HAL
open science

Scenario realizability with constraint optimization

Rouwaida Abdallah, Arnaud Gotlieb, Loïc Hélouët, Claude Jard

To cite this version:

Rouwaida Abdallah, Arnaud Gotlieb, Loïc Hélouët, Claude Jard. Scenario realizability with constraint optimization. [Research Report] 2013. hal-00769656

HAL Id: hal-00769656

https://inria.hal.science/hal-00769656

Submitted on 2 Jan 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Scenario realizability with constraint optimization

Rouwaida Abdallah[1], Arnaud Gotlieb[2], Loïc Hélouët[3],Claude Jard[4]

[1]ENS Cachan (antenne de Bretagne), [2] SIMULA, Norway, [3] INRIA Rennes, [4]Université de Nantes
rouwaida.abdallah@irisa.fr,{arnaud.gotlieb,loic.helouet}@inria.fr, claude.jard@univ-nantes.fr

Abstract. This work considers implementation of requirements expressed as High-level Message Sequence Charts (HMSCs). All HMSCs are not implementable, and the question of whether an HMSC specification can be implemented by communicating machines is undecidable in general. However, several subclasses such as local HMSCs can be implemented using a simple projection operation. This paper proposes a new technique to transform an arbitrary HMSC specification into a local HMSC, hence allowing implementation. We show that this transformation can be automated as a constraint optimization problem. The impact of modifications brought to the original specification can be minimized w.r.t. a cost function. The approach was evaluated on a large number of randomly generated HMSCs. The results of this experimentation are presented and analyzed. In particular, the evaluation shows an average runtime of a few seconds, which demonstrates applicability of the technique.

1 Introduction

In many system development methodologies, the user first specifies the system's use cases. Some specific instantiations of each use case are then aggregated and described using a formal language. In the context of distributed applications we consider, *high-level message sequence charts* (HMSCs) and their variants are very popular. They are standardized by the ITU [9], and a variant called *sequence diagrams* is now part of the UML framework [5]. HMSCs are particularly useful in the early stages of system development to describe patterns of interactions between processes (or objects).

In a later modeling step of the development, state diagrams (i.e. automata) prescribe a behavior for each of the processes. Finally, the processes are implemented as code in a specific programming language. **Parts of this design flow can be automated**. We consider here the automated transformation of HMSCs into the model of *communicating finite state machines* (CFSMs), which will serve as a skeleton for the development of future code. The produced CFSM model is called the (abstract) implementation of the HMSC.

However, HMSCs are not always implementable. A typical situation that prevents HMSCs' implementation is when two distant processes may choose to

execute different scenarios (i.e. finite sequences of interactions). This can lead to deadlock situations, and thus should be avoided. Even worse, realizability of HMSCs, that is the question of whether an HMSC specification can be implemented by an equivalent CFSM is undecidable in general [2, 11]. HMSCs in which such distributed choices do not occur are called *local HMSCs*.

Several works have shown that this subclass of HMSCs can be implemented by simple projection of the original specification on its processes [8, 10, 1]. Non-local HMSCs are generally rejected in the considered methodology as too incomplete or too abstract to be implemented. Based on this state of the art, we propose to extend the possibility of automated production of CFSMs by the use of a **localization procedure** that transforms any non-local HMSC into a local one. It guarantees that every choice in the transformed local HMSC has a *leader process*, which chooses one scenario and communicates its choices to the other processes. This can be achieved by adding new messages and processes in scenarios. Trivial but uninteresting solutions to the localization problem exist. We are thus interested in finding solutions with the minimal number of added messages because they correspond to the less disturbing transformation of the specification. In our work, we propose to address the localization problem with a constraint optimization technique. We build a constraint model where variables represent leader processes or processes contributing to a scenario and constraints capture the localization properties of HMSCs. A cost function is then proposed to minimize the number of added messages. The experiments we ran on a large class of randomly generated HMSCs, with a prototype tool implementation, show that the localization problem can be solved in general in a few seconds on ordinary machines.

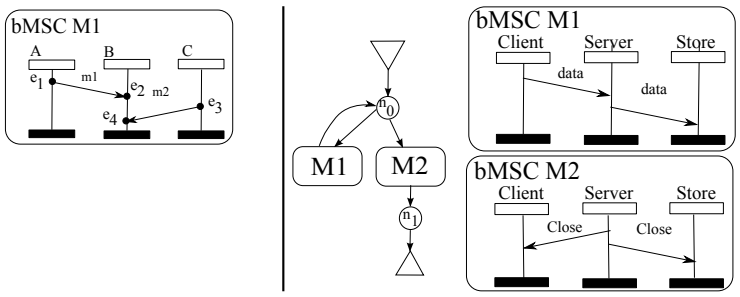
This paper is organized as follows: Section 2 gives the basic formal definitions on HMSCs. Section 3 defines localization of HMSCs. Section 4 proposes an encoding of minimal localization as a constraint optimization problem, and shows the correctness of the approach. Section 5 describes an experimentation conducted to evaluate the performance of our localization procedure, and comments the results. Section 6 concludes this work.

2 Basic definitions

Message Sequence Charts (MSCs for short) describe the behavior of entities of a system called *instances*. It is frequently assumed that instances represent processes of a distributed system. MSCs have a graphical and textual notation, but are also equipped with a standardized semantics [13]. The language is composed of two kinds of diagrams. At the lowest level, basic MSCs (or bMSCs for short) describe interactions among instances. The communications are asynchronous. A second layer of formalism, namely High-level MSCs (HMSCs for short), is used to compose these basic diagrams. Roughly speaking, an HMSC is an automaton which transitions are labeled by bMSCs, or by references to other HMSCs. However, in the paper, we will consider without loss of generality that our specifications are given by only two layers of formalism: a set of bMSCs, and an HMSC with transitions labeled by these bMSCs.

Definition 1 (bMSCs). *A* bMSC *defined over a set of instances $\mathcal{P}$ is a tuple $M = (E, \leq, \lambda, \phi, \mu)$ where E is a finite set of events, $\phi : E \longrightarrow \mathcal{P}$ localizes each event on one instance. $\lambda : E \longrightarrow \Sigma$ is a labeling function that associates a type of action to each event. The label attached to a sending event is of the form $p!q(m)$ denoting a sending of message m from p to q. Similarly, the label attached to a reception is of the form $p?q(m)$ denoting a reception on p of a message m sent by q. Last, events labeled by $p(a)$ represent a local action a of process p. Labeling defines a partition of E into sets of sending events, reception events, and local actions, respectively denoted by E_S,E_R and E_L. $\mu : E_S \longrightarrow E_R$ is a bijection that maps sending events with a corresponding reception. If $\mu(e) = f$, then $\lambda(e) = p!q(m)$ for some p, q, m and $\lambda(f) = q?p(m)$. $\leq \subseteq E^2$ is a partial order relation called the* causal order.

