Analysing Message Sequence Graph Specifications

Joy Chakraborty[1], Deepak D'Souza[2], and K. Narayan Kumar[3]

[1] Motorola India Private Limited
C.V. Raman Nagar
Bangalore 560093, India
j.chakraborty@motorola.com
[2] Computer Science and Automation
Indian Institute of Science
Bangalore 560012
India
deepakd@csa.iisc.ernet.in
[3] Chennai Mathematical Institute
H1 SIPCOT IT Park
Siruseri 603103, India
kumar@cmi.ac.in

Abstract. We give a detailed construction of a finite-state transition system for a com-connected Message Sequence Graph. Though this result is well-known in the literature and forms the basis for the solution to several analysis and verification problems concerning MSG specifications, the constructions given in the literature are either not amenable to implementation, or imprecise, or simply incorrect. In contrast we give a detailed construction along with a proof of its correctness. Our transition system is amenable to implementation, and can also be used for a bounded analysis of general (not necessarily com-connected) MSG specifications.

1 Introduction

Message Sequence Chart (MSC) based specifications are a popular model of early system design, whose use is particularly widespread in the telecom and software industry. A message sequence chart describes a finite sequence, or more accurately a partially ordered sequence, of message exchanges between agents in the system. These are typically "scenarios" that a system user and developer alike can use to communicate and validate system requirements. Messages may be exchanged "synchronously" as in a handshake protocol, or "asynchronously" with separate send and receive events and a message channel to buffer undelivered messages. Message Sequence Graphs (MSG's), also sometimes referred to as "high-level" MSC's, are an activity diagram-like notation that is often used to describe infinite collections of system behaviour. They are finite graphs whose vertices are labeled by MSC's, each of which represents a single logical unit of

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 549–563, 2010.

interaction. The behaviours specified by an MSG are obtained by taking a path in the MSG beginning at the initial node, and collecting the behaviours given by the "concatenation" of the MSC's associated with the nodes along the path.

Given that MSC-based specifications provide an early encapsulation of system design, from an analysis and verification point of view there are some natural problems that one would like to address. Several of these have been considered in the literature, including detecting race conditions (differences in the "visual" ordering and "execution" ordering), timing conflicts, and confluence or "completability." We would like to focus on the following two problems:

1. The model-checking problem [2]: Here we are given a system description in terms of an MSG, and a property in the form of a finite-state automaton describing say undesirable behaviours. We would like to check that the system does not exhibit any of the undesirable behaviours.
2. Detecting implied scenarios [13,1]: Given a description of system behaviour in terms of an MSG, there is a natural, distributed, system model induced by the MSG. This system model is "minimal" in that any distributed implementation of the system that exhibits all the behaviours specified by the given MSG, must necessarily exhibit all the behaviours in this model. However, the minimal system model may exhibit behaviours that are outside the ones specified by the MSG: these behaviours are called *implied scenarios*. We are interested in identifying such behaviours so that the system designer can be alerted (for example to the fact that the exact behaviour specified by the MSG is not realizable by a distributed implementation).

Message Sequence Chart based specifications have received a fair amount of attention from the Computer Science theory community (see [5,6] for surveys). In particular the analysis problems mentioned above have been addressed in the following works. Alur and Yannakakis [2] show that the model-checking problem for asynchronous MSG's is undecidable in general. They propose a condition on the MSG, called "com-connectedness" (essentially that all processes that take part in any loop of the MSG must communicate directly or indirectly with each other in the loop), which is sufficient to ensure that the model-checking problem is decidable. The main task is to show that in such a case the language of behaviours defined by the MSG is regular, i.e. acceptable by a finite-state transition system. However the details of the construction are not spelt out precisely, and there is no proof of correctness given. Independently in [8], Muscholl and Peled also give a construction of a finite-state automaton for a com-connected MSG (called "loop-connected" there), in order to give decision procedures for the problems of confluence and race conditions they consider. While their construction is fairly detailed, there is no accompanying proof of correctness. Furthermore both constructions in [2,8] are not amenable to an "on-the-fly" analysis as they generate behaviours that are not part of the MSG's behaviour (these behaviours would not reach a final state in the constructed automaton). Muscholl and Peled also point to the formal language theoretic result of Clerbout and Latteux [4] which shows that the "trace-closure" of the Kleene of a regular expression R is regular provided that the dependency graph of words in R is strongly-connected, from

which the result follows. Once again this construction is not conducive to implemenation as it is done in terms of regular grammars and is aimed at a general class of semi-commutative grammar systems.

The software engineering community have parallely developed several tools and methodologies for analysing MSC-based specifications. However some of these works are based on an incorrect understanding of the result concerning regularity of com-connected MSG's. We point out some of these cases, without detracting from the several other contributions made in these papers.

- In [13], Uchitel, Kramer, and Magee claim to solve the problem of detecting implied scenarios for general (not necessarily com-connected) MSG's with synchronous messaging. This is done without explicitly building a transition system for the given MSG. No complete proofs are given in the paper or the cited technical report. This claim is incorrect as the problem is in fact undecidable (i.e for general synchronous MSG's) [3].
- In [7] Muccini gives a technique for detecting implied scenarios based on identifying "augmented" behaviours in the components of the system model for a given general synchronous MSG. The technique is validated on a few examples, but the paper gives no proofs and admits that the "correctness and completeness are still under analysis."
- The thesis of Uchitel [11] gives the construction of a finite-state transition system, called there the "trace model," for a given com-connected synchronous MSG. This construction is implemented in a tool called LTSA-MSC [10], and used as a basis for analysis in [14]. However, as we show in Sec. 6, the trace-model constructed is incomplete: it does *not* accept all behaviours specified by the MSG. As a result the tool also incorrectly flags certain behaviours of the induced system model as implied scenarios.

