# Concurrency: Languages, Programming and Theory

# – Introduction –

# Session 1 – Oct 22, 2003

Uwe Nestmann, Martin Odersky

EPFL-LAMP

# Why Concurrency Matters

- Between June 1985 and January 27, a computerized radiation therapy machine called Therac-25 caused 6 known accidents of massive radiation overdoses.
- Concurrent programming errors played an important role in these accidents.
- “Race conditions” between different concurrent activities in the control program resulted in bad control outputs.
- Because problems occured only sporadically, they took a long time to be detected and fixed.

# Why Concurrent Programming is Hard

What makes concurrent programming harder than sequential one?

- A very large number of possible execution histories, depending on the order in which instructions of individual processes (or: threads) are processed.
- Hence, concurrent programs are hard to write and verify.
- They are almost impossible to “debug”, at least with standard techniques.
- Necessary: Some theory. This includes
  - Understanding formally the meaning of a program.
  - Being able to reason whether two programs are equivalent, or whether an implementation meets a specification.

# What this Course is about

- **concurrency**
  - "things" running in parallel, or on distributed locations
  - synchronization through communication
  - mobility (of code and computation, not of devices)
- **theory**
  - a simple calculus of concurrent systems: CCS
  - a calculus of dynamic and mobile concurrent systems: $\pi$
  - *equivalences* and *congruences*
  - formal analysis and proof techniques

- **languages**
  - formal *syntax*
  - formal, operational *semantics* (describing the execution of programs)
  - (informal) type systems
  - 3 mini-languages (calculi): $\lambda$, CCS, $\pi$
- **programming**
  - concurrent programs that implement CCS and $\pi$-calculus specifications.
  - relationship with concurrent programming in Java.

# Objectives (I)

- ***Why*** *communicating/mobile systems ?*
  - increasing number of *existing systems*
  - tend to be *complex*
  - tend to be *error-prone*
  - …
- ***Why*** *calculi ?*
  - *compositional*: break big things into several small things
  - *algebraic*: ease mechanical verification
  - *syntactic*: provide basis for programming languages

# Objectives (II)

Course preparing for research on:

- conception and implementation of verification tools
- conception of “high-level” languages, API’s or design patterns for
  - mobile/distributed systems
- modeling and analyzing systems of this kind
- meta-theoretical problems of the base calculi
- . . .

# Objectives (III)

**After** the course, participants **should be able to**:

- ☐ easily read syntax & operational semantics
- ☐ compare/evaluate languages based on their semantics
- ☐ write specifications and implementations of concurrent programs
- ☐ reason formally about (toy) examples
- ☐ “play” with concurrent languages
  - write your own formal semantics for a given language
  - argue for it! (i.e., analyze its properties)

# Course Material & Support

- a web site
  `http://lamp.epfl.ch/teaching`
- a newsgroup
  `news://news.epfl.ch/epfl.ic.cours.clpt`
- an introductory book (by R. Milner)
  ***communicating and mobile systems: the pi-calculus***
  - 3 copies in the IN-library
  - 3 copies in the LAMP-library
- an advanced book (by D. Sangiorgi, D. Walker)
  ***The pi-Calculus: A Theory of Mobile Processes***
- slides
- papers

# Course Overview

| | | | |
|---|---|---|---|
| 1 | — | October 22, 2003: | Introduction, Functional Programming |
| 2 | — | October 29, 2003: | Foundations of Sequential Programming: The Lambda Calculus |
| 3 | — | November 5, 2003: | Encoding Functional Programs in Lambda |
| 4 | — | November 12, 2003: | A Calculus for Concurrent Processes (CCS) |
| 5 | — | November 19, 2003: | CCS for Programming |
| 6 | — | November 26, 2003: | $\pi$-Calculus Basics |
| 7 | — | December 3, 2003: | $\pi$-Calculus for Programming |
| 8 | — | December 10, 2003: | Higher-Level Concurrent Programming Idioms |
| 9 | — | December 17, 2003: | Exercises or Presentations |
| 10 | — | January 7, 2004: | Equivalences and Congruences of Programs |
| 11 | — | January 14, 2004: | Bisimulation in CCS |
| 12 | — | January 21, 2004: | CCS Verification with the Concurrency Workbench |
| 13 | — | January 28, 2004: | Bisimulation in $\pi$ |
| 14 | — | February 4, 2004: | $\pi$-Calculus Verification |

# Course Organization

- Three streams:
  1. Concurrency theory
  2. Foundations of programming languages
  3. Concurrent programming
- (1) and (2) will be done “ex-cathedra” in class.
- (3) will be done through programming exercises.

# Tool Support

- **verification tools**
  - concurrency workbench (CWB)
  - mobility workbench (MWB)
  - another bisimulation checker (ABC)
- **calculus-based programming**
  - in Scala, using CCS and $\pi$-Calculus API's

# Credits

- **oral exam + bonus** (during the semester)
- **bonus option 1: presentation**
  - of a paper
  - of some selected material from one of the books
- **bonus option 2: miniproject**
  - modeling and analysis for a chosen problem
  - meta-theoretical proofwork
- **bonus option 3: Xmas exam**
- availability of the options will depend on number of participants

# Communication and Feedback

- **Uwe.Nestmann@epfl.ch**
  **Martin.Odersky@epfl.ch**
  **Vincent.Cremet@epfl.ch**
  **Sebastien.Briais@epfl.ch**
- **news://news.epfl.ch/epfl.ic.cours.clpt**
- INR 317, 318, 319, 329