Pattern Matching and Membership for Hierarchical Message Sequence Charts

Blaise Genest and Anca Muscholl

LIAFA, Université Paris VII 2, pl. Jussieu, case 7014 F-75251 Paris cedex 05 e-mail: \{genest.muscholl\}@liafa.jussieu.fr

Abstract. Several formalisms and tools for software development use hierarchy for 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 includes also hierarchical HMSCs in form of High-level MSCs (HMSC). Algorithms on MSCs rarely take into account all possibilities covered by the norm. In particular, hierarchy is not taken into account since the model usually considered are 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 consider the membership and the pattern matching problem to illustrate the way to proceed. We show that the membership problem for hierarchical HMSCs is P SPACE-complete. Second, we describe a polynomial-time algorithm for the pattern-matching problem on hierarchical MSCs.

1 Introduction

It is common to use macros to write a program or to specify the behavior of a system. Macros or hierarchical specifications allow a modular design of complex systems and have the advantage of being more succinct and user-friendly. Several formalisms and tools for software development use hierarchy for 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 (MSCs). While statecharts extend finite state machines by hierarchy and communication mechanisms, MSCs are 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 abstracts away 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 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.

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 (hierarchical) HMSCs. However, complexity can increase exponentially by unfolding. The aim of this paper is to show that this exponential blow-up is avoidable in many cases, by avoiding the expensive unfolding and using the hierarchy for computing the desired results in a modular way. We use techniques stemming from combinatorics on compressed texts, since hierarchical MSC definitions can be seen as a kind of compression by means of Straight-Line Programs (SLP).

In this paper we consider two fundamental problems for hierarchical HMSCs, that are called here nested high-level MSCs (nHMSCs for short): membership problem and pattern matching. However, we think the techniques described here can be used to solve other algorithmic problems on nHMSCs as well. The membership problem is a basic question, asking for instance whether a negative scenario occurs in a system specification, or asking whether a positive scenario is redundant, since already covered by the specification. Without hierarchy, the membership problem for HMSCs has been shown to be NP-complete. The reason for this complexity blow-up (compared to finite-state machines) is that MSCs are partial-order models. We show that hierarchy yields a small increase in complexity, precisely we show that the membership problem for nHMSCs is PSPACE-complete. Surprisingly, hierarchy alone is the source of the 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 already 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 $O(|C_M|^2\cdot|M|^2\cdot|N|^2)$, where $|M|, |N|$ denote the sizes of the description of $M$ and $N$, and $|C_M|$ 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. Regarding the complexity of extended finite state machines, [12] considers the reachability and trace equivalence problems for communicating FSMs (Finite States Machines). 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.

2
Several verification problems on MSCs and MSC-graphs have been considered recently, e.g. detecting races [2,18], model-checking [5], pattern matching with gaps [19], inference [1] and realizability [17,9,14], model-checking against partial-order logics [16,21]. Hierarchical MSCs have been also considered in [5] for the model-checking problem. We note however that our definition of nested HMSCs captures a larger class of MSC specifications than [5].

An extended abstract of this paper was presented at LATIN’02 [8]. As additional result we show here 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 \( M = \langle P, E, C, \ell, m, \prec \rangle \) where:

- \( P \) is a finite set of processes,
- \( E \) is a finite set of events,
- \( C \) is a finite set of names for messages and local actions,
- \( \ell : E \rightarrow T = \{ !j(a), ?j(a), \bar{i}(a) \mid i \neq j \in P, a \in C \} \) labels each event with its type: on process \( i \in P, \) the type is either a send \( l\!j(a) \) of message \( a \) to process \( j, \) or a receive \( ?j(a) \) of message \( a \) from process \( j, \) or a local event \( \bar{i}(a). \) 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 \( \ell(s) = !j(a) \) and \( \ell(r) = ?j(a) \) for some processes \( i, j \in P \) and some message name \( a \in C. \) We denote the events \( s,r \) as matching events and the pair \( (s,r) \) as message.
- \( \prec \subseteq E \times E \) is an acyclic relation between events consisting of:
  - a total order on \( E_i, \) for every process \( i \in P, \) and
  - \( s \prec r, \) 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.

For the questions considered here, message names are irrelevant. Thus, send events will be of type \( \bar{i}l \) and receive events of type \( \bar{l}j. \) 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 delivers messages first-in-first-out (FIFO). We assume the FIFO condition throughout the paper. That is, for all messages \( (e_k,f_k), k = 1,2, \) such
that \( \ell(e_1) = \ell(e_2) \) and \( \ell(f_1) = \ell(f_2) \) we require that \( e_1 < e_2 \) iff \( f_1 < f_2 \). The reflexive-transitive closure \( \leq \) of the acyclic relation \( < \) is a partial order called visual order. Every total order on \( E \) extending \( \leq \) is then called linearization of \( M \). A configuration (prefix) \( C \) of an MSC \( M \) is a downward closed subset of events, that is, if \( e < f \in E \) with \( f \in C \), then \( e \in C \).

Note that with the FIFO message order, any total order on a set of events \( E \) defines at most one MSC. We obtain this MSC from the event sequence 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 \( M = (P, E, C, \ell, m, <) \) and a process \( i \in P \) we let \( M_i \) denote the projection of \( M \) on the set \( E_i \) of events located on \( i \). We 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 projection on any process is equal, that is \( M_i = N_i \) for each \( i \in P \) (up to isomorphism). Note that both tests rely on the FIFO order of messages. Without the FIFO order, a linearization (or the projections on processes) does not suffice for recovering 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 \( M_q = (P_q, E_q, B_q, \varphi_q, C, \ell_q, m_q, <_q) \).

Each macro \( M_q \) consists of:

- A finite set \( E_q \) of events.
- A finite set \( P_q \) of processes.
- 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 \( \varphi_q(b) \leq \| P_q \| \).
- The type function \( \ell_q : E_q \rightarrow T_q \), that associates each event with a type \( i|j, i|j \) or \( i(a) \), with \( i, j \in P_q, i \neq j \) and \( a \in C \). 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.,\( P(e) = i \) iff \( e \in E_{q,i} \)).
- The message function \( m_q : S_q \rightarrow R_q \) that maps each (send) event of type \( i|j \) with a (receive) event of type \( j|?_i \), for all \( i \neq j \).
- 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 \( \prec_q \) 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 \( q_1 < \cdots < q_{d+1} \) with \( \varphi_{q_i}(b) = q_{j+1} \) for some \( b \in B_{q_j} \), for all \( 1 \leq j \leq d \).

We define the indices such that the lowest levels of hierarchy, which stands for levels which do not use references to other level, corresponds to small indices.

![Figure 1](image)

**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 a nMSC forbids \((f,e)\) to be a message, with \( f \) the send and \( e \) the receive, since this would contradict the acyclicity of \( <_P \), even in the case where \( M \) would be empty.