It is required that events of the same instance are totally ordered: $\forall (e_1, e_2) \in E^2, \phi(e_1) = \phi(e_2) \Longrightarrow (e_1 \leq e_2) \vee (e_2 \leq e_1)$. For an instance p, let us call $\leq_p$ this total order. $\leq$ must also reflect the causality induced by the message exchanges, i.e. $\leq = (\bigcup_{p \in \mathcal{P}} \leq_p \cup \mu)^*$. The graphical representation of bMSCs defines instances as vertical lines. All events executed by an instance are ordered from top to bottom. Horizontal arrows represent messages from one instance to another. Figure 1-a) is an example of bMSC, with three instances A, B, C, exchanging two messages m_1,m_2. The events $e1$ and $e3$ are sending events, and the events $e2$ and $e4$ are the corresponding receptions.



(a) (b)

Fig. 1: An example bMSC and an example HMSC

For a bMSC M, we will denote by $min(M) = \{e \in E \mid \forall e' \in E,\ e' \leq e \Rightarrow e' = e\}$, the set of minimal events of M. Intuitively, each event in $min(M)$ can be the first event executed in M. An instance is called *minimal* if it carries a minimal event. A minimal instance (i.e. a process in $\phi(min(M))$) is an instance that can execute the first event in M. In other words, it can *decide* to start executing M rather than another scenario. A bMSC is *local* if it has a single minimal instance.

The semantics of a bMSC M is denoted by $Lin(M)$, and defined as the linearizations of M, that is sequences of actions that follow the causal ordering imposed by M. We refer interested readers to the appendix for details. Basic MSCs

only describe finite interactions. They have been extended with HMSCs [14] to allow iteration, choices, and parallel composition. For simplicity, we restrict to HMSCs without parallel frames, and with only one hierarchical level. With these assumptions, HMSCs can be seen as automata labeled by bMSCs.

Definition 2 (HMSCs). *An* HMSC *is a graph* $H = (I, N, \rightarrow, \mathcal{M}, n_0)$*, where* I *is a finite set of instances,* N *is a finite set of nodes,* $n_0 \in N$ *is the initial node of* H*,* $\mathcal{M}$ *is a finite set of bMSCs, defined over disjoint set of events, and* $\rightarrow \subseteq N \times \mathcal{M} \times N$ *is the transition relation.*

In the rest of the paper, we consider without loss of generality that all nodes, except possibly the initial node and sink nodes, are choice nodes (i.e. have several successors by the transition relation). We will denote by $\mathcal{P}_i$ the set of active processes that interacts within a bMSC $M_i \in \mathcal{M}$. HMSCs also have a graphical representation: Nodes are represented by circles, references to bMSCs by boxes. The initial node of an HMSC is connected to a downward pointing triangle, and final nodes to an upward pointing triangle. The example of Figure 1-b) shows an example of HMSC, with two nodes n_0, n_1. Intuitively, this HMSC depicts a protocol in which three processes exchange data, before closing a session. This very simple HMSC contains an iteration: bMSC $M1$ can be repeated several times before the execution of bMSC $M2$. Note however that even if a bMSC $M1$ is seen before a bMSC $M2$ along a path of H, this does not mean that all events of $M1$ are executed before $M2$ starts. The semantics of an HMSC is defined using *sequential composition* of bMSCs.

We do not define formally sequential composition of bMSCs, and refer interested readers to the appendix. Intuitively, composing sequentially two bMSCs M_1 and M_2 consists in drawing M_2 below M_1 to obtain a new bMSC. The sequential composition is denoted by $M_1 \circ M_2$. From this intuitive definition, we can immediately notice that if some events in $min(M_2)$ are located on processes that do not appear in M_1, then they are also minimal in $M_1 \circ M_2$. This raises two important remarks: First, executing $M_1 \circ M_2$ does not mean executing M_1 then M_2, but rather executing the bMSC obtained by concatenation of M_1 and M_2, and then minimal events in a concatenation $M_1 \circ \ldots M_k$ are not all located in M_1.

A *path* of H is a sequence of transitions $\rho = (n_1, M_1, n'_1) \ldots (n_k, M_1, n'_k)$ such that $n'_i = n_{i+1}$. A path is called *initial* if it starts from node n_0. Each path $\rho = (n_1, M_1, n'_1) \ldots (n_k, M_1, n'_k)$ defines a unique bMSC $M^\rho = M_1 \circ M_2 \cdots \circ M_k$. The semantics $\mathcal{L}(H)$ of a HMSC H is defined as the set of linearizations of all bMSCs M^ρ such that ρ is an initial path of H. With this semantics, HMSCs are very expressive. They are more expressive than finite state automata (they can describe non-regular behaviors, as shown by the example of Figure 1-b)).

The *implementation problem* consists in building a set of communicating machines $\mathcal{A} = \{A_1, \ldots A_k\}$ (one frequently uses as model the Communicating Finite State Machines (CFSM) proposed by [6]) such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(H)$. It is frequently assumed that these communicating machines are obtained by simple projection of the original HMSC on each instance. The *realizability problem* consists in

deciding whether there exists an implementation $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(H)$. Realizability was shown undecidable in general [2, 11]. On the other hand, several papers have shown automatic and correct synthesis techniques for subclasses of HMSCs [8, 10, 3, 1]. The largest known subclass is that of local HMSCs [4]. These results clearly show that automatic implementation can not apply in general to HMSC. However, non-local HMSCs can be considered as too abstract to be implemented, and need some refinement to be implementable. In the rest of the paper, we hence focus on a transformation mechanism that transforms an arbitrary HMSC into a local (and hence implementable) HMSC.

Let us consider the example of Figure 1-b). Node n_0 is a choice node, depicting a choice between two behaviors: either continue to send data (bMSC M_1), or close the data transmission (bMSC M_2). However, the deciding instance in M_1 is the *Client*, while the deciding instance in M_2 is the *Server*. At implementation time, this may result in a situation where *Client* decides to perform M_1 and *Server* decide concurrently to perform M_2, leading to a deadlock of the protocol. Such situation is called a non-local choice, and obviously causes implementation problems. It is then safer to implement HMSCs without non-local choices. At each choice, a single deciding instance chooses to perform one scenario, and all other non-deciding instances must conform to this choice.

