

# Pattern Matching and Membership for Hierarchical Message Sequence Charts

Blaise Genest<sup>1\*</sup> and Anca Muscholl<sup>2</sup>

<sup>1</sup> IRISA/CNRS, Campus de Beaulieu, F-35042 Rennes

<sup>2</sup> LIAFA, Université Paris VII, case 7014, 2 pl. Jussieu, F-75251 Paris cedex 05

**Abstract.** Several formalisms and tools for software development use hierarchy in system design, for instance statecharts and diagrams in UML. Message sequence charts (MSCs) are a standardized notation for asynchronously communicating processes. The norm Z.120 also includes hierarchical HMSCs. Algorithms on MSCs rarely take into account all possibilities covered by the norm. In particular, hierarchy is not taken into account since the models that are usually considered are (flat) MSC-graphs, that correspond to the unfolding of hierarchical HMSCs. However, complexity can increase exponentially by unfolding. The aim of this paper is to show that basic algorithms can be designed such that they avoid the costly unfolding of hierarchical MSCs and HMSCs. We show this for the membership and the pattern matching problem. We prove that the membership problem for hierarchical HMSCs is PSPACE-complete. Then we describe a polynomial time algorithm for the pattern matching problem on hierarchical MSCs.

## 1 Introduction

It is common to use macros when writing a program or modeling a system. Macros (or hierarchical models) enable the modular design of complex systems. They also present the advantage of succinctness and better readability. Several formalisms and tools for software development use hierarchy in system design. One of the most prominent examples is the formalism of statecharts [11], which is a component of several object-oriented notations, such as the Unified Modeling Language (UML). Besides statecharts, UML widely uses several kinds of diagrams (activity, interaction diagrams etc), all based on the ITU standard Z.120 of message sequence charts (MSC for short). While statecharts extend finite state machines (FSM for short) by hierarchy and communication mechanisms, MSC is a visual notation for asynchronously communicating processes. The usual application of MSCs in telecommunication is for capturing requirements of communication protocols in form of scenarios in early design stages. MSCs usually represent incomplete specifications, obtained from a preliminary view of the system that omits several details, such as variables or message contents. High-level MSCs (HMSCs) combine basic MSCs using choice and iteration, thus describing possibly infinite collections of scenarios. For abstract specifications as with

---

\* The results were obtained while being affiliated with LIAFA, Université Paris 7.

HMSCs, hierarchy is of primary importance. Since a scenario corresponds to a specification level which can be very abstract, a designer should be able to merge different specification cases yielding the same abstract scenario and to use this scenario as a macro. By using macros designers may identify sub-scenarios which have to be refined at a later stage. Thus we focus in this paper on hierarchical MSCs (or nested MSC, nMSC for short) and hierarchical HMSCs (nested HMSC, nHMSC for short).

Algorithms on MSCs rarely take into account the whole spectrum of the HMSC standard definition. In particular, hierarchy is not taken into account since the models usually considered are MSC-graphs (that correspond to the unfolding of nHMSCs). However, complexity can increase exponentially by unfolding. The aim of this paper is to show that this exponential blow-up is often unnecessary, since the costly unfolding can be avoided.

In this paper we consider two basic problems for the algorithmic verification of nMSCs and nHMSCs, the *membership problem* and *pattern matching*. We believe that the techniques described here can be used to solve other problems on nHMSCs as well. The membership problem occurs for instance when a negative scenario must be excluded from the specification, or when we check that a positive scenario is already covered by the specification. Without hierarchy, membership of an MSC against an HMSC is NP-complete [1]. The reason for this complexity blow-up (compared to FSM) is that MSCs are partial order models. We show that hierarchy yields a small increase in complexity, precisely we show that the membership problem of an nMSC against an nHMSC is PSPACE-complete. Surprisingly, hierarchy alone is the source of this complexity. We show namely that the membership problem for hierarchical automata is already PSPACE-complete. This result shows a difference between membership and reachability, since reachability for communicating hierarchical automata is EXPSPACE-complete [12].

The second problem considered in this paper is pattern matching for nMSCs. Given two nMSCs  $M, N$ , we want to know whether  $M$  occurs as a pattern of  $N$ . A polynomial time solution for this problem is not immediate. We apply some nice combinatorial techniques stemming from pattern matching on compressed texts and we obtain an algorithm of time  $O(|\mathcal{C}_M|^2 \cdot |M|^2 \cdot |N|^2)$ , where  $|M|, |N|$  denote the sizes of the description of  $M$  and  $N$ , and  $|\mathcal{C}_M|$  is the number of connected components in the communication graph of  $M$ . This question subsumes the test of equality of two nMSC, and shows that equality is decidable in PTIME as well.

*Related work.* For extended FSMs, [12] considers the reachability and trace equivalence problems for communicating FSMs. Model checking hierarchical FSMs against LTL and CTL properties is the topic of [4]. The paper [3] combines hierarchy and concurrency, analyzing the complexity of several problems (reachability, equivalence etc.) for communicating, hierarchical FSMs.

Several verification problems on MSCs and MSC-graphs have been considered over the last years, such as detecting races [2, 19], model checking [5], pattern matching with gaps [20], inference [1], realizability [18, 9, 15], and model checking against partial order logics [17, 22]. Hierarchical MSCs have been also considered

in [5] for the model checking problem. We note however that our definition of nHMSCs captures a larger class of MSC specifications than [5].

An extended abstract of this paper was presented at LATIN'02 [8]. As additional result here we show how to extend the polynomial time algorithm for pattern matching nMSCs to the case where the pattern is not connected.

## 2 Syntax and Semantics of Nested MSCs

We adopt the definition of (basic) message sequence charts (MSC for short), as described in the ITU-standard [13].

**Definition 1. (Message Sequence Charts.)** *A message sequence chart is a tuple  $M = \langle P, E, \mathcal{C}, \ell, m, < \rangle$  where:*

- $P$  is a finite set of processes,
- $E$  is a finite set of events, each located on some process from  $P$ ,
- $\mathcal{C}$  is a finite set of names for messages and local actions,
- $\ell : E \rightarrow \mathcal{T} = \{i!j(c), i?j(c), i(c) \mid i \neq j \in P, c \in \mathcal{C}\}$  labels each event with its type: on process  $i \in P$ , the type is either a send  $i!j(c)$  of message  $c$  to process  $j$ , or a receive  $i?j(c)$  of message  $c$  from process  $j$ , or a local event  $i(c)$ . The labeling  $\ell$  partitions the set of events by type (send, receive, or local),  $E = S \cup R \cup L$ , and by process,  $E = \bigcup_{i \in P} E_i$ . We denote by  $P(e)$  the process of event  $e$  (i.e.,  $P(e) = i$  if  $e \in E_i$ ).
- $m : S \rightarrow R$  is a bijection matching each send to the corresponding receive. If  $m(s) = r$ , then  $\ell(s) = i!j(c)$  and  $\ell(r) = j?i(c)$  for some processes  $i, j \in P$  and some message content  $c \in \mathcal{C}$ . We denote the events  $s, r$  as matching events and the pair  $(s, r)$  as message.
- $< \subseteq E \times E$  is an acyclic relation between events consisting of:
  - a total order on  $E_i$ , for every process  $i \in P$ , and
  - $s < r$ , whenever  $m(s) = r$ .

The upper left part of Figure 1 depicts an MSC  $M$  on three processes with two messages and four events. Each vertical line corresponds to a process, with time increasing from top to bottom. By  $P(M)$  we denote the set of processes of  $M$ .

For the questions considered here, message names are irrelevant. Thus, send events will be of type  $i!j$  and receive events of type  $i?j$ . Moreover, whenever we refer to an MSC in this paper, we mean actually its isomorphism class, where an isomorphism on the set of events  $E$  is a bijection that is compatible with the type function  $\ell$  and the message function  $m$ .

For communication protocols it is natural to assume that each communication channel  $(i, j)$ ,  $i \neq j$ , delivers messages first-in-first-out (*FIFO rule*). We assume the FIFO condition throughout the paper. That is, for all messages  $(s_k, r_k)$ ,  $k = 1, 2$ , such that  $\ell(s_1) = \ell(s_2)$  and  $\ell(r_1) = \ell(r_2)$  we require that  $s_1 < s_2$  iff  $r_1 < r_2$ . The reflexive-transitive closure of the acyclic relation  $<$  is a partial order called *visual order* and is denoted as  $\leq$ . A total order on  $E$  extending  $\leq$

is called a *linearization* of  $M$ . A *configuration* (prefix)  $C$  of an MSC  $M$  is a downward closed subset of events, that is, if  $e \leq f \in E$  with  $f \in C$ , then  $e \in C$ .

Note that the FIFO rule implies that an MSC can be reconstructed from any of its linearizations. We obtain the MSC from the event sequence simply by matching the  $n$ -th send from  $i$  to  $j$  with the  $n$ -th receive on  $j$  from  $i$ , for each pair of distinct processes  $(i, j)$ .

A special case of the pattern matching problem considered in the paper is the equality test of two (nested) MSCs. In order to check the equality of two MSCs  $M, N$  (i.e., up to isomorphism) one can choose any linearization of  $M$  and check whether it is a linearization of  $N$ , too. An alternative approach, that will be used in our algorithms, is to check equality on each process. Thus, for an MSC  $M = \langle P, E, \mathcal{C}, \ell, m, < \rangle$  and a process  $i \in P$  we let  $M|_i$  denote the projection of  $M$  on the set  $E_i$  of events located on  $i$ . That is,  $M|_i$  is the sequence of events of  $M$  on process  $i$ . We then have  $M = N$  if and only if  $M$  and  $N$  have the same set of processes, that is  $P(M) = P(N) = P$ , and if their projections are equal, that is  $M|_i = N|_i$  for each  $i \in P$  (up to isomorphism). Note that both tests rely on the FIFO rule. Without this rule, a linearization (or the projections) does not suffice for reconstructing the MSC. For example, the linearization  $s_1 s_2 r_1 r_2$  where  $s_1, s_2$  are sends and  $r_1, r_2$  are receives from process 1 to process 2, can produce two MSCs, one where  $m(s_1) = r_1, m(s_2) = r_2$  and one where  $m(s_1) = r_2, m(s_2) = r_1$ .

We follow the ITU norm and define *nested MSCs* (nMSC for short) by allowing the reuse of an already defined MSC in a definition. The definition we give below aims at preserving the visual character of MSCs (see also Figure 1).

**Definition 2. (Nested MSC, nMSC.)** *A nested MSC  $M = (M_q)_{q=1,n}$  is a finite sequence of macros of the form  $M_q = \langle P_q, E_q, B_q, \varphi_q, \mathcal{C}, \ell_q, m_q, <_q \rangle$ .*

*Each macro  $M_q$  consists of:*

- *A finite set  $P_q$  of processes.*
- *A finite set  $E_q$  of events, each event is located on some process from  $P_q$ .*
- *A finite set  $B_q$  of references (boxes) used by  $M_q$ .*
- *A function  $\varphi_q$  that associates each reference  $b \in B_q$  with an index  $q < \varphi_q(b) \leq n$ . Thus, reference  $b$  refers to the macro  $M_{\varphi_q(b)}$ . We require that  $P_{\varphi_q(b)} \subseteq P_q$ .*
- *The type function  $\ell_q : E_q \rightarrow \mathcal{T}$ , that associates each event with a type  $il?j(c), i?j(c)$  or  $i(c)$ , with  $i, j \in P_q, i \neq j$  and  $c \in \mathcal{C}$ . The labeling  $\ell$  partitions the set of events by type (send, receive, or local),  $E_q = S_q \cup R_q \cup L_q$ , and by process,  $E_q = \bigcup_{i \in P} E_{q,i}$ . We denote by  $P(e)$  the process of event  $e$  (i.e.,  $P(e) = i$  if  $e \in E_{q,i}$ ).*
- *The message function  $m_q : S_q \rightarrow R_q$  that maps each (send) event of type  $il?j(c)$  with a (receive) event of type  $j?i(c)$ , for some  $i \neq j, c \in \mathcal{C}$ .*
- *The acyclic relation  $<_q$  over the set of events and references  $E_q \cup B_q$ , defined by:*
  - *For each process  $k \in P_q$ , the relation  $<_q$  is a total order over the set  $E_{q,k}$  of events located on  $k$  and the set of references  $b \in B_q$  with  $k \in P_{\varphi_q(b)}$ .*
  - *$e <_q f$  whenever  $m_q(e) = f$  in  $M_q$ .*

The nesting depth of  $M$  is the maximal  $d$  such that there exists some sequence  $q_1 < \dots < q_{d+1}$  with  $\varphi_{q_j}(b) = q_{j+1}$  for some  $b \in B_{q_j}$ , for all  $1 \leq j \leq d$ .

In the spirit of straight-line program notation, higher levels of hierarchy correspond to lower indices. Thus, the MSC  $M$  defined by  $M = (M_q)_{q=1,n}$  will be  $M_1$ . We will depict references in pictures (see Figure 1) as boxes that overlap the processes that occur in the corresponding macro.

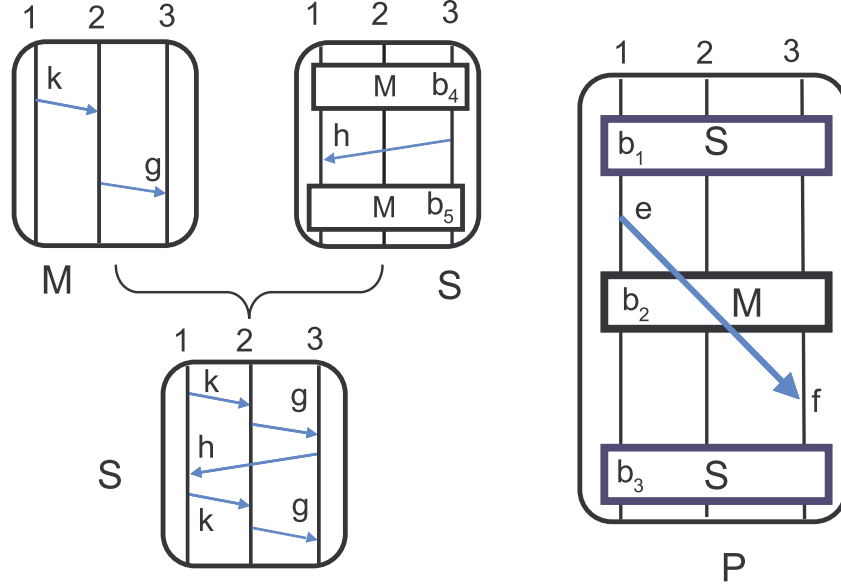


Fig. 1. An nMSC  $P$  using two references,  $S$  and  $M$ .

