Abstracts

Logic and Verification
Moshe Y. Vardi (Rice University, USA)
Mathematical logic developed as an attempt to provide formal foundations for mathematics. The success of that project can be questioned, as the logical foundations of mathematics proved to be incomplete, possibly inconsistent, and undecidable. Logic, on the other hand, proved to be highly successful in providing formal foundations for reasoning about computing systems, where it is deployed today in industrial tools. This tutorial will focus on one application of logic to verification, which is the temporal analysis of systems.

Runtime Verification
Martin Leucker (Technische Universität München, Germany)
Starting from a definition of runtime verification, we develop a taxonomy that explains the different aspects of runtime verification. We explain the core idea of runtime verification by showing how monitors can be attached to existing programs, be used to verify certain aspects of the underlying program as well as be used to guide the program execution. The main part of the presentation deals with synthesis techniques that, starting from a high level correctness specifications, derive suitable monitors automatically. We start with properties expressed in linear temporal logic (LTL), first with a semantics on finite traces and then extended to a semantics over infinite traces.

Computer-Aided Cryptographic Proofs and Designs
Gilles Barthe (University of Manchester, UK)
EasyCrypt is a tool for constructing and verifying cryptographic proofs. EasyCrypt can be used as a stand-alone application, or as a verifying back-end for cryptographic compilers. SyntheCrypt is a new tool that synthesizes public-key encryption schemes and generates proofs of security in EasyCrypt.
The presentation will outline the language-based methods that underlie the design of both tools and illustrate some of their applications.

Probabilistic model checking
Marta Kwiatkowska (University of Oxford, UK)
Probabilistic model checking is a formal verication technique for the analysis of systems that exhibit stochastic behaviour. Such behaviour occurs, for example, due to component failure or randomisation, commonly used as a symmetry breaker in distributed coordination and communication protocols. The techniques have been implemented in tools such as PRISM (www.prismmodelchecker.org) and enable a range of quantitative analyses of probabilistic models against specifications such as the worst-case probability of failure within 10 seconds or the minimum expected power consumption over all possible schedulings. This course will give an overview of probabilistic model checking discrete-time Markov chains and Markov decision processes, explaining the underlying theory and model checking algorithms for temporal logics such as PCTL and LTL. The material will be illustrated with several case studies that have been modelled and analysed in PRISM.

Static analysis by abstract interpretation of run-time errors in synchronous and multi-threaded embedded C programs
Antoine Miné (CNRS, ENS, France)
In the realm of embedded critical systems, it is crucial to guarantee the correctness of programs before they are deployed. We present semantic-based two static analyzers developed in our research group to help addressing this need: Astrée focuses on synchronous control-command software in C; its extension AstréeA analyzes multi-threaded embedded C software. These analyzers detect statically (at compile time) all the arithmetic and memory run-time errors in the program. They are based on abstract interpretation, a general theory of the approximation of program semantics that ensures soundness: the analyzers cannot miss any threat. They may however output false alarms. We show how specializing Astrée with novel abstractions allowed us to reach the zero false alarm goal (that is, a proof of absence of run-time error) on some industrial avionic applications.

Unfoldings: A Partial order Approach to Model Checking
Javier Esparza (TU München, Germany)
State space methods are the most popular approach to the automatic verification of concurrent systems. In their basic form, these methods explore the transition system associated to the concurrent system. Loosely speaking, the transition system is a graph having the reachable states of the system as nodes, and an edge from a state s to another state s' whenever the system can make a move from s to s'. In the worst case, state space methods need to explore all nodes and transitions of the transition system. The main problem of transition systems as a basis for state space methods is the well-known state-explosion problem. Imagine a concurrent system consisting of n sequential subsystems, communicating in some way, and assume further that each of these subsystems can be in one out of m possible states. The global state of the concurrent system is given by the local states of its components, and so the system may have up to m^n reachable states; in fact, this bound is already reached by the rather uninteresting system in whose components run independently of each other, without communicating at all. So very small concurrent systems may generate very large transition systems. As a consequence, naive state space methods may have huge time and space requirements even for very small and simple systems.
The unfolding method is a technique for alleviating the state-explosion problem. It uses results of the theory of true concurrency to replace transition systems by special partially ordered graphs. While these graphs contain full information about the reachable states of the system, their nodes are not reachable states themselves. In particular, the number of nodes of the graph does not grow linearly in the number of reachable states.
The goal of the course is to provide a gentle introduction to the basics of the unfolding method, and in particular to introduce an unfolding-based algorithm for model checking concurrent systems against properties specified as formulas of Linear Temporal Logic (LTL). The course is based on the book

Javier Esparza and Keijo Heljanko. Unfoldings. A Partial-Order Approach to Model

Checking. EATCS Monographs on Theoretical Computer Science. Springer, 2008.

The book is available online (for free) at http://www7.in.tum.de/ esparza/bookunf.html It is recommended to download the book and have it available during the course. Some familiarity with Petri nets will be of advantage.

