Introduction

1 The Need for Formal Methods

One can often hear the expression "with computer precision" which is usually used to describe a flawless impeccable execution of some task. Unfortunately, the computers do not always work correctly. Moreover, the errors of software and hardware systems applied in safety-critical systems like nuclear power plants, medical equipments, highway and air traffic control, railways, electronic commerce, can be potentially disastrous. Such grim predictions are supported by famous failures, like the Ariane 5 space rocket disaster or the Pentium bug, costing billions of dollars. But as information technology penetrates in our daily life, there will be even more prosaic reasons why the errors will not be tolerated. Just imagine a football aficionado whose TV crashes during the penalty shutouts because of the embedded software, or you being cut by your intelligent shaver. There are predictions that in the future the main problem for the application of information technology will not be the lack of raw computational power, but our inability to develop complex systems with sufficient confidence in their correctness.

The traditional engineering techniques for validation, like peer review, simulation, and testing, have often proved inadequate and too expensive to avoid errors in information processing artifacts. One reason for this is that they explore only a part of the possible behavior of the system. As a result some erroneous behavior often escapes undetected.

In the last two decades, alternative formal approaches for validation have emerged. They are based on *formal methods*, i.e., on methods that are more systematic and that have solid mathematical foundations. One of the main advantages of formal methods is that they perform exhaustive exploration of all possible behaviors. In this course we concentrate on *model checking* as a formal approach for debugging and verification of hardware and software.

2 Model Checking

Roughly speaking, model checking is an automated technique that, given a model of the system and some property, checks whether the model satisfies the property. Originally suggested in the beginning of the eighties independently by Emerson and Clarke, and Quelle and Sifakis, nowadays the technique gains popularity, with several major companies developing in-house model-checking tools. Compared to the other (semi)automated formal techniques (for instance, deductive methods, like theorem provers) model checking is relatively easy to use. The specification of the model is very similar to programming and as such it does not require much of additional expertise from the user. The verification procedure is completely automated and often takes only several minutes. Another important advantage of the method is that, if the verification fails, the possible erroneous behavior of the system can be reproduced. This significantly facilitates the location and correction of the errors.

2.1 The Characteristics of the Analyzed Systems.

In this course we deal with *concurrent* systems, i.e., systems composed of components that can operate *concurrently* and *communicate* with each other. It is assumed that the (components of the) systems are *reactive*, i.e. continuously interact with their environment. Typical representatives of reactive systems are communication protocols, operating systems, process control software, and aircraft control. This is in contrast with traditional sequential programs which can be seen as data transformers with single input and single output, i.e., interacting with the environment only at their beginning and end. Concurrency is modeled by *interleaving*, i.e., it is assumed that only one component executes an action at a time and the concurrent actions are arbitrarily ordered. Considering the reactive nature of the concurrent systems that are analyzed, we are interested in the verification of *control (interaction) properties*, rather than properties that are related to data.

The Formal Framework and the Scope. Each framework for formal analysis consists of four components:

- *formal semantics* in which the system and the property are interpreted;
- *formal language* for describing *the system*;
- *formal language* for describing *the property*;
- *formal, preferably automated, techniques* with which it can be checked whether the system satisfies the property;

We use variants of *(labeled) transition systems* to give the semantics of the systems and properties. In other words, the semantics of the system is its *state space*, i.e., the set of all possible states that the system can reach and the transitions between them. The state transition systems are represented in a natural way as labeled graphs. Model checking by definition is applied on systems with *finite* state space.

We specify the properties either directly in the model, or in *linear temporal logic (LTL)* and more general *Büchi automata.*

The course is centered around the model checker *Spin* with its input language *Promela*, developed by Gerard Holzmann at Bell Laboratories.

Further, we consider model checking algorithms based on explicit *enumeration* of the state space (as opposed to the symbolic algorithms based on binary decision diagrams.

Although it is safe to conjecture that at least some of the considered techniques can be used for hardware verification, a tacit assumption throughout the course will be that we are targeting *debugging and verification of software systems.*

2.2 The State Space Explosion Problem.

Model checking requires search of the state space of the system, which usually increases exponentially with the size of the system description. As a consequence, one of the major bottlenecks in model checking is the so called *state space explosion.* The state space explosion is simply the combinatorial explosion caused by the interaction (interleaving) of the components and/or by the usage of data structures ranging over many different values. There exist numerous techniques to combat this problem, like symbolic verification, on-the-fly verification, abstraction, partial order reduction, symmetry reduction, etc. During the course we will briefly outline some of them.