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 \( M_q = (P_q, E_q, B_q, \varphi_q, C, \ell_q, m_q, <_q) \). For simplifying the notation below, we write instead of \( \varphi_1(b) \) just \( b \).

The MSC \( \langle P, E, C, \ell, m, < \rangle \) defined by \( M = (M_q)_{q=1}^n \) is given by \( P = P_1, E = \bigcup_{b \in B_1} E_b \bigcup E_1, \ell = \bigcup_{q=1}^n \ell_q \) and \( m = \bigcup_{q=1}^n m_q \). The visual order \( < \) is defined by \( e < f \) 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 \( e <_1 f \),
- \( e, f \in E_b \) and \( e <_b f \),
- \( e \in E_1, f \in E_b \) and \( e <_1 b \),
- \( e \in E_b, f \in E_1 \) and \( b <_1 f \),
- \( e \in E_b, f \in E_{b'} \) and \( b <_1 b' \),

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 \( b_1 <_1 e \), 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 \( e \).

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 (polynomial time) whether an nMSC satisfies the FIFO condition. To test for the FIFO condition, it suffices to test that there is no \( e < g < h < f \) and no \( e < b < f \) 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 \( \varphi \) 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 given as a tuple \( G = (V, E, s, f, \varphi) \), 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 $(V_q, E_q, \varphi_q, s_q, f_q)$ 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].

We first need to define the composition of two MSCs $N_1N_2$ with $N_k = (P_k, E_k, C_k, \ell_k, m_k, <_k)$. Intuitively, we just glue together the two diagrams process-wise. Let $N_1N_2 = (P, E, C, \ell, m, <)$ with $E = E_1 \cup E_2$, $P = P_1 \cup P_2$, $C = C_1 \cup C_2$, $\ell = \ell_1 \cup \ell_2$, $m = m_1 \cup m_2$ 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 (possibly infinite) 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 $G_q$. Let us 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 $v_1, \ldots, v_n$ 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]).

---

1. Actually, [4] allows several final nodes in each automaton, which counts for the complexity of their algorithms.
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 $M \in L(G)$ with $M$ an MSC and $G$ an MSC-graph was considered in [1], together with the weak membership problem $M \in L^w(G)$. 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\(^2\), 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 $M \in L(G)$ in polynomial space.

\(^2\) This is a slightly improved runtime compared to the result stated in [1].
Proof. The idea of the algorithm is 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 $G$ (as long as we do not exclude any MSC in $L(G)$, that is as long as we do not exclude every linearization of one MSC). We consider only the linearizations in $Lin^0(G)$, where $Lin^0(G)$ is defined recursively. If $G_q$ is an nMSC, then $Lin^0(G_q)$ is the set of linearization of $G_q$.

With a path $v_1, \ldots, v_n$ in $G_q$ we associate the set of all linearizations $u_1 \cdots u_n$, where $u_i \in Lin^0(G_{\varphi(v_i)})$ for all $1 \leq i \leq n$. Let us consider a labeled graph $G_q$. Then $Lin^0(G_q)$ is the set of linearizations associated with the accepting paths of $G_q$, that is, paths starting in $s_q$ and ending in $f_q$. We define $Lin^0(G) = Lin^0(G_1)$. Intuitively, it means that we do not consider linearizations $uvbw$ of path $v_1 \cdots v_n$ where $a$ belongs to a node $v_i$ and $b$ to $v_j$ with $j < i$, that is every node needs to be fully executed before the next node can be considered.

We need to store a configuration of $M$, corresponding to the events already matched with the events from $G$. Since a configuration is a downward closed set of events, it can be stored as a tuple of $\varphi$ events (remind that $\varphi$ 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 $M = (M_q)_{q=1}^n$ will be represented by a sequence $b_1, \ldots, b_m$ of references corresponding to the unfolding of references yielding $e$. That is, we inductively remind $b_m$ for $e \in E_{\varphi(b_m)}$ where $b_m$ is a reference of $\varphi(b_{m-1})$, plus the position of $e$ in $M_{\varphi(b_m)}$. Thus, each event can be stored using linear-size memory. In our 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 $Lin^0(G)$ in space polynomial in $|G|$ (an event of $G$ is represented by a sequence of nodes then of references). Since a new node is started only when the previous node is fully executed, the last event for every process belongs to the same node. The non-deterministic 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 $Lin^0(G)$. 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.

\[ \square \]

Theorem 2 below shows that PSPACE is the lowest complexity we can obtain 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 in the non-hierarchical 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 \( M \in L^\omega(G) \) [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 \( V = \{X_1, \ldots, X_k\} \), 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 \( |X_i| \) of an SLP is the sum of the sizes of the rules. 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 \( \varphi \) SLPs \( L^1, \ldots, L^p \). 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 \( X \in \{M_q \mid q = 1 \cdots n\} \). The initial variable of each \( L^i \) is \( M_{n_i} \). Actually, the SLPs are not in Chomsky normal form to preserve this representation of nMSCs.

**Example 3.** For the nMSC \( P \) is Figure 1 we have the following SLP generating the projection on process 1: \( P_{11} \rightarrow S_{11} e M_{11}, S_{11} \rightarrow M_{11}, hM_{11} \) and \( M_{11} \rightarrow k \).

**Theorem 2** It is \( PSPACE \)-complete to check whether \( W \in L(A) \) for some SLP \( W \) and hierarchical automaton \( A \). If the alphabet is unary, then the membership problem is \( NP \)-complete.

**Remark 1** The NP-hardness result in the unary case follows also from [23].

**Proof.** We first reduce (1-in-3) SAT to the unary membership problem, since we use this reduction in the general case too. This problem is \( NP \)-complete, see [24, 6].

Let \( \varphi = \land_{j=1}^m C(\alpha_j, \beta_j, \gamma_j) \) be an instance of (1-in-3) SAT over \( n \) variables \( (x_i)_{i=1}^n \). Here, a clause \( C(\alpha_j, \beta_j, \gamma_j) \) is true if and only one of the literals \( \alpha_j, \beta_j, \gamma_j \) is true. We use the unary alphabet \( \{a_i\} \). Note that any word \( x \in a^* \) is uniquely defined by its length.

