Foundation of Software Technology (FaST)

by Marcello Bonsangue
Members

Professors
- F. Arbab, Head of Cluster
- Joost Kok, Head of Cluster
- Frank de Boer

Associate Professors
- Jetty Kleijn
- Marcello Bonsangue
- Luuk Groenewegen (retired)

Assistant Professors

PostDocs
- Natallia Kokash
- Michiel Helvenstein
- Stijn de Gouw

PhD Students
- Bahamn Pourvatan
- Jurriaan Rot
- Vlad Serbanescu
- Nikolaos Bezirgiannis
- Kayvan Azadbakht
- Kasper Dokter
- Sung Jongmans
- Natallia Kokash
- Chetan Nagarajagowda
- Behrooz Nobakht

Secretary
- Marloes van der Nat
Mission

- Development of formalisms, methods, techniques, and tools to design, analyze, and construct software systems out of components and services.

- **Ingredients**
  - Classes/Objects
  - Components
  - Services

- **Construction**
  - Composition
  - Correctness

- **Issues**
  - Concurrency
  - Coordination
  - Model

- **Approach**
  - Formal methods
  - Experimental systems
  - Empirical studies
Vision

- Mastering the complexity of modern software systems
  - Embedded systems
  - Systems of independent components
  - Service composition
  - Multi-core chips

- Characteristics
  - Concurrency
  - Distribution
  - Mobility
  - Dynamic reconfiguration / self-adaptation
  - Multiple, independent providers
  - Third-party black-box composition
  - Compositional end-to-end QoS
Areas

- Dynamically reconfigurable systems
- Testing, deductive verification, and model checking
- Formal models of concurrent, distributed, object oriented, and component-based systems.
- Formal semantics, process algebras and logics for reasoning about such systems
- Quality of service
Major challenge

Development of techniques for effectively establishing behavioral properties of dynamical systems
Activities – F. Arbab

- Coordination models and languages
  - Coordinated composition of software intensive systems
  - Coordination language Reo
  - Constraint automata

- Use of coordination
  - Compositional QoS
  - Code generation for multi-core systems
  - Service oriented computing
  - Testing
Activities – F.S. de Boer

- **Software correctness**
  - Programming logics
    - Deductive proof methods for the verification of programs
  - Object Orientation
    - Verification and Testing
  - Concurrency
    - Semantics
    - Multi-core programming
    - Cloud aware programming
  - Integrated Formal Methods
    - Testing
    - Model checking
    - Deductive verification
    - Abstraction
Activities – M. Bonsangue

- **Formal Methods**
  - Mathematically-based techniques for the specification, development and verification of software and hardware systems
    - Testing
    - Semantics and model checking of software connectors
    - Semantics and verification of dynamical evolving systems

- **Algebra, Coalgebra and Logic**
  - Mathematical frameworks for the specification of the reactive behaviour of systems
    - Process algebra, regular expressions
    - (Probabilistic, non-deterministic, ...) automata
    - Modal logics
Activities – L. Groenewegen

- Coordination models
  - Paradigm language
    - PARallelism, its Analysis, Design and Implementation by a General Method
    - Modeling behaviors and constraints thereon

- Coordination patterns
  - Self-adaptation patterns
    - McPal
    - On-the-fly migration through coordination
    - Managing changing processes
Activities – J. Kleijn

- Concurrency
  - Using Petri nets, Team Automata, and other automata formalisms to model, analyze, and verify the behavior of:
    - Computation
    - Biological systems
    - Hardware and software components
Interest in formal methods

- **Maths for formal methods**
  Algebras, coalgebras and logics.

- **Deductive verification**
  Study of logical formalism with the goal of proving formally that the software satisfies its specification.

- **Model checking**
  Development of technique to automatically check that the software satisfies its specification.

- **Testing**
  Executing a program with the intent of finding bugs.
Few bachelor projects

- Equations and automata
- Parsing trees from derivatives
- Recursive guarded languages
- Monitoring and runtime verification
- Flow graph for a OO-language
- Resource aware programming
- Application specific scheduling
Equations and automata

- **Equation**: $aab = ba$

- **Problem**: Find a complete set of equations satisfied by an automaton.
Parsing trees from derivatives

- \( S \rightarrow aSb | ab \)

- \( S_{ab} = (S_a)_b = (Sb | b)_b = S | \Lambda \)

- **Problem**: Construct a parsing tree from the derivatives.
Recursive guarded language

- Regular guarded language
  - 0: empty language
  - 1: empty word language
  - b: Boolean actions
  - a: ordinary actions
  - r + r, r;r: choice, seq. composition
  - r*: recursion

**Problem**: Extend it with full recursion.