*Example 1.* Consider the nMSC  $P$  in Figure 1. It uses three references,  $B_P = \{b_1, b_2, b_3\}$  that correspond to  $\varphi_P(b_1) = \varphi_P(b_3) = S$  and  $\varphi_P(b_2) = M$ . The nesting depth of  $P$  is 2. The visual order  $<_P$  of  $P$  requires on process 1 the order  $b_1 <_P e <_P b_2 <_P b_3$ . Notice that the definition of an nMSC would not allow  $(f, e)$  to be a message, since this would contradict the acyclicity of the relation  $<_P$ .

The semantics of an nMSC is the MSC defined by replacing each reference of  $M$  by the corresponding MSC. Inductively it suffices to define the semantics of nMSCs of nesting depth one. Let  $M = (M_q)_{q=1,n}$  be an nMSC of nesting depth one, with  $M_q = \langle P_q, E_q, B_q, \varphi_q, C, \ell_q, m_q, <_q \rangle$ . For simplifying the notation below, we write instead of  $\varphi_1(b)$  just  $b$ .

The MSC  $\langle P, E, \mathcal{C}, \ell, m, \prec \rangle$  defined by  $M = (M_q)_{q=1,n}$  is given by  $P = P_1$ ,  $E = \bigcup_{b \in B_1} E_b \cup E_1$ ,  $\ell = \cup_{q=1,n} \ell_q$  and  $m = \cup_{q=1,n} m_q$ . The visual order  $\prec$  is defined by  $e \prec f$  if and only if either  $m(e) = f$ , or  $P(e) = P(f)$  and one of the following conditions holds:

- $e, f \in E_1$  and  $e \prec_1 f$ ,
- $e, f \in E_b$  and  $e \prec_b f$ ,
- $e \in E_1, f \in E_b$  and  $e \prec_1 b$ ,
- $e \in E_b, f \in E_1$  and  $b \prec_1 f$ ,
- $e \in E_b, f \in E_{b'}$  and  $b \prec_1 b'$ ,

where  $b, b' \in B_1$ . For simplicity, we denote the MSC defined by  $M = (M_q)_{q=1,n}$  as  $M$ , too.

*Example 2.* For the nMSC  $P$  in Figure 1, the lower right part of the picture shows the MSC defined by  $S$ . Note that event  $g \in E_M$  occurs twice in  $S$  – for simplicity, we denote both occurrences as  $g$ .

Note also that the semantics requires that  $b_1 \prec_1 e$ , but this does not mean that all events of  $S = \varphi_P(b_1)$  must happen before  $e \in E_P$ . For instance, the first occurrence of  $g$  in  $S$  precedes event  $e$  of  $P$ , but the second occurrence is concurrent with  $e$ .

*Remark 1.* Obviously, a syntactically correct nMSC  $M$  might not yield an MSC because of the FIFO order. For example, the message  $(e, f)$  of  $P$  would violate the FIFO condition if  $M$  would contain a message from process 1 to process 3. Fortunately, it can be verified easily (in polynomial time) whether an nMSC satisfies the FIFO condition. For checking the FIFO condition, it suffices to test that there is no  $e \prec g \prec h \prec f$  and no  $e \prec b \prec f$  with  $b$  containing a send from  $i$  to  $j$ , where  $(e, f), (g, h)$  are two messages from  $i$  to  $j$ .

**Size of nMSC.** For complexity estimations we will denote by  $\wp$  the overall number of processes. The size of an nMSC  $M$  is denoted as  $|M|$ . It represents the size of the syntactical description of  $M$ , where an event is of size one and the size of a reference is the number of its processes.

### 3 Nested High-Level MSC

An MSC can only describe a finite scenario. For specifying more complex behaviors, in particular infinite sets of scenarios, the ITU norm proposes to compose MSCs in form of MSC-graphs, by using choice and iteration.

**Definition 3. (MSC-graph)** An MSC-graph is a tuple  $G = \langle V, E, s, f, \varphi \rangle$ , where:

- $(V, E)$  is a directed graph with starting vertex  $s \in V$  and final vertex  $f \in V$ .
- Each vertex  $v$  is labeled by the MSC  $\varphi(v)$ .

In the same way as we defined nested MSCs from (flat) MSCs we can generalize MSC-graphs to *hierarchical HMSCs* (or *nested high-level MSCs*, nHMSC for short).

**Definition 4. (Nested high-level MSC.)** An nHMSC is a finite sequence  $G = (G_q)_{q=1,n}$ , where each  $G_q$  is either a labeled graph or an nMSC. A labeled graph  $G_q$  is a tuple  $\langle V_q, E_q, \varphi_q, s_q, f_q \rangle$  consisting of:

- A directed graph  $(V_q, E_q)$  with starting vertex  $s_q$  and final vertex  $f_q$ .
- A function  $\varphi_q$  that associates each vertex  $v$  with a reference  $q < \varphi_q(v) \leq n$ , representing  $G_{\varphi_q(v)}$ .

Thus, a node in an nHMSC can be mapped either to some graph or to an nMSC. This definition combines hierarchical automata as defined in [4] with our definition of nMSC. The special case where there is only one process (i.e., no concurrency) yields the hierarchical automata used in [4]<sup>1</sup>.

We first need to define the composition of two MSCs  $N_1N_2$  with  $N_k = \langle P_k, E_k, \mathcal{C}_k, \ell_k, m_k, <_k \rangle$ . The intuition behind the composition is simple, we just glue together the two diagrams process by process. So let  $N_1N_2 = \langle P, E, \mathcal{C}, \ell, m, < \rangle$  with  $E = E_1 \cup E_2$ ,  $P = P_1 \cup P_2$ ,  $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ ,  $\ell = \ell_1 \cup \ell_2$ ,  $m = m_1 \cup m_2$  and

$$< = <_1 \cup <_2 \cup \bigcup_{i \in P} E_{1,i} \times E_{2,i}.$$

The semantics of an nHMSC  $G = (G_q)_{q=1,n}$  is a set of MSCs  $L(G)$  defined recursively. If  $G_q$  is an nMSC, then  $L(G_q)$  is a singleton consisting of the MSC defined by  $G_q$ . Consider a labeled graph  $G_q$ . Then  $L(G_q)$  is the set of MSCs associated with the accepting paths of  $G_q$ , that is, paths starting in  $s_q$  and ending in  $f_q$ . With a path  $v_1, \dots, v_n$  in  $G_q$  we associate the set of all MSCs  $M_1 \cdots M_n$ , where  $M_i \in L(G_{\varphi_q(v_i)})$  for all  $1 \leq i \leq n$ . The set of executions of  $G$  is defined as  $L(G) = L(G_1)$ .

As in [1] we also consider a weaker semantics for nHMSCs, that does not use the composition of MSCs (called *weak closure* in [1]). This semantics is based on taking the product of the sequential behaviors of single processes. Several algorithmic problems can be solved more efficiently for the weak closure of MSC-graphs. This makes it interesting to compare it with the usual semantics also in the setting of nHMSCs.

**Weak closure of nHMSC.** Let  $G$  be an nHMSC. Then  $L^w(G)$  denotes the set of MSCs  $M$  such that for each process  $i$  there is some MSC  $N \in L(G)$  such that  $M|_i$  is equal to  $N|_i$ . Note that  $L(G) \subseteq L^w(G)$  and that the inclusion is strict, in general (see [1]).

<sup>1</sup> Actually, [4] allows several final nodes in each automaton, which counts for the complexity of their algorithms.

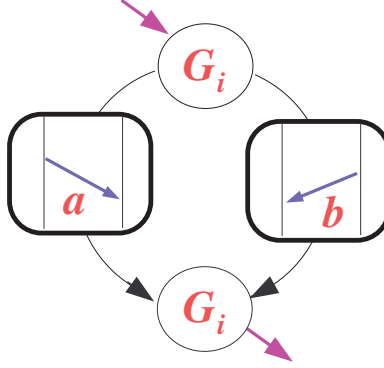


Fig. 2. An nHMSC  $G_{i+1}$  generating  $(a + b)^{2^i - 1}$  with  $G_1 = \epsilon$ .

## 4 Membership Problem

Checking the membership of an MSC  $M$  in an MSC-graph  $G$  is used typically for checking that no bad scenario can occur in a given specification. Another application is checking whether a good scenario is already covered by the specification. Checking membership is not an easy task already because of the concurrency implied by the MSC composition, all the more in the presence of hierarchy. The MSC membership problem  $M \stackrel{?}{\in} L(G)$  with  $M$  an MSC and  $G$  an MSC-graph was considered in [1], together with the weak membership problem  $M \stackrel{?}{\in} L^w(G)$ . The results of [1] can be summarized as follows:

- The MSC membership problem is NP-complete. A deterministic algorithm of time  $O(|G| \cdot |M|^\varphi)$  solves it<sup>2</sup>, where  $\varphi$  is the number of processes.
- The *weak* MSC membership problem is solvable in time  $O(|G| \cdot |M|)$ .

So the MSC membership problem is solvable in polynomial time if we fix the number of processes.

### 4.1 Hierarchical Membership Problem

The membership problem seems *a priori* more difficult for an nMSC  $M$  against an nHMSC  $G$ , since the naive approach of guessing a path of  $G$  and checking equality with  $M$  is too expensive (both the path of  $G$  and the MSC defined by  $M$  can be of exponential size). However, it is easy to show that we can test membership in polynomial space:

**Theorem 1 (Hierarchical MSC Membership Problem)** *Given an nMSC  $M$  and an nHMSC  $G$ , we can decide whether  $M \stackrel{?}{\in} L(G)$  in polynomial space.*

<sup>2</sup> This is a slightly improved runtime compared to the result stated in [1].



*Proof.* The idea of the algorithm is quite straightforward. We guess an MSC in  $L(G)$  and we match it against the nMSC  $M$ , however expanding neither  $M$  nor  $G$ . Recall that for testing equality of two MSCs  $M, N$ , it suffices to choose one linearization of  $N$  and check whether it is a linearization of  $M$ . Hence, we can choose the linearization of the MSC in  $G$ . We consider only linearizations in  $\text{Lin}^0(G)$ , where  $\text{Lin}^0(G)$  is defined recursively. If  $G_q$  is an nMSC, then  $\text{Lin}^0(G_q)$  is the set of linearizations of  $G_q$ . With a path  $v_1, \dots, v_n$  in  $G_q$  we associate the set of all linearizations  $u_1 \cdots u_n$ , where  $u_i \in \text{Lin}^0(G_{\varphi_q(v_i)})$  for all  $1 \leq i \leq n$ . Let us consider a labeled graph  $G_q$ . Then  $\text{Lin}^0(G_q)$  is the set of linearizations associated with accepting paths of  $G_q$ , that is, paths starting in  $s_q$  and ending in  $f_q$ . We define  $\text{Lin}^0(G) = \text{Lin}^0(G_1)$ .

We need also to store a configuration of  $M$  in polynomial space, corresponding to the events already matched with the events from  $G$ . Since a configuration in an MSC is a downward closed set of events, it can be stored as a tuple of  $\wp$  events (recall that  $\wp$  is the number of processes), representing the last event of the configuration on each process. Such a tuple is of linear size w.r.t. the size of  $M$ . Each event  $e$  of the (unfolded)  $M = (M_q)_{q=1,n}$  is represented by a sequence  $b_1 < \dots < b_m$  of references corresponding to the unfolding of references yielding  $e$ . That is, we inductively store  $b_m$ , where event  $e$  belongs to  $M_{\varphi(b_m)}$  (and  $b_m$  is a reference used by  $M_{\varphi(b_{m-1})}$ ), plus the position of  $e$  in  $M_{\varphi(b_m)}$ . Thus, each event can be stored in linear space. In Figure 1, the first occurrence of  $g$  in  $P$  corresponds to  $(b_1, b_4, g)$ , the second occurrence to  $(b_1, b_5, g)$ , and so on.

Similarly, we can store the current configuration of the linearization in  $\text{Lin}^0(G)$  in polynomial space (an event of  $G$  is represented by a sequence of nodes and references). Since a new node is started only after the linearization of the previous node is completed, the last events on each process belong to the same node. The nondeterministic algorithm consists in guessing a successor configuration of  $G$ , obtained by extending the current configuration by an event  $e$  such that the new configuration is still a prefix of some linearization in  $\text{Lin}^0(G)$ . Then we check that  $e$  can extend the current linearization of  $M$  as well. The algorithm stops when the configuration that corresponds to the path being guessed in  $G$  is equal to  $M$  and the path of  $G$  is accepting. □

Theorem 2 below shows that PSPACE is the best complexity for the hierarchical membership problem. The lower bound holds even if there is only one process (Theorem 2), or if the graph  $G$  is not hierarchical (Theorem 3), but not both (Theorem 4). This shows also that fixing the number of processes does not lower the complexity of the problem, unlike the nonhierarchical case.

We show the PSPACE lower bound for the following problem: given a straight-line program  $W$  (see below) and a hierarchical automaton  $\mathcal{A}$ , test whether  $W \in L(\mathcal{A})$ . This question corresponds to the hierarchical membership problem with a single process. Notice also that the weak membership problem  $M \stackrel{?}{\in} L^w(G)$  [1] can be reduced to this question.

**Straight-line programs.** A straight-line program (SLP for short) over the alphabet  $\Sigma$  is a context-free grammar with variables  $V = \{X_1, \dots, X_k\}$ , initial variable  $X_1$  and rules from  $V \times (V \cup \Sigma)^+$ . The rules are such that there is exactly one rule for each left-hand side variable and if  $X_i \rightarrow \alpha$ , then each  $X_j$  in  $\alpha$  satisfies  $j > i$ .

The constraints on the rules make that any variable  $X_i$  generates a unique word. For convenience, we denote the word generated by the variable  $X_i$  also as  $X_i$ . The length of a variable  $X_i$  represents the length of the word generated by  $X_i$  and is denoted as  $\|X_i\|$ . Clearly,  $\|X_i\|$  can be at most exponential in the number of rules. The size of an SLP  $X_i$  is the sum of the sizes of the rules and is denoted by  $|X_i|$ . Without loss of generality, we can assume that rules are of size 2, that is of the form  $X \rightarrow YZ$  with  $Y, Z \in V \cup \Sigma$ .