We associate with each clause \( C_j = C(\alpha_j, \beta_j, \gamma_j) \) the word \( w_j \in a^* \) of length \( 4^j \). This word can be defined by an SLP of polynomial size. Let \( W = w_1 \cdots w_m \in a^* \) be the word of length \( \sum_{j=1}^m 4^j \). The automaton \( A \) consists of a sequence of choices with transitions labeled by \( t_i \) and \( f_i \), for \( i \) varying from 1 to \( n \), where \( t_i = \sum_{j \in R_i} 4^j \) and \( R_i = \{j \mid x_i \in \{\alpha_j, \beta_j, \gamma_j\}\} \). In the same way, \( f_i = \sum_{j \in S_i} 4^j \) and \( S_i = \{j \mid (\neg x_i) \in \{\alpha_j, \beta_j, \gamma_j\}\} \).
Any path $\rho$ of $A$ 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 $(n_m n_{m-1} \ldots n_1 0)_4$. We have $W = L$ iff $(11 \ldots 10)_4 = (n_m n_{m-1} \ldots n_1 0)_4$, thus if $n_j = 1$ for all $j$. That is, there is a path in $A$ labeled by $W$ if there is a valuation satisfying $\varphi$, which implies that the membership problem for hierarchical automaton on a unary alphabet is NP-hard.

We now show the first statement of Theorem 2. We reduce the (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 $\land_{j=1}^{m} C(a_j, \beta_j, \gamma_j)$. As before, a clause $C(a_j, \beta_j, \gamma_j)$ is true iff exactly one literal is true. The PSPACE-hardness of this problem is shown in [24,6].

The idea is to make the valuations of the variables correspond to paths in the hierarchical automaton $(A_i)_{i=0,n}$ and to validate the valuations using the SLPs $(W_i)_{i=0,n}$. We define the automata $A_i$ and the SLPs $W_i$ by induction on $i = 0, \ldots, n$. Here, we use 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 $w_j$ is associated with clause $C_j$ and $t_i, f_i$ are associated with 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 $A_0$ 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:

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

The recursive definition of the automata $(A_i)_{i=1,n}$ 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 $A_i$ when $Q_i = \exists$, the automaton in the middle defines $A_i$ when $Q_i = \forall$, and the automaton on the right defines $S_i$. Note that the symbol $b$ is only generated by $S_0$. In the figure we recall its position by marking a $b$ aside each $S_i$. 

\[\]
The overall idea is as follows. The values of $x_{i+1}, \ldots, x_n$ are already chosen when an automaton calls $A_i$ (from a higher hierarchy level). The automaton $A_i$ on the left sets $x_i$ true, then uses $S_{n-i}$ to recover the fixed values of $x_{i+1}, \ldots, x_n$, and finally it sets $x_i$ false. The automaton $A_i$ in the middle guesses whether $x_i$ is true (by taking the transition labeled by $x_t_i$) or false (by choosing the transition labeled by $x_f_i$). If it chooses both transitions labeled by $x_t_i, x_f_i$ 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 $A_i$ works on figure 3, that shows the unfolding of the automaton $A_2$ 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 $x_{i+1}, \ldots, x_n$, we show $S_{n-i}$ for $n = 9, i = 7$ in the figure below.

$A_i$ and $S_i$ are designed so that any path of $A_i$ is labeled by at most one $x_t_i$ and at most one $x_f_i$ 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 $x_t_i$ and $x_f_i$, but not by two $x_f_i$ or two $x_t_i$. By contradiction, assume that there are two consecutive $b$’s in $A_i$ such that there is a path from one to another labeled by two $x_t_j$ (the case $x_f_j$ is symmetric). We take the minimal $A_i$, which ensures this. By the minimality of $A_i$, this can only happen either because of the first $x_t_j$ transition of $A_i$, or between $S_{n-i}$ and one of the two $A_{i-1}$. Since in $S_{n-i}$ all $x_t_k$ occur after the (unique) $b$, there is no $x_t_j$. 

---

**Footnotes:**

1. In the example, the numbers $1, 2, \ldots, n$ are used to denote the states of the automata, and $b$ is the symbol for the fictive $b$ transition. The figure shows the unfoldings of $A_2$ for two different $\varphi$ formulas.

2. The $S_{n-i}$ automata are designed to recover the fixed values of $x_{i+1}, \ldots, x_n$ from the $A_i$ automata. The transitions are labeled with $x_t_i$ or $x_f_i$, depending on whether $x_i$ is true or false.

3. The minimality of $A_i$ ensures that the conditions for the occurrence of $x_t_j$ or $x_f_j$ cannot occur in any other way than described above.

---

**References:**

1. [Automata Theory](https://example.com)
2. [Formal Language Theory](https://example.com)

---

**Further Reading:**

1. [Automata Theory: A Comprehensive Guide](https://example.com)
2. [Introduction to Formal Languages and Automata](https://example.com)
Fig. 3. Unfolding of $A_2$ for $Q_2x_2Q_1x_1 = \forall x_2 \forall x_1$ on the left, and on the right, unfolding of $A_2$ for $Q_2x_2Q_1x_1 = \exists x_2 \forall x_1$
in $A_{i-1}$ before its first $b$ (if any). It already shows a contradiction in 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 $x_{tj}$ between the last $b$ of $A_{i-1}$ and the $b$ in $S_{n-i}$, for all $k < i$. Finally, between the $b$ of $S_{n-i}$ and the first $b$ of the second $A_{i-1}$ there can be at most one $x_{tk}$ with $k > i$ (from $S_{n-i}$) and at most one $x_{tk}$ with $k < i$ (from $A_{i-1}$). Thus, in all cases we contradict the assumption on $A_i$.

Let us prove that $W_n \in L(A_n)$ 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 $0$) is labeled by $x_n$ and all nodes on level $l$ are labeled by $x_l$. The leaves are on level 0, and are unlabeled. A node $v$ labeled by $x_i$ corresponds to a valuation $\sigma(v)$ of the variables $x_{i+1}, \ldots, x_n$. For instance, if the valuation for a node is $x_n$ is true, then its children must evaluate $x_n$ to true, and evaluate $x_{n-1}$ either to true or false. Moreover, a node on level $k$ have two children if $x_k$ is universally quantified (one child evaluate $x_k$ true and the other one false), and one child if $x_k$ 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 valuation of every leaf, $\psi$ is true.

Using the property we just showed, we can note that between any two consecutive $b$'s of any path of $A_n$, 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 four for determining whether a clause is true, is still applicable. Hence, a path $\rho$ of $A_n$ 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.

Assume 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 $x_j$ 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 $\rho(v) = T^{-1}(v)W_iF^{-1}(v)$. We recall that $T(v), F(v)$ are words over $a^*$, hence $T^{-1}(v)W_iF^{-1}(v)$ 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 $L(A_i)$ 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 $T(v)F(v) = W_0$ using the same argument as in the NP-hardness case. Hence $\rho(v) = W_0W_0^{-1} = \epsilon \in L(A_0)$.

