

Benedikt Bollig¹ and Stefan Haar and Loïc Hélouët²

¹ LSV, ENS Cachan, CNRS, INRIA, France

² IRISA, INRIA, Rennes, France

bollig@lsv.ens-cachan.fr, stefan.haar@inria.fr, loic.helouet@irisa.fr

One of the key challenges when a fault occurs in a distributed system is to find the original causes for the failure. In some cases, a seemingly unimportant phenomenon leads to a complete network collapse, and finding the root(s) of the dysfunction, as well as the way the fault has spread through the system, is a key issue for correcting the faulty system. Usually, running systems are equipped with probes, i.e. software or hardware mechanisms that record occurrences of some events. These logs are then used to retrieve faults. However, logs grow rapidly, and finding explanations without automated tools is almost unfeasible. Therefore, it is preferable to record only a subset of events that occur, and to base diagnosis on a behavioral model of the system. The observations are then correlated to runs of the model to find *explanations*, that is, the set of runs that might have lead to the observation (the logged events). Usually, an explanation is characterized by the existence of an embedding from the observation into the considered run.

This problem has been studied for interleaved models such as finite state machines [10], or concurrent ones such as Petri nets [2] or scenarios [5]. However, these models describe behaviors involving only a bounded number of processes. A challenge is then to propose diagnosis for dynamic and concurrent models. A graph grammar based solution was proposed by [1] to deal with dynamically evolving topologies, but does not consider buffered communications among processes.

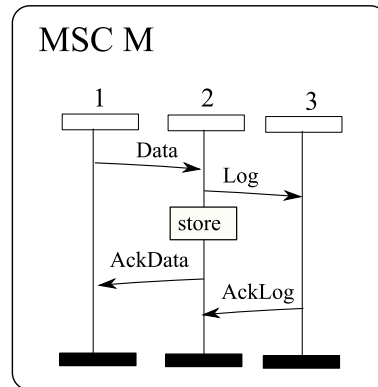


Figure 1. An example of basic MSC

This work defines a diagnosis framework for a scenario based dynamic model, close to the dynamic MSCs proposed in [7]. This model is called *MSC grammars* [3], and consists in context free grammars that compose basic MSCs (descriptions of finite and asynchronous interactions between processes). Figure 1 shows an example of basic MSC over three processes 1,2,3. The semantics of communications in MSCs is asynchronous message exchanges, and it is frequently assumed that these communications are performed using FIFO queues. We will also adopt this convention throughout the paper. MSC grammars are strictly more expressive than High-level MSCs [6], a scenario model that compose basic MSCs defined over a finite set of processes by means of automata. As usual for context free grammars, our MSC grammars contain an axiom, a set of non-terminals and rewriting rules, but terminals are basic MSCs. We furthermore allow renaming of processes in the terminal basic MSCs, which allows for the definition of behaviors over an arbitrary number of processes. The semantics of an MSC grammar \mathcal{G} is the set of MSCs that can be derived from the axiom of \mathcal{G} . Within this setting, a derivation of an MSC grammar can be seen as a run ρ of our model, that produces an MSC M_ρ .

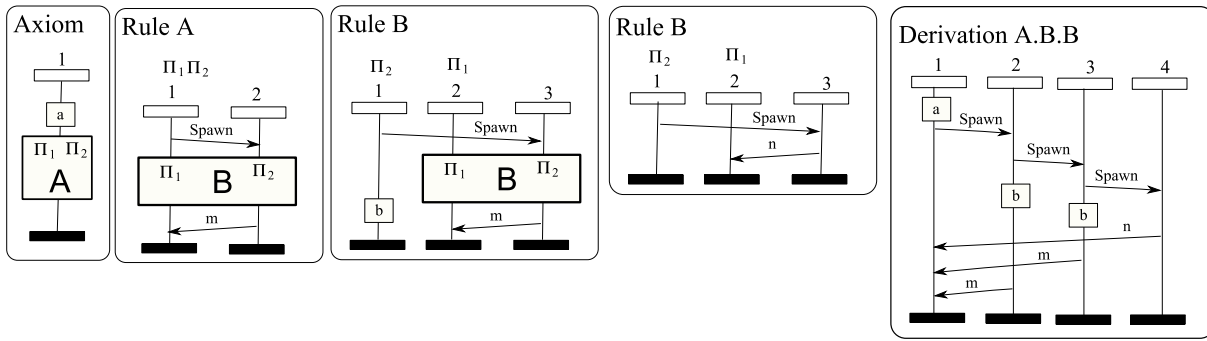


Figure 2. An example of MSC grammar and one of its derivations

Figure 2 shows an example of MSC grammar. Non terminals are named A, B , and identifiers π_1, π_2 are used to refer to some processes during rewriting. Intuitively, these process identifiers play the role of hyperedge vertices in hyperedge replacement grammars [4]. The rightmost MSC in Figure 2 shows a derivation of the grammar, obtained by rewriting successively non-terminals A, B, B , using rules A, B, B . The set of MSCs described by this grammar defines behaviors in which processes are dynamically (and recursively) created from an initial single process. Then all processes execute action b and send message m to the initial process in their reverse order of creation, except for the last process, which sends a message of type n to the initial process, and does not execute action b .

