMSC Morphisms. Consider the leftmost morphism. Event morphism is represented as a plain arrow and instance morphism as a dashed arrow. For the sake of clarity, labels morphism is not represented, but is obvious on this example. bMSC $Interface$ can clearly be injected in bMSC $B1$. Each event of $Interface$ has an image in $B1$, messages and order are preserved. Instances $Sender$ and $Medium$ are renamed by the instance mapping into X and Y.

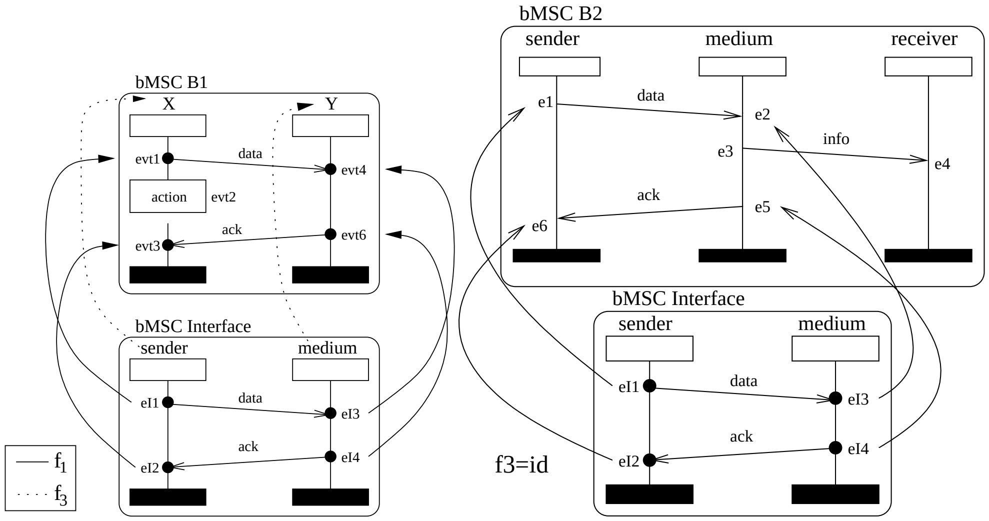


Fig. 3. Two example morphisms

Definition 3 (Amalgamated Sum of Two Sets). *Let I, J and K be three finite sets. Let $f : I \rightarrow J$ and $g : I \rightarrow K$ be two injective maps. The amalgamated sum $J_f{+}_gK$ is defined as $J_f{+}_gK = \big(J \backslash f(I)\big) \biguplus \big(K \backslash g(I)\big) \biguplus I$. The amalgamated sum yields two injections $\tilde{f} : J \rightarrow J\ _f +_g K$ and $\tilde{g} : K \rightarrow J\ _f +_g K$ defined as follows:*

$$\begin{cases} \forall i \in f(I), & \tilde{f}(i) = f^{-1}(i) \\ \forall i \in J \setminus f(I), & \tilde{f}(i) = i \end{cases} \qquad \begin{cases} \forall i \in g(I), & \tilde{g}(i) = g^{-1}(i) \\ \forall i \in K \setminus g(I), & \tilde{g}(i) = i \end{cases}$$

Amalgamated sum of sets is a disjoint union of these sets where events that are identified through the interface are not duplicated. It will be used to amalgamate sets of instances, events or actions of two bMSCs.

Definition 4. *Let B_1,B_2, B_I be three bMSCS and let $f : B_I \longrightarrow B_1$ and $g : B_I \longrightarrow B_2$ be two bMSC morphisms. The* amalgamated sum *of B_1 and B_2 with respect to interface B_I and morphisms f and g is denoted $B_1\ _f +_g B_2$, and is defined as $B_1\ _f +_g B_2 = (E, \leq, A, I, \alpha, \phi, m)$, where:*

- $E = E_1\ _{f_1} +_{g_1} E_2$
- $\leq$ *is the preorder relation obtained by transitive closure of* $\tilde{f}_1(\leq_1) \cup \tilde{g}_1(\leq_2)$;
- $A = A_1\ _{f_2} +_{g_2} A_2$
- $I = I_1\ _{f_0} +_{g_0} I_2$
- $\forall e \in E, \alpha(e) = \begin{cases} \alpha_1(e) \ \textit{if}\ e \in E_1 \backslash f_1(E_I) \\ \alpha_2(e) \ \textit{if}\ e \in E_2 \backslash f_2(E_I) \\ \alpha_0(e) \ \textit{otherwise} \end{cases}, \qquad \phi(e) = \begin{cases} \phi_1(e) \ \textit{if}\ e \in E_1 \backslash f_1(E_I) \\ \phi_2(e) \ \textit{if}\ e \in E_2 \backslash f_2(E_I) \\ \phi_0(e) \ \textit{otherwise} \end{cases}$
- $m = \tilde{f}_1(m_1) \cup \tilde{g}_1(m_2)$

Using the two morphisms of Figure 3, the amalgamated sum between bMSCs $B1$ and $B2$ with bMSC $Interface$ produces the bMSC of Figure 2. Note that $\leq$

is not always acyclic, even when B_1 and B_2 are well-formed bMSCs. We will say that an amalgamated sum is *well-formed* whenever $\leq$ is a partial order relation (transitive, reflexive, antisymmetric). Figure 4 shows an example of amalgamated sum for which the ordering between actions a and b are inconsistent. The result produces a bMSC that is not well-formed.