Consider 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 $L(A_{i-1})$. Then,

$$\rho(v) = T^{-1}(v)W_iF^{-1}(v) = T^{-1}(v)W_{i-1}bW_{i-1}F^{-1}(v)$$
$$= T^{-1}(v)T(v_1)\rho(v_1)bT(v_2)\rho(v_2)F(v_2)F^{-1}(v)$$
$$= xt_1\rho(v_1)bT(v_2)\rho(v_2)xf_1$$

We used in the equations above $T^{-1}(v)T(v_1) = xt_1$ for the positive child $v_1$ of $v$ and $F^{-1}(v)F(v_2) = xf_1$ for the negative child $v_2$ of $v$. Moreover, $F(v_1)bT(v_2) =
$F(v) \land T(v) \in L(S_{n-1})$ since the indices of false variables in $\sigma(v_1)$ and of true variables in $\sigma(v_2)$ form a partition of $\{i+1, \ldots, n\}$. This shows that $\rho(v) \in L(A_i)$.

Consider 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 $L(A_{i-1})$. It is easy to show now that $\rho(v) \in L(A_i)$ using:

$$
\rho(v) = T^{-1}(v)W_iF^{-1}(v) = T^{-1}(v)W_{i-1}F^{-1}(v) \\
= T^{-1}(v)T(v_1)\rho(v_1)F(v_1)F^{-1}(v) \\
= x_i\rho(v_1)
$$

For the reverse direction the arguments are similar. From a word $W = W_n$ of $A = A_n$, we obtain subwords $\rho(v)$ in $L(A_i)$ as above, labeled by $T^{-1}(v)W_iF^{-1}(v)$. For each leaf $v$ this means that $\sigma(v)$ satisfies exactly one literal per clause.

\[ \square \]

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 $M \in L(G)$ 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 (1-in-3)QBF. Let $F$ be an instance of (1-in-3)QBF of the form $F = (Q_n x_n) \ldots (Q_1 x_1) \varphi$, where $Q_i \in \{\exists, \forall\}$ and the formula $\varphi$ is of the form $\land_{j=1, \ldots, m} R(\alpha_{j,1}, \alpha_{j,2}, \alpha_{j,3})$, 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 $F = F_n$. Let $F_i = (Q_i x_i) F_{i-1}$, with $F_0 = \varphi$. Each $F_i$ will determine $G_i, M_i$.

The processes used in the construction are $SC_1, \ldots, SC_m$ and $RC_1, \ldots, RC_m$, plus $VY_1, \ldots, VN_n$ and $VN_1, \ldots, VN_n$. 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 $4n$ vertices, labeled by $MY_i, MN_i$, or $\emptyset$. The graph chooses between $MY_i$ and $MN_i$ for all $i$, as depicted below:

15
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 will use the order between $a_i$, $b_i$ as follows: The sequence $a_i b_i$ means that $x_i$ is true, while $b_i a_i$ 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 $F_{i-1}$. For simplicity, we denote $a = a_i$ and $b = b_i$. Note that in a valuation tree for $F$ showing that $F$ 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 usual, by assigning each universally quantified variable two children labeled 0 and 1, respectively each existentially quantified variable one child labeled 0 or 1.

If $F_i = \forall x_i F_{i-1}$, then let $M_i = (ab)^2 M_{i-1} S_i (ba)^2 M_{i-1}$ (see Figure 4.1). 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.

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

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 $MN_i$, so $\rho_j$ consumes all occurrences of $a, b$ in $(ab)^{2^f}$. In particular, all 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 forced to be false.
Suppose now that \( F_i = \exists x_i F_{i-1} \). Let \( M_i = (ab)^{2f}(a)M_{i-1} \), and \( G_i = (V_i, E_i) \), where \( V_i = V_{i-1} \cup \{e_0, e_1, e_2, e_3\} \). Let \( E_i = E_{i-1} \cup \{(e_0, E_i), (e_3, e_1), (e_1, e_0), (e_2, e_3)\} \), where as above \( E_i \) is the initial vertex and \( F_i \) 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 \( e_2 \) with \( a \), and \( e_0 \) et \( e_3 \) with \( \emptyset \).

The underlying idea in this case is that the additional occurrence of \( a \) in \( M_i \) must be matched by \( e_3 \) 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 easy, see the proof of theorem 2.

\[ \Box \]

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 \( A \) are flat, the membership problem is solvable in polynomial time.

**Theorem 4** 1. Let \( W \) be a word defined by an SLP and let \( A \) be an NFA.
Deciding whether \( W \in L(A) \) can be achieved in time \( O(\|W\| \cdot |A|^3) \).
2. Let \( W \) be a word and let \( A \) be a hierarchical automaton (hNFA for short).
Deciding whether \( W \in L(A) \) can be achieved in time \( O(\|W\|^3 \cdot |A|^3) \).

For the first statement in the theorem above a similar result (for Lempel-Ziv compressed words and regular expressions) has been shown in [23].

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

**Membership** \([(X_i)_{i=1,n} \text{ SLP-compressed word, } A=\langle V, E, a_0, a_f \rangle \text{ NFA)}\]
For each variable \( X_i \) in the lowest level:
\[ T_{X_i} = \{ (a, b) \in V \times V \mid a X_i \rightarrow b \} ; \]
For $i = 1 \ldots n$:
Let $T_{X_i} = \emptyset$;
Let $Y, Z$ s.t. $X_i \rightarrow Y Z$;
For all vertices $a, b, c \in V$:
If $(a, b) \in T_Y$ and $(b, c) \in T_Z$:
$T_{X_i} = T_{X_i} \cup \{(a, c)\}$;
Return $(a_0, a_f) \in T_{X_i}$;

The second algorithm computes for each sub-automaton $B$ of a hNFA $A$ the set $T_B$ of factors $W_1 \cdots W_j$ of a word $W$ that it accepts. This algorithm uses another dynamic programming algorithm calculating for each factor $W_i \cdots W_j$ the set $T_{i, j}$ of pairs $(a, b)$ of states of $B$ between which a path labeled by $W_i \cdots W_j$ exists. For convenience, we assume without loss of generality that every transition is a sub-automaton, but those on the lowest level of the hierarchy (else, we just replace every transition labeled by $a$ by a sub-automaton in the lowest level of the hierarchy with a unique transition from the initial to the final state labeled by $a$). 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 $W_i \cdots W_j$, or else the path labeled by $W_i \cdots W_j$ 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 first the sets $T_{k, k+d}$ for small $d$, which allows us not to use a costly fix point algorithm.

**Membership (W word, A=($V, E, a_0, a_f$) hNFA)**
For each sub-automaton $B$ of $A$ in the lowest level of hierarchy:
$T_B = \{(i, j) \mid W_i \cdots W_j$ is accepted by $B\}$;
For each sub-automaton $B$ of $A$, by increasing hierarchical level:
For $d = 0, \ldots, |W|$, for $i = 1, \ldots, |W| - d$,
$D_{i, i+d} = \{(a, b) \mid \exists \text{ subaut. } C \text{ s.t. } a \xrightarrow{C} b$
and $(i, i+d) \in T_C\}$;
For each $e < d$ and every $a, b, c$ vertices of $B$,
If $(a, b) \in D_{i, i+e}$ and $(b, c) \in D_{i+e, i+d}$:
$D_{i, i+d} = D_{i, i+e} \cup \{(a, c)\}$;
$T_B = \{(i, j) \mid (a_0, a_f) \in D_{i, j}\}$;
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 since it is already NP-hard for $H$ an HMSC [1], and it suffices to guess a path of $G$ of the size of $M$, which is polynomial, and check whether it is labeled by $M$. 