Definition 3 (Local choice node). *Let $H = (I, N, \rightarrow, \mathcal{M}, n_0)$ be an HMSC. Let $c \in N$, c is a* local choice *if and only if for every pair of (non necessarily distinct) paths $\rho = (c, M_1, n_1)(n_1, M_2, n_2) \ldots (n_k, M_k, n_{k+1})$ and $\rho' = (c, M'_1, n'_1)(n'_1, M'_2, n'_2) \ldots (n'_q, M'_k, n'_{q+1})$ there is a single minimal instance in M^ρ and in $M^{\rho'}$ (i.e. $\phi(min(M^\rho)) = \phi(min(M^{\rho'}))$ and $|\phi(min(M^\rho))| = 1$). H is called a* local *HMSC if all its nodes are local.*

Due to the semantics of concatenation, non-locality can not be checked on a pair of bMSCs leaving node c, but has to be checked for pairs of paths. Intuitively, locality of an HMSC H guarantees that every choice in H is controlled by a unique deciding instance. Checking whether an HMSC is local is decidable [10], and one can easily show that this question is in co-NP [1]. It was shown in [8] that for a local HMSC $H = (I, N, \rightarrow, \mathcal{M}, n_0)$, if for every $M_i \in \mathcal{M}$ we have $\mathcal{P}_i = I$, then there exists a CFSM $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(H)$. A solution was proposed to leverage this restriction in [1], hence allowing to implement *any* local HMSC.

An immediate question that arises is: How to implement non-local HMSCs? In the rest of the paper, we propose a solution that transforms any non-local HMSC into a local one, hence allowing its implementation. This results in slight modifications of the original specification. We allow additional active instances and new messages in bMSCs, but do not change the structure of the HMSC. Consider the example of Figure 1-b). Replacing M_1 by the bMSC M_3 of Figure 2 solves the non-local choice problem. Similarly, replacing M_1 and M_2 respectively by M_4 and M_5 solves the the non-locality problem, but needs more messages.

This example raises several remarks. First, the proposed transformations are purely syntactic, and modifying the set of minimal instances does not always

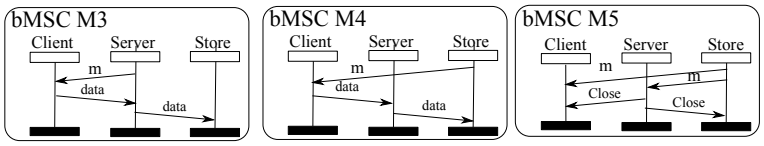


Fig. 2: Solutions for localization of HMSC in Figure 1-b)

produce a meaningful specification. For this reason, the examples exhibit changes involving a single message type m. A meaning for additional message has to be chosen adequately by the designer once an HMSC is localized. The second remark is that there are several possibilities for localization. The first solution proposed adds one message in bMSC M_1 to obtain M_3. The second solution adds two messages to M_1 and one to M_2, and one can notice that in M_5, the message between *Store* and *Client* is useless. Indeed, there exists an infinite number of transformations to localize an HMSC. This calls for the following solutions: We want to restrict to cheapest solutions (for instance solutions with a minimal number of added messages). As we will show later, once a deciding instance for a choice is fixed, one can compute the minimal number of messages needed to localize this choice. As a consequence, the solutions to a localization problem can be given in terms of choosing a deciding instances at each choice, and instances participating to bMSCs. Then, the localization can be easily tuned using different cost functions.

3 Localization of HMSCs

In this section, we show how to transform a non-local HMSC into a local one. This procedure called *localization* consists in choosing a single deciding instance for each bMSC M in the HMSC so that all choices become local, and then ensure that all other instances execute their minimal events only after the first event (the *choice*) of the deciding instance. This is done by adding messages, as in the examples of Fig. 2.

Definition 4. *Let M be a bMSC over a set of events E and processes P, with minimal events $e_1, \ldots, e_k$. A* localized extension *of M is a bMSC M' over a set of events $E' \supseteq E$ and over $P' \supseteq P$, such that there exists a minimal event $e_{min} \in E'$ and for every $e \leq f \in E$, we have $e \leq' f$. The unique minimal instance in a localized bMSC M is called the* leader *of M.*

Note that as there exists an infinite number of extensions for a bMSC M, choosing extensions that are as close as possible to the original model is desirable. The impact of localization can be simply measured as the number of added messages. A more generic approach is to associate a cost to communications between processes, and to choose extensions with minimal cost. This makes sense, as for instance the cost and delays for communications via satellite are higher than with ground networks. Similarly, the configuration of a system may prevent two processes p and q from exchanging messages. To avoid solutions with communications between p and q, one can design a cost function that associates a redhibitory (or even infinite) cost to such communications.

For a given bMSC M with k minimal events, there exists a localized extension over the same set of processes that contains exactly $k-1$ additional messages. This localized extension is built when one picks up a deciding instance d among the minimal instances of M, and create causal dependencies from the minimal event on instance d to all other minimal events with additional messages. Only $k-1$ messages are necessary in this case, regardless to their respective ordering and place of insertion in the orginal bMSC. Another possibility is to pick up another non-minimal process among those of M that do not carry a minimal event as a leader, or even add a new process to M. In such cases, a localized extension can always be built with exactly k additional messages.

Localization of HMSCs is more complex than localization of bMSCs. For each non-local choice c, we have to ensure that *every branch* leaving c has the same leader. Hence, this is not a property purely local to bMSCs. As for bMSCs, we can define a notion of localized extension of a HMSC as follows:

Definition 5. *Let $H = (I, N, \longrightarrow, \mathcal{M}, n_0)$ be an HMSC. $H' = (I, N, \longrightarrow', \mathcal{M}', n_0)$ is a* localized extension *of H iff there is a bijection $f : \mathcal{M} \to \mathcal{M}'$ such that $\forall M \in \mathcal{M}$, $f(M)$ is a localized extension of M, $\longrightarrow' = f(\longrightarrow)$, and H' is a local HMSC.*