Since any MSC  $M$  is determined by its projections  $(M|_i)_{i \in P}$ , an nMSC  $M$  can be identified with  $\wp$  SLPs  $L^i$ ,  $i \in P$ . The SLP  $L^i$  generates the projection  $M|_i$  of  $M$  on the set of events of process  $i \in P$ . We denote the variables used by  $L^i$  as  $X|_i$ , where the variables  $X$  are related one-to-one to the macros  $M_q$  from the definition of the nMSC  $M$ . The initial variable of each  $L^i$  is thus  $M_1|_i$ . These SLPs can be translated in polynomial time into Chomsky normal form.

*Example 3.* For the nMSC  $P$  in Figure 1 we have the following SLP generating the projection on process 1:  $P|_1 \rightarrow S|_1 e M|_1 S|_1$ ,  $S|_1 \rightarrow M|_1 h M|_1$  and  $M|_1 \rightarrow k$ . By adding new variables we can transform these 3 rules into equivalent rules in Chomsky normal form.

A *hierarchical automaton* (hNFA for short) corresponds roughly to an nHMSC over a single process. For clarity we give the definition formally. An hNFA is a sequence of edge-labeled graphs  $\mathcal{A} = (\mathcal{A}_q)_{q=1,n}$ , where  $\mathcal{A}_q = \langle V_q, R_q, \delta_q, \varphi_q, s_q, f_q \rangle$ , with  $V_q$  the finite set of states,  $R_q$  the finite set of references,  $s_q, f_q \in V_q$  the initial and final state. The transition relation  $\delta_q$  is a subset of  $(V_q \cup R_q) \times (\Sigma \cup \{\epsilon\}) \times (V_q \cup R_q)$ . The mapping  $\varphi_q$  associates a reference  $R$  with a subautomaton,  $q < \varphi_q(R) \leq n$ . A transition of the form  $(R, a, v)$  with  $R \in R_q, v \in V_q$  means an  $a$ -labeled transition from the final state of the subautomaton  $\mathcal{A}_{\varphi_q(R)}$  to the state  $v$  of  $\mathcal{A}_q$ . The meaning of transitions  $(v, a, R)$  and  $(R', a, R)$ , is similar, with the transition ending in the initial state of  $\mathcal{A}_{\varphi_q(R)}$ .

**Theorem 2** *It is PSPACE-complete to check whether  $W \in L(\mathcal{A})$  for an SLP  $W$  and an hNFA  $\mathcal{A}$ . If the alphabet is unary, then the membership problem is NP-complete.*

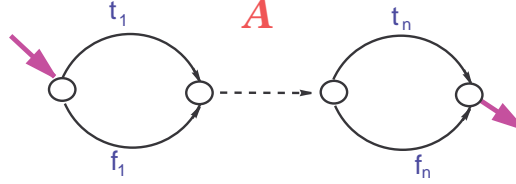
**Remark 1** The NP-hardness result in the unary case also follows from [24].

*Proof.* We first reduce (1-in-3) SAT to the unary membership problem, since we use this reduction in the general case, too. This variant of SAT is still NP-complete, see [25, 6].

Let  $\varphi = \bigwedge_{j=1}^m C(\alpha_j, \beta_j, \gamma_j)$  be an instance of (1-in-3) SAT over  $n$  variables  $(x_i)_{i=1,n}$ . Here, (1-in-3) means that a clause  $C(\alpha_j, \beta_j, \gamma_j)$  is true if *exactly one* of

the literals  $\alpha_j, \beta_j, \gamma_j$  is true. We use the unary alphabet  $\{a\}$ . Clearly, any word  $x \in a^*$  is uniquely defined by its length.

For each integer  $j$ , it is easy to define an SLP (or an hNFA)  $L(j)$  of size polynomial in  $j$  that generates the word  $a^{4^j}$ . We associate with each clause  $C_j = C(\alpha_j, \beta_j, \gamma_j)$  the word  $w_j = L(j)$ . Thus, let  $W = w_1 \cdots w_m \in a^*$  be the word of length  $\sum_{j=1}^m 4^j$ . The hNFA  $\mathcal{A}$  consists of a sequence of choices with transitions labeled by  $t_i$  and  $f_i$ , for  $i$  varying from 1 to  $n$ , where  $t_i \in a^*$  is the word of length  $\sum_{j \in R_i} 4^j$  and  $R_i = \{j \mid x_i \in \{\alpha_j, \beta_j, \gamma_j\}\}$ . In the same way,  $f_i \in a^*$  is the word of length  $\sum_{j \in S_i} 4^j$  and  $S_i = \{j \mid (\neg x_i) \in \{\alpha_j, \beta_j, \gamma_j\}\}$ . Formally, a transition labeled by  $t_i$  corresponds to sequencing the automata accepting  $L(j)$ , for  $j \in R_i$  (similarly for  $f_i$ ).



A maximal path  $\rho$  of  $\mathcal{A}$  corresponds to a valuation  $\sigma$  where each variable  $x_i$  is true if the path chooses  $t_i$ , and false if it chooses  $f_i$ . Let  $n_j$  be the number of literals of  $C_j$  that are set true by  $\sigma$ . Recall that  $\sigma$  satisfies the formula  $\varphi$  iff  $n_j = 1$  for all  $j$ . It is easy to see that  $\rho$  is labeled by the word  $L \in a^*$  of length  $\sum_{j=1}^m n_j 4^j$ . Notice that since each clause has three literals,  $n_j \in \{0, 1, 2, 3\}$  for all  $j$ . The length of  $L$  in base 4 is thus  $(n_m n_{m-1} \dots n_1 0)_4$ . We have  $W = L$  iff  $(11 \dots 10)_4 = (n_m n_{m-1} \dots n_1 0)_4$ , thus iff  $n_j = 1$  for all  $j$ . That is, there is a path in  $\mathcal{A}$  labeled by  $W$  if and only if there is a valuation satisfying  $\varphi$ . This implies that the membership problem for hierarchical automata with a unary alphabet is NP-hard.

We now show the first statement of Theorem 2. We reduce the problem (1-in-3) QBF (one-in-three quantified boolean formula) to the hierarchical membership problem. Let  $\varphi$  be an instance of (1-in-3) QBF of the form  $\varphi = Q_n x_n \cdots Q_1 x_1 \psi$ , where  $Q_i \in \{\exists, \forall\}$  and the formula  $\psi$  is of the form  $\bigwedge_{j=1}^m C(\alpha_j, \beta_j, \gamma_j)$ . As before, a clause  $C_j = C(\alpha_j, \beta_j, \gamma_j)$  is true iff exactly one literal is true. The PSPACE-hardness of this problem is shown in [25, 6].

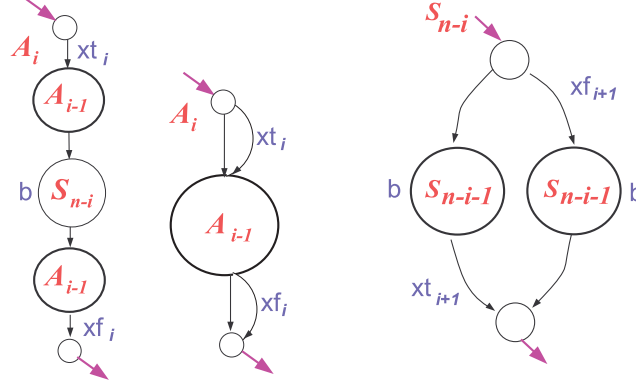
The idea is to make the valuations of the variables correspond to paths in the hierarchical automaton  $(\mathcal{A}_i)_{i=0,n}$  and to validate the valuations using the SLPs  $(W_i)_{i=0,n}$ . We define the automata  $\mathcal{A}_i$  and the SLPs  $W_i$  by induction on  $i = 0, \dots, n$ . We use now the binary alphabet  $\{a, b\}$ . The letter  $a$  will have the same meaning as in the NP case, and the letter  $b$  will be used as a delimiting symbol.

We define the words  $w_j, t_i, f_i \in a^*$  with respect to  $\psi$  as before. That is, each  $w_j = L(j)$  is associated with the clause  $C_j$  and  $t_i, f_i$  are associated with the variable  $x_i$ . Moreover, we associate with each variable  $x_i$  the word  $w_{i+m} \in a^*$  of length  $4^{i+m}$ . Let  $W_0 = w_1 \cdots w_{n+m}$  be the word of  $a^*$  of length  $\sum_{j=1}^{n+m} 4^j$ , and let  $\mathcal{A}_0$  be an automaton consisting of one  $\epsilon$ -transition from its initial state to its

final state. Let also  $S_0$  be an automaton consisting of one transition labeled by  $b$ . The SLP-compressed words  $(W_i)_{i=1,n}$ , are defined by:

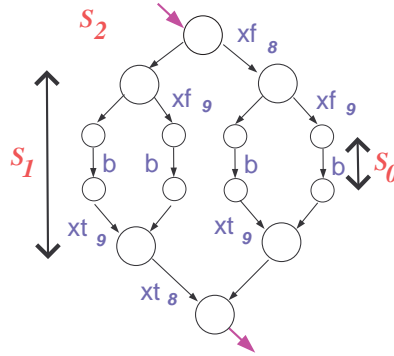
- $W_i \longrightarrow W_{i-1}$ , if  $Q_i = \exists$ ,
- $W_i \longrightarrow W_{i-1} b W_{i-1}$ , if  $Q_i = \forall$ .

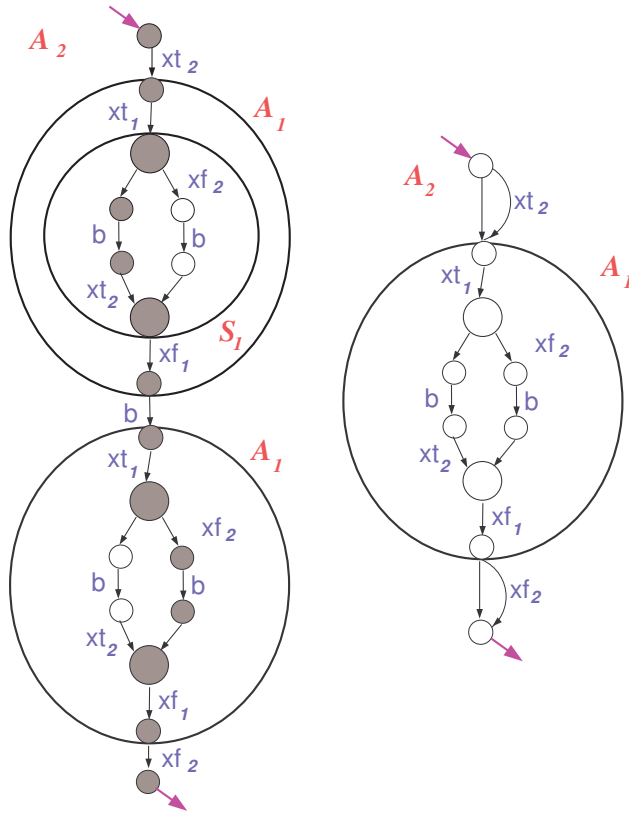
The recursive definition of the hNFA  $(\mathcal{A}_i)_{i=1,n}$  and  $(S_i)_{i=0,n-1}$  is illustrated in the figure below. Transitions are either labeled by  $\epsilon$ , or by  $xt_i = t_i w_{i+m}$  or  $xf_i = f_i w_{i+m}$ . The automaton on the left defines  $\mathcal{A}_i$  when  $Q_i = \forall$ , the automaton in the middle defines  $\mathcal{A}_i$  when  $Q_i = \exists$ , and the automaton on the right defines  $S_i$ . Note that the symbol  $b$  is only generated by  $S_0$ .



The overall idea is as follows. The values of  $x_{i+1}, \dots, x_n$  are already chosen when an automaton calls  $\mathcal{A}_i$  (from a higher hierarchy level). The automaton  $\mathcal{A}_i$  on the left sets  $x_i$  true, then uses  $S_{n-i}$  to recover the fixed values of  $x_{i+1}, \dots, x_n$ , and finally it sets  $x_i$  false. The automaton  $\mathcal{A}_i$  in the middle guesses whether  $x_i$  is true (by taking the transition labeled by  $xt_i$ ) or false (by choosing the transition labeled by  $xf_i$ ). If it chooses both transitions labeled by  $xt_i, xf_i$  (or none of them), then the word labeling this path will not be equal to  $W_n$  because  $W_n$  contains exactly one occurrence of  $w_{i+m}$  between any two consecutive  $b$ 's. We illustrate how  $\mathcal{A}_i$  works in Figure 3, that shows the unfolding of the automaton  $\mathcal{A}_2$  for  $\varphi = \forall x_2 \forall x_1 \psi$  on the left and for  $\varphi = \exists x_2 \forall x_1 \psi$  on the right.

To illustrate how  $S_{n-i}$  recovers the values of  $x_{i+1}, \dots, x_n$ , we show  $S_{n-i}$  for  $n = 9, i = 7$  in the figure below.





**Fig. 3.** Unfolding of  $\mathcal{A}_2$  for  $Q_2x_2Q_1x_1 = \forall x_2\forall x_1$  on the left, and on the right, unfolding of  $\mathcal{A}_2$  for  $Q_2x_2Q_1x_1 = \exists x_2\forall x_1$

The hNFA  $\mathcal{A}_i$  and  $S_i$  are designed so that any path of  $\mathcal{A}_i$  is labeled by at most one  $xt_i$  and at most one  $xf_i$  between any two consecutive  $b$ 's, for each  $i$  (for convenience, we suppose that each automaton starts and ends with a fictive  $b$  transition). That is, a path can be labeled by  $xt_i$  and  $xf_i$ , but not by two  $xt_i$  or two  $xf_i$ . By contradiction, assume that there are two consecutive  $b$ 's in  $\mathcal{A}_i$  such that there is a path from one  $b$  to the other one labeled by two  $xt_j$  (the case  $xf_j$  is symmetric). We take the minimal  $i$  which ensures this. By the minimality of  $i$ , this can only happen either because of the first  $xt_i$  transition of  $\mathcal{A}_i$ , or between  $S_{n-i}$  and one of the two  $\mathcal{A}_{i-1}$ . Since in  $S_{n-i}$  all  $xt_j$  occur after the (unique)  $b$ , there is no  $xt_i$  in  $\mathcal{A}_{i-1}$  before its first  $b$  (if any). This already settles the case where  $Q_i = \exists$ . Consider now the case  $Q_i = \forall$ . For the same reason as before, there can be at most one  $xt_j$  between the last  $b$  of  $\mathcal{A}_{i-1}$  and the  $b$  in  $S_{n-i}$ , for all  $j < i$ . Finally, between the  $b$  of  $S_{n-i}$  and the first  $b$  of the second  $\mathcal{A}_{i-1}$  there can be at most one  $xt_j$  with  $j > i$  (from  $S_{n-i}$ ) and at most one  $xt_j$  with  $j < i$  (from  $\mathcal{A}_{i-1}$ ). Thus, in all cases we contradict the assumption on  $\mathcal{A}_i$ .