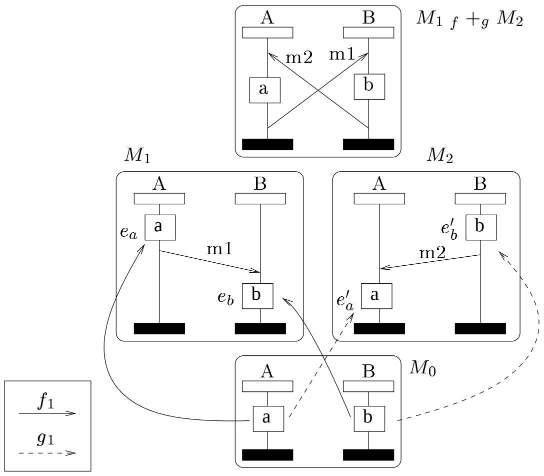


Fig. 4. An amalgamated sum that is not well-formed

Proposition 1. *Let B_1, B_2, B_I, be three bMSCs and $f : B_I \longrightarrow B_1, g : B_I \longrightarrow B_2$ be two bMSC morphisms. Let us denote by $\mathcal{P}_{f,g} : E_1 \longrightarrow E_2$ the partial bijective function that pairs events of E_1 and E_2, i.e $\mathcal{P}_{f,g}(e_1) = e_2$ iff $\exists e_i \in B_I, f(e_i) = e_1$ and $g(e_i) = e_2$ The following propositions are equivalent:*

1. *$B_1 \; {}_f+_g \; B_2$ is well-formed*
2. $\forall e_1 \in f(B_I), \uparrow(e_1) \cap \mathcal{P}^{-1}_{f,g}\left(\downarrow\left(\mathcal{P}_{f,g}(e_1)\right)\right) = \{e_1\}$
3. $\forall e_2 \in g(B_I), \uparrow(e2) \cap \mathcal{P}_{f,g}\left(\downarrow\left(\mathcal{P}^{-1}_{f,g}(e_2)\right)\right) = \{e_2\}$

From this proposition, we know that checking for well-formedness of an amalgamated sum resumes to finding connected components in $\leq_2 \cup \leq_2 \cup \mathcal{P} \cup \mathcal{P}^{-1}$. This can be done in polynomial time using Tarjan's algorithm [11]. Note however that the complexity of finding a valid interface from a pair of bMSCs is due to the number of possible interfaces more than to the complexity of checking whether $B_1 \; {}_f+_g \; B_2$ is well-formed.

3 Common Part

Intuitively, in an amalgamated sum, the interface bMSC B_I represents the common part of B_1 and B_2. The amalgamated sum merges B_1 and B_2 without duplicating the common part. When an amalgamatehed sum is not well formed, then it means that operands do not agree of the respective orders of common events. Then either some events that were identified in the interface should not be considered identical, or the two descriptions are not compatible and should not be composed. For a well-formed amalgamated sum $B_1\, {}_f\!+_g B_2$, we will often call the triple (B_I, f, g) the *common part* of B_1 and B_2. The empty common part C_ϵ is a common part such that $B_I = B_\epsilon$.

Automating the amalgamated sum resumes to finding common parts for a pair of bMSCs B_1, B_2. Note however that the design of an interface for a couple of bMSCs can be a difficult task. In our definition, nothing forces the interfaced events to have the same label, nor to be located on the same instance. The only restriction is the general case is that the bMSC morphisms should be injective. That is, an interface can have the size of the smallest operand of the sum. Kleitman and Rotschild gave a bound for the number of partial orders of size n which is in $O\left(2^{\frac{n^2}{4}\cdot\frac{3n}{2}.ln(n)}\right)$ [7]. Note however that the number of interfaces of size n describing the behavior of l instances (without considering labeling, that may yeld an infinite number of different bMSCs) is much lower, as it is in l^n (everytime you add an event, it can be located on l instances, and ordering does not matter as it can be deduced from the ordering in $B1, B2$). Anyway, this number remains too big to imagine searching exhaustively for the whole set of interfaces and morphisms: if m is the size of the minimal operand (and M the size of the largest operand) for an amalgamated sum, the number of possible interfaces is $\underset{i\in 1..m}{\Sigma} l^i$, and the worst number of possible common parts is hence in $\underset{i\in 1..m}{\Sigma} l^i.\frac{m!}{(m-i)!}.\frac{M!}{(M-i)!}$. A sensible restriction of the amalgamated sum is to consider that when two events in B_1 and B_2 are the image of a single event in the interface, then they are located on the same instance and have identical label. i.e, $\forall e_1 \in E_1, e_2 \in E_2$, when $\exists e \in E_I, f_1(e) = e1$ and $g_1(e) = e_2$ we have $\phi(e_1) = \phi(e_2)$ and $\alpha(e_1) = \alpha(e_2)$. With this restriction in mind, the interface can be built using the same labels and the same instances as the summed operands (i.e. $A_I \subseteq A_1 \cap A_2$ and $I_I \subseteq I_1 \cap I_2$). Hence, if we call $K = |\alpha^{-1}(A_1 \cap A_2) \cap \phi^{-1}(I_1 \cap I_2)|$, the number of possible interfaces becomes $\underset{i\in 1..K}{\Sigma} l^i.\frac{m!}{(m-i)!}.\frac{M!}{(M-i)!}$. So, from now, we will consider that in any bMSC morphism $f = (f_1, f_2, f_3)$ $f_2 = id$ and $f_3 = id$. But even with this restriction, the number of common parts remains huge.

Now, we have to remind that the main justification of amalgamated sum is to exploit redundancy in different views. Hence a common part with a large interface can be considered as better than the empty common part C_ϵ, but also as any smaller interface. The notion of *extension* of a common part captures this intuition, and provides a structure to the set of common parts.