Usually, observation mechanisms collect a subset of events that have occurred in a system, and store them as one or more log files. Observing a fixed set of communicating processes is easy, as the use of vectorial clocks [9] allows to record the exact computation played by a system. However, observing all events and their respective causal ordering in a system is too costly. Hence, one usually relies on a partial observation, i.e. defines an observation alphabet, and processes signal only occurrences of actions within this alphabet. Similarly, logs recall only a subset of the actual ordering among observed events. However, a frequent and reasonable assumption is that the set of events observed on a single process are totally ordered. In a dynamic context where the set of running processes is not known a priori, we can suppose that dynamically created processes execute several copies of some instrumented code, hence yielding an alphabet of observable actions per process and also a total ordering of observed events on each process. The causal ordering among events located on distinct processes is necessarily weaker in a dynamic context, as finite width vectorial clocks cannot be used. However, one can still tag observations with useful information (for instance if a message was sent to some process since the last observation) to recover some ordering among observed events. We can hence suppose that the observations produced by an instrumented system are *partial orders*. Figure 3 depicts a typical observation architecture. Processes are represented by squares, and their communication links as lines. Dynamically created processes are represented by dashed lines. Each process continuously sends information about the observable events it has executed, and the observers collect these information to form a coherent observation. When a diagnosis is needed, observers send the collected observation to a diagnoser that compares the observed partial order with a model of the system, and outputs a diagnosis.

Usually, the question whether an MSC M is a valid explanation for an observation O resumes to finding an *embedding* relation that maps events of the observation onto events of the MSC, while respecting locality of events (i.e events are executed by the same process in the observation if and only if they are mapped onto events located on the same process in the MSC), labeling, and causal ordering (ordered events in the observation are mapped to ordered events in the explanation). The example of Figure 4 shows an observation O (on the left) and a possible explanation M (on the right) when observable events are atomic actions a and b , and reception of message n . In the observation, causal ordering among events is symbolized by a dashed line. The embedding relation from observation O to MSC M is depicted as a thick dotted line. Note that in a dynamic context, the identity of processes is irrelevant, so events of a process can be mapped onto events located on a process with different name. Note also that all processes need not be observed, and that the creation of processes need not be observable. In the example of Figure 4, no event executed by process 4 is observed.