Using the property we just showed, we can note that between any two consecutive  $b$ 's of any path of  $\mathcal{A}_n$ , there are at most three  $w_j$  and two  $w_{i+m}$  for any  $1 \leq j \leq m, 1 \leq i \leq n$ . Thus our coding in base 4 for determining whether a clause is true, is still applicable. Hence, a path  $\rho$  of  $\mathcal{A}_n$  is labeled by  $W_n$  iff for all  $1 \leq k \leq n + m$  there is exactly one  $w_k$  between any two consecutive  $b$ 's.

Let us show now that  $W_n \in L(\mathcal{A}_n)$  iff there exists a satisfying valuation tree VT for  $\varphi$ . A valuation tree VT is a binary tree of height  $n + 1$  such that its root (level  $n$ ) is labeled by  $x_n$  and all nodes on level  $l$  are labeled by  $x_l$ . The leaves are on level 0, labeled by true or false. A node  $v$  labeled by  $x_l$  corresponds to a valuation  $\sigma(v)$  of the variables  $x_{l+1}, \dots, x_n$ . Moreover, a node on level  $l$  has two children if  $x_l$  is universally quantified (one child evaluates  $x_l$  to true and the other one to false), and one child if  $x_l$  is existentially quantified. We say that a valuation tree satisfies a QBF formula  $\varphi = Q_n x_n \cdots Q_1 x_1 \psi$  if for every leaf, the associated valuation makes  $\psi$  true.

Assume first that VT is a valuation tree showing that  $\varphi$  is true. A valuation  $\sigma(v)$  defines two words  $T(v), F(v)$  as follows: the word  $T(v)$  is the concatenation of all  $xt_j$  where  $j > i$  and  $x_j$  is true in  $\sigma(v)$ . The word  $F(v)$  is the concatenation of all  $xf_j$  where  $j > i$  and  $x_j$  is false in  $\sigma(v)$ . Let  $v$  be a node of VT labeled by  $x_i$ . We define the word  $\rho(v) = T^{-1}(v)W_i F^{-1}(v)$ . We recall that  $T(v), F(v)$  are words over  $a^*$ , hence  $T^{-1}(v)W_i F^{-1}(v)$  is the word that results from  $W_i$  by deleting  $|T(v)|$  many  $a$ 's in the prefix and by deleting  $|F(v)|$  many  $a$ 's in the suffix.

Let us show by induction on level  $i$  that  $\rho(v)$  is in  $L(\mathcal{A}_i)$  for any node  $v$  of VT on level  $i$ .

If  $v$  is a leaf of VT, then it defines an accepting valuation for  $\psi$ , hence  $T(v)F(v) = W_0$  due to the (1-in-3) restriction. Hence  $\rho(v) = W_0 W_0^{-1} = \epsilon \in L(\mathcal{A}_0)$ .

Consider first an internal node  $v$  labeled by  $x_i$  with  $Q_i = \forall$ . Let  $v_1, v_2$  be the children of  $v$ , with  $v_1$  corresponding to  $x_i$  true, and  $v_2$  to  $x_i$  false. By induction let us suppose that  $\rho(v_1), \rho(v_2)$  are in  $L(\mathcal{A}_{i-1})$ . Then,

$$\begin{aligned}
\rho(v) &= T^{-1}(v)W_iF^{-1}(v) = T^{-1}(v)W_{i-1}bW_{i-1}F^{-1}(v) \\
&= T^{-1}(v)T(v_1)\rho(v_1)F(v_1)bT(v_2)\rho(v_2)F(v_2)F^{-1}(v) \\
&= xt_i\rho(v_1)F(v_1)bT(v_2)\rho(v_2)xf_i
\end{aligned}$$

We used in the equations above  $T^{-1}(v)T(v_1) = xt_i$  for the positive child  $v_1$  of  $v$  and  $F^{-1}(v)F(v_2) = xf_i$  for the negative child  $v_2$  of  $v$ . Moreover,  $F(v_1)bT(v_2) = F(v)bT(v) \in L(S_{n-i})$  since  $v_1$  corresponds to  $x_i$  true, and  $v_2$  corresponds to  $x_i$  false. This shows that  $\rho(v) \in L(\mathcal{A}_i)$ .

Consider now an internal node  $v$  that is labeled by  $x_i$  with  $Q_i = \exists$ . Assume by symmetry that  $v_1$  is the child of  $v$  in VT (thus,  $x_i$  is true). By induction we assume that  $\rho(v_1)$  is in  $L(\mathcal{A}_{i-1})$ . It is easy to show now that  $\rho(v) \in L(\mathcal{A}_i)$  using:

$$\begin{aligned}
\rho(v) &= T^{-1}(v)W_iF^{-1}(v) = T^{-1}(v)W_{i-1}F^{-1}(v) \\
&= T^{-1}(v)T(v_1)\rho(v_1)F(v_1)F^{-1}(v) \\
&= xt_i\rho(v_1)
\end{aligned}$$

For the reverse direction the arguments are similar. From a word  $W = W_n$  accepted by  $\mathcal{A} = \mathcal{A}_n$ , we obtain subwords  $\rho(v)$  in  $L(\mathcal{A}_i)$  as above, labeled by  $T^{-1}(v)W_iF^{-1}(v)$ . This means that for each leaf node  $v$ , the valuation  $\sigma(v)$  satisfies exactly one literal per clause. □

Theorem 2 shows immediately that the hierarchical membership problem is PSPACE-hard even with one process, by encoding the alphabet  $\{a, b\}$  by local actions on a single process. Similar arguments can be used for the case where  $G$  is an MSC-graph (with no hierarchy) as shown in the following theorem.

**Theorem 3** *The hierarchical MSC membership problem  $M \stackrel{?}{\in} L(G)$  is PSPACE-complete. The lower bound holds even if  $G$  is an MSC-graph, or if there is only one process.*

*Proof.* The problem we reduce from is again (1-in-3)QBF. Let  $\varphi$  be an instance of (1-in-3)QBF of the form  $\varphi = (Q_n x_n) \dots (Q_1 x_1) \psi$ , where  $Q_i \in \{\exists, \forall\}$  and the formula  $\psi$  is of the form  $\bigwedge_{j=1 \dots m} C(\alpha_{j,1}, \alpha_{j,2}, \alpha_{j,3})$ , with  $\alpha_{j,k}$  literals.

The idea is to let valuations of the variables to correspond to paths of  $G$  and to validate the valuations using the nMSC  $M$ . We define the graph  $G$  and the nMSC  $M$  by induction on  $\varphi = \varphi_n$ . Let  $\varphi_i = (Q_i x_i) \varphi_{i-1}$ , with  $\varphi_0 = \psi$ . Each  $\varphi_i$  will determine  $G_i, M_i$ .

The processes used in the construction are  $SC_1, \dots, SC_m$  and  $RC_1, \dots, RC_m$ , plus  $VY_1, \dots, VN_n$  and  $VN_1, \dots, VN_n$ . Here  $V$  means a variable and  $C$  a clause,  $S$  stands for “send”,  $R$  for “receive”,  $Y$  for “yes” and  $N$  for “no”.

For all  $i$ , let  $MY_i$  be the MSC consisting of a message from  $VY_i$  to  $VN_i$ , then back from  $VN_i$  to  $VY_i$ , and a message from  $SC_j$  to  $RC_j$  for all  $j$  such

that  $x_i \in \{\alpha_{j,1}, \alpha_{j,2}, \alpha_{j,3}\}$ . Symmetrically, let  $MN_i$  be the MSC consisting of a message from  $VN_i$  to  $VY_i$ , then back from  $VY_i$  to  $VN_i$ , and a message from  $SC_j$  to  $RC_j$  for all  $j$  such that  $\neg x_i \in \{\alpha_{j,1}, \alpha_{j,2}, \alpha_{j,3}\}$ .

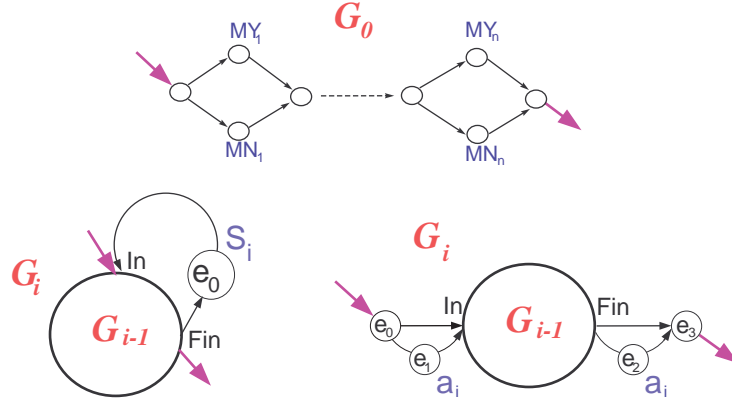
$M_0$  is an MSC consisting of one message from  $SC_j$  to  $RC_j$ , for all  $j$ . The MSC-graph  $G_0$  consists of  $3n + 1$  nodes, labeled by  $MY_i$ ,  $MN_i$ , or  $\emptyset$ . The graph chooses between  $MY_i$  and  $MN_i$  for all  $i$ , as depicted on figure 4.

Note that all messages defined above commute, except for the ones between  $VY_i$  and  $VN_i$ . Let  $a_i$  be the message from  $VY_i$  to  $VN_i$ , and  $b_i$  the message from  $VN_i$  to  $VY_i$ . We use the order between  $a_i$ ,  $b_i$  as follows: The sequence  $MY_i = a_i b_i$  means that  $x_i$  is true, while  $MN_i = b_i a_i$  means that  $x_i$  is false.

Assume now that  $G_{i-1}, M_{i-1}$  are already defined, and that there are  $f$  universal quantifiers in  $\varphi_{i-1}$ . For simplicity, we denote  $a = a_i$  and  $b = b_i$ . Note that in a valuation tree for  $\varphi$  showing that  $\varphi$  is true, each value 0 or 1 assigned to the variable  $x_i$  is used by  $2^f$  leaves. A valuation tree is defined as in Theorem 2.

If  $\varphi_i = \forall x_i \varphi_{i-1}$ , then let  $M_i = (ab)^{2^f} M_{i-1} S_i (ba)^{2^f} M_{i-1}$ . The MSC  $S_i$  is used for synchronizing processes occurring in  $M_i$ . It contains a message between each (ordered) pair of processes of  $M_i$  (in some arbitrary order). Note that using the hierarchy we can describe  $(ab)^{2^f}$ , and thus  $M_i$ , by an expression of polynomial size. Note also that each  $G_i$  is defined as a (flat) MSC-graph.

Let  $G_i = (V_i, E_i)$ , where  $V_i = V_{i-1} \cup \{e_0\}$  and  $E_i = E_{i-1} \cup \{(Fin, e_0), (e_0, In)\}$ . The initial node In (the final node Fin, respectively) of  $G_i$  is the same as for  $G_{i-1}$ . The vertex  $e_0$  is labeled by the synchronization MSC  $S_i$ .



**Fig. 4.** The MSC-graph on top is  $G_0$ . The MSC-graph lower on the left inductively defines  $G_i$  from  $G_{i-1}$  in the universal case. The MSC-graph lower on the right inductively defines  $G_i$  from  $G_{i-1}$  in the existential case.

The definition of  $M_i, G_i$  can be explained intuitively as follows. Let  $\rho$  be a path of  $G_i$  labeled by  $M_i$ . Note that the MSC  $S_i$  occurring in  $M_i$  has to match



the MSC  $S_i$  of  $e_0$ . Thus  $\rho = \rho_1 e_0 \rho_2$ , with  $\rho_1$  an accepting path of  $G_{i-1}$  labeled by  $(ab)^{2^f} M_{i-1}$  and  $\rho_2$  an accepting path of  $G_{i-1}$  labeled by  $(ba)^{2^f} M_{i-1}$ . Each time  $\rho_j$  goes through  $G_0$  (which happens  $2^f$  times),  $\rho_j$  consumes either  $ab$  of  $MY_i$  or  $ba$  of  $MN_i$ . In particular, all  $2^f$  occurrences consumed by  $\rho_1$  are of the form  $ab$ , which ensures that the valuation of  $x_i$  associated with  $\rho_1$  is consistent ( $x_i$  is true). The same holds for the path  $\rho_2$ , where the value of  $x_i$  is ensured to be false.

Suppose now that  $\varphi_i = \exists x_i \varphi_{i-1}$ . Let  $M_i = (ab)^{2^f} (a) M_{i-1}$ , and  $G_i = (V_i, E_i)$ , where  $V_i = V_{i-1} \cup \{e_0, e_1, e_2, e_3\}$ . Let  $E_i = E_{i-1} \cup \{(e_0, \text{In}), (\text{Fin}, e_3), (e_0, e_1), (e_1, \text{In}), (\text{Fin}, e_2), (e_2, e_3)\}$ , where as above In is the initial vertex and Fin is the final vertex of  $G_{i-1}$ . The initial and final vertices of  $G_i$  are  $e_0$  et  $e_3$ . We label  $e_1$  and  $e_2$  by  $a$ , and  $e_0$  et  $e_3$  by the empty MSC.

The underlying idea in this case is that the additional occurrence of  $a$  in  $M_i$  must be matched by  $e_1$  or  $e_2$  (nowhere else there is an  $a$ ). If it is  $e_1$ , every time the path  $\rho$  goes through  $G_0$ , it must choose  $ba$ , hence it goes through  $VN_i$ . The corresponding value for  $x_i$  is then forced to be false. If it is  $e_2$ , then  $\rho$  must choose  $ab$ , hence it goes through  $VY_i$ . The rest of the proof is similar to the proof of Theorem 2. □

However, if there is only one process and hierarchy is not allowed for the graph  $G$  (or the MSC/word  $M$ ), then our lower bound proof does not work anymore. Indeed, we show below that in the case where the word  $W$  or the automaton  $\mathcal{A}$  are flat, the membership problem is solvable in polynomial time.