Definition 5. *Let $C = (B_I, f, g)$ be a common part for a pair of bMSCs B_1, B_2. We will say that $C' = (B'_I, f', g')$ is an* extension *of C and write $C \sqsubseteq C'$ if and only if:*

- *(B'_I, f', g') is a common part of B_1, B_2*
- *$E_I \subseteq E'_I$*
- *$\forall e \in E_I, f'(e) = f(e)$ and $g'(e) = g(e)$*

We will say that (B'_I, f', g') is a sequential extension *of (B_I, f, g) when no event of $E'_I \setminus E_I$ (respectively $f'(E'_I \setminus E_I)$ and $g'(E'_I \setminus E_I)$) is located in the causal past of E_I (respectively $f(E_I)$ and $g(E_I)$).*

For a common part C, let us denote by $Ext(C)$ the set of extensions of C and by $SExt(C)$ the set of sequential extensions of C. These extension sets contain a set of maximal elements $\widehat{C}$ and $\overline{C}$ such that $Ext(\widehat{C}) = \widehat{C}$ and $SExt(\overline{C}) = \overline{C}$.

Proposition 2. *For any pair of bMSCs B_1, B_2, $\widehat{C_\epsilon} \subseteq \overline{C_\epsilon}$.*

This proposition gives the intuition for an efficient and incremental search for interfaces. The set of common parts is in fact the search space that has to be explored to produce the best interfaces. $(Ext(C_\epsilon), \sqsubseteq)$ and $(SExt(C_\epsilon), \sqsubseteq)$ form two lower semi-lattices. However, the size of $SExt(C_\epsilon)$ is lower than the size of $Ext(C_\epsilon)$. Furthermore, the maximal elements in the lattice of extensions are also maximal elements in the lattice of sequential extensions, i.e. no optimal solution is lost. This means that the best maximal common part can be built incrementally (and more efficiently) by exploration of a linearization of one of the operands. Let us clarify this intuition with the example bMSCs of figure 5. The semi-lattices of common parts extensions and sequential extensions for this pair of bMSCs are represented in Figure 5. The table in Figure 5 describes how events are matched in each common part: common part $C1$ identifies two events e_1 and e'_1, $C2$ identifies e_2 and e'_2, C3 e_3 and e'_3, etc. We know that $e_1 \leq e_2 \leq e_4 \leq e_3$, and furthermore, no common part may contain at the same time e_2 and e_4. Note also that events associated to messages m, n, o and t, u, v never appear in a common part (one can not find event with similar labels). The leftmost graph in Figure 5 shows the Hasse diagram of a possible extension semi-lattice: vertices represent common parts, and an arrow between two vertices C_i, C_j indicates that $C_i \sqsubseteq C_j$. Maximal elements in this semi-lattice are depicted by squares. The rightmost graph shows the semi-lattice of sequential extensions. Note that due to the ordering among events, some edges do not appear anymore in the diagram. Hence, some common parts (such as $C3$, $C6$, and $C7$) become maximal w.r.t the sequential extension relation. However, they are not maximal w.r.t extension, and most of them will be ignored during the heuristic search.

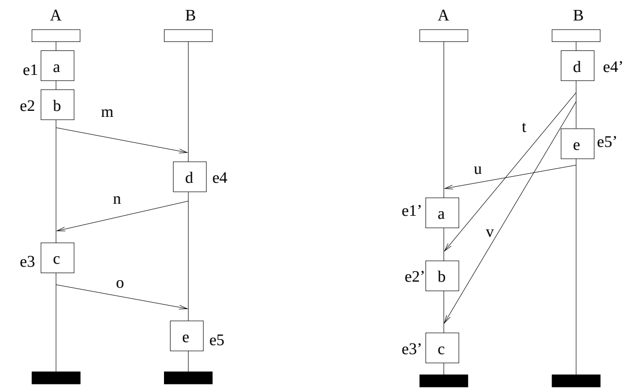
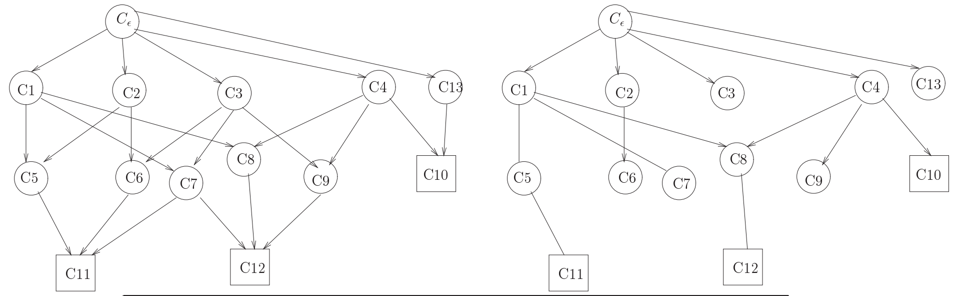


Fig. 5. A pair of bMSCs



$C1 : \mathcal{P}(\{e_1\}) = \{e'_1\}$	$C7 \;: \mathcal{P}(\{e_1, e_3\}) = \{e'_1, e'_3\}$
$C2 : \mathcal{P}(\{e_2\}) = \{e'_2\}$	$C8 \;: \mathcal{P}(\{e_1, e_4\}) = \{e'_1, e'_4\}$
$C3 : \mathcal{P}(\{e_3\}) = \{e'_3\}$	$C9 \;: \mathcal{P}(\{e_3, e_4\}) = \{e'_3, e'_4\}$
$C4 : \mathcal{P}(\{e_4\}) = \{e'_4\}$	$C10 : \mathcal{P}(\{e_4, e_5\}) = \{e'_4, e'_5\}$
$C5 : \mathcal{P}(\{e_1, e_2\}) = \{e'_1, e'_2\}$	$C11 : \mathcal{P}(\{e_1, e_2, e_3\}) = \{e'_1, e'_2, e'_3\}$
$C6 : \mathcal{P}(\{e_2, e_3\}) = \{e'_2, e'_3\}$	$C12 : \mathcal{P}(\{e_1, e_3, e_4\}) = \{e'_1, e'_3, e'_4\}$
	$C13 : \mathcal{P}(\{e_5\}) = \{e'_5\}$