Localizing an HMSC H consists in finding $\mathcal{M}'$ and the bijection f. As mentionned above, as there exists a (potentially) infinite number of solutions, we consider the solutions with the smallest number of changes to the original model. We propose to address this problem with a cost function $\mathcal{F}$ that evaluates the cost of each possible transformation of H. The goal of our localization algorithm is thus to minimize $\mathcal{F}$. For the sake of simplicity in this paper, $\mathcal{F}$ counts the total number of messages and instances added in $\mathcal{M}'$. As localization transforms $\mathcal{M}$ into $\mathcal{M}'$, $\mathcal{F}$ is defined as a sum of individual costs of modifications. Formally,

$$\mathcal{F}(H, H') \triangleq \sum_{M \in \mathcal{M}} c_{M, f(M)}$$

where $c_{M,M'}$ is the individual cost to transform M into M'. When H is clear from the context, we will write $\mathcal{F}(H')$ instead of $\mathcal{F}(H, H')$. Let $M_i \in \mathcal{M}$ be a bMSC, $M'_i = f(M_i)$, I_{M_i}, $I_{M'_i}$ be the set of instances in M_i and M'_i. Let $k = |min(M_i)|$ be the number of minimal instances in M_i, l be the leader instance of M'_i, and $x = |I_{M'_i}| - |I_{M_i}|$ be the number of new instances in M'_i. We choose a constant $\theta \in [0, 1]$ and define the cost c_{M_i, M'_i} for transforming M_i into M'_i as follows:

$$c_{M_i, M'_i} \triangleq \begin{cases} x * \theta + (k + x - 1) * (1 - \theta) & \text{if } l \in \phi(min(M_i)) \text{ or } l \notin I_{M_i} \\ x * \theta + (k + x) * (1 - \theta) & \text{otherwise} \end{cases}$$

Intuitively, c_{M_i, M'_i} is the *barycenter* between the number of added messages, and the number of added instances, weighted by θ. We already know that the number of messages to add is at most $k-1$ if we have k minimal instances. Adding x instances to M_i hence yields adding $(k + x - 1)$ messages if l is chosen among the minimal instances of M_i or among the new instances. Similarly, if the leader instance is chosen among instances that are not minimal w.r.t the causal ordering, one needs to add $k + x$ messages to localize M_i. The value θ is

chosen to penalize more the number of added processes or the number of added messages.

Let us illustrate the computation of $\mathcal{F}$ on an example. Let H_c be the HMSC on the right of Figure 3, and let H'_c be a localization of H_c. As mentionned above, $\mathcal{F}(H_c, H'_c) = c_{M_1,M'_1} + c_{M_2,M'_2} + c_{M_3,M'_3}$. The leader of M'_1 is C and, as $C \in min(M_1) = \{A, C\}$, then $c_{M'_1} = 1 - \theta$ (there is a single message added in M'_1). The leader of M'_2 is also C, but C is not an instance of M_2, so $c_{M'_2} = \theta + (1-\theta) = 1$ (there is a single message and a single instance added in M'_2). The leader of M'_3 is again C. As C is an instance of M_3 but not a minimal instance we have $c_{M'_3} = 2 * (1 - \theta)$. As a result, $\mathcal{F}(H_c, H'_c) = 1 - \theta + 1 + 2 * (1 - \theta) = 4 - 3 * \theta$. One can easily notice that H'_c is local. If we compare H_c with another localization H''_c, depicted at the right of Figure 3, we get that $c_{M_1,M''_1} = 1$, $c_{M_2,M''_2} = \theta$, $c_{M_3,M''_3} = 0$ and finally, $\mathcal{F}(H_c, H''_c) = 1 + \theta$. If $\theta = 0.5$ then $\mathcal{F}(H_c, H''_c) < \mathcal{F}(H_c, H'_c)$ and thus, localization H''_c shoud be preferred to H'_c. On the other hand, if $\theta = 1$, then H'_c should be preferred. This example shows that the cost funtion influences the choice of a particular localization solution.

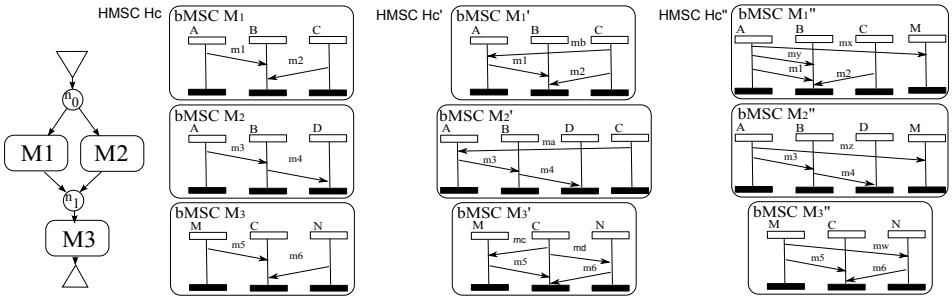


Fig. 3: Localizing the HMSC Hc

The cost function $\mathcal{F}$ defined above that counts the number of new messages and processes in bMSCs is only an example, and other functions can be considered. For instance, a cost function can consider concurrency among events as an important property to preserve, and thus impose a penality everytime a pair of events e, e' is causally ordered in $f(M)$, but not in M.

Note also that several localization solutions can have the same cost. For instance, if $\mathcal{F}$ is used, the order in which messages are exchanged to obtain localized bMSCs is ignored. Considering that the cost function is influenced only by the number of added messages and added processes, we define $\mathcal{F}(H, \{(I_{M'}, l_{M'})\}_{M \in \mathcal{M}})$ as being the cost of a localization of H that satisfies $I_{M'} = I_{f(M)}$, where $f(M)$ has $l_{M'}$ as leader for every $M \in \mathcal{M}$. The localization problem can be formally defined as follows:

Definition 6. *Let $H = (I, N, \longrightarrow, \mathcal{M}, n_0)$ be a non-local HMSC, and $\mathcal{F}$ be a cost function. The* localization problem *for $H, \mathcal{F}$ consists in returning solutions $s_1, \ldots, s_k$, where each s_i is of the form $s_i = \{(I_{M'}, l_{M'})\}_{M \in \mathcal{M}}$ such that $\mathcal{F}(H, \{(I_{M'}, l_{M'})\}_{M \in \mathcal{M}})$ is minimal, and where for each $M \in \mathcal{M}$, $I_{M'} \subseteq I$ is a set of instances appearing in $M' = f(M)$ and $l_{M'} \in I_{M'}$ is the leader of M'.*

4 Localization as a constraint optimization problem

This section explains how a finite domain constraint optimization model is constructed from a given HMSC, to minimize the cost of the localization.