- Theorem 4** 1. *Deciding for an SLP  $W$  and an NFA  $\mathcal{A}$  whether  $W \in L(\mathcal{A})$  can be done in time  $O(|W| \cdot |\mathcal{A}|^3)$ .*  
 2. *Deciding for a word  $W$  and a hNFA  $\mathcal{A}$  whether  $W \in L(\mathcal{A})$  can be done in time  $O(|W|^3 \cdot |\mathcal{A}|^3)$ .*

For the first statement in the theorem above a similar result (for Lempel-Ziv compressed words and regular expressions) has been shown in [24].

The polynomial time algorithms for Theorem 4 are stated below. The first algorithm computes by dynamic programming the set  $T_X$  of pairs  $(a, b)$  of states of a NFA  $\mathcal{A}$  between which a path labeled by  $X$  exists, for each variable  $X$  of the SLP. A variable  $X$  is said to belong to the lowest level, if the rule associated with  $X$  is terminal.

**Membership**  $((X_i)_{i=1,n}$  SLP,  $A=(V, E, a_0, a_f)$  NFA)

For each variable  $X_i$  on the lowest level:

$$T_{X_i} = \{ (a, b) \in V \times V \mid a \xrightarrow{X_i} b \};$$

For  $i = 1 \dots n$ :

$$\text{Let } T_{X_i} = \emptyset;$$

$$\text{Let } Y, Z \text{ s.t. } X_i \rightarrow Y Z;$$

For all vertices  $a, b, c \in V$ :

$$\text{If } (a, b) \in T_Y \text{ and } (b, c) \in T_Z:$$

$$T_{X_i} = T_{X_i} \cup \{(a, c)\};$$

Return  $(a_0, a_f) \in T_{X_1}$ ;

The second algorithm computes for each sub-automaton  $\mathcal{B}$  of a hNFA  $\mathcal{A}$  the set  $T_{\mathcal{B}}$  of factors of a word  $W$  that it accepts. We denote as  $W[i \dots j]$  the factor of  $W$  from position  $i$  to position  $j$ ,  $i \leq j$ . The algorithm actually computes for each  $i \leq j$  the set  $T_{i,j}$  of pairs  $(a, b)$  of states of  $\mathcal{B}$  between which a  $W[i \dots j]$ -labeled path exists. For convenience, we assume without loss of generality that all transitions (except for the lowest hierarchy level) correspond to sub-automata. We use the fact that  $(a, b) \in T_{i,j}$  if either there is a transition from  $a$  to  $b$  labeled by a sub-automaton  $C$  accepting  $W[i \dots j]$ , or else the path labeled by  $W[i \dots j]$  can be decomposed as  $a, c$  and  $c, b$ , and then there exists  $0 < e < j - i$  such that  $(a, c) \in T_{i, i+e}$  and  $(c, b) \in T_{i+e, j}$ . We thus compute first the lower levels of hierarchy, and we compute then for each sub-automaton the sets  $D_{i, i+d}$ , for increasing  $d$ .

**Membership** ( $W$  word,  $A=(V, E, a_0, a_f)$  hNFA)

For each sub-automaton  $B$  of  $A$  on the lowest level of hierarchy:  
 $T_B = \{(i, j) \mid W[i \dots j] \text{ is accepted by } B\}$ ;  
For each sub-automaton  $B$  of  $A$ , by increasing hierarchical level:  
For  $d = 0, \dots, |W|$ , for  $i = 1, \dots, |W| - d$ ,  
 $D_{i, i+d} = \{ (a, b) \mid a, b \text{ vertices of } B \text{ s.t. } a \xrightarrow{C} b \text{ for some } C \text{ with } (i, i+d) \in T_C \}$ ;  
For each  $e < d$  and every  $a, b, c$  vertices of  $B$ ,  
If  $(a, b) \in D_{i, i+e}$  and  $(b, c) \in D_{i+e+1, i+d}$ :  
 $D_{i, i+d} = D_{i, i+d} \cup \{(a, c)\}$ ;  
 $T_B = \{(i, j) \mid (a_0, a_f) \in D_{i, j}\}$ ;  
Return  $(1, |W|) \in T_A$

The figure below summarizes the complexities of the different variants for the hierarchical MSC membership problem, as considered in this section. The last two columns correspond to the case of a single process (word case) and to the general MSC case, respectively. The fact that the membership problem is NP-complete for an MSC  $M$  and an nHMSC  $G$  is easy to show. The lower bound holds already for MSC-graphs  $G$  [1], and for the upper bound it suffices to guess a path of  $G$  of the size of  $M$ , which is polynomial, and check whether it is labeled by  $M$ .

$M$	$G$	words	MSC
Flat	Nested	P	NP-complete
Nested	Flat	P	PSPACE-complete
Nested	Nested	PSPACE-complete	PSPACE-complete

**Fig. 5.** Complexity of the membership problem.

## 5 Pattern Matching of nMSCs

The aim of this section is to show that the pattern matching problem for nMSCs can be solved in polynomial time, without unfolding the nMSCs. We first consider a special case of pattern matching, namely testing the equality of two nMSCs. Then we describe first a pattern matching algorithm when the pattern nMSC is connected, and second the additional work for non-connected patterns.

### 5.1 Equality of nMSCs

Recall first that the FIFO rule allows to test the equality of two MSCs  $M$  and  $N$  process-wise, which amounts to test the equality of  $\wp$  pairs of words (over the type alphabet  $\mathcal{T}$ ). In the hierarchical case we already used in Section 4.1 the representation of an nMSC  $M$  by  $\wp$  straight-line programs  $L^i$ , where SLP  $L^i$  generates the projection  $M|_i$  of  $M$  on process  $i$ .

In order to test the equality of two nMSCs in polynomial time, we can use directly the following result:

**Theorem 5 ([23])** *Let  $P$  be an SLP, and  $A, B$  be two variables of  $P$ . We can determine whether  $A$  and  $B$  generate the same word in time  $O(|P|^5 \log(|P|))$ .*

The theorem above provides an algorithm for testing  $M = N$  of time  $O((|M| + |N|)^5 \log(|M| + |N|))$ . We can improve the running time by using the pattern matching algorithm described in the next section.

### 5.2 Pattern Matching nMSCs

**Definition 5.** *The pattern matching problem for two MSCs  $M$  and  $N = \langle P, E, \mathcal{C}, \ell, m, \langle \rangle \rangle$  consists in knowing whether there exists some subset  $F \subseteq E$  of events of  $N$  such that the restriction of the mappings  $\ell, m$  to  $F$  equals  $M$ . Moreover, we require that  $F$  is convex, that is if  $e, f \in F$  and  $e < g < f$ , then  $g \in F$ . We call such an event set  $F$  an occurrence of  $M$  in  $N$ .*

*If  $M, N$  are nMSCs, then  $M$  occurs as a pattern in  $N$  if the MSC defined by  $M$  is a pattern in the MSC defined by  $N$ , and we write  $M \subseteq N$  in this case.*

It is easy to see that for an MSC  $M$  to be a pattern of an MSC  $N$  it does not suffice to have each  $M|_i$  a pattern of  $N|_i$ . But of course, this condition is necessary. Before we consider the nested case, we show a simple algorithm for the flat case:

**Theorem 6** *Let  $M, N$  be two MSCs. We can check whether  $M$  is a pattern of  $N$  in linear time.*

*Proof.* The main idea comes from pattern matching in trace monoids, [16]. We use the linear time algorithm of Knuth-Morris-Pratt for determining occurrences of  $M|_i$  in  $N|_i$ , for all  $i \in P$ . We search for tuples of occurrences of  $(M|_i)_{i \in P}$  that form a factor of  $N$ . That is, we look for a configuration of  $N$  such that on each

process  $i$ , we have  $M|_i$  as a suffix. This is done by progressing one event at a time from a configuration  $C$  of  $N$  to the next configuration  $C'$  as follows. For a process  $j$ , let  $\text{next}(C, j)$  be  $C \cup \{e\}$ , where  $e$  is the next event on  $j$  and if  $C \cup \{e\}$  is a configuration (otherwise,  $\text{next}(C, j)$  is undefined).

For the current configuration  $C$  of  $N$  we will record the set  $J$  of processes  $i$  such that  $M|_i$  is a suffix of  $C$  on process  $i$ . From  $C$  we look for a process  $j \notin J$  such that  $\text{next}(C, j)$  holds. If such a  $j$  exists, then we set  $C' = \text{next}(C, j)$  and update  $J$  by possibly adding  $j$ . Otherwise, the next event on every  $j \notin J$  is a receive from some  $i \in J$ , where the corresponding send does not belong to  $C$ . Let  $J_0$  be the set of all such processes  $i$ . Note that the occurrence found on any of the processes from  $J_0$  cannot form an occurrence of  $M$  in  $N$ . So we can progress on any of the processes in  $J_0$  (if possible). We first try to find some  $i \in J_0$  such that  $\text{next}(C, i)$  is defined. If such an  $i$  exists, then we set  $C' = \text{next}(C, i)$  and update  $J$  by possibly removing  $i$ . If not, then we surely find some  $i \in J \setminus J_0$  such that  $\text{next}(C, i)$  is defined (otherwise  $C$  cannot be extended at all, which means that  $N$  is not an MSC). Then we apply the same reasoning to this  $i$ .

The overall complexity of the algorithm is linear, by taking care that each event in  $N$  is considered at most a constant number of times. We need for this to record in addition the set  $X$  of processes  $i$  such that  $\text{next}(C, i)$  is undefined, although there is some next event  $e$  on  $i$ . This is the case where  $e$  is a receive on  $i$ , and the matching send  $f$  does not belong to  $C$ . Together with  $i \in X$  we store the process  $j$  of  $f$ . Altogether we record the four sets  $J \setminus X$ ,  $J \cap X$ ,  $\bar{J} \setminus X$  and  $\bar{J} \cap X$ . Whenever we add an event  $e$  on process  $i$ , we update the membership of  $i$  in one of these sets by looking at the next event  $e$  on  $i$ . Moreover, if  $e$  is a send with matching receive  $f$  on  $j$ , then the membership of  $j$  is also updated. Thus, we can choose the process where we progress in constant time, and every update can be done in constant time, too.  $\square$

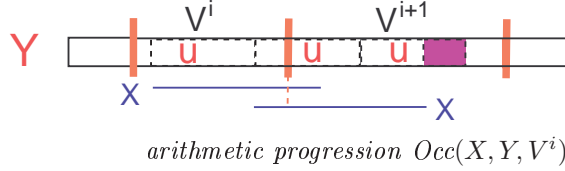
**Definition 6.** Let  $N = (N_i)_{i=1,n}$  be an nMSC (or an SLP), and let  $i, j \leq n$ .

1. We write  $N_i < N_j$  whenever  $N_i$  is used in the definition of  $N_j$  or in the definition of  $Z$  with  $Z < N_j$ . We write  $N_i \leq N_j$  when  $i = j$  or  $N_i < N_j$ . The variable  $N_i$  is then called lower than the variable  $N_j$ .
2. We say that  $N_i$  occurs literally in  $N_j$  when  $N_i$  is used as a reference (variable resp.) in the definition of  $N_j$ , and we write  $N_i \in N_j$  if this is the case.

The strategy we will use for nMSC pattern matching is to compute an implicit representation of all positions where  $M|_i$  occurs as a pattern in  $N|_i$ . In a second step we compute all positions where the projections  $M|_i$  form an MSC factor. The basis of our algorithm is a pattern matching algorithm for SLP-compressed words, that was proposed in [21] (based on ideas from [23])<sup>3</sup>:

**Theorem 7 ([21])** Let  $P$  be an SLP and let  $A, B$  be two variables of  $P$ . An implicit representation of all occurrences of the word defined by  $A$  in the word defined by  $B$  can be computed in time  $O(|A|^2|B|^2)$ .

<sup>3</sup> Very recently, an improved algorithm of complexity  $O(|A||B|^2)$  was described in [14].



The idea of the algorithm in [21] is based on word combinatorics, as we describe next. First, we suppose that the right-hand sides of the rules of the SLPs are either terminal (consisting only of terminal symbols, here types from  $\mathcal{T}$ ), or consist of variables only. For a variable  $Y$  we denote by  $|Y|$  the length of its right-hand side.

Let  $X$  be a variable of the SLP  $A$  and suppose that  $X$  occurs in  $B$ , i.e. the word defined by  $X$  is a factor of the word defined by  $B$ . Suppose that  $X$  does not appear as a factor inside any variable  $Y$  of  $B$  with terminal rule  $Y \rightarrow \alpha$ . Then  $X$  occurs in a variable  $Y$  with  $Y \rightarrow V^1 \dots V^k$ . Let  $i$  be such that  $V^i$  is the first symbol that this occurrence of  $X$  overlaps, and the occurrence ends beyond  $V^i$  (see also the figure above). In particular,  $Y$  is the lowest variable that contains this occurrence of  $X$ . We let  $\text{Occ}(X, Y, V^i)$  denote the set of positions of  $Y$  at which an occurrence of  $X$  starts within  $V^i$  and ends beyond  $V^i$ . Let  $\text{Occ}(X, Y) = \bigcup_{i=1}^k \text{Occ}(X, Y, V^i)$  if the rule for  $Y$  is nonterminal, otherwise it denotes the set of positions of  $Y$  where  $X$  occurs.

Using a combinatorial argument (lemma of Fine and Wilf, [7]), it is shown in [21] that  $\text{Occ}(X, Y, V^i)$  is an arithmetic progression that can be computed by dynamic programming in polynomial time. Therefore,  $\text{Occ}(X, Y)$  consists of at most  $|Y|$  arithmetic progressions, if the rule of  $Y$  is nonterminal (otherwise,  $\text{Occ}(X, Y)$  is of size at most  $|Y|$ ). We represent each set  $\text{Occ}(X, Y, V^i)$  by a triple of numbers  $(n, s, k)$  where  $n$  and  $p + s$  are the positions in  $Y$  of the two first occurrences of  $X$  in  $\text{Occ}(X, Y, V^i)$ , and  $k = \#\text{Occ}(X, Y, V^i)$  is the number of occurrences of  $X$  in  $\text{Occ}(X, Y, V^i)$ . That is, we have  $Y = Y_1 X Y_2$  with  $\|Y_1\| = n + si$ , for all  $0 \leq i < k$ . As an example, consider the words  $Y = aaababababb$  and  $X = ababab$ . The arithmetic progression which corresponds to the occurrences of  $X$  in  $Y$  is  $(2, 2, 3)$  (the first position in a word being 0).