18
\[
\begin{array}{|c|c|c|c|}
\hline
M & G & \text{words} & \text{MSC} \\
\hline
\text{Flat} & \text{Nested} & P & \text{NP-complete} \\
\hline
\text{Nested} & \text{Flat} & P & \text{PSPACE-complete} \\
\hline
\text{Nested} & \text{Nested} & \text{PSPACE-complete} & \text{PSPACE-complete} \\
\hline
\end{array}
\]

**Fig. 4.** Complexity of membership problems.

## 5 Pattern Matching of nMSCs

The aim of this section is to show that pattern matching on nMSCs can be achieved in polynomial time, i.e., without unfolding the nMSCs. We first consider a special case of pattern matching, namely testing equality of nMSCs. Then we describe first a pattern matching algorithm when the pattern nMSC is connected, and second the additional work to do when the pattern is not connected.

### 5.1 Equality of nMSCs

Recall first that the FIFO message order allows testing the equality of two MSCs \(M\) and \(N\) process-wise, which amounts to testing the equality of \(\varphi\) pairs of words (over the type alphabet \(T\)). In the hierarchical case we already used in Section 4.1 the representation of an nMSC \(M\) by \(\varphi\) straight-line programs \(L^i\), where the SLP \(L^i\) generates the projection \(M_i\) of \(M\) on process \(i\).

Thus, for testing the equality of two nMSCs in polynomial time, we can use directly the following result:

**Theorem 5 ([22])** 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 \(N = (P, E, C, \ell, m, <)\) 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 \(e < q < f\), then \(q \in F\).

In particular, the message mapping \(m\) must be one-to-one between the send and receive events 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 \(N_i\). Of course, this condition is necessary. Before to consider the hierarchical case, we show a simple algorithm for the non-hierarchical 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, [15]. We need the linear time algorithm of Knuth-Morris-Pratt for determining occurrence 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 $N$. Thus, we look for a configuration of $N$ such that on each process $i$, we have $M_i$ as a suffix. This is done by recording the set $J$ of processes $i$ satisfying this condition and progressing one event at a time on processes $j \notin J$. If this is not possible, the next event on every $i \notin J$ is a receive from some $j \in J$, while the corresponding send from $j$ to $i$ in $N$ has not been seen yet. We then progress on $j$, and update $J$ by using Knuth-Morris-Pratt algorithm to know whether $j \in J$ or $j \notin J$. The overall complexity of the algorithm is linear, by taking care that each event in $N$ is considered at most once. □

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

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 $N_i < N_j$.
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 it 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 a factor $M$. The basis of our algorithm is a pattern matching algorithm for SLP-compressed words, that was proposed in [20] (see also [22]):

**Theorem 7 ([20])** Let $P$ be an SLP and let $A, B$ be two variables of $P$. One can determine all occurrences of the word defined by $A$ in the word defined by $B$ in time $O(|A|^2|B|^2)$.

![Diagram](image)

**arithmetic progression** $\text{Occ}(X, Y, V^i)$

The idea of the algorithm in [20] is based on word combinatorics. Let $X$ be a variable of $A$ 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 rule $Y \rightarrow \alpha \in \Sigma^*$. Then $X$ occurs in a variable $Y$ with $Y \rightarrow V^1 \ldots V^k$. Let $i$ be such that $V^i$ is the first symbol (variable or letter) that this occurrence of $X$ overlaps, and the occurrence ends beyond $V^i$ (see also figure above). In particular, $Y$ is the lowest variable that contains an 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 $V^i$, or starts and ends within $V^i$ if $V^i$ is in the lowest level of the hierarchy. Let $\text{Occ}(X, Y) = \bigcup_i \text{Occ}(X, Y, V^i)$. 

20
Using a combinatorial argument (lemma of Fine and Wilf, [7]), it is shown in [20] that \( \text{Occ}(X, Y, V') \) 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, precisely at most one for each \( i \) (and \(|V'| \) for \( V' \) on the lowest level of the hierarchy). That is, we can represent \( \text{Occ}(X, Y, V') \) by a triple of numbers \((n, s, k)\) where \( n \) and \( n + s \) are the positions in \( Y \) of the two first occurrences of \( X \) in \( \text{Occ}(X, Y, V') \), and \( k = \# \text{Occ}(X, Y, V') \) is the number of occurrences of \( X \) in \( \text{Occ}(X, Y, V') \). That is, we have \( Y = Y_1XY_2 \) with \(||Y_1|| = n + is\), for all \( 0 \leq i < k \). As an example, consider the words \( Y = aaabababab\) and \( X = abab \). 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** Using the algorithm of [20] we immediately obtain 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 by Flandowski in [22].

Throughout the section we denote occurrences of projections \( M_i \) using superscripts. That is, \( M_i^i \) will correspond to a given starting position of \( M_i \) as pattern of \( N_i \). Suppose that \( M_i \) is a factor of \( N_i \), for all \( i \in P \). We say that \((M_i^i)_{i=1}^P \) forms a factor of \( N \) if there exists \( M \) a factor of \( N \) such that \( M_i = M^i \) for all \( i \in P \).

### 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 in this section that \( M \) cannot be written as \( M_1M_2 \), where \( M_1, M_2 \) are non-empty MSCs with no common process.

**Definition 7.** Let \( M_i^i \) and \( M_j^j \) be occurrences of \( M_i \) in \( N_i \), resp. of \( M_j \) in \( N_j \). We say that \( M_i^i \) and \( M_j^j \) are compatible, if the first send (resp. receive) between the processes \( i \) and \( j \) on \( M_i^i \) matches the first receive (resp. send) on \( M_j^j \) (if \( i, j \) communicate in \( M \)). More generally, we call the indices corresponding to \( M_i^i \), \( M_j^j \) in a given arithmetic progression compatible.

**Lemma 1.** Let \((M_i^i)_{i=1}^P \) be occurrences of \( M_i \) in \( N_i \). Then \((M_i^i)_{i=1}^P \) forms a factor of \( N \) iff \((M_i^i)_{i=1}^P \) are pairwise compatible.

