Introduction to the Course on Automatic System Verification: Theory and Applications

Angelo Montanari

Department of Mathematics, Computer Science, and Physics
Università degli Studi di Udine
Udine, Italy

Aim of the course

The course aims at developing techniques, methodologies, and tools for the **formal specification** and **verification** of **reactive programs/systems** (and of the environment with which they interact / that they control) by making use of **temporal logics**

Key words:

- techniques, methodologies, and tools for **formal specification** and **verification** (to check the consistency of a specification and the validity of an expected property of a program/system)
- **reactive programs/systems** vs. transformational (input/output) programs
- **temporal logic** (logical specification formalisms; tools for satisfiability and model checking; tools for the synthesis of controllers)

Reactive programs/systems

A **reactive program/sistem** is a program/system whose goal is to maintain a proper **interaction with the environment** within which it operates over time, that is, indefinitely, instead of terminating its execution returning a suitable final value

Examples of reactive programs/systems:

operating systems, concurrent and real-time programs, programs for process control; embedded programs/systems (programs whose software module is an integrated component of the complete system)

Concurrency

Concurrency is a distinctive feature of reactive programs/systems

There two different forms of concurrency (as a matter of fact, they can be reduced to a single one):

- a reactive program/system which operates concurrently with its environment
- the majority of reactive programs/systems consist of a set of processes that are concurrently executed

We will investigate **formal techniques** for the specification and the analysis of the interactions among the components of programs/systems that operate in a concurrent way

Models, languages, and algorithms

- We will describe a **computational model**
  - first, *fair transition systems* and a simple programming language for reactive programs/systems, and
  - then *automata* on infinite objects (words, trees, graphs)
- We will use **temporal logic** as a formal specification language
- We will develop **algorithms** for satisfiability and model checking

Process communication - 1

As for the programming language, we will analyze the **mechanisms** for the **communication** and the **synchronization** of concurrent processes

There are fondamental forms of communication:

- via **shared variables**

- via **message passing**

We will present a **uniform approach** to the communication among reactive programs/systems, which does not depend on the specific communication mechanism/modality one adopts

Process communication - 2

In particular, we will show how some **fundamental paradigms** of concurrent programming, such as

- **mutual exclusion**
- **producer/consumer schema**

can be programmed by means of both communication mechanisms/modalities

We will also investigate the relationships between **true concurrency** and **interleaving**

True concurrency vs. interleaving

True concurrency: **actual** (physical) concurrent execution of processes

Interleaving: concurrency modeled as the **alternation** (interleaving) of atomic actions selected once a time from the various concurrent processes (there are some analogies with the management of concurrent transations in database systems)

We will identify the **syntactic restriction**s on the considered programs and the **fairness requirements** on computations that must be imposed to allow us to assume interleaving as a **faithful model** of true concurrency

Properties of reactive programs/systems

We will analyze the most significant classes of properties of reactive programs/systems

- **safety**
- **liveness**
- **reactivity**

Safety: bad situations that must never occur (universal conditions)

Liveness: good situations that sooner or later must occur (existential conditions)

Model checking

Model checking for most common temporal logics (LTL, CTL, CTL^*, μ-calculus)

Computational complexity of model checking (the problem of state explotion)

Possible solutions: symbolic model checking, OBDD (Ordered Binary Decision Diagram) and their variants, partial order reduction

Automaton-based model checking

Some significant **model checkers** (SPIN, SMV, NuSMV), with examples

Bibliography - 1

Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer, 1992

Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems: Safety, Springer, 1995

A. Montanari, Sistemi Reattivi: Automi, Logiche e Algoritmi (note al corso).

W. Thomas, Automata on Infinite Objects, in Handbook of Theoretical Computer Science, Vol B (Capitolo 4), J. van Leeuwen (ed.), Elsevier, 1990.

W. Thomas, Languages, Automata, and Logic, in Handbook of Formal Languages, Vol. III, G. Rozenberg, A. Solomaa (eds.), Springer, 1997, pp. 389-455.

Bibliography - 2

D. Perrin, J. E. Pin, Infinite Words: Automata, Semigroups, Logic and Games, Pure and Applied Mathematics Series, Elsevier, 2004.

A. Montanari, G. Puppis, Verification of Infinite State Systems, Note al Corso della 18th European Summer School in Logic, Language and Information (ESSLLI), Malaga, Spain, 2006.

E. A. Emerson, Temporal and Modal Logic, Handbook of Theoretical Computer Science, Vol B (Capitolo 16), J. van Leeuwen (ed.), Elsevier, 1990, pp. 995-1072.

E.M. Clarke, O. Grumberg, D. Peled, Model Checking, The Mit Press, 2000.

M. Huth, M. Ryan, Logic In Computer Science: Modelling And Reasoning About Systems, Cambridge University Press, 1999.

Articoli su specifici temi del corso.