Fig. 6. Semi lattices of extensions and sequential extensions

4 Automation of MSC sums

According to [9], performance of heuristic search depends on two criterions: a good representation and organization of the search space, and a good evaluation function for a place in this search space. For each place of the search space, we have to be able to compute a set of successors, and chose to explore the more promising among them (w.r.t. the evaluation function). The semi-lattice of sequential extensions is smaller than the lattice of extensions. Furthermore, the maximal elements w.r.t extension are the same in both lattices, and the extension relation guarantees that we are progressing toward a better solution. Hence the lattice of semi-extensions can be considered as a good organization of the search space. A good way to explore the lattice of sequential extensions of a pair of bMSCs B_1,B_2 is to study events of the smallest operand according to the causal ordering (i.e. chose a peculiar linearization). For each event e, we have to find a similar event e' in the other operand (same label and same instance), and make sure that after identifying e and e', the amalgamated sum of B_1 and B_2 is still well-formed.

In the sequel, we will denote by $B \circ \{e\}$ the bMSC obtained by sequential concatenation of B and an event e, and slightly abusing the morphisms notation, we will denote by $f' = f \cup \{(e, e')\}$ the morphism f' defined on $dom(f) \cup \{e\}$ such that : $f'(x) = \begin{cases} f(x) \text{ if } x \neq e \\ e' \text{ otherwise} \end{cases}$

Proposition 3. *Let $C = (B, f, g)$ be a common part of B_1, B_2, let e be an atomic action of $E_1 \setminus \downarrow f(E_B)$ and e' be an atomic action of $E_2 \setminus \downarrow g(E_B)$ such that $\phi(e) = \phi(e')$ and $\alpha(e) = \alpha(e')$. Let e_i be a freshly created event such that $\phi(e_i) = \phi(e)$ and $\alpha(e_i) = \alpha(e)$. Then, $C' = (B \circ \{ei\}, f \cup \{(e_i, e)\}, g \cup \{(e_i, e')\})$ is a sequential extension of C.*

This property shows that atomic actions can be easily paired as long as they do not appear in the past of an already matched event. Hence, searching for a paired event for an atomic action resumes to finding this event in the future of already matched events.

Proposition 4. *Let $C = (B, f, g)$ be a common part of B_1, B_2, and let e be an event of $E_1 \setminus \downarrow f(E_B)$ and e' be an event of $E_2 \setminus \downarrow g(E_B)$ such that $\phi_1(e) = \phi_2(e')$ and $\alpha_1(e) = \alpha_2(e')$. If e is a message emission, then there is an extension $C' = (B', f', g')$ of C containing e and $m(e)$ iff:*

- $\exists e' \in E_2 \setminus \downarrow g(E_B)$ *such that* $\phi(e) = \phi(e')$ *and* $\alpha(e) = \alpha(e')$.
- $\exists e'' \in E_2 \setminus \downarrow g(E_B)$ *such that* $\phi(m(e)) = \phi(e'')$ *and* $\alpha(m(e)) = \alpha(e'')$.
- $m(e') = e''$
- $\mathcal{P}^{-1}_{f',g'}(\downarrow (e'')) \cap \mathcal{P}_{f',g'} \uparrow (e) = e$

This proposition means that if we try to build a common part by exploring a linearization of one of the operands and matching its events one after another, then we can not conclude immediately that a message emission will appear in a common part, and we have to study its reception before concluding. Figure 7 shows a situation where events e_1 and e'_1 can not be identified through an interface when e_2 and e'_2 have already been matched.

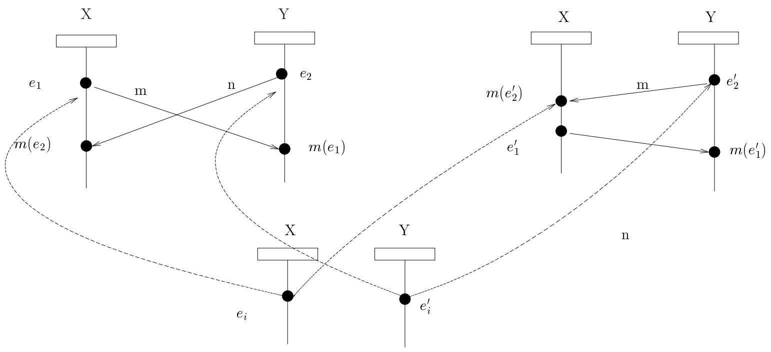


Fig. 7. Matching messages

With this properties, we know how to explore the semi-lattice of sequential extensions, following a linearization of a bMSC. However, this exploration is still a complete one, and we need a way to discriminate common parts which extensions can not be among the best ones to avoid exploring the whole lattice. This can be done through the definition of a quality criterion to guide the exploration and abandon hopeless interfaces. Of course, to fit the properties of extensions and their lattices, quality should be a function of the size of a common parts.

Definition 6. *The* quality *of a common part is the ratio* $Q(B_I, f, g) = \frac{|B_I|}{min(|B_1|,|B_2|)}$.

With the definition of a quality criterion, the search for the best common parts can be defined as the construction of the set
$C_{Best} = \arg\max\{Q(B_I) \mid \exists (B_I, f, g)$ common part of $B_1, B_2\}$.

Proposition 5. *Let B_1, B_2 be two bMSCs, and let C_{Best} be the best common parts found w.r.t quality Then $C_{Best} \subseteq \overline{C_\epsilon}$*

Note however that this property only holds when the quality of an interface is a function of its size. The main idea is to avoid searching the whole space of possible common parts by pruning systematically parts of the lattice for which the best reachable sequential extensions will have a lower quality than the best quality already computed. And of course, this decision must be taken without exploring the abandoned part. We adopt a classical approach in operational research: common parts are built sequentially and incrementally. The exploration is guided by the quality that must be maximized on all maximal common parts, and by an heuristic, that gives an upper approximation of the maximal quality reachable from a common part. For a detailed survey on heuristics, interested readers may consult [9].