Our aim in this paper is to give a precise and complete description of a finite-state transition system accepting exactly the behaviours specified by a given com-connected MSG G. We give a precise description of a "reduced" transition system called $\mathcal{T}'_G$ in Sec. 4 which is guaranteed to be finite-state when the given MSG is com-connected. We give a detailed proof of correctness of our construction. Once we have such a transition system, the analysis problems we mentioned earlier can be solved easily for com-connected MSG's with synchronous messages.

It is also worth pointing out that the transition system $\mathcal{T}'_G$ we describe is sound and complete for *general* MSG's (i.e. not necessarily com-connected) as well – though of course, it may not be finite-state in this case. Our construction also handles both synchronous and asynchronous messaging in the MSG's. Further, it has no ϵ-transitions (i.e. hidden or silent transitions), and has a bounded number of transitions applicable in any state. Thus, this transition system can be used to perform a bounded analysis or model-checking to check properties like "there are no property violations by behaviours of length ≤ 15" in the given MSG model.

We thus hope that the construction we give will be a basis on which the software engineering community can build more accurate tools for analysing MSC-based specifications. Due to lack of space, we give only sketches of some of the proofs. The complete proofs are available in the technical report [3].

2 Message Sequence Charts

We begin with some preliminary notions. For a finite alphabet A, we denote the set of finite words over A by A^*. The empty word is denoted ϵ. For words u and v over A, we denote the concatenation of u followed by v by $u \cdot v$ or simply uv. We write $u \preceq v$ to denote the fact that u is a prefix of v.

A *transition system* is a tuple $\mathcal{T} = (Q, A, q_0, \rightarrow)$, Q is a set of states, A is the set of labels or alphabet of the transition system, q_0 is the initial state, $\rightarrow \subseteq Q \times A \times Q$ is the labeled transition relation. A *run* of $\mathcal{T}$ from q_1 to q_2 on a word $w \in A^*$ is denoted by $q_1 \xrightarrow{w}{}^* q_2$ and defined in the standard way. The language generated by $\mathcal{T}$, denoted $L(\mathcal{T})$, is defined to be $\{w \in A^* \mid q_0 \xrightarrow{w}{}^* q \text{ for some } q \in Q\}$. For a state $q \in Q$, we denote the language generated by $\mathcal{T}$ starting at q by $L_q(T)$ and define it to be $\{w \in A^* \mid q \xrightarrow{w}{}^* r \text{ for some } r \in Q\}$.

A *message sequence chart* (*MSC*) is a tuple $M = \langle P, E, C, \lambda, B, \{<_p\}_{p \in P}\rangle$ where:

- P is a finite set of processes, E is a finite set of events, and C is a finite set of message labels.

  The set of actions of M is defined to be $\Sigma_M = P \times \{!, ?\} \times P \times C$ where $(p, !, q, m)$ signifies process p sending message m to q, while action $(p, ?, q, m)$ signifies p receiving message m from q. All actions of the form $(p, !, q, m)$ are called *send* actions, while actions of the form $(p, ?, q, m)$ are called *receive* actions.

  For a process $p \in P$, the set $\Sigma_p = \{a \in \Sigma_M \mid a = (p, !, q, m) \text{ or } a = (p, ?, q, m)\}$ where $q \in P$ and $m \in C$, is the set of all actions in which p participates.
- $\lambda : E \rightarrow \Sigma_M$ is the labeling function which maps events to actions. For a process $p \in P$, the set $E_p = \{e \mid \lambda(e) \in \Sigma_p\}$ are the events in which p participates.

  E is further partitioned into send events $S = \{e \,|\, \lambda(e) = (p, !, q, m),\ p, q \in P,\ m \in C\}$ and receive events $R = \{e \,|\, \lambda(e) = (p, ?, q, m),\ p, q \in P,\ m \in C\}$ respectively.
- $B : S \rightarrow R$ is a bijective map which maps each send event to its corresponding receive event. We require that if $\lambda(e) = (p, !, q, m)$ then $\lambda(B(e)) = (q, ?, p, m)$. We refer to B as the "matching receive" map.
- For each $p \in P$, $<_p$ is a strict total order on E_p. In addition, the matching receive map B induces a strict partial order $<_B$ on E, which says that a receive event has to be preceded by the corresponding send event, defined by $e <_B e'$ iff $B(e) = e'$.

  We define $<_M$ as the transitive closure, and $\leq_M$ as the reflexive transitive closure, of $(\bigcup_{p \in P} <_p) \cup <_B$ respectively. It is required that the relation $\leq_M$ must be a partial-order.

A *linearization* of the events in an MSC M as defined above is a sequence of events $w = e_1 e_2 \cdots e_n \in E^*$ containing all the events of E without repetitions, and respecting the partial order $\leq_M$ in the sense that for no $i < j \leq n$ do we have $e_j \leq_M e_i$. We denote the set of linearizations of M by $lin(M)$. We define the

event language of M, written $L^e(M)$, to be the set of all prefixes of linearizations of M. Thus, $L^e(M) = \{x \mid x \preceq y,\ y \in lin(M)\}$.

Following [2], we define a *cut* in an MSC M to be a subset c of the events E of M which is closed with respect to the partial order $\leq_M$: i.e. if $e \in c$ and $e' \leq_M e$, then $e' \in c$. Each prefix of a linearization of an MSC corresponds to a sequence of "incremental" cuts as described below:

Lemma 1. *Let M be an MSC with event set E. Then $w \in E^*$ is a prefix of some linearization of M if and only if there exists a sequence of cuts $\langle c_x \rangle_{x \preceq w}$ in M such that $c_\epsilon = \emptyset$, and for each $x \cdot e \preceq w$, we have $c_{x \cdot e} - c_x = \{e\}$.* □

We now define the notion of a message sequence graph. A *Message Sequence Graph* (MSG) is a vertex-labeled graph G of the form $\langle V, v_0, \Delta, \mathcal{M}, \mu \rangle$, where V is the set of vertices of the MSG, $v_0 \in V$ is the initial vertex, $\Delta \subseteq (V \times V)$ is the set of directed arcs, $\mathcal{M}$ is the set of MSC's associated with the MSG, and $\mu : V \to \mathcal{M}$ maps each vertex of G to one of the MSCs in $\mathcal{M}$. We assume that the MSC's in $\mathcal{M}$ are all over a common set of processes and labels, and also that the events across the MSC's in $\mathcal{M}$ are distinct. The set of events of G is denoted E^G, and is defined to be $\bigcup_{v \in V} E_{\mu(v)}$. We denote the set of events where process p participates by E^G_p, defined in a similar way as for a single MSC. The set of action labels for G is defined to be $\Sigma_G = \bigcup_{v \in V} \Sigma_{\mu(v)}$. Fig. 1 shows an example MSG.