4.1 Constraint solving over finite domains

A *constraint solving problem* is composed of a finite set of *variables* $X_1, \ldots, X_n$, where each variable X_i ranges over a finite *domain*, noted $D(X_i)$. An assignment of a variable is a choice of a value from its domain. A set of *constraints* $C_1, \ldots, C_m$ is defined over the variables and the goal in a constraint solving problem is to find *solutions*, i.e., assignments for all variables, that satisfy all constraints. A constraint solving problem is *satisfiable* if it allows at least one solution. When a *cost function* $\mathcal{F}$ is associated to each assignment, the problem becomes a *constraint optimization problem* (COP) where the goal is to find a solution that optimizes the cost. Such a solution is called an *optimal solution*.

Constraint solving frequently uses filtering and propagation. Roughly speaking, the underlying idea is to consider each constraint in isolation, as a filter over the domains. *Filtering* a domain means eliminating inconsistent values w.r.t. a given a constraint. For example, if $D(X) = \{1, 3, 4\}$ and $D(Y) = \{2, 3, 4, 5\}$, the constraint $X > Y$ filters $D(X)$ to $\{3, 4\}$ and $D(Y)$ to $\{2, 3\}$. Once a reduction is performed on the domain of a variable, *constraint propagation* awakes the other constraints that hold on this variable, in order to propagate the reduction. Constraint propagation is a polynomial process: It takes $O(n * m * d)$ where n is the number of variables, m is the number of constraints and d is the maximum number of possible values in the domains.

Constraint propagation and filtering alone do not guarantee satisfiability, and just prune the domains without trying to instantiate variables. For example, considering the constraint system shown above, the constraint $X > Y$ prunes the domains $D(X)$ to $\{3, 4\}$ and $D(Y)$ to $\{2, 3\}$ but $(3, 3)$ is not a solution of the constraint. The constraint system may even be unsatisfiable, while constraint propagation and filtering does not detect it (i.e., they ensure only *partial satisfiability*). Hence, an additional step called *labeling search* is needed to exhibit solutions. Labeling search consists in exploring the search space composed of the domains of uninstantiated variables. Interestingly, a labeling procedure can awake constraint propagation and filtering, allowing an early pruning of the search space. In the previous example, if X is labeled by 3 then the constraint $X > Y$ is awoken and automatically reduce the domain of Y to $\{2\}$. A labeling search procedure is *complete* when the whole search space is explored. Complete labeling search can eventually determine satisfiability (or unsatisfiability) of a constraint solving problem over finite domains. However, it is an exponential procedure in the worst case. This is not surprising as determining satisfiability of a constraint problem over finite domains is NP-hard [15].

During labeling search, when a solution s is found, the value $m = \mathcal{F}(s)$ of the cost function can be recorded, and backtracking can then be enforced by adding the constraint $\mathcal{F}(...) < m$ to the set of constraints (or $\mathcal{F}(...) \leq m$ if one

wants to explore **all** optimal solutions). If another solution is found, then the cost function $\mathcal{F}$ will necessarily have a cost smaller than m. This procedure, called branch&bound [12], can be controlled by a timeout that interrupts the search when a given time threshold is reached. Of course, the current value of $\mathcal{F}$ in this case may not be a global minimum, but it is already an interesting value for the cost function, something that we call a *quasi-optimum*. For localization of HMSCs, selecting the local HMSC with the smallest cost is desirable but not always essential. On the other hand, mastering the time spent for localization is essential to scale to real-size problems.

4.2 From HMSC to COP

Variables. Localizing an HMSC H consists in selecting a set of participating instances and a minimal process for each bMSC appearing in H, such that every choice in the HMSC becomes a local choice. As this selection is not unique, we use constraint optimization techniques to provide characteristics of localized HMSCs with minimal cost. We propose to transform any HMSC into a constraint optimization problem, as follows: A couple of variables (X_i, Y_i) is associated to each bMSC $M_i \in \mathcal{M}$, where X_i represents the set of instances chosen for the bMSC $f(M_i)$, and Y_i represents the leader in $f(M_i)$. If I is the set of instances of H, every X_i takes its possible values in 2^I while Y_i takes a value in I.

Constraints. Our constraint model is composed of domain, equality and inclusion constraints. *Domain constraints*, noted DOM, are used to specify the domains of X_i and Y_i. Obviously, if a bMSC M_i is defined over a set of processes $\mathcal{P}_i$, we have $\mathcal{P}_i \subseteq X_i \subseteq I$. *Equality constraints*, noted EQU, enforce the locality property. For two bMSCs M_i, M_j such that there exists two transitions (n, M_i, n_1) and (n, M_j, n_2) in $\rightarrow$ originating from the same node n, $f(M_i)$ and $f(M_j)$ must have the same leader, i.e., $Y_i = Y_j$. We write $M_i \otimes M_j$, when such choice between M_i and M_j exists in H. Locality of HMSCs is also enforced by using *inclusion constraints*, noted $INCL$. Let $M_i, M_j \in \mathcal{M}$ be two bMSCs. We write $M_i \triangleright M_j$ when there exists a path $(n, M_i, n')(n', M_j, n'')$, i.e., when M_i is the predecessor of M_j in H. In such case, in any localization of H, the minimal instance of $f(M_j)$, represented by variable Y_j, must appear in the set of instances of $f(M_i)$, represented by variable X_i. In our constraint model, this is expressed by the constraint $Y_j \in X_i$. Similarly, the leader of a bMSC in the localized solution can only be one of its instances, so we have $Y_i \in X_i$ for every $M_i \in \mathcal{M}$.

It is worth noticing that the localization problem is always satisfiable, as there exists at least one trivial solution: Select an instance in I as leader for all bMSCs, then add this instance if needed to every bMSC, and messages from this instance to all other instances. However, this trivial and uninteresting solution is not necessarily minimal w.r.t. the chosen cost function. We can now prove that our approach is *sound* and *complete* by considering the following definition:

Definition 7. *Let $H = (I, N, \rightarrow, \mathcal{M}, n_0)$ be an HMSC, the constraint optimization model associated to H is $CP_H = (\mathcal{X}, \mathcal{Y}, \mathcal{C})$ where $\mathcal{X} = \{X_1, \ldots, X_{|\mathcal{M}|}\}$*

associates a variable to the set of instances appearing in each bMSC of $f(\mathcal{M})$, $\mathcal{Y} = \{Y_1, \ldots, Y_{|\mathcal{M}|}\}$ *associates a variable to the leader selected for each bMSC of* $f(\mathcal{M})$, *and* $\mathcal{C} = DOM \cup EQU \cup INCL$ *is a set of constraints defined as follows:*

- $DOM = \bigwedge_{i \in 1 \ldots |\mathcal{M}|} X_i \in 2^I \wedge \mathcal{P}_i \subseteq X_i \wedge Y_i \in I$;
- $EQU = \bigwedge_{M_i, M_j | M_i \otimes M_j} Y_i = Y_j$
- $INCL = \bigwedge_{M_i, M_j | M_i \triangleright M_j} Y_j \in X_i \wedge \bigwedge_{i \in 1 \ldots |\mathcal{M}|} Y_i \in X_i$

Then, solving the localization problem for an HMSC H amounts to find an optimal solution for CP_H, w.r.t. cost function $\mathcal{F}$. We have:

Theorem 1. *Computing solutions for a localization problem using an optimal solution search for the corresponding COP is a sound and complete algorithm.*

This result is not really surprising, as CP_H represents what is needed for an HMSC to become local. For the sake of completeness, a formal proof of this theorem is available in appendix.