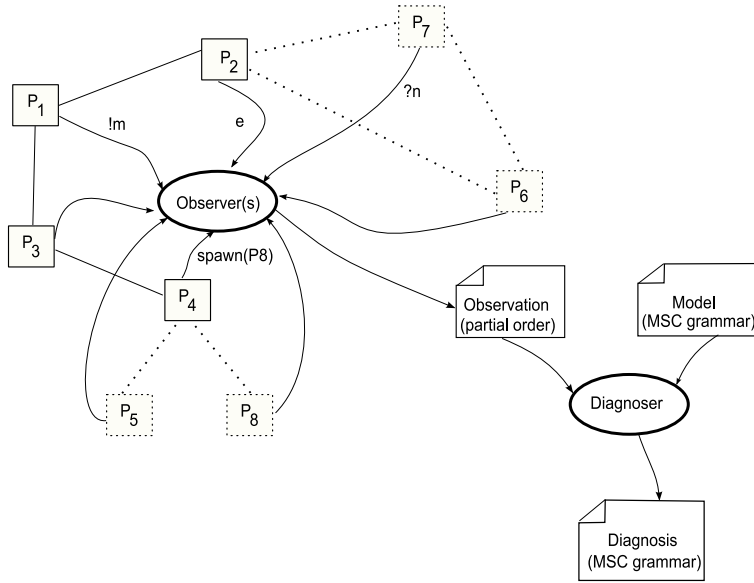


Figure 3. An observation architecture

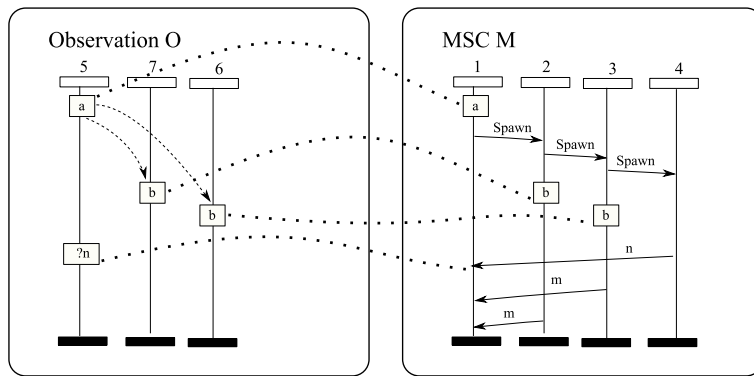


Figure 4. An observation and a possible explanation for it

In this work, we first show that the question whether an MSC M_ρ embeds an observation O is equivalent to the question whether M_ρ satisfies an MSO property ϕ_O computed from the observation. Indeed, it is possible to consider the MSCs generated by MSC grammars as graphs, whose vertices are events and whose edges symbolize messages, process creations, or the direct successor relation on a process. We can then use the monadic second-order logic (MSO) over MSCs defined in [7] (plus an additional labeling relation) to check properties of MSCs, and in particular the embedding of a given observation. The translation from O to ϕ_O associates an event variable to each event in O , and ensures that causal ordering among observed events is respected in all models of the formula. The syntax of MSO over MSCs is of the form :

$$\begin{aligned} \phi ::= & \text{lab}_a(x) \mid (u, x) \rightarrow (v, y) \mid x \leq y \mid x \in X \mid u \in U \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x \phi \mid \exists X \phi \mid \exists u \phi \mid \exists U \phi \end{aligned}$$

where \rightarrow denotes the message relation and \leq denotes the immediate successor on a process relation, $\text{lab}_a(\cdot)$ denotes the labeling of an event by a . As for variables, x, y are individual first order event variables, and X is a second-order set variable, which is interpreted over sets of events. Variables u and U are respectively first-order and second order process variables. The formula associated to observation O of Figure 4 is :

$$\begin{aligned} \phi_O ::= & \exists x, y, z, t, u, v, w, \text{lab}_a(x) \wedge \text{lab}_b(y) \wedge \text{lab}_b(z) \wedge \text{lab}_{?n}(t) \\ & x \leq t \wedge (u, x) \leq (v, y) \wedge (u, x) \leq (w, z) \end{aligned}$$

In this example, (u, x) denotes an event x located on a process u , and \leq denotes causal ordering, which is easily definable in terms of \leq and \rightarrow as soon as set quantification is allowed [8].

The second step of this work is to show that verifying whether an MSC grammar \mathcal{G} satisfies an MSO formula ϕ (i.e. if there exists a run of the grammar satisfying ϕ) is decidable. The proof is obtained by building a tree automaton that recognizes all parse trees of \mathcal{G} , and then decorating the states of this tree automaton with sub-formulae of ϕ (this result relies on standard techniques, that were already used in [7]). This decorated automaton is then a *recognizer* for all MSCs in the language of the grammar that satisfy ϕ . This result provides an immediate means to perform diagnosis from MSC grammars, by construction of a recognizer for the formula ϕ_O attached to observation O . This tree automaton accepts a run of \mathcal{G} if and only if M_ρ embeds the observation, and can be seen as a new grammar \mathcal{G}' that generates only explanations for O .

Though the complexity of model checking MSO formulae may seem prohibitive for practical applications, this first step demonstrates clearly the feasibility of diagnosis from MSC grammars. Furthermore, a clear advantage of this approach is that diagnosis performed this way is *compositional*, as merging two diagnosis obtained from distinct observations simply consists in a product of the tree automata computed for diagnosis. To the best of our knowledge, this is the first time a diagnosis framework is proposed for an infinite state, dynamic and concurrent model with communication buffers.

Références

1. P. Baldan, T. Chatain, S. Haar, and B. König. Unfolding-based diagnosis of systems with an evolving topology. *Information and Computation*, 2010. To appear.
2. A. Benveniste, E. Fabre, C. Jard, and S. Haar. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5) :714–727, May 2003.
3. B. Bollig and L. Hélouët. Realizability of dynamic MSC languages. In *Proc. of CSR 2010, Computer Science in Russia*, June 2010.
4. A. Habel. *Hyperedge Replacement : Grammars and Languages*, volume 643 of *LNCS*. Springer Verlag, 1992.
5. L. Hélouët, T. Gazagnaire, and B. Genest. Diagnosis from scenarios. In *Proc. of WODES'06*, 2006.
6. ITU-TS Recommendation Z.120 : Message Sequence Chart , 2004.
7. M. Leucker, P. Madhusudan, and S. Mukhopadhyay. Dynamic message sequence charts. In *FSTTCS'02*, volume 2556 of *LNCS*, pages 253–264. Springer, 2002.
8. P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The mso theory of connectedly communicating processes. In *FSTTCS*, pages 201–212, 2005.
9. F. Mattern. On the relativistic structure of logical time in distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.
10. M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2) :105–124, 1996.