A *path* in G is a sequence of vertices $v_1, \ldots, v_k$ $(k \geq 0)$ of G such that $(v_i, v_{i+1}) \in \Delta$ for each $i \in \{1, \ldots, k-1\}$. An *initial path* in G is a path beginning at v_0. We will use the convention that α, β, etc. denote paths in G, and u, v etc. denote vertices in G.

Let π be a non-empty path in an MSG G over a common set of processes P and labels C. For each vertex v in G, let each MSC $\mu(v)$ be $\langle P_v, E_v, C, \lambda_v, B_v, \{<^v_p\}_{p \in P} \rangle$. We define the *(weak) concatenation* of the MSCs in the path π to be the MSC $M_\pi = \langle P, E_\pi, C, \lambda_\pi, B_\pi, \{<^\pi_p\}_{p \in P} \rangle$ where:

- $E_\pi = \bigcup_{\rho v \preceq \pi} (E_v \times \{\rho v\})$.
- For each $\rho v \preceq \pi$, we define $\lambda_\pi(e, \rho v) = \lambda_v(e)$.
- For each $\rho v \preceq \pi$, and for all send events $e \in E_v$, we define $B_\pi(e, \rho v) = (B_v(e), \rho v)$.
- For each $p \in P$, $<^\pi_p$ is given as follows: Let $\rho v \preceq \pi$ and $\rho' v' \preceq \pi$ and $e \in E_v$ and $e' \in E_{v'}$. Then $(e, \rho v) <^\pi_p (e', \rho' v')$ iff e and e' are p-events and either $\rho v \prec \rho' v'$ or $\rho v = \rho' v'$ and $e <^v_p e'$.

The set of linearizations of G, denoted by $lin(G)$, is defined to be

$$\{e_1 \cdots e_n \mid \pi \text{ initial path in } G, (e_1, \rho_1) \cdots (e_n, \rho_n) \in lin(M_\pi)\}.$$

The *event language* of G, denoted $L^e(G)$, is defined to be $\{u \mid u \preceq v \text{ and } v \in lin(G)\}$. The *action language* of G is denoted $L^a(G)$, and defined to be

$$\{\lambda_{v_1}(e_1) \cdots \lambda_{v_n}(e_n) \mid e_1 \cdots e_n \in L^e(G),\ \text{each } e_i \in E_{v_i}\}.$$

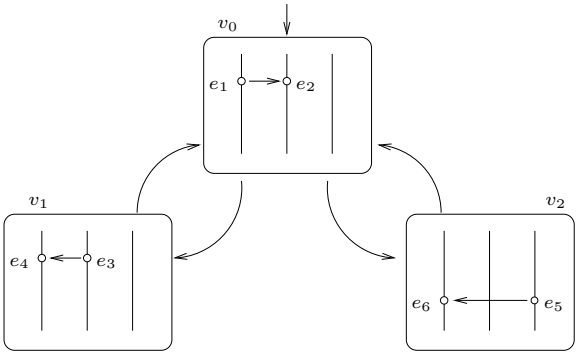

Fig. 1. An example MSG G_1

Finally, for an MSC $M = \langle P, E, C, \lambda, B, \{<_p\}_{p \in P} \rangle$, the *communication graph* of M is the directed graph on the set of processes P of M, where we have an edge (p, q) iff process p sends a message to process q in M (i.e. there exists an event $e \in E_p$ and an event e' in E_q, with $B(e) = e'$). The MSC M is said to be *com-connected* [2] if the communication graph of M contains at most one non-trivial strongly connected component and isolated nodes corresponding to processes not reacting in M. We say an MSG G is *com-connected* if for every loop, i.e. a path of the form $u\alpha u$, in G, the communication graph of $M_{u\alpha}$ is com-connected. We illustrate this using the MSG G_1 in Fig. 1. The communication graph for the loop $v_0 v_1 v_0$ is com-connected, whereas the loop $v_0 v_2 v_0$ is not com-connected. Thus the MSG G_1 is not com-connected.

Our aim in this paper is to provide a constructive proof of the following claim:

Theorem 1 ([2,8]). *Let G be a com-connected MSG. Then $L^e(G)$ is regular (i.e. it can be generated by a finite-state transition system).*

We will prove this claim in Sec. 5. In Sec 3 we first show that for a general (not necessarily com-connected) MSG G we can associate a (possibly infinite state) transition system $\mathcal{T}_G$ which generates precisely the language $L^e(G)$. In Sec 4 we give a couple of rules by which we can reduce the state space of $\mathcal{T}_G$, while preserving its language, to obtain a transition system $\mathcal{T}'_G$. Finally in Sec. 5 we show that when the given MSG is com-connected, this reduced transition system $\mathcal{T}'_G$ will indeed have a finite number of states.

3 Transition System for an MSG

In this section we show how we can associate a transition system $\mathcal{T}_G$ with a given *general* MSG G, which generates exactly the same event language as G. This transition system will, in general, have an infinite number of states. For the rest of this section we fix an MSG $G = \langle V, v_0, \Delta, \mathcal{M}, \mu \rangle$.