5 Implementation and experimental results

To evaluate the approach proposed in the paper, we implemented a systematic transformation from HMSC descriptions to COPs and conducted an experimental analysis over a large number of randomly generated HMSCs. Our implementation contains three main components G, A and S and is described in Figure 4. G is a random HMSC generator, A is an analyzer that transforms a localization problem for a given HMSC into a COP, as described in the previous section. Finally S is a constraint optimization solver: We used the `clpfd` library of SICStus Prolog [7]. The generator G takes an expected number of distinct HMSCs to generate (nbH), a number of bMSCs in each HMSC(nbB), and a number of active processes in each HMSC (nbP) as inputs. As output, it produces an xml file containing nbH randomly generated HMSCs.

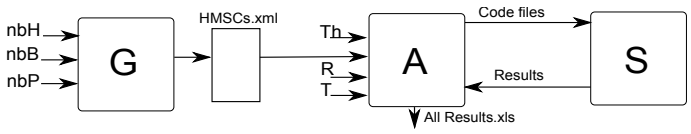


Fig. 4: The input and outputs of the generator, the analyser and the solver.

The analyser A takes Th, the parameter θ of the cost function $\mathcal{F}$, a set of heuristics R, and a sequence of time-out values T, as inputs. These values can be considered as internal parameters for the constraint optimization solver: They allow us to evaluate different strategies. In the experiments, we considered several labeling heuristics to choose the variable and the value to enumerate first, e.g., *leftmost*, *first-fail*, *ffc*, *step* or *bisect*. *Leftmost* is a variable-choice heuristic

that selects the first unassigned variable from a statically ordered list. *First-fail* is a dynamic variable-choice heuristic that selects first the variable with the current smallest domain. *Ffc* is an extension of *first-fail* that uses the number of constraints on a given variable as a tie-break when two variables have the same domain size. *Step* is choice-value heuristic that consists in traversing incrementally the variation domain of the current variable. Finally, *bisect* implements *domain splitting* which consists in dividing a domain into two subdomains and propagating the subdomain informations. For example, if x takes a value in an interval $[a, b]$, then *bisect* will propagate first $x \in [a, \frac{a+b}{2}]$, and then $x \in [\frac{a+b}{2}, b]$ upon backtracking. For each generated HMSC H and each heuristic $h_i \in R$, the analyser creates a prolog file that contains the corresponding COP. For efficiency reasons, a special attention has been paid to the encoding of variation domains and constraints. Subset domains were encoded using a binary representation and sets inclusion using efficient div/mod operations. The prolog file is then used as input of the solver. The sequence T represents the various instants at which the optimization process must temporarily stop, and returns the current value of the cost function. These values are quasi-optima, representing approximations of the global optimum. The combination between heuristics and time-out values is useful to compare different labeling strategies. Finally, the analyser A collects all the results returned by the solver with the time needed to provide a solution, and stores them for a systematic comparison.

The first step of the experiment consisted in a systematic evaluation of the performance of several heuristics to guide the solver. During this step, we considered several heuristics and time-outs. We do not report here all the results, but show only the results for one illustrative model. Figure 5 shows the time-aware minimization of the cost value with 12 different heuristics and time-outs between 1s and 14s for a chosen localization problem. Heuristics descriptions use the following syntax: *[b — u] / [left — ff — ffc] / [bisect — step] / [XYC — YXC — CXY — CYX]*, where b and u stand resp. for *bounded costs* and *unbounded costs*. The heuristic *bounded cost* evaluates a lower bound on the cost of a solution that can be reached from a given state. *Left*, *ff* and *ffc* stand for a variable-choice heuristic, *bisect* and *step* stand for value-choice heuristic, and *XYC*, etc. stand for the static order in which variables are fed to the solver. Bold values indicate *proved global minima*, non-bold values indicate *quasi-optima*, – indicates absence of result in the given time contract.

In Figure 5, heuristics 3,9,10 give the best results. The series of experiment that we run shows that heuristics with an estimation of cost, and a static ordering of variable evaluations have the best performance. Overall, the heuristics number 10 combining *bisect* (domain splitting), *left* (static variable ordering), and a cost evaluation exhibited the best results and was selected for the next steps of the experiment.

As next steps, we generated 11 groups of 100 random HMSCs, with 10 bMSCs, which is a reasonably large number, according to the existing literature on HMSCs.We then let the number of processes grow from 4 to 14. We also generated 12 groups of 100 HMSCs, containing exactly 8 processes, and let the

	heuristics \ runtime	1 s	2 s	3 s	4 s	5 s	6 s	7 s	12 s	13 s	14 s
0	u/left/step/XYC	23	23	22	22	22	22	19	19	19	19
1	u /left/step/YXC	19	**19**								
2	b/left/step/XYC	19	**19**								
3	b/left/step/YXC	**19**									
4	b/left/step/CYX	-	-	-	-	-	-	-	-	21	**19**
5	b/ff/step/XYC	22	19	19	19	19	**19**				
6	b/ff/step/YXC	30	19	19	19	19	**19**				
7	b/ff/step/CYX	30	19	19	19	19	**19**				
8	b/ffc/step/CYX	27	22	19	19	19	19	**19**			
9	b/left/bisect/XYC	**19**									
10	b/left/bisect/YXC	**19**									
11	b/left/bisect/CYX	-	-	-	-	21	**19**				

Fig. 5: Comparing heuristics with one representative example.

number of bMSCs grow from 4 to 15. The goal of these series of experiments was to evaluate the influence of both the number of processes and the number of bMSCs on the runtime of our localization approach. We expected these parameters to influence the performance of localization, as increasing the number of bMSCs increases the number of variables, and increasing the number of processes increases the size of variables' domains. However, we have obtained solutions for all HMSCs, which allowed us to evaluate the impact of both parameters. The evaluation was performed on a machine equipped with INTEL $P9600$ core2 Duo at $2,53$ Ghz, with $4Go$ of RAM. Results of both experiences are given in Figure 6 and Figure 7, using box-and-whiskers plots to show the statistical distribution of datasets.