As in the previous section we will denote by \( \text{Occ}(M, Y) \) the set of occurrences \( M^0 \) of \( M \) in \( Y \) such that \( M^0 \) does not occur in any \( Z < Y \). We denote by \( \text{Occ}(M, Y, V) \subseteq \text{Occ}(M, Y) \) those occurrences that start within \( V \) (ending beyond \( V \)), where \( V \subseteq Y \) is a reference occurring literally in \( Y \). It means that one event of the occurrence has to occur in \( V \) and one (not necessarily on the same process) has to occur not in \( V \). Obviously, no event may occur before \( V \).

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 $M^0_{ij} \in \text{Occ}(M_{ij}, Y)$ there can be at most one occurrence $M^0_{ij}$ in $Y$ that is compatible with $M^0_{ij}$.

2. For each occurrence $M^0$ in $\text{Occ}(M, Y, V)$ there exists some process $h$ such that $M^0_{ih} \in \text{Occ}(M_{ih}, Y, V)$. We call such a process $h$ a leading process for $M^0$. Thus, any pairwise compatible tuple $(M^0_{ik})_{k \neq h} \subseteq Y$ is determined by the occurrence $M^0_{ih}$, because of Fact 1.1.

**Example 4.** For the nMSC $P$ in Figure 1 and the pattern $N$ in Figure 5 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 $(0, 0, 0)$.

![Figure 5. Pattern MSC N](image)

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 $i = n (j = 0)$ or $i = n + (k - 1)s (j = k - 1)$. Any non external index is called an internal index.

The next proposition provides the main argument that the search for a pairwise compatible tuple of occurrences $(M^0_{i})_{i \in P}$ can be done in polynomial time. Intuitively, we must show that the occurrences of $(M^0_{i})_{i \in P}$ 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 $N_q$.

**Proposition 1** Assume that $M^0 \in \text{Occ}(M, Y, V)$ with $M^0_{i} \in \text{Occ}(M_{i}, Y^i, V^i)$, 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 $i$. 

22
2. For some leading process $h$ for $M^0$ (i.e., $V^h = V$ 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 there is no leading process $h$ such that $M^0|_h$ is an external index of $\text{Occ}(M|_h, Y^h, V^h)$. Assume also that there is a message from process $i$ to process $j$ in $M$. We decompose $M|_i = A_{i,j}B^*_{i,j}C_{i,j}$ such that $B^*_{i,j}$ begins with the first send from $i$ to $j$, and ends with the last one. Similarly, we decompose $M|_j = A_{j,i}B^*_{j,i}C_{j,i}$ such that $B^*_{j,i}$ 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.

**Lemma 2.** Let $\pi = \text{Occ}(M|_i, Y, V)$ be an arithmetic progression consisting of at least three indices. Then each $B^*_{i,j}$ associated with some internal index of $\pi$ belongs to $\text{Occ}(B^*_{i,j}, Y, V)$.

**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 $d \geq 2$ 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 $C_{i,j} = a_{k+1} \cdots a_1 \cdots a_m$, 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 $B^*_{i,j}$ also contains this position, except possibly for the first and the last $B^*_{i,j}$. Hence, every $B^*_{i,j}$ associated with an internal index of $\pi$ is in $\text{Occ}(B^*_{i,j}, Y, V)$.

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 that $B^*_{h,j} \in \text{Occ}(B^*_{h,j}, Y, V)$. Hence, we also have $B^*_{h,j}, h \in \text{Occ}(B^*_{h,j}, Y, V)$, 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 $B^*_{h,j} \in \text{Occ}(B^*_{h,j}, Y, V)$ 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 $j$, 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 $O(|M|^2|N|^2)$.
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 \(\text{Occ}(M_1,Y,V), \ldots, \text{Occ}(M_p,Y,V)\);
For every variable \(Y\) of \(N\):
  For every \(pos(h)\) at the beginning or end of an
  arithmetic progression of \(\text{Occ}(M_h,Y)\):
    Let \((M_h)^{\text{pos}(h)}\) be the corresponding occurrence of \(M_h\):
    If there exists \(((M_k)^{\text{pos}(k)})_{k \neq h}\) compatible with \((M_h)^{\text{pos}(h)}\)
    where for all \(k\), \(\text{pos}(k) \in \text{Occ}(M_k, Z^k)\) with \(Z^k \leq Y\):
      Return \((Y, (\text{pos}(k))_k \in \rho)\);
  For every \(V \in Y\) s.t. \(\forall i, \pi_i = \text{Occ}(M,i,Y,V) \neq \emptyset:\
    For each \(i\), let \(\pi_i = (n_i, s_i, k_i)\);
    Let \((t_1, \ldots, t_p, e_1, \ldots, e_p) = \text{Periods}(\text{Reduce}(\pi_1, \ldots, \pi_p))\);
    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 \(\text{pos}(k)\) to be inside \(Y\) for every \(k\) to ensure that \(h\) is leading, which ensures the uniqueness of \(\text{pos}(k)\) 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 other process \(j > i\). The algorithm first computes the occurrences \(M_{\lfloor i\rfloor}\) 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 \(M_0 \subseteq \text{Occ}(M,i,Y,V)\) 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 \(i\) 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 \text{Reduce} and \text{Periods} and show that our algorithm returns only occurrences of \(M\) which are indeed factors of \(N\). The subroutine \text{Reduce} restricts the arithmetic progressions \((\pi_1, \ldots, \pi_p)\) by adding an offset to each arithmetic progression \(\pi_i\). 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 \text{Reduce} with \(u'_1\), 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 a 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 of \(\pi_j\). We then compute the events which fulfills every constraint.

24
Let $\pi_1, \ldots, \pi_\varphi$ be arithmetic progressions of occurrences of $M_1, \ldots, M_\varphi$, such that for each pair $i \neq j$ there exists a message between $i, j$ from each $u_i$ in $\pi_i$ to some $u_j$ in $\pi_j$, and vice-versa. That is, $\pi_1, \ldots, \pi_\varphi$ is the result of a call of Reduce. Let $u_0^i$ be the first index of each arithmetic progression. The only problem that remains for deciding whether there exist compatible occurrences $M_1, M_2$ 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 $(u_1, \ldots, u_\psi)$ corresponding to the starting positions of pairwise compatible tuples $((M_1)^0, \ldots, (M_\psi)^0)$. As we show later, such tuples occur periodically, hence we just need to determine some periods $(\mu_1, \ldots, \mu_\psi) \in \mathbb{N}^\psi$ and the first positions $(u_1^0, \ldots, u_\psi^0)$ from which we can apply these periods.