We begin with some preliminary notions. We define a *configuration* of G to be a pair of the form (π, c) where π is a path in G and c is a cut in M_π. Configurations will play the role of states in $\mathcal{T}_G$. We can view a configuration (π, c) as a snapshot of the events each process has completed in its process line in the MSC M_π. Each process p can be viewed to be positioned at last event it has performed.

Next we introduce some notation regarding insertion and deletion in paths and cuts in G. For a path $\pi = \alpha\beta$ in G, by $\pi + \alpha/\gamma$ we denote the sequence of nodes $\alpha\gamma\beta$ in G. We note that $\pi + \alpha/\gamma$ may not be a path in G. Similarly, if $\pi = \alpha\beta\gamma$ is a path in G, then we define $\pi - \alpha/\beta$ to mean the sequence of nodes $\alpha\gamma$ in G.

Let (π, c) be a configuration of G, and let $\pi = \alpha\beta$. Then we define the set of events corresponding to c in $\pi + \alpha/\gamma$, denoted $c + \alpha/\gamma$, to be

$$\{(e, \rho) \in c \mid \rho \preceq \alpha\} \cup \{(e, \alpha\gamma\rho) \mid (e, \alpha\rho) \in c \text{ and } \rho \neq \epsilon\}.$$

Once again, it is not necessary that $c + \alpha/\gamma$ is a cut in $M_{\alpha\gamma\beta}$.

Similarly, if (π, c) is a configuration of G, and $\pi = \alpha\beta\gamma$, we define the set of events corresponding to c in $\pi - \alpha/\beta$, denoted $c - \alpha/\beta$ to be

$$\{(e, \rho) \in c \mid \rho \preceq \alpha\} \cup \{(e, \alpha\rho) \mid (e, \alpha\beta\rho) \in c,\ \rho \neq \epsilon\}.$$

Let (π, c) be a configuration of G, with $\pi = \alpha\beta\gamma$. We say that β is *completely traversed* in c if all the events in β are included in c, and all processes which react in β have their maximal event in c located in γ. Formally, we represent this as a predicate $Ex(\pi, \alpha, \beta, c)$ which is true iff the conditions below are true:

- $E_{\alpha\beta} - E_\alpha \subseteq c$, and
- For each $p \in P$, if $(e, \alpha\rho v) \in c$ with $e \in E_p$ and $\rho v \preceq \beta$, then there exists $e' \in E_p$ and $\rho' v' \preceq \gamma$ such that $(e', \alpha\beta\rho' v') \in c$.

Similarly, we say β is *completely unexecuted* in c if none of the events in β are included in c: i.e. $(E_{\alpha\beta} - E_\alpha) \cap c = \emptyset$.

Consider a configuration (π, c) of G. We now want to define a way of cutting out loops in π which are completely unexecuted in c. We say that $\alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$ with $n \geq 0$ is an *unexecuted loop decomposition* of π wrt c, if the following holds: $\pi = \alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$; each $\alpha_i u_i$ has no completely unexecuted loops; each $\beta_i u_i$ is completely unexecuted; and γ does not have any completely unexecuted loops. It is not difficult to see that if we remove an unexecuted loop from a configuration, we get another valid configuration. The unexecuted loop-free configuration of (π, c) corresponding to the decomposition $\alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$ is obtained by cutting out each of the $\beta_i u_i$'s, and is defined to be:

$$(\alpha_1 u_1 \cdots \alpha_n u_n \gamma, (\cdots (c - \alpha_1 u_1/\beta_1 u_1) \cdots) \ -\ \alpha_1 u_1 \cdots \alpha_n u_n/\beta_n u_n).$$

There can be several unexecuted loop decompositions for a given configuration, and hence also several unexecuted loop-free configurations. We denote by $\lfloor (\pi, c) \rfloor_{ue}$ the set of all unexecuted loop-free configurations of (π, c).

We also define a (unique) *left-most maximal* unexecuted loop decomposition of a configuration (π, c). We define this to be $\alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$ with $n \geq 0$ where:

- $\pi = \alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$
- Each $\beta_i u_i$ is the left-most unexecuted loop in the segment $\alpha_i u_i \beta_i u_i \cdots \beta_n u_n \gamma$. That is, for each $\tau_1 v \tau_2$ such that $\tau_1 v \tau_2 = \alpha_i$, there is no $\tau_3 v \preceq u_i \beta_i u_i \cdots \alpha_n u_n \beta_n u_n \gamma$ such that $\tau_2 \tau_3 v$ is completely unexecuted.
- Each $\beta_i u_i$ is maximal: That is, no prefix τu_i of $\alpha_{i+1} u_{i+1} \beta_{i+1} u_{i+1} \cdots \alpha_n u_n \beta_n u_n \gamma$ is completely unexecuted.
- γ does not have any completely unexecuted loops.

Each $\alpha_i u_i$ marks the beginning of the i-th maximal loop which is completely unexecuted in c, starting from the left of π. It is easy to see that this decomposition is unique. We define the (unique) *left-most maximal* unexecuted loop configuration induced by (π, c), denoted $\lfloor (\pi, c) \rfloor_{lue}$, to be:

$$(\alpha_1 u_1 \cdots \alpha_n u_n \gamma, (\cdots (c - \alpha_1 u_1 / \beta_1 u_1) \cdots) \ - \ \alpha_1 u_1 \cdots \alpha_n u_n / \beta_n u_n).$$

We are now in a position to describe a transition system corresponding to the given MSG G. We define $\mathcal{T}_G = (Q, E_G, q_0, \rightarrow)$ where: Q is the set of configurations of G; E_G is the set of events of G as defined in the last section; the initial state is $q_0 = (\epsilon, \emptyset)$; and the transition relation $\rightarrow$ is given by the following rules:

(T1) $(\pi, c) \xrightarrow{e} (\pi, c')$ provided $c' = c \cup \{(e, \rho)\}$ for some $\rho \preceq \pi$ and $(e, \rho) \notin c$.