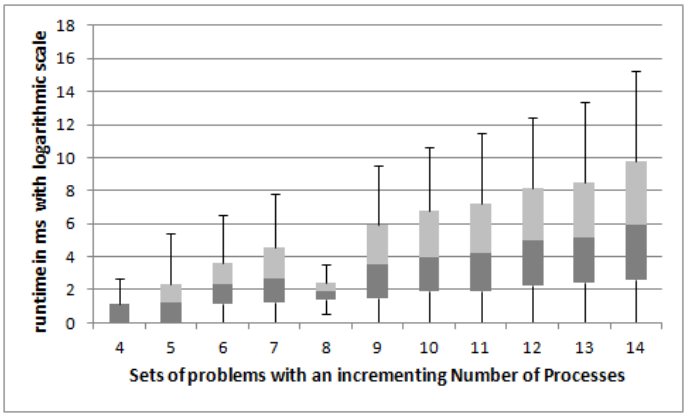


Fig. 6: Influence of the number of processes on the runtime execution

Both plots use logarithmic scales to tackle the big variance between runtime measurements. As expected, the plots show exponential curves but the runtime for each group remains quite low. For randomly generated HMSCs of reasonable size (such as the ones found in the literature), our experimental results show that localization using constraint optimization takes a few minutes in the worst cases,

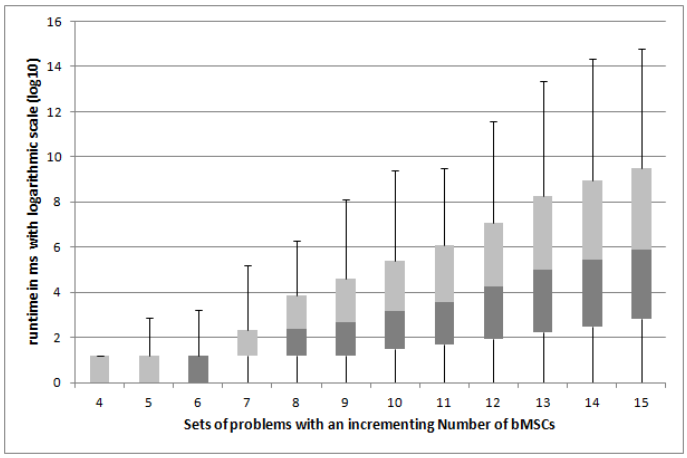


Fig. 7: Influence of the number of bMSCs on the runtime execution

and an average duration of a few seconds. Actually, even for the largest cases (15 bMSCs with 14 processes), the runtime of our localization approach did not exceed 40 minutes. Although solving COPs over finite domains is NP-hard [15], as examples of existing HMSCs usually contain less than 15 bMSCs, our localization process appears to be of practical interest. Our results are encouraging, and show that the approach is fast enough to be used in practice. However, they have been obtained on random instances only, and thus further experiments on non-random instances are necessary to confirm this judgment.

6 Conclusion and future work

This paper has proposed a sound and complete method to transform arbitrary HMSCs into implementable ones. Our approach transforms an HMSC in a constraint optimization problem. The solution returned by a solver can be used to build an optimal localized version of the original specification, without changing the overall architecture of the HMSC. Once an HMSC is localized by addition of messages and processes in bMSCs, automatic implementation techniques can generate code for communicating processes. Our approach has been implemented and tested on a benchmark of 2300 randomly generated HMSCs. The experimental results show that our approach is of practical interest: It usually takes less than a few minutes to localize an HMSC.

There are four foreseen extensions of this work. First, other cost functions can be considered as our approach does not depend on the choice of a particular cost function. For instance, we plan to study localization with functions that accounts for the cost of communications between instances. Second, we plan to allow modifications of the HMSC, in addition to those brought to the bMSCs of the specification. Considering architectural constraints that disallow communications between some processes is another challenging issue, as in this case existence of a solution is not guaranteed. Finally, noticing that localization is a rather syntactic procedure, the question of designating a process as a leader or adding messages should also be addressed in more semantics terms.

Further work also includes the experimentation of our approach on industrial case studies, to evaluate its performance on non-random HMSCs. We have started a collaboration with a company that develops communicating systems and wants to generate test cases based on requirements design. Our approach will be useful to derive automatically test cases from HMSCs that were designed without any requirement on implementability.

References

1. R. Abdallah, C. Jard, and L. Hélouët. Distributed implementation of message sequence charts. *Software and Systems Modeling*, page to appear, 2012.
2. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of msc graphs. In *ICALP*, pages 797–808, 2001.
3. N. Baudru and R. Morin. Synthesis of safe message-passing systems. In *FSTTCS*, pages 277–289, 2007.
4. H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Proc. of TACAS'97*, volume 1217 of *LNCS*, pages 259 – 274, 1997.
5. G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.
6. D. Brand and P. Zafiropoulo. On communicating finite state machines. Technical Report 1053, IBM Zurich Research Lab., 1981.
7. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *PLILP*, pages 191–206, 1997.
8. B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state high-level MSCs: Model-checking and realizability. *Journal on Comp. and System Sciences*, 72(4):617–647, 2006.
9. O. Haugen. in message sequence charts (msc). In *ITU, Z.120*, Editor. 1999, ITU-T: Geneva. p. 126.
10. L. Hélouët and C. Jard. Conditions for synthesis of communicating automata from HMSCs. In *Proc. of FMICS 2000*, 2000.
11. M. Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Theoretical Computer Science*, 309(1-3):529–554, 2003.
12. K. Marriott and P.J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.
13. M. Reniers and S. Mauw. High-level Message Sequence Charts. In *SDL97: Time for Testing - SDL, MSC and Trends*, Proc. of the 8th SDL Forum, pages 291–306, 1997.
14. E. Rudolph, J. Grabowski, and P. Graubmann. Tutorial on Message Sequence Charts (MSC '96). Tutorial, In FORTE/PSTV'96, 1996.
15. P. Van Hentenryck, V.A. Saraswat, and Y. Yves Deville. Design, implementation, and evaluation of the constraint language cc(fd). *J. Log. Program.*, 37(1-3):139–164, 1998.

7 Appendix

This appendix is included for the convenience of reviewers, and is not part of the submitted paper. In case of acceptance, it will be removed from the final version, that will refer to a complete technical report including this appendix.

