Concurrency Theory

Winter Semester 2017/18

Lecture 1: Introduction

Joost-Pieter Katoen and Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ws-1718/ct/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Preliminaries

Staff

- Lectures:
  - Joost-Pieter Katoen (`katoen@cs.rwth-aachen.de`)
  - Thomas Noll (`noll@cs.rwth-aachen.de`)
- Exercise classes:
  - Philipp Berger (`berger@cs.rwth-aachen.de`)
  - Sebastian Junges (`sebastian.junges@cs.rwth-aachen.de`)
- Student assistants:
  - Moritz Dederichs
  - Justus Fesefeldt

Target Audience

- Master program Informatik
  - Theoretische Informatik
- Master program Software Systems Engineering
  - Theoretical Foundations of SSE
- In general:
  - interest in formal models for concurrent (software) systems
  - application of mathematical modelling and reasoning methods
- Expected: basic knowledge in
  - essential concepts of operating systems and system software
  - formal languages and automata theory
  - mathematical logic

Course Objectives

Objectives

- Understand the foundations of concurrent systems
- Model (and compare) concurrent systems in a rigorous manner
- Understand the main semantical underpinnings of concurrency

Motivation

- Supporting the design phase
  - "Programming Concurrent Systems"
  - synchronisation, scheduling, semaphores, ...
- Verifying functional correctness properties
  - "Model Checking"
  - validation of mutual exclusion, fairness, absence of deadlocks, ...
- Comparing expressivity of models of concurrency
  - "interleaving" vs. "true concurrency"
  - equivalence, refinement, abstraction, ...

Organisation

- Schedule:
  - Lecture Mon 14:15–15:45 9U10 (starting 09 Oct)
  - Lecture Thu 14:15–15:45 9U10 (starting 12 Oct)
  - Exercise class Fri 14:15–15:45 9U10 (starting 20 Oct)
- Irregular lecture dates – checkout web page!
- 1st assignment sheet: 13 Oct on web page
  - submission by 20 Oct before exercise class
  - presentation on 20 Oct
- Work on assignments in groups of three
- Examination (6 ECTS credits):
  - oral or written (depending on number of participants)
- Admission requires at least 50% of the points in the exercises
- Solutions to exercises and exam in English or German

Moodle for Theoretical Computer Science

- Developed by Models and Theory of Distributed Systems group at TU Berlin (Prof. Nestmann)
- Learning units (in German):
  - A: fixed-point theory
  - B: bisimulation
- Procedure:
  - initial questionnaire (motivation, knowledge level)
  - division into groups A/B
  - online access to learning units (for two weeks)
  - final questionnaire
- Full details provided next week
- Please support this activity!

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: 0123}$$
$$\quad 13 \qquad\qquad\qquad 2$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1, or 3
- If exclusive access to shared memory and atomic execution of assignments guaranteed
  $\Longrightarrow$ only possible outcome: 3

Concurrency and Interaction

The problem arises due to the combination of

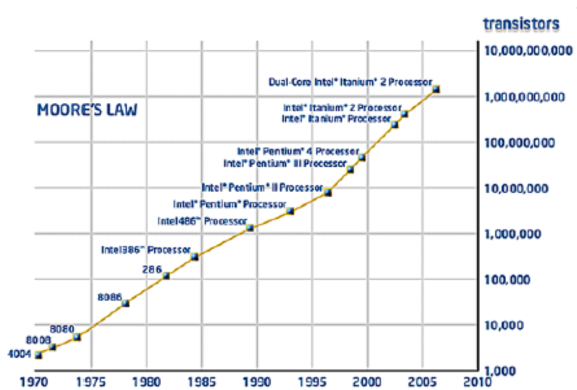
- concurrency and
- interaction (here: via shared memory)

Conclusion

When modelling concurrent systems, the precise description of the mechanisms of both concurrency and interaction is crucially important.