(T2) $(\pi, c) \xrightarrow{e} (\pi \rho v, c')$ provided $c' = c \cup \{(e, \pi \rho v)\}$ and ρv is loop-free. Also, if $\pi = \epsilon$ then ρv has to be an initial path in G.

(T3) $(\pi_1 u \pi_3, c) \xrightarrow{e} \lfloor (\pi_1 u \pi_2 u \pi_3, c') \rfloor_{lue}$ provided there exists a non-empty and loop-free α and a loop-free β such that $\pi_2 u = \alpha\beta$ and $c + \pi_1 u / \pi_2 u$ is a cut in $M_{\pi_1 u \pi_2 u \pi_3}$ and c' is $(c + \pi_1 u / \pi_2 u) \cup \{(e, \pi_1 u \alpha)\}$.

We illustrate these transition rules in Fig. 2. We note that all configurations (π, c) of $\mathcal{T}_G$ reachable from the initial state, satisfy that properties that they always have a process in the last node of π, and also that they are always unexecuted loop free.

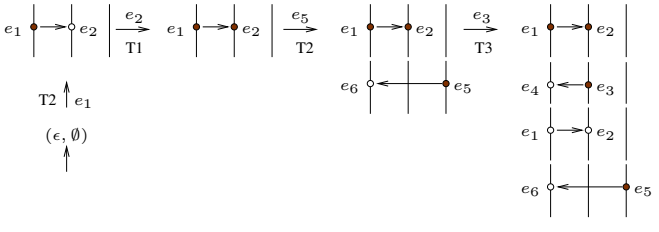

Fig. 2. Initial transitions in $\mathcal{T}_G$ for MSG G_1.

We now sketch a proof of the correctness of our construction of $\mathcal{T}_G$, by showing that it accepts exactly the language $L^e(G)$. We first show that $\mathcal{T}_G$ is "complete" in the sense that it accepts all event sequences in $L^e(G)$. Let $w \in L^e(G)$ with $w = e_1 e_2 \cdots e_n$. Then we know by Lemma 1 that there is a sequence of incremental cuts $c_0, c_1, \cdots, c_n$ in M_π for some initial path π in G, such that $c_0 = \emptyset$ and for each $i \in \{0, \ldots, n-1\}$, $c_{i+1} - c_i = \{(e_{i+1}, \rho_{i+1})\}$ for some $\rho_{i+1} \preceq \pi$. For each $i \in \{0, \ldots, n\}$, let θ_i be $\max\{\rho_j \mid j \leq i\}$. We claim that w has a run

$$(\pi_0, d_0) \xrightarrow{e_1} (\pi_1, d_1) \xrightarrow{e_2} \cdots \xrightarrow{e_n} (\pi_n, d_n)$$

in $\mathcal{T}_G$, where $\pi_0 = \epsilon$, $d_0 = \emptyset$, and for all $j \in \{0, \ldots, n\}$, $(\pi_j, d_j) \in \lfloor (\theta_j, c_j) \rfloor_{ue}$.

We do this by showing, using induction on i, that for each $i \in \{0, \ldots, n\}$ we can produce a run

$$(\pi_0, d_0) \xrightarrow{e_1} (\pi_1, d_1) \xrightarrow{e_2} \cdots \xrightarrow{e_i} (\pi_i, d_i)$$

in $\mathcal{T}_G$ such that $(\pi_0, d_0) = (\epsilon, \emptyset)$, and for each $j \in \{1, \ldots, i\}$, $(\pi_j, d_j) \in \lfloor (\theta_j, c_j) \rfloor_{ue}$. The proof is fairly routine and the complete details can be found in [3].

We now argue the "soundness" of $\mathcal{T}_G$, by showing that if $w \in L(\mathcal{T}_G)$ then $w \in L^e(G)$. Let $w \in L(\mathcal{T}_G)$ with $w = e_1 e_2 \cdots e_n$ such that $(\pi_0, c_0) \xrightarrow{e_1} (\pi_1, c_1) \cdots \xrightarrow{e_n} (\pi_n, c_n)$ is a run in $\mathcal{T}_G$. We note that each π_i must be an initial path in G. We claim that w will produce a sequence of incremental cuts $c'_0, c'_1, \ldots, c'_n$ in M_{π_n} where $c'_0 = \emptyset$, $c'_n = c_n$, and for each $i \in \{0, \ldots, n-1\}$, $c'_{i+1} - c'_i = \{(e_{i+1}, \rho_{i+1})\}$ for some $\rho_{i+1} \preceq \pi_n$. We prove this by induction on length of w, and again the details can be found in [3].

4 Reducing $\mathcal{T}_G$

In this section our aim is to show that the state-space of the transition system $\mathcal{T}_G$ can be reduced, by observing that we can remove fully traversed prefixes and loops from configurations without affecting the language generated by the transition system. To do this it will be convenient to make use of the notion of bisimilarity and some simple results concerning it.

Let $\mathcal{T} = (Q, A, q_0, \rightarrow)$ be a transition system and let $\mathcal{R}$ be a bisimulation relation on $\mathcal{T}$. We represent the reflexive transitive closure of $\mathcal{R}$ as $\mathcal{R}^*$ and clearly, it is also a bisimilar relation.

The lemma below shows how we can reduce the state space of a transition system using a bisimulation relation on it.