**Remark 2** By the algorithm of [21] we note that the equality of two SLPs  $M, N$  can be checked in time  $O(|M|^2|N|^2)$ , which improves the complexity provided by the algorithm proposed in [23].

Throughout the section we denote occurrences of projections  $M|_i$  using superscripts. That is,  $M|_i^1$  will correspond to a given starting position of  $M|_i$  as pattern of  $N|_i$ . Suppose that for each  $i \in P$ ,  $M^i$  occurs in  $N|_i$  as a factor, and let  $E_i$  be the corresponding set of events (positions). We say that  $(M^i)_{i \in P}$  forms a factor of  $N$  if the set of events  $F = \cup_{i \in P} E_i$  satisfies Definition 5 (factor MSC).

### 5.3 Pattern Matching for Connected Patterns

We turn now to the pattern matching problem for nMSCs  $M, N$  where the pattern  $M$  is connected. That is, we suppose throughout this section that  $M$

cannot be written as  $M_1M_2$ , where  $M_1, M_2$  are nonempty MSCs with no common process.

Following the definitions of the previous section we will denote by  $\text{Occ}(M, Y)$  the set of occurrences  $M^0$  of the nMSC  $M$  in the nMSC  $Y$ , such that  $M^0$  does not occur in any reference  $Z < Y$ . We denote by  $\text{Occ}(M, Y, V) \subseteq \text{Occ}(M, Y)$  those occurrences that start within  $V$  and end beyond  $V$ , where  $V \in Y$  is a reference occurring literally in  $Y$ . This means that 1) all events of  $M^0$  must occur within or after  $V$ , 2) for at least one process  $i$ , the occurrence  $M^0|_i$  starts within  $V$  and ends after  $V$ . Notice that for a process  $i$  as in point 2), we have  $M^0|_i \in \text{Occ}(M|_i, Y, V)$ .

**Definition 7.** Let  $M|_i^1$  and  $M|_j^2$  be occurrences of  $M|_i$  in  $N|_i$ , resp. of  $M|_j$  in  $N|_j$ . We say that  $M|_i^1$  and  $M|_j^2$  are compatible, if the first send (resp. receive) between the processes  $i$  and  $j$  on  $M|_i^1$  matches the first receive (resp. send) on  $M|_j^2$  (if  $i, j$  communicate in  $M$ ). More generally, we call the indices corresponding to  $M|_i^1, M|_j^2$  in a given arithmetic progression compatible.

**Lemma 1.** Let  $(M^0|_i)_{i \in P}$  be occurrences of  $M|_i$  in  $N|_i$ . Then  $(M^0|_i)_{i \in P}$  forms a factor of  $N$  iff  $(M^0|_i)_{i \in P}$  are pairwise compatible.

Our search for compatible occurrences uses the following properties, that are easily shown using the fact that  $M$  is connected:

- Fact 1**
1. Let  $Y$  be a variable of  $N$  and  $h \neq j$  two processes. Then for each  $M^0|_h \in \text{Occ}(M|_h, Y)$  there can be at most one occurrence  $M^0|_j$  in  $Y$  that is compatible with  $M^0|_h$ .
  2. For each occurrence  $M^0$  in  $\text{Occ}(M, Y, V)$  there exists some process  $h$  such that  $M^0|_h \in \text{Occ}(M|_h, Y, V)$ . We call such a process  $h$  a leading process for  $M^0$ . Thus, any pairwise compatible tuple  $(M^0|_k)_{k \neq h} \subseteq Y$  is determined by the occurrence  $M^0|_h$ , because of 1).

*Example 4.* For the nMSC  $P$  in Figure 1 and the pattern  $N$  in Figure 6 we have  $\text{Occ}(N, P) = \emptyset$  and  $\text{Occ}(N, S)$  is a singleton, corresponding to the unique occurrence of  $N$  in  $S$ . The leading processes are 1 and 3, since e.g.  $\text{Occ}(N|_3, S|_3) = \{0\}$ . Note that  $\text{Occ}(N|_2, S|_2) = \emptyset$  and  $\text{Occ}(N|_2, M|_2) = \{0\}$  is the arithmetic progression  $(0, 0, 0)$ .

An **index**  $i = n + js$ ,  $j < k$ , of an arithmetic progression  $(n, s, k)$  in  $Y$  is called *external*, if it is either the first or the last index of the progression, that is either  $i = n$  or  $i = n + (k - 1)s$ . Any nonexternal index is called an *internal index*.

The next proposition provides the main argument that the search for a pairwise compatible tuple of occurrences  $(M|_i)_{i \in P}$  can be done in polynomial time. Intuitively, we show that the occurrences of  $(M|_i)_{i \in P}$  can be located in the same variable  $Y$  of  $N$ , up to polynomially many exceptions. Without this property we would have to consider different variables  $Y^i$  for different processes  $i \in P$ . We recall that for every message  $(e, f)$  in an nMSC  $N = (N_q)_{q=1, n}$  the events  $e$  and  $f$  appear literally in the same macro  $N_q$ .

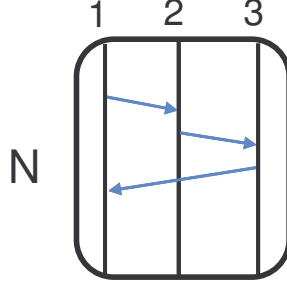


Fig. 6. Pattern MSC  $N$

**Proposition 1** *Assume that  $M^0 \in \text{Occ}(M, Y, V)$  and that  $M^0|_i \in \text{Occ}(M|_i, Y^i, V^i)$  for  $i \in P$ , where  $Y, Y^i, V^i$  are variables of  $N$ . Then we have one of the following two cases:*

1.  $Y^i = Y$  and  $V^i = V$  for all  $i \in P$ .
2. For some leading process  $h$  for  $M^0$  (in particular,  $V^h = V$  and  $Y^h = Y$ ), the occurrence  $M^0|_h$  is an external index of  $\text{Occ}(M|_h, Y^h, V^h)$ .

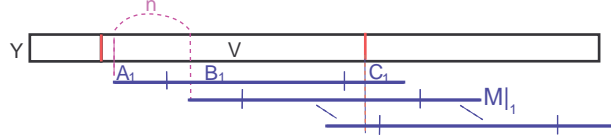
*Proof.* Suppose that for every leading process  $h$ , the occurrence  $M^0|_h$  is an internal index of  $\text{Occ}(M|_h, Y^h, V^h)$ . We want to infer that  $Y^i = Y$  and  $V^i = V$  for all  $i \in P$ . Assume also that there is a message from process  $i$  to process  $j$  in  $M$ . We decompose  $M|_i = A_{i,j}B_{i,j}^sC_{i,j}$  such that the word  $B_{i,j}^s$  begins with the first send from  $i$  to  $j$ , and ends with the last one. Similarly, we decompose  $M|_j = A_{j,i}B_{j,i}^rC_{j,i}$  such that the word  $B_{j,i}^r$  begins with the first receive on  $j$  from  $i$ , and ends with the last one. We need the next lemma to infer that if an occurrence  $M^0$  is such that  $M^0|_i \in \text{Occ}(M|_i, Y^i, V^i)$  and  $M^0|_j \in \text{Occ}(M|_j, Y^j, V^j)$  are both internal indices, then we have  $Y^i = Y^j$  and  $V^i = V^j$ . This will allow finishing the proof of the proposition, using the fact that  $M$  is connected.

**Lemma 2.** *Assume that the arithmetic progression  $\pi = \text{Occ}(M|_i, Y, V)$  consists of at least three indices. Then each occurrence of  $B_{i,j}^s$  that corresponds to some internal index of  $\pi$ , belongs to  $\text{Occ}(B_{i,j}^s, Y, V)$ .*

**Proof of lemma:** Since  $M|_i$  belongs to an arithmetic progression consisting of at least three indices,  $M|_i$  is of the form  $(a_1 \cdots a_n)^d(a_1 \cdots a_m)$ , where  $d \geq 3$  and  $m < n$ .

By assumption, there is a message from  $i$  to  $j$  in  $M|_i$ , hence  $a_k = i!j$  for some  $k$ . Since  $A_{i,j}$  and  $C_{i,j}$  have no  $i!j$ , we obtain  $A_{i,j} = a_1 \cdots a_{k-1}$  and  $C_{i,j} = a_{l+1} \cdots a_n a_1 \cdots a_m$ , with  $l > m$ .

In particular, we have  $|A_{i,j}| < n$  and  $|C_{i,j}| < n$ . Since each  $M|_i$  contains the last position of the word generated by  $V$ , the subword  $B_{i,j}^s$  also contains this position, except possibly for the first and the last  $B_{i,j}^s$ . Hence, every  $B_{i,j}^s$  associated with an internal index of  $\pi$  is in  $\text{Occ}(B_{i,j}^s, Y, V)$ .



□

Let now  $h$  be a leading process, thus  $Y^h = Y$  and  $V^h = V$ . Let also  $j \neq h$  such that  $j, h$  communicate in  $M$ . Since  $M^0|_h$  is an internal index of  $\text{Occ}(M|_h, Y, V)$  we can apply Lemma 2 and we obtain for the corresponding occurrence  $B_{h,j}^{s,0} \in \text{Occ}(B_{h,j}^s, Y, V)$ . Hence, we also have  $B_{j,h}^{r,0} \in \text{Occ}(B_{j,h}^r, Y, V)$ , since matching sends and receives always appear literally in the same variable. Recall that  $M^0|_j \in \text{Occ}(M|_j, Y^j, V^j)$  with  $Y^j \leq Y$ . Using  $B_{j,h}^{r,0} \in \text{Occ}(B_{j,h}^r, Y, V)$  we obtain that  $Y \leq Y^j$ , hence  $Y^j = Y$ . Applying the lemma again to  $M^0|_j$  we obtain also  $V^j = V$ , that is,  $j$  is a leading process, too. The result follows for all processes  $i$ , due to  $M$  being connected. □

**Theorem 8** *Let  $M, N$  be two nMSCs, with  $M$  connected. We can check whether  $M$  occurs in  $N$  in time  $O(|M|^2|N|^2)$ .*

The algorithm below returns occurrences of  $M$  in  $N$ , in form of pairs  $(Y, \pi)$ , where  $Y \leq N$  and  $\pi$  is an arithmetic progression designating a set of positions within  $Y$  that correspond to occurrences of  $M$ . We denote below the number of processes by  $p$ .

#### Pattern-Matching (nMSC $M, N$ )

For each variable  $X$  on the lowest level of hierarchy:  
  If  $M \subseteq X$  at position  $pos$  then return  $(X, pos)$ ;  
For all variables  $Y, V$  of  $N$  with  $V \in Y$ :  
  Compute  $\text{Occ}(M|_1, Y, V), \dots, \text{Occ}(M|_p, Y, V)$ ;  
For every variable  $Y$  of  $N$ :  
  For every process  $h$ :  
    For every  $pos(h)$  at the beginning or end of an  
      arithmetic progression of  $\text{Occ}(M|_h, Y)$ :  
      Let  $(M|_h)^{pos(h)}$  be the corresponding occurrence of  $M|_h$ :  
      If there exist  $((M|_k)^{pos(k)})_{k \neq h}$  compatible with  $(M|_h)^{pos(h)}$   
      where for all  $k$ ,  $pos(k) \in \text{Occ}(M|_k, Z^k)$  with  $Z^k \leq Y$ :  
      return  $(Y, (pos(k))_{k \in P})$ ;  
  For every  $V \in Y$  s.t. for all  $i$ :  $\pi_i = \text{Occ}(M|_i, Y, V) \neq \emptyset$ :  
    For each  $i$ , let  $\pi_i = (n_i, s_i, k_i)$ ;  
    Let  $(t_1, \dots, t_p, e_1, \dots, e_p) = \text{Periods}(\text{Reduce}(\pi_1, \dots, \pi_p))$ ;  
    Let  $\pi'_i = (n_i + t_i s_i, s_i e_i, (k_i - t_i)/e_i)$   
    If  $(\pi'_i)_i \neq \emptyset$  then return  $(Y, (\pi'_i)_i)$

Notice that we have to restrict  $pos(k)$  to be inside  $Y$  for every  $k$  to ensure that  $h$  is leading, which ensures the uniqueness of  $pos(k)$  for every  $k$ . For simplifying the presentation of the algorithm we will assume below that every process  $i$  in



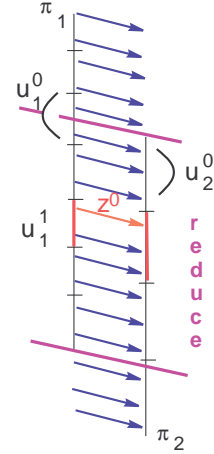
$M$  sends at least one message to every process  $j > i$ . This is just a technical assumption, which makes the presentation nicer. The algorithm first computes the occurrences  $M|_i$  process-wise. Then, in the third for-loop, it first considers external indices, corresponding to the second case of Proposition 1. If no pattern is found, the algorithm looks for an occurrence corresponding to the first case of Proposition 1, where  $M^0|_i \in \text{Occ}(M|_i, Y, V)$  for every process  $i$ . The arithmetic progression  $\text{Occ}(M|_i, Y, V)$  is denoted by  $\pi_i = (n_i, s_i, k_i)$  above. We denote by  $u_i$  the word consisting of the  $s_i$  first symbols of  $M|_i$ . By assumption, each  $u_i$  contains both symbols  $i!j$  and  $i?j$ , for all  $j > i$ . For each  $i < j$  we denote by  $m_{i,j}$  the number of sends from  $i$  to  $j$  in  $u_i$ , and by  $m_{j,i}$  the number of receives from  $i$  to  $j$  in  $u_j$ .

We describe now the subroutines **Reduce** and **Periods** and show that our algorithm returns only occurrences of  $M$  which are indeed factors of  $N$ . The subroutine **Reduce** restricts the arithmetic progressions  $(\pi_1, \dots, \pi_p)$  by adding an offset to each  $n_i$  of an arithmetic progression  $\pi_i = (n_i, s_i, k_i)$ , and reducing  $k_i$ . This is done such that for all pairs of distinct processes  $i, j$  there exists a send to process  $j$  and a receive from  $j$  in every occurrence from  $\pi_i$ , such that the matching event belongs to  $\pi_j$ . For instance, in the example below the arithmetic progression  $\pi_1$  will start after a call of **Reduce** with  $u_1^0$ , since the two copies of  $u_1$  before have no send to process 2 such that the matching receive belongs to  $\pi_2$ . Thus, the first two occurrences of  $u_1$  in  $\pi_1$  will not be used for looking for compatible occurrences. It also reduces the number of occurrences of arithmetic progressions. **Reduce** takes quadratic time by computing for every pair of processes  $i, j$  the first and the last event on  $i$  that sends or receives a message from an occurrence from  $\pi_j$ . We then compute the events which fulfill every constraint.

