Analysing Message Sequence Graph Specifications

Joy Chakraborty\textsuperscript{1}, Deepak D’Souza\textsuperscript{2}, and K. Narayan Kumar\textsuperscript{3}

\textsuperscript{1} Motorola India Private Limited
C.V. Raman Nagar
Bangalore 560093, India
j.chakraborty@motorola.com

\textsuperscript{2} Computer Science and Automation
Indian Institute of Science
Bangalore 560012
India
deeakd@csa.iisc.ernet.in

\textsuperscript{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
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