This appendix contains a definition of the semantics of HMSCs, and the proof of Theorem 1 proving soundness and completeness of the approach.

7.1 HMSC Semantics

The semantics of a bMSC M is given in terms of sequences of actions allowed by the causal ordering $\leq$. More formally, we have:

Definition 8. *A linearization of a bMSC M is a word $w = a_1....a_{|E|}$ that is the labeling of some linear extension of M (i.e. a total order on E respecting the causal ordering $\leq$). The semantics of M is the set of all its linearizations, and is denoted $Lin(M)$.*

Let us now define the sequential composition of two basic MSCs M_1, M_2.

Definition 9. *Let $M_1 = (E_1, \leq_1, \lambda_1, \phi_1, \mu_1)$, $M_2 = (E_2, \leq_2, \lambda_2, \phi_2, \mu_2)$ be two bMSCs. The* sequential composition *of M_1 and M_2 is denoted $M_1 \circ M_2$, and is the bMSC $M_1 \circ M_2 = (E_1 \uplus E_2, \leq_{1\circ 2}, \lambda_1 \uplus \lambda_2, \phi_1 \uplus \phi_2, \mu_1 \uplus \mu_2)$, where $\leq_{1\circ 2} = (\leq_1 \cup \leq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \phi(e_1) = \phi(e_2)\})^*$, with $\uplus$ denoting disjoint union, and $f_1 \uplus f_2$ denotes a function defined over $Dom(f_1) \uplus Dom(f_2)$, that associates $f_1(x)$ to any $x \in Dom(f_1)$ and $f_2(x)$ to any $x \in Dom(f_2)$.*

The semantics of an HMSC $H = (I, N, \longrightarrow, \mathcal{M}, n_0)$ is the set of words

$$\mathcal{L}(H) = \{Lin(M^\rho) \mid \rho \text{ initial path of } H\}$$

7.2 Proof of correctness

Theorem 1. *Computing solutions for a localization problem using an optimal solution search for the corresponding constaint model is both a sound and complete algorithm.*

To establish soundness anc correctness, we first need a technical lemma establishing correspondence between solutions of CP_H and localized HMSCs.

Lemma 1. *Let H be a HMSC, and CP_H be the associated constaint problem. For every (not necessarily miminal) solution s of CP_H of cost $\mathcal{F}(s)$, there exists a localized extension H_s of H such that $\mathcal{F}(H_s) = \mathcal{F}(s)$.*

Proof : Obviously, for every solution $s = \{X_i, Y_i\}_{i \in 1..|\mathcal{M}|}$, there exists a localized extension H_s such that for every $M_i \in \mathcal{M}$, $f(M_i)$ has X_i as instance set, and Y_i as leader instance. The cost $\mathcal{F}(H_s)$ of any localized extension H_s is $\sum c(M_i, f(M_i))$. The value $\sum c(M_i, f(M_i))$ be achieved by designing each M'_i as follows: set $M'_i = prefix_i \circ M_i$, where $prefix_i$ is:

- a bMSC containing messages from Y_i to any instance in $\big(\phi(min(M_i))\backslash\{Y_i\}\big)\cup \big(\phi(M'_i)\setminus\phi(M_i)\big)$ if $Y_i \in \phi(min(M_i))$
- a bMSC containing messages from Y_i to any instance in $\phi(min(M_i)) \cup \big(\phi(M'_i)\setminus\phi(M_i)\big)\setminus\{Y_i\}$ if $Y_i \in \big(\phi((M'_i))\setminus\phi(M_i)\big)$
- a bMSC containing messages from Y_i to any instance in $\phi(min(M_i)) \cup (\phi(M'_i)\setminus\phi(M_i))$ if $Y_i \in \phi(M_i)\setminus\phi(min(M_i))$

Obviously, taking as leaders and instance sets the choices indicated by solution s to CP_H, and designing $\mathcal{M}' = \{M'_i\}_{i\in 1..|\mathcal{M}|}$ as defined above, we necessarily have that H_s is localized: all bMSCs are localized, and equality constraints impose that two transitions originating from the same node are labeled by bMSCs with the same leaders. Inclusion constraints has as a consequence that for every path $\rho = (n, M'_1, n_1)\dots(n_{k-1}, M'_k, n_k)$, $\phi(min(M^\rho)) = \phi(min(M'_1))$.□

Proof (of Theorem 1): We can now proceed in two directions, showing first soundness, i.e for every optimal solution $\{X_i, Y_i\}$ to a constraint problem CP_H, there exists a localized extension H' which is minimal w.r.t the cost function $\mathcal{F}$. Let s be an optimal solution. By lemma 1, we know that there exists a localized extension H_s with the same cost as s. Now, suppose that there exists H' a localized extension of H, such that $\mathcal{F}(H') < \mathcal{F}(H_s)$. Let f' be the relation mapping bMSCs of $\mathcal{M}$ to bMSCs of $\mathcal{M}'$. Then there are several bMSCs $M_{i_1}, \dots, M_{i_k}$ such that $c(M_i, f'(M_i)) < c(M_i, f(M_i))$, i.e. they are defined over sets of instances $X_{i_1}, \dots, X_{i_k}$ with leaders $Y_{i_1}, \dots, Y_{i_k}$, and still satisfy equality and inclusion constraints. Hence, the solution s can not be optimal as s' obtained by replacing each X_i (resp Y_i) by $\phi(f'(M_i))$ (resp $\phi(min(f'(M_i)))$ is better than s. Contradiction.

Let us now prove completeness, that is that every optimal localized extension H' of H is defined over a set of bMSCs $\{M'_i = f(M_i)\}_{i\in 1..|\mathcal{M}|}$ is such that $s = \{(\phi(M'_i), \phi(min(M'_I)))\}_{i\in 1..|\mathcal{M}|}$ is also an optimal solution for CP_H. Obviously, as H' is localized, s satisfies all equality and inclusion constaints imposed by CP_H, otherwise one can find a path ρ in H with $|\phi(min(M^\rho))| > 1$, or two paths with distinct minimal instances. Now, let us suppose that s is not optimal, that is, there exists $s' = \{(X_i, Y_i)\}_{i\in 1..|\mathcal{M}|}$ such that $\mathcal{F}(s') < \mathcal{F}(s)$. Using lemma 1, we know that there exists a localized extension $H_{s'}$ with the same cost $\mathcal{F}(H_{s'}) = \mathcal{F}(s')$. Hence H' is not optimal, contradiction. □