The estimated quality $\mathcal{E}Q(C) = qs(C) + h(C)$ of an common part C is an over approximation of the quality of any extension of C. It is defined as the sum of two functions: a fixed value $qs(C)$, the quality of C (i.e. the number of events matched for sure so far and in any extension), and an heuristic $h(C)$ that is an over approximation of the number of events that can still be matched in an extension of C. At each step, we chose to extend the part that has the best estimated quality.

The heuristic should satisfy two main principles :

- when a common part C is maximal, then $h(I) = 0$.
- For any extension C' of a common part C, $qs(C') + h(C') \leq qs(C) + h(C)$

The first requirement means that when no extension is possible for a common part C, then the estimated quality which is also the best quality reachable should be the actual quality of C. The second requirement means that the heuristic should always provide an over-approximation of the quality that may be achieved from a common part. This way, we can be sure that a path leading to a solution is never forgotten due to underestimation of the reachable maximal quality. Hence, the problem of automating the amalgamated sum is brought back to an exploration of a subset of common parts such that if a common part C' is explored after a common part C, then $\mathcal{E}Q(C') \geq \mathcal{E}Q(C)$.

During the exploration we will study events one after another in a linearization of the smallest operand. To keep track of the events explored so far, we just need to recall an integer n. However, we can not be sure that a message emission will be eventually matched in the maximal extension reachable from the state of our research. Hence, we will keep some events as matched for sure in an interface B_I, with associated morphisms f and g, and we will also memorize temporary events in another potential interface B'_I, with associated morphisms f' and g'. During the exploration, we will compute an estimated quality associated to an exploration state rather that to a common part (in a state, some potential solutions have already been ruled out). For a pair of bMSCS B_1, B_2, and in a given state $s = (n, B_I, B'_I, f, g, f', g')$, we will have $\mathcal{E}Q(s) = qs(s) + h(s)$, where:

$$qs(n, B_I, B'_I, f, g, f', g') = \frac{|B_I|}{min(|B_1|, |B_2|)}$$

, and

$$h(n, B_I, B'_I, f, g, f', g') = \frac{|B'_I| + \left| \left\{ \begin{array}{l} e \in E_1 \setminus \downarrow (f(B_I) \cup f(B'_I)) \mid \\ \quad \exists e' \in E_2 \setminus \downarrow (g(B_I) \cup g'(B'_I)), \\ \quad \alpha(e) = \alpha(e') \wedge \phi(e) = \phi(e') \end{array} \right\} \right|}{min(|B_1|, |B_2|)}$$

Here, function qs describes the ratio between the number of event matched in a state and the size of bMSCs. Function h gives the ratio between events that may still be matched in an exploration state and the size of bMSCs. Obviously, the estimated quality of common part can only decrease with sequential extension. Hence, if we have a maximal sequential extension $C \in \overline{C_\epsilon}$, and an extension C' such that $Q(C) \geq \mathcal{E}Q(C')$. Then, for any sequential extension C'' of C', $Q(C) \geq \mathcal{E}Q(C'')$. This also holds for exploration states, which means that during explorations, some common parts do not have to be extended, as we already know (using the evaluation function) that the quality of their sequential extensions will be lower than a local maxima that was previously discovered.

5 Algorithm

From the results of previous section, the exploration algorithm is rather straightforward : start from the empty common part, and find its best extensions. The heuristic is used to avoid exploring common parts for which we already know a better extension of C_ϵ. Extensions are computed as follows : events are studied one after another. For each event, we can decide to match it or not, hence creating two exploration paths. For atomic actions, events can be matched without the risk to create a cyclic preorder in the sum $B_1 \,_f{+}_g\, B_2$. Whether an emission can be matched depends on the future extensions, hence these matching are just potential. A reception must be matched if the corresponding emission is matched and if it does not create a cycle in the preorder relation of the amalgamated sum.

The complete exploration algorithms are described by algorithms 1 and 2 next pages. Note that the algorithm does not exactly explore the set of extensions, as when a message emission is under study, we have to guess whether it should be matched or not (this guess is depicted by B'_I). Note also that the heuristic allows to rule out automatically some parts of the search space. For the example of Figure 5, the linearization among events is $e1.e2.!m.?m.e4.!n.?n.e3.!o.?o.e5$, and the set of extensions explored would be $C_\epsilon, C1, C2, C4, C5, C6, C7, C8, C11, C12$. The best common parts computed by the algorithm are $C11$ and $C12$.

6 Example

Let us show on a concrete example how amalgamated sum can be used to compose services. Figure 8 presents an example of a cartography service build over a SOA architecture. In this kind of architecture, a service is described as an orchestration of basic services. Each basic service is designed in an independent way. The expected behavior is thus a combination of the individual basic services behaviors.

