Formal Verification of STATEMATE-Statecharts

1 Introduction

During the software development process it is important to use powerful techniques for proving the expected behavior of the system and hence avoiding failures in real applications later on. Therefore, an effective means to increase confidence in the development process is to use analysis techniques based on the formal descriptions used in early phases (requirements specification).

The STATEMATE tool [HLN+90] is widely used for developing industrial software systems, e.g. avionic systems, automotive systems, computer operating systems etc. This paper concentrates on the STATEMATE-statecharts [Har87] as a description language for dynamic system behavior. Statecharts is a highly structured language extending conventional state transition diagrams by hierarchy, concurrency and communication.

Within the STATEMATE tool users specify and validate these descriptions by a simulation tool which is driven interactively. Simulation is bounded in the way that it only shows some parts of a system and cannot provide evidence of requirements being satisfied over the whole life time of a system. The industrial practice of Statemate naturally leads to the point where developers require means for ensuring properties, e.g. the safety or security properties in avionic systems. Unfortunately there are no practical tools implemented for such verification tasks.

This paper introduces an approach which provides software engineers with fully automatic verification. Starting from a first system description of requirements the developer describes the dynamic behavior of the system in statecharts and formulates the requirements either as statecharts or as so-called temporal properties. Afterwards he can check automatically whether the system satisfies a requirement or not. Fig 1 shows our framework for translating statecharts and properties into a verification tool.

At first we use the timeless, synchronous model of Statemate as underlying formal semantics of statecharts [HN95]. The translation is defined over a subset of the statechart language which represents a fully abstraction from data. Secondly we chose CSP [Hoa78, Hoa85] as underlying formal semantics because there exists tool support [Sca95], in particular the process-algebraic verification tool FDR (Failure Divergence Refinement) [For97]. FDR is a commercial tool which stands under continuous development. The verification task is done by using the implemented refinement calculus of CSP which will be explained later on.

In section 2 we introduce the translation of STATEMATE statecharts into CSP processes in detail. This is the premise for the analysis of statechart properties.
shown in section 3. We conclude with an overview of the current work and give an idea of a verification pool consisting several verification tools for different purposes.

2 Statecharts in CSP

The model of a statechart specification in CSP is an embedding of the syntax and the step semantics of STATEMATE. The model is divided into three separate components:

- **statechart term**
  A statechart is represented as a CSP process. Different processes for sequential, parallel, and hierarchical statecharts are specified.

- **environment**
  For the representation of the STATEMATE environment two processes are defined for every statechart event which provide (pre-process) and receive (post-process) events during a step.

- **step semantical process** The semantics of a STATEMATE-step is modeled by a global step process called global scheduler.

2.1 Statecharts Terms

The translation of statecharts can be carried out compositionally by restricting the original language, i.e. you can not use interlevel-transitions. Separate parts of the statechart structure are translated separately too. XOR-terms and their parallel as well as hierarchical compositions are introduced gradually. A modification of the well known mutual exclusion problem is used to illustrate the translation.

Fig. 2 shows such a modification of the Mutual Exclusion. Two users, represented by two XOR-terms User1 and User2, compete for the use of a resource which is represented by the XOR-chart Ress. User1 can request the use of a resource by req1, which will block User2 if User1 is accepted by Ress. This means User2 can