Timed automata and their quantitative extensions
Kim G. Larsen (Aalborg University, Denmark)

Timed automata is by now a well-established formalism for modeling and analyzing real-time systems, including real-time controllers, communication protocols. Over the years a number of symbolic techniques have been developed for the efficient analysis of timed automata and with implementation in the tool suite UPPAAL (www.uppaal.{com,org}). The UPPAAL modeling formalism allows for the efficient analysis of safety and (time-bounded) liveness properties of networks of interacting timed automata extended with discrete variables, structured and user-defined types as well as user-defined functions. The course will give an overview of the modeling formalism of timed automata and the basic symbolic model checking algorithms. More recently the formalism of timed automata has been extended with continuous observer variables allowing for issues related to e.g energy consumption in embedded systems. The course reviews a number of results for the resulting notion of priced timed automata, including recent results on energy-bounded infinite runs in the case when energy may both be consumed as well as harvested. Most recently a stochastic semantics of (priced) timed automata has been put forward enabling the expression of performance properties such as the probability of violating a deadline or the expected energy consumption. A range of so-called highly scalable statistical model-checking techniques have been implemented in UPPAAL, allowing estimation and testing of such performance metrics to be obtained through simulation, where the simulation effort increases with the desired level of confidence. The lecture will contain demonstration of the UPPAAL tool, the new statistical model checking engine as well as several case studies that have been dealt with by the tool.

Games for Verification and Synthesis
Hugo Gimbert (LaBRI & CNRS, France)
Game playing is a powerful metaphor that fits many situations in which interaction between autonomous agents plays a central role. Numerous tasks in computer science and AI such as design, synthesis, verification, testing, query evaluation, planning, etc. can be formulated in game-theoretic terms. Viewing them abstractly as games reveals the underlying algorithmic questions, and helps to clarify the relationships between problem domains.
This talk will illustrate how games can be used in several ways in this context: as algorithmically tractable models of controllable open systems (e.g. games on graphs and stochastic games) as algorithmic tools (e.g. for mu-calculus model-checking) as well as proof tools (e.g. to prove stability under complementation of recognizable set of trees).

Verification of concurrent systems
Ahmed Bouajjani (LIAFA, Univ. Paris Diderot (Paris 7), France)
The verification of concurrent programs is a challenging problem. This is due to the huge number of orderings in which actions of different threads can be executed, and to the intricacy of the interactions between these threads (especially in presence of dynamic thread creation, recursion, etc). Basic problems such as the state reachability problem, that is relevant in checking safety properties, are undecidable in general, even when the manipulated data are in a finite domain. Therefore, restrictions either on the considered class of program models or on the class of explored behaviors during the analysis, must be considered in order to obtain decidable and/or tractable analysis problems.
In this talk, we will present program models capturing relevant classes of programs (including for instance asynchronous programs) and study the decidability and complexity of their state reachability problem. Moreover, we will present bounded analysis techniques (such as context-bounding) that are used for efficient bug detection in concurrent programs.

Safety, Dependability and Performance Analysis of Extended AADL Models
Alessandro Cimatti (IRST, Italy)
and Thomas Noll (RWTH Aachen University, Germany)
This tutorial presents a component-based modeling approach to system-software co-engineering of real-time embedded systems, in particular aerospace systems. Our method is centered around the standardized Architecture Analysis and Design Language (AADL) modeling framework. Taking the core features of AADL and its recent Error Model Annex, we have set up a modeling framework that supports a variety of system analysis and verification methods. Its major distinguishing aspects are the possibility to describe hardware and software components and its nominal operations, hybrid (and timing) aspects, as well as probabilistic faults and their propagation and recovery. Moreover, it supports dynamic (i.e., on-the-fly) reconfiguration of components and inter-component connections. The operational semantics gives a precise interpretation of specifications by providing a mapping onto networks of event-data automata. These networks are then subject to different kinds of formal analysis such as model checking, safety and dependability analysis, and performance evaluation. We demonstrate tool support realizing these analyses and report on industrial case studies that have been carried out in the context of aerospace systems. The tool is publicly available.

Software Synthesis
Ruzica Piskac (Max Planck Institute for Software Systems, Germany)
Software synthesis is a technique for automatically generating code given a specification. The goal of software synthesis is to make coding easier while increasing both the productivity of the programmer and the correctness of the produced code. In this talk we describe an approach to synthesis that relies on the use of automated reasoning and decision procedures. The complex specifications are handled by employing efficient algorithms for reasoning about the domain of the specification. We show how to generalize some decision procedures into predictable and complete synthesis procedures.

Here completeness means that the procedure is guaranteed to find code that satisfies the given specification. Moreover, code produced this way is correct by construction. The synthesis procedure also outputs preconditions on input values that guarantee the existence of the output values.
In addition, we also outline a synthesis procedure for specifications given in the form of type constraints. The procedure takes into account polymorphic type constraints as well as code behavior and derives code snippets that use given library functions. The constraints can have multiple solutions and hence more than one code snippet can be a good candidate. We use an additional weight function to rank the derived snippets. The tools implementing these synthesis algorithms are publicly available.