Algorithm 1 BestCommonParts(B_1, B_2)

Chose a linearization $w \in Lin(B_1)$
$T = \{(0, \emptyset, \emptyset, f_\epsilon, g_\epsilon, f_\epsilon, g_\epsilon)\}$
$Best = \{\}$
$fbest = 0$
while $T \neq \emptyset$ **do**
  select $t \in T$ such that $\forall t' \in T, qs(t') + h(t') \leq qs(t) + h(t)$
  $T = T \setminus \{t\}$
  $Succ = Successors(t, w)$
  for all $s \in Succ$ **do**
    if s is a leaf **then**
      if $qs(s) \geq fbest$ **then**
        $fbest = qs(s)$
        $Best = \{s' \in Best \cup \{s\} \mid qs(s') \geq fbest\}$
      end if
    else
      $T = T \cup \{s\}$
    end if
  end for
  $T = \{t \in T | qs(t) + h(t) \geq fbest\}$
end while
$C_{Best} = \emptyset$
for all $(n, E_I, E'_I, f, g, f', g') \in Best$ **do**
  $B = (E_I, \{(e, e') \mid f(e) \leq_1 f(e') \wedge g(e) \leq_2 g(e')\}, \phi(E_I), \alpha(E_I), \alpha_1 \circ f, \phi_1 \circ f, m_1 \circ f)$
  $C_{Best} = C_{Best} \cup \{(B, f, g)\}$
end for
return (C_{Best})

BMSC $UserServiceInteraction$ (USI) describes the behavior expected by an user. BMSC $ServiceDirectoryInteraction$ (SDI) describes the service from the directory provider's point of view: when a user asks for a map, the service call the directory provider for the destination address. BMSC $ServiceLocalizationInteraction$ (SLI) describes the relation between the service and the localization provider: when a user asks for a map and the destination address has been found, the service ask the localization provider for the user's location. Finally, the last bMSC $ServiceMapProviderInteraction$ (SMPI) describes the the service as seen by the Map Provider: when the destination address and the user location have been found, the service asks for a map.

All these bMSCs can be automatically composed to obtain a single behavior, that refines the UserServiceInteraction. The bMSC provided in Figure 9 can be computed as the amalgamated sum $((USI\,{}_{f1_{g}1}\!+_S DI)\,{}_{f2}\!+_{g2} SLI)\,{}_{f3}\!+_{g3} SMPI$. Note that in this example, the best common parts are unique, and hence do not require interaction with an user. This may not always be the case, and the integration order for bMSCs may also influence the final result.

Algorithm 2 Successor$\big((n, I, I', f, g, f', g'), w)\big)$

where:

- n number of events studied so far in the linearization of B_1
- E_I events of the interface computed so far
- E_I' events that may appear in the maximal interface reachable from current state (essentially sending events).
- $f : E_I \longrightarrow E_1$ morphism from I to E_1
- $g : E_I \longrightarrow E_2$ morphism from I to E_2
- $f' : E_I' \longrightarrow E_1$ potential morphism from I' to E_1
- $g' : E_I' \longrightarrow E_2$ potential morphism from I' to E_2

$Succ = \emptyset$
choose $e_1 = w_{[n+1]}$
if e_1 is an atomic action **then**
    $Succ := Succ \cup \{(n+1, E_I, E_I', f, g, f', g')\}$
    /* we can choose not to match an event */
    $Match = \{e_2 \in E_2 \setminus \downarrow (g(E_I) \cup g'(E_I'))$ such that $\phi(e') = \phi(e_2)$ and $\alpha(e_2) = \alpha(e')$ and $\uparrow (e_1) \cap \mathcal{P}_{f,g}^{-1} \downarrow (\mathcal{P}_{f,g}(e_1)) = e_1\}$
    for all $e_2 \in Match$ **do**
        Create a new event e_i
        $Succ := Succ \cup \{n+1, E_I \cup \{e_i\}, E_I', f \cup \{(e_i, e_1)\}, g \cup \{(e_i, e_2)\}, f', g')\}$
    end for
end if
if e_1 is a message emission **then**
    $Succ := Succ \cup \{(n+1, E_I, E_I', f, g, f', g')\}$
    /* we can choose not to match an event */
    $Match = \{e_2 \in E_2 \setminus \downarrow (g(E_I) \cup g'(E_I'))$ such that $\phi(e') = \phi(e_2)$ and $\alpha(e_2) = \alpha(e')$ and $\uparrow (e_1) \cap \mathcal{P}_{f,g}^{-1} \downarrow (\mathcal{P}_{f,g}(e_1)) = e_1\}$
    for all $e_2 \in Match$ **do**
        Create a new event e_i
        $Succ := Succ \cup \{(n+1, E_I, E_I' \cup \{e_i\}, f, g, f' \cup \{(e_i, e_1)\}, g' \cup \{(e_i, e_2)\})\}$
    end for
end if
if e_1 is a reception **then**
    if $m^{-1}(e_1) \notin dom(f')$ **then**
        /* The emission corresponding to e_1 was not matched */
        /* Hence e_1 must not be matched */
        $Succ := Succ \cup \{(n+1, E_I, E_I', f, g, f', g')\}$
    else
        $e_i = f'^{-1}(m^{-1}(e_1))$
        /* e_i is the representation of the emission of message */
        /* received in e_1 in the interface */
        $e_2 = m(g'(e_i))$
        if pairing e_1, e_2 does not create cycles in the sum **then**
            create a new event e_i'
            $Succ := Succ \cup \left\{ \begin{pmatrix} n+1, E_I \cup \{e_1, e_2\} \\ f \cup \{(ei, f'(e_i))\} \cup \{(e_i', e_1)\}, \\ g \cup \{(ei, g'(e_i))\} \cup \{(e_i', e_2)\}, \\ f' \setminus \{(ei, f'(e_i))\}, \\ g' \cup \{(ei, g'(e_i))\} \end{pmatrix} \right\}$
        else
            $Succ := Succ \cup \left\{ \big(n+1, E_I, E_I', f, g, f' \setminus \{(ei, f'(e_i))\}, g' \cup \{(ei, g'(e_i))\}\big) \right\}$
        end if
    end if
end if