For all $i < j$ let $z_{i,j} < m_{j,i}$ be the number of events $ilj$ in $u_0^i$ before the first one that has a matching receive in $\pi_j$. Let $z_{j,i} < m_{j,i}$ be the number of $j?i$ in $u_0^j$ before the first that has a matching send in $\pi_i$. In the figure aside, $m_{1,2} = 2$, $m_{2,1} = 3$, $z_{1,2} = 1$ and $z_{2,1} = 0$. Let $z_{i,j}^0$ be such that after reading the first $z_{i,j}^0$ we arrive at a message consisting of the first $ilj$ of some $u_i$ and the first $j?i$ of some $u_j$. In the example, we marked as $z^0$ the earliest message consisting of the first $i?j$ of some $u_i$ and the first $2?i$ of some $u_2$, and $z_{1,2}^0 = 3$. So $z_{i,j}^0 + z_{j,i} \equiv 0 \pmod{m_{i,j}}$ 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 $z_{i,j}^0$ such that $z_{i,j}^0 + z_{j,i} \equiv 0 \pmod{\text{lcm}(m_{i,j}, m_{j,i})}$ to the above equations in time $O(\min(|M_1|, |M_2|)^3)$. 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 $u_i^0$. Notice that $\mu_i$ divides $\text{lcm}\{m_{i,j} \mid i < j\}$. The restriction of the arithmetic progression $\pi_i$ according to $\mu_i, u_i^0$ is denoted $\pi_i'$.

The first $ilj$ of each $u_i$ in the restricted arithmetic progression $\pi_i'$ corresponds to 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 $(\pi_i')$. Let $x_{i,j}$ be an integer denoting the number of $u_i$ between $u_0^j$ and the reception of the first message from $u_0^j$. We want to compute all tuples $(u_i)_{i=1,\varphi}$ such that the first $ilj$ of $u_i$ matches the first $j?i$ of $u_j$. That is, we need a solution $(t_i)_{i=1,\varphi}$ of the following system of $\varphi(\varphi - 1)$ 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 $t_i$ determines each $t_i$, modulo some value $e_i$ depending on the $(m_{i,j})_{i,j}$. We can combine the equation for $(1, i)$ with the equation for $(i, j)$ to obtain a system of $\varphi(\varphi - 1)$ equations:

$$\delta_{i,j} t_1 = y_{i,j} + \nu_{i,j} t_j$$
Let \( j \leq \varphi \). Notice that several of these equations (for different \( i \)) concerns the same \( t_1 \) and \( t_j \): either all these equations are equivalent, or there exists a unique or no solution at all (we just combine two equations by multiplying per \( \nu_{i,j} \) one and by \( \nu_{i,j} \) the other and subtracting one equation with the other one). If there is a unique solution, we stop the procedure and test this solution in each equation. If this is indeed a solution of the system, we return its value, else we will not find an occurrence of \( M \) in this level. Hence, we can assume for the following that there is a unique equation (since all are equivalent) for each \( j \), that is we have a system of \( \varphi \) equations, where \( i \) is fixed.

If \( \gcd(\delta_{i,j}, \nu_{i,j}) \) 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 \( \gcd(\delta_{i,j}, \nu_{i,j}) \), and thus consider only the case where \( \gcd(\delta_{i,j}, \nu_{i,j}) = 1 \).

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

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 \texttt{Periods} costs time \( O(|M|^4) \).

\textbf{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.

\subsection*{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 \( O(|C_M|^2) \leq O(\varphi^2) \), 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 \( M' = M : \prod_{i \in P_N \setminus P_M} \text{loc}_i \). Obviously, \( M' \) occurs in \( N' \) iff \( M \) occurs in \( N \).

\begin{center}
\begin{tikzpicture}
\node (1) at (0,0) {1};
\node (2) at (1,0) {2};
\node (3) at (2,0) {3};
\node (4) at (3,0) {4};
\node (5) at (4,0) {5};
\node (a1) at (0,-1) {\text{Occ}(M_{1,2})};
\node (a2) at (4,-1) {\text{Occ}(M_{3,4})};
\draw (1) -- (2) -- (3) -- (4) -- (5);
\end{tikzpicture} \Rightarrow \begin{tikzpicture}
\node (1) at (0,0) {1};
\node (2) at (1,0) {2};
\node (3) at (2,0) {3};
\node (4) at (3,0) {4};
\node (5) at (4,0) {5};
\node (a1) at (0,-1) {\text{loc}_i};
\node (a2) at (4,-1) {\text{Occ}(M_{3,4})};
\draw (1) -- (2) -- (3) -- (4) -- (5);
\end{tikzpicture}
\end{center}

Let \( M, N \) be nMSCs. For each reference \( X \) of \( M \) or \( N \), let \( C_X \subseteq 2^{P_M} \) 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 \( C \subseteq \mathcal{C}_X \). In other words, \( X = (X|_C)_{C \subseteq \mathcal{C}_X} \) represents the decomposition of the MSC associated with \( X \) into weakly connected nMSCs. It follows from the previous section that we can compute in time \( O(|\mathcal{M}|^2 N^2) \) a compact representation of all occurrences of \( M|_C \) in \( N \), for each \( C \subseteq \mathcal{C}_M \). The next definition states when a tuple of occurrences \( (M|_C)_{C \subseteq \mathcal{C}_M} \) of the weakly 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 weakly connected components of \( M \), where \( C, D \subseteq \mathcal{C}_M \) 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 \( C \subseteq \mathcal{C}_M \). Then \( (a_C)_{C \subseteq \mathcal{C}_M} \) is an occurrence of \( M \) in \( Y \) iff \( a_C, a_D \) are compatible for all \( C, D \subseteq \mathcal{C}_M \), \( C \neq D \).

**Proof.** The implication from left to right follows directly from the definition of pattern. For the converse assume that \( (a_C)_{C \subseteq \mathcal{C}_M} \) 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 \( s_1, r \) precedes \( s_{i+1} \), and \( r_m \) precedes \( a_D \). Since all processes appear in \( M \), there exist some \( k \) and \( C', D' \subseteq \mathcal{C}_M \) 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.

Let \( C \subseteq \mathcal{C}_M \). Note that the occurrences of the weakly 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 weakly connected component in the proposition below.

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

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 \( Z \subseteq Y \) 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 $(\text{Occ}(M_{\mid C}, N))_{C \in C_M}$). 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 $M_{\mid C}$ 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 $C \in C_M$. The occurrence $\downarrow C$ for component $C$ means an occurrence of $M_{\mid C}$ after $Y$, while $\uparrow C$ for $C$ means an occurrence of $M_{\mid C}$ before $Y$. Thus, we let $\uparrow C < a_C < \downarrow C$ for all $a_C \in \text{Occ}(M_{\mid C}, Y)$. For $C \neq D$, we say that $\uparrow C, a_D \in \text{Occ}(M_{\mid C}, Y)$ 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 $E \subseteq \{\#1, \#1\mid C \in C_M\}$ be a set of constraints. We define $\text{Min}^Y_E = (a_C)_{C \in C_M}$ as the minimal tuple satisfying the following conditions:

1. For each $C \in C_M$, $a_C \in \text{Occ}(M_{\mid C}, Y) \cup \{\downarrow C, \uparrow C\}$.
2. The occurrences $(a_C)_{C \in C_M}$ are pairwise compatible.
3. $(a_C)_{C \in C_M}$ satisfies the constraint $E$. That is, $(\# \downarrow D) \in E$ implies that $a_D \neq \downarrow D$ and $(\# \uparrow D) \in E$ 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 $(a_C)_{C \in C_M}$ 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

- $\text{Min}_{\#1} = (\downarrow 1, \downarrow 2, \downarrow 4, \downarrow 5)$, and
- $\text{Min}_{=1} = (\downarrow 1, \downarrow 2, \downarrow 4, \downarrow 5)$.