Lemma 2. *Let $T = (Q, A, q_0, \rightarrow)$ be a transition system, and let $\mathcal{R}$ be bisimulation relation on $\mathcal{T}$. Consider a transition system $\mathcal{T}' = (Q', A, q_0, \Rightarrow)$ where $Q' \subseteq Q$, and $\Rightarrow$ satisfies the following conditions for any $q' \in Q$:*

- *whenever $q' \xrightarrow{a} r$ in $\mathcal{T}$, there exists a state $r' \in Q'$ such that $q' \stackrel{a}{\Rightarrow} r'$ in $\mathcal{T}'$, and $(r, r') \in \mathcal{R}$.*
- *whenever $q' \stackrel{a}{\Rightarrow} r'$ in $\mathcal{T}'$, there exists $r \in Q$ such that $q' \xrightarrow{a} r$ in $\mathcal{T}$ and $(r, r') \in \mathcal{R}$.*

Then $L(\mathcal{T}) = L(\mathcal{T}')$. □

We return now to our transition system $\mathcal{T}_G$ corresponding to the given MSG G. Consider the relation $\rightsquigarrow_1$ on the states of $\mathcal{T}_G$ defined below, which relates a configuration with the one obtained from it by deleting a fully traversed prefix. We define $\rightsquigarrow_1$ as follows: $(\alpha\beta, c) \rightsquigarrow_1 (\beta, c')$ provided

- α is non-empty,
- $Ex(\alpha\beta, \epsilon, \alpha, c)$ holds, and
- $c' = c - \epsilon/\alpha$.

The relation $\rightsquigarrow_2$ relates configurations of $\mathcal{T}_G$ based on deleting completely traversed loops. We have $(\alpha u\beta u\gamma, c) \rightsquigarrow_2 (\alpha u\gamma, c')$ provided

- $Ex(\alpha u\beta u\gamma, \alpha, u\beta, c)$ holds, and
- $c' = c - \alpha/u\beta$.

Lemma 3. *The relations $\rightsquigarrow_1$ and $\rightsquigarrow_2$ are bimulation relations on the transition system $\mathcal{T}_G$.* □

We will now reduce the state space of $\mathcal{T}_G$ using the bisimulation relations defined above.

For a configuration (π, c) of G, we define its *maximal reducible decomposition* to be $\alpha\alpha_1 u_1\beta_1 u_1 \cdots \alpha_n u_n\beta_n u_n\gamma$ with $n \geq 0$, where

- $\pi = \alpha\alpha_1 u_1\beta_1 u_1 \cdots \alpha_n u_n\beta_n u_n\gamma$.
- α is the maximal prefix of π which is completely traversed.
- Each $u_i\beta_i$ is the leftmost completely traversed loop in the segment $\alpha_i u_i\beta_i u_i \cdots \alpha_n u_n\beta_n u_n\gamma$.
- Each $u_i\beta_i$ is maximal.
- γ does not have any completely traversed loop.

The maximal reducible decomposition of (π, c) can be seen to be unique.

Let $\alpha\alpha_1 u_1\beta_1 u_1 \cdots \alpha_n u_n\beta_n u_n\gamma$ be the maximal reducible decomposition of (π, c). Then we define the *reduced form* of (π, c), denoted $\lfloor(\pi, c)\rfloor$, to be the configuration

$$(\alpha_1 u_1 \cdots \alpha_n u_n\gamma, \\ (\cdots((c - \epsilon/\alpha) - \alpha_1 u_1\beta_1) - \cdots) - \alpha_1 u_1 \cdots \alpha_n u_n/u_n\beta_n).$$

Clearly, the reduced form of a configuration does not contain any completely traversed prefix or loops.

We now define the reduced transition system corresponding to the MSG G, denoted $\mathcal{T}'_G$, as follows: $\mathcal{T}'_G = (Q', E_G, q_0, \Rightarrow)$ where

- $Q' = \{\lfloor(\pi, c)\rfloor \mid (\pi, c)$ a configuration of $G\}$.
- E_G is the set of events of G.
- $q_0 = (\epsilon, \emptyset)$.
- Let $q', r' \in Q'$. Then, $q' \stackrel{e}{\Rightarrow} r'$ iff there exists $r \in Q$ such that $q' \stackrel{e}{\rightarrow} r$ in $\mathcal{T}_G$, and $r' = \lfloor r\rfloor$.

We note that the reduced form of the initial configuration $(\epsilon, \emptyset)$ is itself.

We now want to argue that $\mathcal{T}'_G$ generates the same language as $\mathcal{T}_G$. We have already shown that $\rightsquigarrow_1$ and $\rightsquigarrow_2$ are bisimulation relations on $\mathcal{T}_G$. Hence so is the relation $(\rightsquigarrow_1 \cup \rightsquigarrow_2)^*$. Further, for any configuration (π, c), it is clear that $((\pi, c), \lfloor(\pi, c)\rfloor) \in (\rightsquigarrow_1 \cup \rightsquigarrow_2)^*$. Finally, it is immediate to check that T'_G satisfies the conditions of Lemma 2 with respect to $\mathcal{T}_G$ and the bisimulation relation $\mathcal{R} = (\rightsquigarrow_1 \cup \rightsquigarrow_2)^*$. Hence by Lemma 2 we can conclude that $L(\mathcal{T}'_G) = L(\mathcal{T}_G)$.

We summarise the result of this section in the following theorem:

Theorem 2. *For any MSG G, the reduced transition system $\mathcal{T}'_G$ generates precisely the language $L^e(G)$.*

We note that though $\mathcal{T}'_G$ has fewer states than $\mathcal{T}_G$ in general, it may still have an infinite number of states. Consider the MSG G_2 in Fig. 3. The configurations of the form $(v_0^n, \{(e_1, v_0), (e_1, v_0^2), \cdots, (e_1, v_0^n)\})$ where $n > 0$ do not have any unexecuted loops or completely traversed loops. So, for each n, these are distinct states in $\mathcal{T}'_{G_2}$, and thus $\mathcal{T}'_{G_2}$ has an infinite number of states. In the next section we show that this is not possible for a com-connected MSG.