Let  $(\pi_i)_{i=1,p}$  be arithmetic progressions of occurrences of  $M|_1, \dots, M|_p$ , such that for each pair  $i < j$  there is a message from each  $u_i$  in  $\pi_i$  to some  $u_j$  in  $\pi_j$ . That is,  $(\pi_i)_{i \in P}$  is the result of a call of **Reduce**. Let  $u_i^0$  be the first index of each arithmetic progression  $\pi_i$ . The only problem that remains for deciding whether there exist compatible occurrences  $M|_i, M|_j$  is that the existence of messages from  $u_i$  in  $\pi_i$  to  $u_j$  in  $\pi_j$  does not mean that the events match correctly w.r.t.  $M$ . We will look for tuples of occurrences of the  $M|_i$  that are pairwise compatible by considering sub-progressions of the  $\pi_i$ .

From now on we want to determine all tuples  $(u_i)_{i \in P}$  corresponding to the starting positions of pairwise compatible tuples  $((M|_1)^0, \dots, (M|_p)^0)$ . As we show later, such tuples occur periodically, hence we just need to determine some periods  $(\mu_1, \dots, \mu_p) \in \mathbb{N}^p$  and the first positions  $(u_1^1, \dots, u_p^1)$  from which we can apply these periods.

For all  $i < j$  let  $z_{i,j} < m_{i,j}$  be the number of events of type  $i!j$  in  $u_i^0$  before the first event that has a matching event in  $\pi_j$ . Let also  $z_{j,i} < m_{j,i}$  be the number of events of type  $j?i$  in  $u_j^0$  before the first one that has a matching



send in  $\pi_i$ . In the figure above we have  $i = 1, j = 2, m_{1,2} = 2$  (there are two sends in each  $u_1$ ),  $m_{2,1} = 3$  (there are three receives in each  $u_2$ ),  $z_{1,2} = 1$  (the first send of  $u_1^0$  has no matching receive in  $\pi_2$ ) and  $z_{2,1} = 0$ . Let  $z_{i,j}^0$  be such that after reading the first  $z_{i,j}^0 + z_{i,j}$  sends from  $\pi_i$  to  $\pi_j$  we arrive at a message consisting of the first  $i!j$  of some  $u_i$  and the first  $j?i$  of some  $u_j$ . In the example, we marked as  $z^0 = z_{1,2}^0$  the earliest message consisting of the first  $1!2$  of some  $u_1$  and the first  $2?1$  of some  $u_2$ , and  $z_{1,2}^0 = 3$ . So  $z_{i,j}^0 + z_{i,j} \equiv 0 \pmod{m_{i,j}}$  and  $z_{i,j}^0 + z_{j,i} \equiv 0 \pmod{m_{j,i}}$ . Using the Chinese Remainder Theorem the subroutine `Periods` first computes the least solutions  $z_{i,j}^0$  modulo  $\text{lcm}(m_{i,j}, m_{j,i})$  to the above equations in time  $O(\min(|M|_i, |M|_j)^3)$ . We perform this computation for each pair of processes in overall time  $O(|M|^3)$  for obtaining the new period  $\mu_i$  and the new offset  $u_i^1$ . Notice that  $\mu_i = \text{lcm}\{m_{i,j} \mid i < j\}$ . The restriction of the arithmetic progression  $\pi_i$  according to  $\mu_i, u_i^1$  is denoted  $\pi_i'$ .

By definition, the first  $i!j$  of each  $u_i$  in the restricted arithmetic progression  $\pi_i'$  matches the first  $j?i$  of some  $u_j$  of the unrestricted arithmetic progression  $\pi_j$ . The final step of `Periods` is to compute occurrences of  $M$  from  $(\pi_i')_{i=1,p}$ . Let  $x_{i,j}$  denote the number of  $u_j$  between the occurrence  $u_j^1$  and the occurrence containing the receive of the first message from  $u_i^1$ . We want to compute all tuples  $(u_i)_{i=1,p}$  such that the first  $i!j$  of  $u_i$  matches the first  $j?i$  of  $u_j$ . That is, we need a solution  $(t_i)_{i=1,p}$  of the following system of  $p(p-1)$  linear equations:

$$\mu_i m_{i,j} t_i = x_{i,j} m_{j,i} + \mu_j m_{j,i} t_j.$$

Thus, the value of  $t_1$  determines each  $t_i$ , modulo some value  $e_i$  depending on the constants  $(m_{i,j})_{i,j}$ . We can combine the equation for  $(1, i)$  with the equation for  $(i, j)$  to obtain a system of  $p(p-1)$  equations:

$$\delta_{i,j} t_1 = y_{i,j} + \nu_{i,j} t_j.$$

Let  $j \in P$ . Notice that several of these equations (for different  $i$ ) involve the same pair of variables  $t_1$  and  $t_j$ . Either all these equations are equivalent, or they yield a unique solution, or no solution at all. If there is a unique solution, then we stop the procedure and test this solution in each equation. If this is indeed a solution of the system, we return its value. If there is no solution, we do not find an occurrence of  $M$  at this level. Hence, we can assume for the following that the equations for  $j$  are all equivalent. Then it suffices to consider a system of  $p$  equations of the above form (i.e., we fix some  $i$  for each  $j$ ).

If  $\text{gcd}(\delta_{i,j}, \nu_{i,j})$  does not divide  $y_{i,j}$ , there is no solution to our system. Else, we can divide  $\delta_{i,j}, y_{i,j}, \nu_{i,j}$  by  $\text{gcd}(\delta_{i,j}, \nu_{i,j})$ , and thus consider only the case where  $\text{gcd}(\delta_{i,j}, \nu_{i,j}) = 1$ .

Let  $\gamma_{i,j}$  be the inverse of  $\delta_{i,j}$  modulo  $\nu_{i,j}$ . Hence the equations are reduced to  $p$  simple equations of the form  $t_1 \equiv y_{i,j} \gamma_{i,j} \pmod{\nu_{i,j}}$ . The subroutine `Periods` finally computes a solution  $(t_1, \dots, t_p)$  using again the Chinese Remainder Theorem and returns  $(t_i + u_i^1 - u_i^0, e_i)_i$ .

Since the intersection of an arithmetic progression with the periodic set is still an arithmetic progression, in the end we have arithmetic progressions of

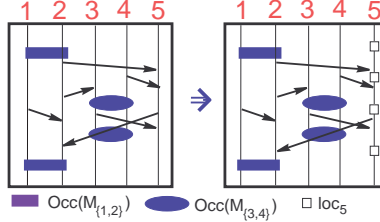
periods increased by a factor of  $e_i$ , that contains only compatible occurrences. A call of `Periods` costs time  $O(|M|^3)$ .

**Remark 3** We can slightly adapt the algorithm for computing all occurrences of  $M$  in  $N$ . Note that the number of occurrences might be exponential (as in the word case), thus the representation of all occurrences will be implicit.

#### 5.4 Pattern Matching for Non-Connected Patterns

We turn now to the general case where the nMSC pattern  $M$  is not connected. We show that the complexity of the algorithm increases just by a factor  $O(|\mathcal{C}_M|^2) \leq O(\wp^2)$ , namely the square of the number of weakly connected components of  $M$ .

It will be helpful in the following to have all processes of  $N$  appear in  $M$ . This can be enforced by a simple modification of  $M, N$ , as depicted below. For each reference  $Y$  of  $N$  and each process  $i \in P_N \setminus P_M$  we add a local action  $\text{loc}_i$  on process  $i$  in  $Y$  before each message or reference on  $i$ , and before the end of  $Y$ . Let  $M' = M \cdot \prod_{i \in P_N \setminus P_M} \text{loc}_i$ . Obviously,  $M'$  occurs in  $N'$  iff  $M$  occurs in  $N$ .



Let  $M, N$  be nMSCs. For each reference  $X$  of  $M$  or  $N$ , let  $\mathcal{C}_X \subseteq 2^{P_M}$  be the set of maximal connected components of the communication graph of  $X$  (this is the graph with vertices corresponding to processes and edges between communicating processes). We will denote by  $X|_C$  the projection of  $X$  over the processes in  $C \in \mathcal{C}_X$ . In other words,  $X = (X|_C)_{C \in \mathcal{C}_X}$  represents the decomposition of the MSC associated with  $X$  into connected nMSCs. It follows from the previous section that we can compute in time  $O(|M|^2|N|^2)$  a compact representation of all occurrences of  $M|_C$  in  $N$ , for each  $C \in \mathcal{C}_M$ . The next definition states when a tuple of occurrences  $(M|_C)_{C \in \mathcal{C}_M}$  of the connected components of  $M$  corresponds to an occurrence of  $M$  in  $N$ .

**Definition 8.** Let  $a \in \text{Occ}(M|_C, Y), b \in \text{Occ}(M|_D, Y)$  be two occurrences of connected components of  $M$ , where  $C, D \in \mathcal{C}_M$  and  $C \neq D$ . Then  $a, b$  are called compatible if there is no message in  $Y$  from some process in  $C$  to some process in  $D$  that is sent after  $a$  and received before  $b$  (or vice versa).

**Lemma 3.** Let  $a_C \in \text{Occ}(M|_C, Y)$ , for all  $C \in \mathcal{C}_M$ . Then  $(a_C)_{C \in \mathcal{C}_M}$  is an occurrence of  $M$  in  $Y$  iff  $a_C, a_D$  are compatible for all  $C, D \in \mathcal{C}_M, C \neq D$ .

*Proof.* The implication from left to right follows directly from the definition of patterns. For the converse assume that  $(a_C)_{C \in \mathcal{C}_M}$  is not an occurrence of  $M$  in  $Y$ . This means that there is some chain of messages  $(s_k, r_k)_{k=1}^m$  with  $P(s_1) \in C$ ,  $P(r_m) \in D$ ,  $P(r_k) = P(s_{k+1})$  for all  $k$ , and such that  $a_C$  precedes  $s_1$ ,  $r_i$  precedes  $s_{i+1}$ , and  $r_m$  precedes  $a_D$ . Since all processes appear in  $M$ , there exist some  $k$  and  $C', D' \in \mathcal{C}_M$  such that  $P(s_k) \in C'$ ,  $P(r_k) \in D'$ ,  $a_{C'}$  precedes  $s_k$  and  $r_k$  precedes  $a_{D'}$ . But this means that  $a_{C'}, a_{D'}$  are not compatible, contradiction.  $\square$

Let  $C \in \mathcal{C}_M$ . Note that the occurrences of the connected components  $M|_C$  in  $Y$  are totally ordered by the visual order of  $Y$ . This justifies the use of min and max on occurrences of the same connected component in the proposition below.

**Proposition 2** *Let  $a = (a_C)_{C \in \mathcal{C}_M}, b = (b_C)_{C \in \mathcal{C}_M} \in (\text{Occ}(M|_C, Y))_{C \in \mathcal{C}_M}$  be two occurrences of  $M$  in  $Y$ . Then  $(\min(a_C, b_C))_{C \in \mathcal{C}_M}$  and  $(\max(a_C, b_C))_{C \in \mathcal{C}_M}$  are also occurrences of  $M$  in  $Y$ .*

*Proof.* By Lemma 3 it suffices to check that  $\min(a_C, b_C), \min(a_D, b_D)$  are compatible, for all  $C, D \in \mathcal{C}_M, C \neq D$ . The only case to verify is when  $\min(a_C, b_C) = a_C < b_C$  and  $\min(a_D, b_D) = b_D < a_D$ . Assume by contradiction that there is a message from  $C$  to  $D$  that is sent after  $a_C$  and received before  $b_D$ . Then  $a_C$  and  $a_D > b_D$  are not compatible, a contradiction. The case where a message is sent after  $b_D$  and received before  $a_C$  is symmetrical.  $\square$

We describe the pattern matching algorithm in a simpler case where the following two conditions hold. First, we assume that every message is on the lowest hierarchical level. This means that macros either consist of references (and local actions) only, or they are MSCs. In other words, we forbid messages crossing references in  $N$ . Second, for all references  $Y, Z$  with  $Z \in Y$  and each occurrence of  $M|_C$  in  $Y$  either  $M|_C$  is included in  $Z$ , or it has an empty intersection with  $Z$ . That is, we assume that no occurrence of  $M|_C$  in  $Y$  is split between several references  $Z \in Y$ . If  $N$  satisfies these conditions w.r.t.  $M$ , then we call the pair  $(M, N)$  **nice**. The general case is technically more involved, but it does not require new ideas.

If  $M$  occurs as a pattern of  $N$ , then Proposition 2 ensures that there is a unique minimal occurrence of  $M$  in  $N$  (minimal with respect to the component wise ordering of tuples from  $(\text{Occ}(M|_C, N))_{C \in \mathcal{C}_M}$ ). In order to find the minimal occurrence of  $M$  in a reference  $X$  of  $N$ , we look for compatible minimal occurrences in each reference  $Y \in X$ . If  $Y$  does not contain the complete  $M$ , then we need more information about possible components  $M|_C$  that are outside  $Y$  and that are compatible with the components within  $Y$ . Since there may be several references  $X$  with  $Y \in X$  we encode this additional information by imaginary occurrences denoted  $\downarrow_C$  and  $\uparrow_C$ , for each component  $C \in \mathcal{C}_M$ . The occurrence  $\downarrow_C$  for component  $C$  means an occurrence of  $M|_C$  after  $Y$ , while  $\uparrow_C$  for  $C$  means an occurrence of  $M|_C$  before  $Y$ . Thus, we let  $\uparrow_C < a_C < \downarrow_C$  for all  $a_C \in \text{Occ}(M|_C, Y)$ . For  $C \neq D$ , we say that  $\uparrow_C, a_D \in \text{Occ}(M|_C, Y)$  are compatible if there is no message from  $C$  to  $D$  that is received before  $a_D$  in  $Y$  (symmetrically for  $\downarrow$ ). The precise definition follows:

**Definition 9.** Let  $Y$  be a reference of  $N$ . Let  $E \subseteq \{\neq\uparrow_C, =\downarrow_C \mid C \in \mathcal{C}_M\}$  be a set of constraints. We define  $\text{Min}_E^Y = (a_C)_{C \in \mathcal{C}_M}$  as the minimal tuple satisfying the following conditions:

1. For each  $C \in \mathcal{C}_M$ ,  $a_C \in \text{Occ}(M|_C, Y) \cup \{\uparrow_C, \downarrow_C\}$ .
2. The occurrences  $(a_C)_{C \in \mathcal{C}_M}$  are pairwise compatible.
3.  $(a_C)_{C \in \mathcal{C}_M}$  satisfies the constraint  $E$ . That is,  $(\neq\uparrow_D) \in E$  implies that  $a_D \neq \uparrow_D$ , and  $(=\downarrow_D) \in E$  implies that  $a_D = \downarrow_D$ .