In the figure to the right we also have:

- $\text{Min}_{\neq \#1} = (a, \downarrow 2, e, \downarrow 4, \downarrow 5) = \text{Min}_{\#1, \neq \#1}$,
- $\text{Min}_{=12} = (b, \downarrow 2, e, \downarrow 4, \downarrow 5)$,
- $\text{Min}_{\neq 1, \neq 4} = (\downarrow 1, \downarrow 2, \downarrow 3, 9, \downarrow 5)$.

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

**Lemma 4.** Let $(b_C)_{C \in C_M} = \text{Min}^N_{(\# \downarrow C)_{C \in C_M}}$. Then $M$ is a pattern of $N$ iff $b_C \neq \downarrow C$, for all $C \in C_M$.

The problem is that we might need the tuples $\text{Min}^Y_E$ for arbitrary sets $E$ of constraints (and there are exponentially many). Fortunately, we can avoid the
exponential blow-up by computing \( \min^Y_E \) only for singletons \( E = \{ \neq 1 \} \) and \( E = \{ 1 \} \). We first show that these tuples suffice for computing in polynomial time \( \min^Y_E \) 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 \( E, F \subseteq \{ \neq 1 \} \subseteq \mathcal{C} \) be two sets of constraints. Then \( \min^Y_{E \cup F} = \max(\min^Y_E, \min^Y_F) \).

**Proof.** Let \( b = (b_C)_C = \max(\min^Y_E, \min^Y_F) \). We have of course \( \min^Y_{E \cup F} \geq \min^Y_E \) and \( \min^Y_{E \cup F} \geq \min^Y_F \), hence \( \min^Y_{E \cup F} \geq b \). But \( \min^Y_{E \cup F} \) 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, \( b = \min^Y_{E \cup F} \). □

\[
\begin{align*}
\min^Y_{\{ \neq \}^1 \{ \neq \}^2} &= (a, b, c, \downarrow 4, d), \\
\min^Y_{\{ \neq \}^1 \{ \neq \}^1} &= (a, \downarrow 2, c, \downarrow 4, d) = \min^Y_{\{ \neq \}^2}, \\
\min^Y_{\{ \neq \}^2} &= (\downarrow 1, b, \downarrow 3, \downarrow 4, \downarrow 5).
\end{align*}
\]

**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 \( \min^Y_{\{ \neq \}^1 \{ \neq \}^1} \) and \( \min^Y_{\{ \neq \}^2} \) can be computed in time \( O(|Y|^2) \) from the tuples \( \min^Z_{\{ \neq \}^2} \) \( \in \mathcal{C}_M \) and \( \min^Z_{\{ \neq \}^1} \) \( \in \mathcal{C}_M \), where \( Z \in Y \).

**Proof.** Assume 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 \( E_\cdot \subseteq \mathcal{C}_M \) 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 $E_1 = \emptyset$ and we augment $E_1$ as long as there exist $a, b$ with the following properties:

- $(ac)_C$ is an occurrence in $Y^1$ with $a_C = \uparrow_C$ iff $C \in E_1$,
- $(bc)_C$ is an occurrence in $Y^2$ with $b_C = \uparrow_C$ iff $C \notin E_1$.

The algorithm for computing $\text{Min}^Y_{\neq 1_D}$ is described below (for $\text{Min}^Y_{\{=1_D\}}$ the reasoning is similar):

1. Let $E_1 = \emptyset$
2. Compute $(ac)_C = \text{Min}^Y_{\neq 1_D}$, with $E = \{\neq | 1_D \} \cup \{= | 1_C \mid C \in E_1\}$
3. Let $E_1 = \{C \mid a_C = \uparrow_C\}$
   // For all $C \in E_1$, $M|_C$ must be in $Y^2$ or after $Y$.
4. Compute $(bc)_C = \text{Min}^Y_{\neq 1_C \in E_1}$
5. Let $E_1 = \{C \mid b_C \neq \uparrow_C\}$. If $E_1$ changes, then goto (2).
6. Let $d_C = b_C$ if $C \in E_1$, and $d_C = a_C$, otherwise.
7. Return $(dc)_C$.

Note that each time the set $E_1$ changes at step (3), it increases by at least one component. Hence, we return to step (2) at most $O(\varphi)$ times.

For the running time let us denote by $E_1^t$ the value of $E_1$ after $t$ iterations. The $t$-th iteration needs time $\varphi(|E_1^t| - |E_1^{t-1}|)$, thus the overall running time is at most $O(\varphi^2)$.

If an nMSC has more than two references, then we define several sets $E_1^t$ to explain the minimal reference $Y^i$ where the occurrence of the projection should be. Considering that for each step, one set $E_1^t$ has to change, the running time is $O(\varphi^3)|Y|$. \hfill \qed

**Theorem 9** We can test whether $M$ occurs as pattern of $N$ in time $O(C_M^2(|M|^2 |N|^2))$.

**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 $C \in C_M$ 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 $\text{Min}^Y_{\neq 1_C}$ and $\text{Min}^Y_{= 1_C}$ for all components $C \in C_M$ and references $Y$ of $N$. We apply Proposition 3 to compute $\text{Min}^Y_{\neq 1_C}$ and $\text{Min}^Y_{= 1_C}$. The time costs are $O(|M|^2|N|^2)$ for the connected components and $O(\varphi^3|N|) \leq O(|M|^2|N|^2)$ 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 $C_M^2$, where $C_M$ is the number of connected components of $M$, expressing additional constraints due to components $M|_C$ that might be split over several references of $N$. \hfill \qed
6 Conclusion

In developing new techniques for algorithms on hierarchical MSCs, we proved that algorithms can benefit from the redundancy provided by the use of macros. Namely, it is not a good idea to unfold the hierarchical system since the redundancy is lost. Moreover, we use the hierarchy to lower the running time. We showed that pattern matching and membership can efficiently use the hierarchy, together with techniques stemming from combinatorics, arithmetics and dynamic programming. We believe that similar results can be stated for many other problems on hierarchical MSCs, such as model-checking against properties expressed by template MSCs [10].

Acknowledgment: We wish to thank Marc Zeitoun for his insightful comments on previous versions of this paper, as well as Markus Lohrey and the referees of LATIN’02 for several suggestions for improvement.

References