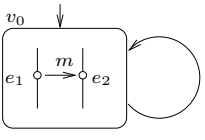

Fig. 3. MSG G_2 with an infinite-state $\mathcal{T}'_{G_2}$

5 Regularity of Com-Connected MSG's

In this section our aim is to supply a proof of Theorem 1. We begin with an observation from [2].

Lemma 4 ([2]). *Let G be a com-connected MSG. Consider a configuration of the form $(\alpha u \beta u \gamma, c)$. Then, either $u\beta$ is completely unexecuted, or $u\beta$ is completely traversed, or there is a process whose last executed event and next unexecuted event are both in $u\beta$.*

Proof. Let us assume the contrary. Then among the processes that take part in an event in the nodes in $u\beta$, there must be processes p and q such that *none* of the p events in $u\beta$ are executed, and *all* of the q events in $u\beta$ are executed. As G is com-connected, we must have a path in the communication graph of $M_{u\beta}$ from p to q. Let this path be $p = r_0 \rightarrow r_1 \rightarrow \cdots \rightarrow r_n = q$ with $n \geq 1$. Then, since q has completed all its events in $u\beta$, it must have also received a message from r_{n-1}. Thus r_{n-1} has taken part in $u\beta$, since it must have sent the message received by q. Now either r_{n-1} has another event to take part in $u\beta$, and we are done; or, it has completed all its events in $u\beta$ and in particular has received a message from r_{n-2}. We repeat this argument till we either find a process r_i

which has participated in $u\beta$ and has an unexecuted event in $u\beta$, or reach a contradiction that process p has participated in an event in $u\beta$. This completes the proof of the lemma. □

Let us now consider the reduced transition system $\mathcal{T}'_G$ for a com-connected MSG G. The states in $\mathcal{T}'_G$ are all in reduced form, and hence have no completely unexecuted loops or completely traversed loops. It follows that in any loop in a state (π, c) of $\mathcal{T}'_G$, there must be at least one process positioned in a node in that loop. Thus no node can occur more than $k+1$ times in π, where k is the number of processes. In fact, because of the property of the reachable configurations of $\mathcal{T}_G$ that there is always a process positioned in the last node of the path, the bound of $k+1$ can be reduced to k. This bounds the length of π by mk, where m is the number of nodes in G. We also get an upper bound of $m^{mk}(mnk)^k$ on the number of states in $\mathcal{T}'_G$, where n is the maximum number of events in any MSC associated with a node in G.

This completes the proof of Theorem 1.

6 Synchronous MSG's

In this section we consider MSC's with synchronous (or "handshake") messages. These synchronous MSC's are defined in a similar manner to the (asynchronous) MSC's of Sec. 2, except that sends and receives are no longer distinct events, but are represented as a single message exchange event. We call MSG's whose nodes are labelled by synchronous MSC's *synchronous* MSG's. The language of event sequences generated by a synchronous MSG is defined in a similar manner to that of (asynchronous) MSG's in Sec. 2. Finally, we say a synchronous MSG G is *com-connected* iff for every loop $u\beta u$ in G, the communication graph of $M_{u\beta}$ has at most one non-trivial connected component. Fig. 4 shows a com-connected synchronous MSG G_4.

The construction of the transition systems $\mathcal{T}_G$ and $\mathcal{T}'_G$ for an asynchronous MSG G, is based purely on the partial order induced by a path in the MSG. Hence it readily applies to synchronous MSG's as well. Further, the proof of

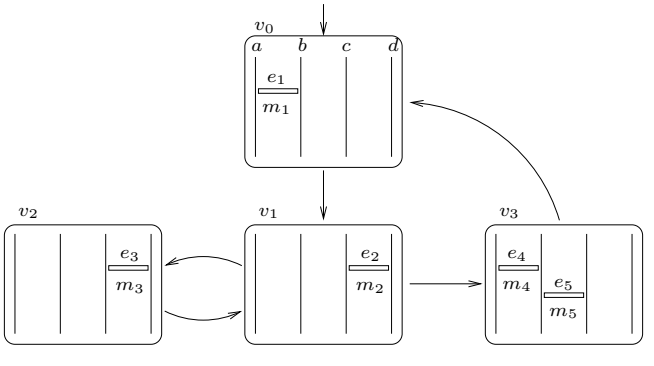

Fig. 4. MSG G_4 with synchronous messages

correctness of $\mathcal{T}_G$ and $\mathcal{T}'_G$ also makes use of certain properties of this partial order, and these properties are easily seen to be satisfied by synchronous MSG's as well. Hence we can conclude:

Theorem 3. *Let G be a synchronous MSG, and let $\mathcal{T}_G$ and $\mathcal{T}'_G$ be the transition systems defined in Sec. 3 and 4 respectively. Then both transition systems generate exactly the language $L^e(G)$.*

Further, Lemma 4 can also be seen to apply for synchronous MSG's, and hence we have:

Theorem 4. *Let G be a com-connected synchronous MSG. Then $L^e(G)$ is regular.*

The state transition diagram of the transition system T'_{G_4} for the MSG G_4 of Fig. 4 can be found in [3].

The MSG of Fig. 4 is useful for pointing out the incompleteness of the transition system constructed by Uchitel in [11]. He constructs a transition system, called the "trace model," which is meant to generate the language of event sequences for a given com-connected synchronous MSG. The construction uses a "coordinator" component which keeps track of the current path in the MSG traced out by processes, and ensures that other processes follow this path consistently. The transition rules are similar to ours, except that there is *no rule* for inserting a path within the current node list. This ommission essentially leads to the trace model being incomplete.