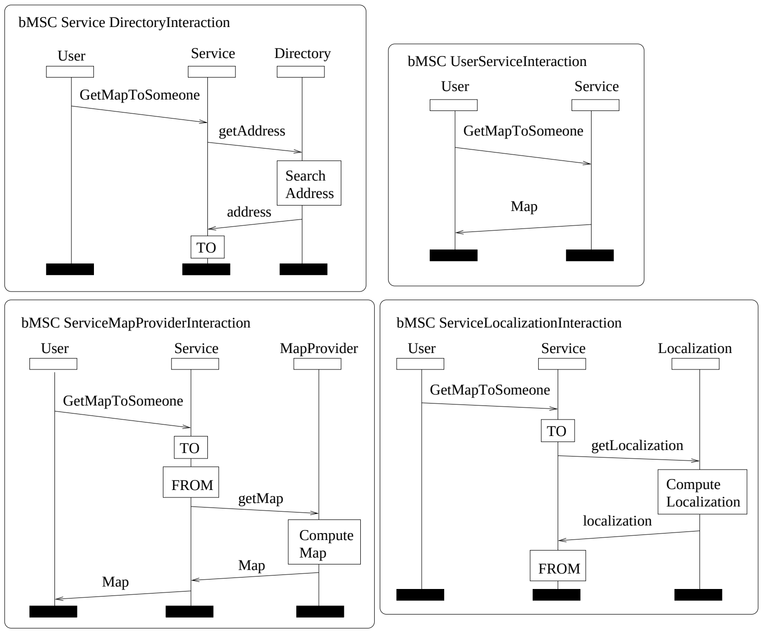


Fig. 8. A complete example

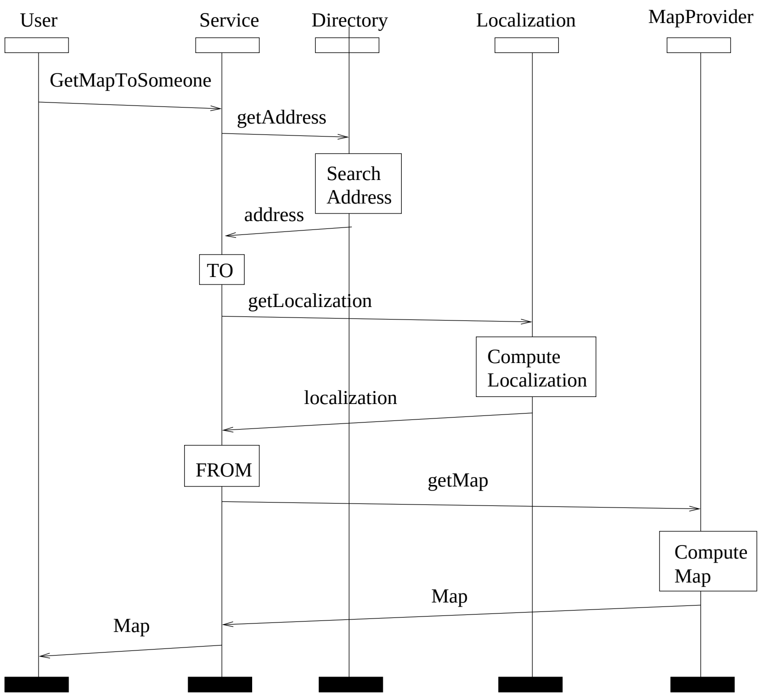


Fig. 9. Auomatic Composition from bMSCs of Figire 8

7 Conclusion

We have proposed an algorithm to compute efficiently the common part of two bMSCs. The method proposed consists in searching with an heuristic for the best possible interfaces. At the end of the algorithm, the result may not be unique, and the final choice of which interface to chose to compose the bMSCs is left to the end-user. However, we believe that the number of possible choices should remain small. We have shown that in the worst case, the number of possible solutions that have to be searched in really huge. The heuristic proposed does not guarantee that the complete search space is not explored. However, operational research algorithms have shown good average complexities. Hence, the automatic composition problem should be feasible in practice. A possible extension of this work is to consider similar automation for HMSCs.

References

1. K. M. Chandy and J. Misra. *Parallel Program Design, A Foundation*. Addison-Wesley, 1988.
2. W Damm and D. Harel. Live sequence charts: Breathing life into message sequence charts. *Journal of System Design*, 2001.
3. L. Hélouët. Distributed system requirements modeling with message sequence charts. *International Journal of Information and Software Technology*, 2002.
4. ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1999.
5. J.P. Katoen and L. Lambert. Pomsets for message sequence charts. In *Proceedings of SAM98:1st conference on SDL and MSC*, pages 281–290, Berlin, Juin 1998.
6. J. Klein, B. Caillaud, and L. Hélouët. Merging scenarios. In *9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, number 133 in ENTCS, pages 209–226, sep. 2004.
7. D.J Kleitman and B.L Rotschild. Asymptotic enumeration of partial orders. *Transactions of the American Mathematical Society*, (205):205–220, 1975.
8. OMG. Uml superstructure specification, v2.0. OMG Document number formal/05-07-04, 2005.
9. J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1995.
10. E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.
11. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2), 1972.

Appendix: Proofs

Proposition 1:

Let B_1, B_2, B_I, be three bMSCs and $f : B_I \longrightarrow B_1, g : B_I \longrightarrow B_2$ be two bMSC morphisms. Let us denote by $\mathcal{P}_{f,g} : E_1 \longrightarrow E_2$ the partial bijective function that pairs events of E_1 and E_2, i.e $\mathcal{P}_{f,g}(e_1) = e_2$ iff $\exists e_i \in B_I, f(e_i) = e_1$ and $g(e_i) = e_2$ The following propositions are equivalent:

1. $B_1 \ {}_f+_g \ B_2$ is well-formed
2. $\forall e_1 \in f(B_I), \uparrow(e_1) \cap \mathcal{P}^{-1}_{f,g}\Big(\downarrow(\mathcal{P}_{f,g}(e_1))\Big) = \{e_1\}$
3. $\forall e_2 \in g(B_I), \uparrow(e2) \cap \mathcal{P}_{f,g}\Big(\downarrow(\mathcal{P}^{-1}_{f,g}(e_2))\Big) = \{e_2\}$

proof: Suppose 2) does not hold. This means that there exists an event $e_1 \in E_1$ that is paired with an event $e_2 \in E_2$, and that this pair will form a unique event e in the amalgamated sum. If $\uparrow(e_1) \cap \mathcal{P}^{-1}_{f,g} \downarrow(\mathcal{P}_{f,g}(e_1)) = e_1$, then there is an event $x \in \downarrow(e_2)$ such that $x \leq e_2$. There is also an event $y \in \uparrow(e_1)$ such that $e_1 \leq y$. Now, if $\mathcal{P}_{f,g}(y) = x$, then there is an event e' in the amalgamated sum such that $e \leq e'$ and $e' \leq e$, and the order relation in the sum is not acyclic 1) does not hold. (Figure 10 illustrates this demonstration).

The same reasoning holds for 3) w.r.t 1). Now suppose that 1) does not hold. This means that the relation $\leq$ is cyclic, i.e. there exists a cycle containing two events e and e' in the amalgamated sum. These events are the images of a pair of event $e_1, e'_1 \in E_1$ and $e_2, e'_2 \in E_2$. Furthermore, $e_1 \leq_1 e'_1$, $e'_2 \leq_2 e_2$, and $\mathcal{P}(e_1) = e2$ and $\mathcal{P}(e'_1) = e'_2$. Hence, we have $e'_1 \in \uparrow(e_1) \cap \mathcal{P}^{-1}_{f,g} \downarrow(\mathcal{P}_{f,g}(e_1))$, and 2) does not hold. The same reasoning holds for 1) w.r.t. 3)$\square$

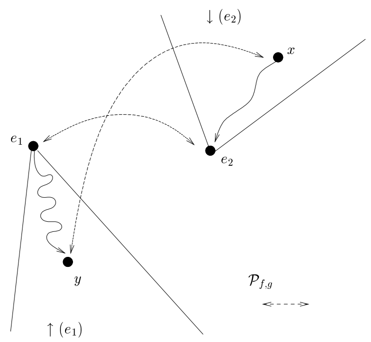


Fig. 10. Illustration of proof of property 1

Proposition 2: For any pair of bMSCs B_1, B_2, $\widehat{C_\epsilon} \subseteq \overline{C_\epsilon}$.

proof:

For any common part C, if $C1$ is an extension of C, $C2$ is an extension of C and if there exists an upper bound C' that extends both $C1$ and $C2$, then the matchings performed in $C1$ and $C2$ can be performed in any order. Any maximal element of $Ext(c_\epsilon)$ is a collection of pairwise compatible matchings, and can hence be obtained as a sequence of matching additions. If we order these additions according to the causal order of operands, we obtain a maximal extension of C_ϵ through sequential extensions. $\square$.

Proposition 3 Let $C = (B, f, g)$ be a common part of B_1, B_2, let e be an atomic action of $E_1 \setminus \downarrow f(E_B)$ and e' be an atomic action of $E_2 \setminus \downarrow g(E_B)$ such that $\phi(e) = \phi(e')$ and $\alpha(e) = \alpha(e')$. Let e_i be a freshly created event such that $\phi(e_i) = \phi(e)$ and $\alpha(e_i) = \alpha(e)$. Then, $C' = (B \circ \{ei\}, f \cup \{(e_i, e)\}, g \cup \{(e_i, e')\})$ is a sequential extension of C.
proof: The proof of this property is obvious. As e and e' are events from $E_1 \setminus \downarrow f(E_B)$ and $E_2 \setminus \downarrow g(E_B)$ and as C is a common part, there is no pair of matched events in the futures of e (respectively e'. Hence, again, for $f(E_i) \cup e$, property 2) is satisfied (and similarly property 3) is satisfied for $g(E_i) \cup \{e'\}$. Hence, the amalgamated sum remains well formed with interface $B \circ \{ei\}$ and morphisms $f \cup \{(e_i, e)\}$,$g \cup \{(e_i, e')\}$), and C' is a common part. $\square$

Proposition 4
Let $C = (B, f, g)$ be a common part of B_1, B_2, and let e be an event of $E_1 \setminus \downarrow f(E_B)$ and e' be an event of $E_2 \setminus \downarrow g(E_B)$ such that If e is a message emission, then there is a extension $C' = (B', f', g')$ of C containing e and $m(e)$ iff:

- $\exists e' \in E_2 \setminus \downarrow g(E_B)$ such that $\phi(e) = \phi(e')$ and $\alpha(e) = \alpha(e')$.
- $\exists e'' \in E_2 \setminus \downarrow g(E_B)$ such that $\phi(m(e)) = \phi(e'')$ and $\alpha(m(e)) = \alpha(e'')$.
- $m(e') = e''$
- $\mathcal{P}^{-1}_{f',g'}(\downarrow (e'')) \cap \mathcal{P}_{f',g'} \uparrow (e) = e$

proof: The first three items come from the definition of morphisms: we can not include a message emission and not the reception. Now suppose that in any morphism containing e and the reception $m(e)$ and extending current morphism, the amalgamated sum forms a cyclic relation. Then, the smallest morphisms that only pair $e, m(e)$ with e', e'' (in addition to already matched events aslo produces a cycle, and from proposition 1, we have $\mathcal{P}^{-1}_{f',g'}(\downarrow (e'')) \cap \mathcal{P}_{f',g'} \uparrow (e) = e$. $\square$

Proposition 5 Let B_1, B_2 be two bMSCS. Then we have: $C_{Best} \subseteq \overline{C_\epsilon}$
proof: We know that if C' is the extension of C, then the size of C' (and hence the quality) increases. Then any maximal element of the semi-lattice of extensions is of better quality than all the common part it extends. $\square$