Concurrency Everywhere

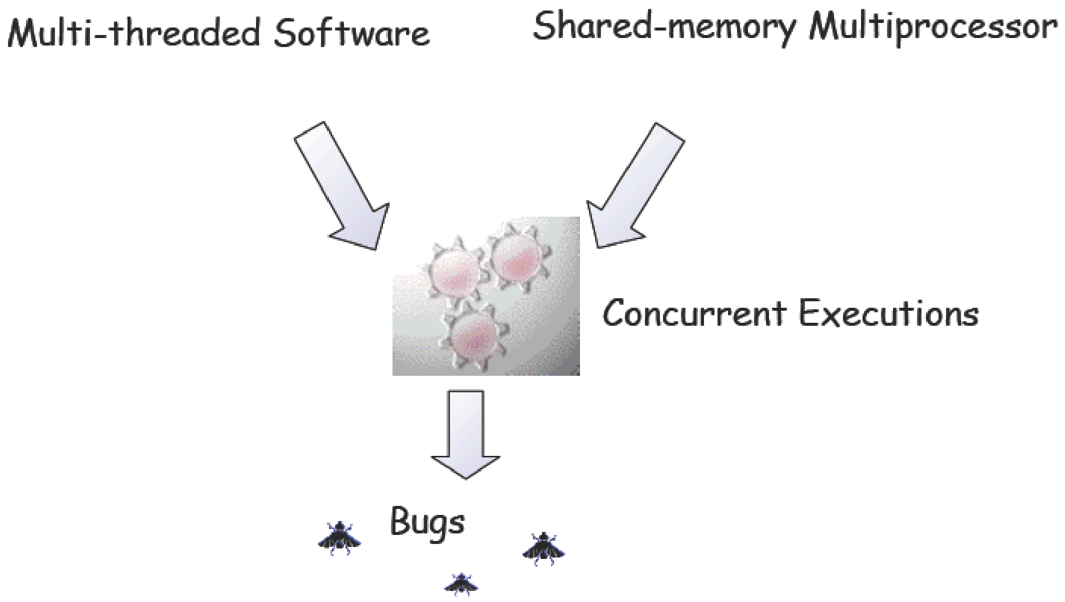
- Operating systems
- Embedded/reactive systems:
  - parallelism (at least) between hardware, software, and environment
- High-end parallel hardware infrastructure
  - high-performance computing
- Low-end parallel hardware infrastructure:
  - increasing performance only achievable by parallelism
  - multi-core computers, GPGPUs, FPGAs



Moore's Law: Transistor density doubles every 2 years

Problems Everywhere

- Operating systems:
  - mutual exclusion
  - fairness
  - no deadlocks, ...
- Shared-memory systems:
  - memory models
  - inconsistencies ("sequential consistency" vs. relaxed notions)
- Embedded systems:
  - safety
  - liveness, ...



Memory Models

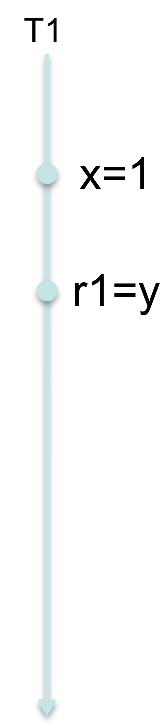
Sequential

An illustrative example

Initially: x = y = 0

<u>thread1</u>: <u>thread2</u>:

thread1	thread2
1: x = 1	3: y = 1
2: r1 = y	4: r2 = x



A Closer Look at Reactive Systems

Reactive Systems I

- Thus: “classical” model for sequential systems

$$\mathit{System} : \mathit{Input} \to \mathit{Output}$$

  (transformational systems) is not adequate
- Missing: aspect of interaction
- Rather: reactive systems which interact with environment and among themselves
- Main interest: not terminating computations but infinite behaviour
  (system maintains ongoing interaction with environment)
- Examples:
  - operating systems
  - embedded systems controlling mechanical or electrical devices (planes, cars, home appliances, ...)
  - power plants, production lines, ...

Reactive Systems II

Observation: reactive systems often safety critical
$\Longrightarrow$ correct behaviour has to be ensured

- Safety properties: “Nothing bad is ever going to happen.”
  E.g., “at most one process in the critical section”
- Liveness properties: “Eventually something good will happen.”
  E.g., “every request will finally be answered by the server”
- Fairness properties: “No component will starve to death.”
  E.g., “any process requiring entry to the critical section will eventually be admitted”

Overview of the Course

1. Introduction and Motivation
2. The "Interleaving" Approach
   - Syntax and semantics of CCS
   - Hennessy-Milner Logic
   - Case study: mutual exclusion
   - Extensions and alternative approaches (value passing, mobility, CSP, ACP, ...)
3. Equivalence, Refinement and Compositionality
   - Behavioural equivalences ((bi-)simulation)
   - Case study: mutual exclusion
   - (Pre-)congruences and compositional abstraction
   - HML and bisimilarity
4. The "True Concurrency" Approach
   - Petri nets: basic concepts
   - Case study: mutual exclusion
   - Branching processes and net unfoldings
   - Analyzing Petri nets
   - Alternative models (trace languages, event structures, ...)
5. Extensions (timed models, ...)

Overview of the Course

Literature

(also see the collection “Handapparat Softwaremodellierung und Verifikation” at the CS Library)

- Fundamental:
  - Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen and Jiří Srba: *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.
  - Wolfgang Reisig: *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer Verlag, 2012.
- Supplementary:
  - Maurice Herlihy and Nir Shavit: *The Art of Multiprocessor Programming*. Elsevier, 2008.
  - Jan Bergstra, Alban Ponse and Scott Smolka (Eds.): *Handbook of Process Algebra*. Elsevier, 2001.