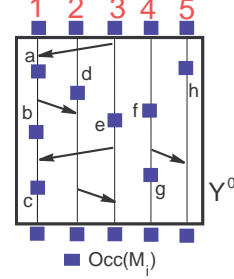
Note that the minimal occurrence in the previous definition is well defined, since there exists at least one tuple  $(a_C)_{C \in \mathcal{C}_M}$  satisfying the three conditions above, namely  $a_C = \downarrow_C$  for all  $C$ . In other words, there may always be an occurrence of  $M$  after  $Y$ .

*Example 5.* The two extreme constraints correspond to

$$\text{Min}_\emptyset = (\uparrow_C)_{C \in \mathcal{C}_M} \text{ and } \text{Min}_{(=\downarrow_C)_{C \in \mathcal{C}_M}} = (\downarrow_C)_{C \in \mathcal{C}_M}.$$

In the figure to the right we also have:

- $\text{Min}_{\{\neq\uparrow_1\}} = (a, \uparrow_2, e, \uparrow_4, \uparrow_5) = \text{Min}_{\{\neq\uparrow_1, \neq\uparrow_3\}}$ .
- $\text{Min}_{\{=\downarrow_2\}} = (b, \downarrow_2, e, \uparrow_4, \uparrow_5)$ .
- $\text{Min}_{\{\neq\uparrow_4, =\downarrow_5\}} = (\uparrow_1, \uparrow_2, \uparrow_3, g, \downarrow_5)$ .



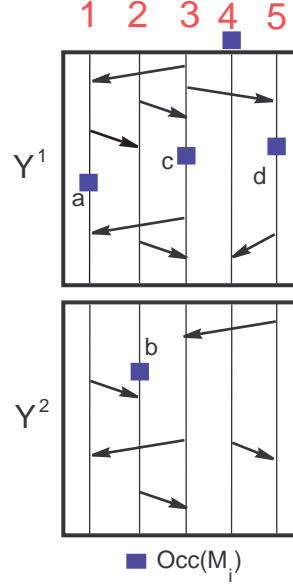
The next lemma shows that it suffices to compute (recursively) the tuples  $\text{Min}_E^Y$ , for suitable constraints  $E$  and references  $Y$  of  $N$ .

**Lemma 4.** Let  $(b_C)_{C \in \mathcal{C}_M} = \text{Min}_{(\neq\uparrow_C)_{C \in \mathcal{C}_M}}^N$ . Then  $M$  is a pattern of  $N$  iff  $b_C \neq \downarrow_C$ , for all  $C \in \mathcal{C}_M$ .

The problem is that we might need the tuples  $\text{Min}_E^Y$  for arbitrary sets  $E$  of constraints (and there are exponentially many). Fortunately, we can avoid the exponential blow-up by computing  $\text{Min}_E^Y$  only for singletons  $E = \{\neq\uparrow_C\}$  and  $E = \{\downarrow_C\}$ ,  $C \in \mathcal{C}_M$ . We first show that these tuples suffice for computing in polynomial time  $\text{Min}_E^Y$  for arbitrary  $E$ . In a second step, we show that we will need only a polynomial number of constraints  $E$  in the recursive step.

**Lemma 5.** Let  $E, F \subseteq \{\neq\uparrow_C, =\downarrow_C \mid C \in \mathcal{C}_M\}$  be two sets of constraints. Then  $\text{Min}_{E \cup F}^Y = \max(\text{Min}_E^Y, \text{Min}_F^Y)$ .

*Proof.* Let  $b = (b_C)_C = \max(\text{Min}_E^Y, \text{Min}_F^Y)$ . We have of course  $\text{Min}_{E \cup F}^Y \geq \text{Min}_E^Y$  and  $\text{Min}_{E \cup F}^Y \geq \text{Min}_F^Y$ , hence  $\text{Min}_{E \cup F}^Y \geq b$ . But  $\text{Min}_{E \cup F}^Y$  is the minimal tuple that satisfies the three properties which  $b$  satisfies, too: the tuple  $b$  has pairwise compatible components  $b_C$  and it satisfies the constraints in  $E \cup F$ . Therefore,  $b = \text{Min}_{E \cup F}^Y$ .  $\square$



$$\begin{aligned} \text{Min}_{\{\neq \uparrow_5\}}^{Y^1 Y^2} &= (a, b, c, \uparrow_4, d). \\ \text{Min}_{\{\neq \uparrow_5\}}^{Y^1} &= (a, \downarrow_2, c, \uparrow_4, d) = \text{Min}_{\{=\downarrow_2, \neq \uparrow_5\}}^{Y^1}. \\ \text{Min}_{\{\neq \uparrow_2\}}^{Y^2} &= (\uparrow_1, b, \uparrow_3, \uparrow_4, \uparrow_5). \end{aligned}$$

**Proposition 3** Assume that the pair  $(M, N)$  is nice and consider some reference  $Y$  of  $N$  and a component  $D \in \mathcal{C}_M$ . Then  $\text{Min}_{\{\neq \uparrow_D\}}^Y$  and  $\text{Min}_{\{=\downarrow_D\}}^Y$  can be computed in time  $O(|Y|\wp^2)$  from the tuples  $(\text{Min}_{\{\neq \uparrow_C\}}^Z)_{C \in \mathcal{C}_M}$  and  $(\text{Min}_{\{=\downarrow_C\}}^Z)_{C \in \mathcal{C}_M}$ , where  $Z \in Y$ .

*Proof.* We can assume without restriction that any reference  $Y$  of  $N$  that is not on the lowest hierarchy level has exactly two subreferences, that is  $Y = Y^1 Y^2$ .

We will compute the set of components  $E_\downarrow \subseteq \mathcal{C}_M$  that consists of all  $C$  such that  $M|_C$  has no occurrence in  $Y^1$  which is compatible with the constraints, thus  $M|_C$  must occur either in  $Y^2$  or after  $Y$ . In order to do this, we start with  $E_\downarrow = \emptyset$  and we augment  $E_\downarrow$  as long as there exist  $a, b$  with the following properties:

- $(a_C)_C$  is an occurrence in  $Y^1$  with  $a_C = \downarrow_C$  iff  $C \in E_\downarrow$ ,
- $(b_C)_C$  is an occurrence in  $Y^2$  with  $b_C = \uparrow_C$  iff  $C \notin E_\downarrow$ .

The algorithm for computing  $\text{Min}_{\{\neq \uparrow_D\}}^Y$  is described below (for  $\text{Min}_{\{=\downarrow_D\}}^Y$  the reasoning is similar):

- (1) Let  $E_\downarrow = \emptyset$
- (2) Compute  $(a_C)_C = \text{Min}_E^{Y^1}$ , with  $E = \{\neq \uparrow_D\} \cup \{=\downarrow_C \mid C \in E_\downarrow\}$
- (3) Let  $E_\downarrow = \{C \mid a_C = \downarrow_C\}$   
 // For all  $C \in E_\downarrow$ ,  $M|_C$  must be in  $Y^2$  or after  $Y$ .

- (4) Compute  $(b_C)_C = \text{Min}_{(\neq \uparrow C)_{C \in E_\downarrow}}^{Y^2}$
- (5) Let  $E_\downarrow = \{ C \mid b_C \neq \uparrow C \}$ . If  $E_\downarrow$  changes, then goto (2).
- (6) Let  $d_C = b_C$  if  $C \in E_\downarrow$ , and  $d_C = a_C$ , otherwise.
- (7) Return  $(d_C)_C$ .

Note that each time the set  $E_\downarrow$  changes at step (3), it increases by at least one component. Hence, we return to step (2) at most  $O(\wp)$  times.

For the running time let us denote by  $E_\downarrow^t$  the value of  $E_\downarrow$  after  $t$  iterations. The  $t$ -th iteration needs time  $\wp(|E_\downarrow^t| - |E_\downarrow^{t-1}|)$ , thus the overall running time is at most  $O(\wp^2)$ .

If an nMSC has more than two references, then we define several sets  $E_\downarrow^i$  to explain the minimal reference  $Y^i$  where the occurrence of the projection should be. Considering that for each step, one set  $E_\downarrow^i$  has to change, the running time is  $\wp^2|Y|$ .  $\square$

**Theorem 9** *We can test whether  $M$  occurs as pattern of  $N$  in time  $O(\mathcal{C}_M^2(|M|^2|N|^2))$ .*

*Proof.* We show the theorem only for the case where  $(M, N)$  is a nice pair. The general case is technically more involved, but does not require new ideas.

Theorem 8 is used for computing first the implicit representation of all occurrences of  $M|_C$  in  $Y$ , for all components  $C \in \mathcal{C}_M$  of  $M$  and all references  $Y$  of  $N$ . For each  $Y$  we need then only the position of the minimal occurrence of each  $M|_C$  in  $Y$  (if any). We compute then  $\text{Min}_{\neq \uparrow C}^Y$  and  $\text{Min}_{= \downarrow C}^Y$  for all components  $C \in \mathcal{C}_M$  and references  $Y$  of  $N$ . We apply Proposition 3 to compute  $\text{Min}_{\neq \uparrow C}^Y$  and  $\text{Min}_{= \downarrow C}^Y$ . The time costs are  $O(|M|^2|N|^2)$  for the connected components and  $O(\wp^3|N|) \leq O(|M|^2|N|^2)$  for the additional algorithms looking for compatible components. The overall running time is thus  $O(|M|^2|N|^2)$ . In the general case we get an additional factor  $\mathcal{C}_M^2$ , where  $\mathcal{C}_M$  is the number of connected components of  $M$ , expressing additional constraints due to components  $M|_C$  that might be split over several references of  $N$ .  $\square$

## 6 Conclusion

In developing new techniques for algorithms on hierarchical MSCs, we provided arguments that algorithms can benefit from the hierarchical structure. We showed that pattern matching and membership algorithms can efficiently use the hierarchy, together with techniques stemming from combinatorics, arithmetics and dynamic programming. We believe that similar techniques can be useful for other problems on hierarchical MSCs, for instance verification of properties expressed by template MSCs [10].

*Acknowledgment:* We wish to thank Marc Zeitoun and Markus Lohrey for insightful comments on previous versions of this paper. A special thank is due to the referees of *TOCS*, who did an enormous work in reviewing all details of our paper and proposing very many improvements and corrections, even in the most technical parts.

## References

1. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *Proc. of ICALP'01*, LNCS 2076, pp. 797–808, 2001. Journal version in *Theor. Comput. Sci.*, 331(1): 97–114 (2005).
2. R. Alur, G. H. Holzmann, and D. A. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.
3. R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *Proc. of ICALP'99*, LNCS 1644, pp. 169–178, 1999.
4. R. Alur and M. Yannakakis. Model checking of hierarchical state machines. In *Proc. of SIGSOFT'98*, pp. 175–188, 1998. Extended version in *ACM Trans. Program. Lang. Syst.* 23(3): 273–303, 2001.
5. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of CONCUR'99*, LNCS 1664, pp. 114–129, 1999.
6. V. Dalmau. *Computational Complexity of Problems over Generalized Formulas*. PhD thesis, Universitat politècnica de Catalunya (UPC), 2000.
7. N.J. Fine and H.S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16, 1965.
8. B. Genest, A. Muscholl. Pattern Matching and Membership for Hierarchical Message Sequence Charts. In *Proc. of LATIN'02*, LNCS 2286, pp. 326–340, 2002.
9. B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state High-level MSCs: Model-checking and realizability. In *Proc. of ICALP'02*, LNCS 2380, pp. 657–668, 2002. Journal version in *J. of Comp. and Sys. Sci.*, 72(4):617–647, 2006.
10. B. Genest, M. Minea, A. Muscholl, and D. Peled. Specifying and verifying partial order properties using template MSCs. In *Proc. of FoSSaCS'04*, LNCS 2987, pp. 195–209, 2004.
11. D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
12. D. Harel, O. Kupferman and M. Y. Vardi. On the complexity of verifying concurrent transition systems. In *Proc. of CONCUR'97*, LNCS 1243, pp. 258–272, 1997. Journal version in *Inf. and Comput.* 173(2): 143–161, 2002.
13. ITU-TS recommendation Z.120, 1996.
14. Yu. Lifshits. Solving Classical String Problems on Compressed Texts. Available at <http://xxx.lanl.gov/abs/cs.DS/0604058>.
15. M. Lohrey. Safe realizability of high-level message sequence charts. In *Proc. of CONCUR'02*, LNCS 2421, pp. 177–192, 2002. Journal version in *Theor. Comput. Sci.* 309(1-3): 529–554, 2003.
16. H. Liu, C. Wrathall and K. Zeger. Efficient Solution of Some Problems in Free Partially Commutative Monoids. *Inf. and Comput.*, 89:180–198, 1990.
17. P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proc. of ICALP'01*, LNCS 2076, pp. 809–820, 2001.
18. M. Mukund, K. Narayan Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *Proc. of CONCUR'00*, LNCS 1877, pp. 521–535, 2000.
19. A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proc. of MFCS'99*, LNCS 1672, pp. 81–91, 1999.
20. A. Muscholl, D. Peled, and Z. Su. Deciding properties of message sequence charts. In *Proc. of FoSSaCS'98*, LNCS 1378, pp. 226–242, 1998.
21. M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. of CPM'97*, LNCS 1264, pp. 1–11, 1997.



22. D. Peled. Specification and verification of Message Sequence Charts. In *Proc. of FORTE/PSTV'00*, pp. 139–154, Kluwer 2000.
23. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. of ESA'94*, LNCS 855, pp. 460–470, 1994.
24. W. Rytter. Algorithms on compressed strings and arrays. In *Proc. of SOFSEM'99*, LNCS 1725, pp. 48–65, 1999.
25. T. J. Schaefer. The complexity of satisfiability problems. In *Proc. of STOC'78*, pp. 216–226, 1978.