The trace model generated by the LTSA-MSC tool [10,12] on the MSG G_4 as input is shown in Fig. 5. The labels of the trace model basically corresponds

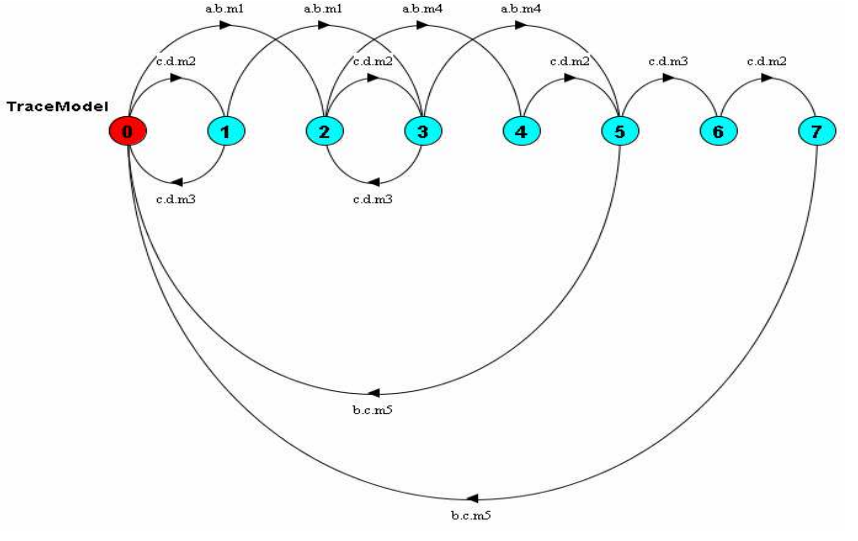

Fig. 5. Transition system generated by LTSA-MSC tool for MSG G_4

to the events of the MSG. For example, the transition from state 0 to state 2 happens with the synchronous exchange of m_1 between process a and b, which is the event e_1 in the MSG G_4. The event sequence $e_1e_4e_2e_3e_2e_3$ is not allowed by the trace model, while it is clearly a legal behaviour of the MSG G_4, being a prefix of a linearisation of M_π, where π is the path $v_0v_1v_2v_1v_2v_1v_3$.

7 Detecting Implied Scenarios

We now sketch how our construction of $\mathcal{T}'_G$ can be used to detect "implied scenarios" in a com-connected synchronous MSG.

A synchronous MSG G induces a natural "minimal" distributed finite-state system model (called the "architecture model" in [11]). Each process has a component obtained by keeping track of which events it can participate in next. The system model induced by the components, can be viewed as the "synchronised product" of the components for each process, where a pair of components are required to simultaneously execute events that are common to their process lines. The system model, which we denote $\mathcal{S}_G$, can be seen to include *all* the behaviours in $L^e(G)$ for a given synchronous MSG G. The problem is that it may sometimes generate a strict superset of the behaviours in $L^e(G)$, and these behaviours are what are refered to as "implied scenarios." Thus an event sequence $w \in E_G^*$ is called an *implied scenario* if $w \in L(\mathcal{S}_G) - L^e(G)$. We refer the reader to [13,11] for some illustrative examples of implied scenarios.

For a com-connected synchronous MSG G implied scenarios can be easily detected from $\mathcal{T}'_G$ and $\mathcal{S}_G$ using standard automata-theoretic techniques. The MSG G_4 can be seen to have no implied scenarios [3]. But, since the trace model generated by the LTSA-MSC tool [10,11,12] is incomplete, it reports $e_1e_4e_2e_3e_2e_3$ as one of the several implied scenarios for this MSG.

8 Conclusion

In this paper we have given a precise construction of a transition system for a given MSG specification, which accepts exactly the set of behaviours specified by the MSG. When the given MSG specification is com-connected, the transition system we give is guaranteed to be finite-state and can thus be used a basis for building sound and complete tools for analysing properties of these specifications. Our transition system is also suitable for analysing MSG specifications in a bounded fashion, even when the given MSG is not com-connected.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Trans. Software Eng. 29(7), 623–633 (2003)
2. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)

3. Chakraborty, J., D'Souza, D., Narayan Kumar, K.: Analysing Message Sequence Graph Specifications, Technical Report IISc-CSA-TR-2009-1, CSA Department, IISc (2009), http://www.csa.iisc.ernet.in/TR/2009/1/
4. Clerbout, M., Latteux, M.: Semi-commutations. Inf. Comput. 73(1), 59–74 (1987)
5. Genest, B., Muscholl, A., Peled, D.: Message sequence charts. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 537–558. Springer, Heidelberg (2004)
6. Henriksen, J.G., Mukund, M., Kumar, K.N., Sohoni, M.A., Thiagarajan, P.S.: A theory of regular msc languages. Inf. Comput. 202(1), 1–38 (2005)
7. Muccini, H.: Detecting implied scenarios analyzing non-local branching choices. In: Pezzé, M. (ed.) FASE 2003. LNCS, vol. 2621, pp. 372–386. Springer, Heidelberg (2003)
8. Muscholl, A., Peled, D.: Message sequence graphs and decision problems on mazurkiewicz traces. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 81–91. Springer, Heidelberg (1999)
9. Muscholl, A., Petersen, H.: A note on the commutative closure of star-free languages. Inf. Process. Lett. 57(2), 71–74 (1996)
10. Uchitel, S.: LTSA-MSC tool (2001), http://www.doc.ic.ac.uk/~su2/Synthesis/
11. Uchitel, S.: Incremental Elaboration of Scenario Based Specifications and Behavior Models Using Implied Scenarios. PhD thesis, Imperial College (2003)
12. Uchitel, S., Chatley, R., Kramer, J., Magee, J.: LTSA-MSC: Tool support for behaviour model elaboration using implied scenarios. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 597–601. Springer, Heidelberg (2003)
13. Uchitel, S., Kramer, J., Magee, J.: Detecting implied scenarios in message sequence chart specifications. In: ESEC / SIGSOFT FSE, pp. 74–82 (2001)
14. Uchitel, S., Kramer, J., Magee, J.: Incremental Elaboration of Scenario-Based Specifications and Behavior Models Using Implied Scenarios. ACM Transactions on Software Engineering and Methodology 13(1), 37–85 (2004)