Teaching Principles of Dependable Distributed Software

Zoltán Horváth
hz@inf.elte.hu

Faculty of Informatics, Eötvös Loránd University, Budapest

14th Workshop on Software Engineering Education and Reverse Engineering
Sinaia, Aug 26, 2014
Software – what our everyday life is depending on

- communication: phone, e-mail, social media
- navigation
- home banking
- healthcare administration, medical devices
- car – in-vehicle software: diagnostic, safety, driver assistance
- aircraft – autopilot control system ("Who’s really flying the plane?")
- nuclear power plant control system safety software
Infamous software bugs

- The Mars Climate Orbiter doesn’t orbit (1998, metric system, $327.6$ million)
- Call waiting ... and waiting ... and waiting (On Jan. 15, 1990, around 60,000 AT&T customers, congestion lead to cascade reset of 114 switches)

- Therac-25 Medical Accelerator disaster (1985, race condition, two modes)

- Soviet early-warning system (Sep. 23, 1983, Petrov)
- Nuclear Power Plant shutdown (June 5, 2008, USA, software update in a distributed system)

- Ford in-vehicle software failure - hw. reset at Körösfő (2014)
- Samsung mobile network connection error (2014, flight mode off and on)
Correctness of Distributed programs

- interleaving, branching time semantics of distributed and parallel programs
- testing of properties, reachable states
- specification of the problem to solve
- static verification, analysis, calculation of reachable states

Zoltán Horváth, hz@inf.elte.hu
Always true is not always invariant

- \( P \) invariant – \( P \) holds initially and \( P \) is preserved (also in unreachable states)
- \( P2 \) always true – \( P2 \) holds in all reachable states

Only invariants do compose!
Motivation for using formal methods:
- safety critical applications
- safe application of software components
- **primary goal**: sound concepts about distributed and parallel programs
We need a formal model, which is appropriate for specification of problems and developing the solutions of problems in case of parallel and distributed systems. The introduced model

- is an extension of a relational model of nondeterministic sequential programs,
- provides tools for stepwise refinement of problems, in a functional approach,
- uses the concept of iterative abstract program of UNITY, and
- the concept of solution is based on the comparison of the problem as a relation and the behaviour relation of the program.
Example problem: the dining philosophers

States: thinking:t, forks in hands:f, eating:e, at home:h,
Some requirements (problem specification): \( \forall i : \)

**unless:** \( f(i).t \triangleright f(i).f \lor f(i).h \)

**unless:** \( f(i).f \triangleright f(i).e \)

**ensures:** \( f(i).e \leftrightarrow f(i).t \)

**inevitable leads-to:** \( f(i).t \hookrightarrow f(i).h \)

**invariant:** \( (f(i).e \rightarrow (\neg f(i + 1).e \land f(i - 1).e)) \in \text{inv} \)

**fixed point:** \( \text{FP} \Rightarrow f(i).h \)

**termination:** \( \forall i : f(i).t \in \text{TERM} \)
Example for abstract program

\[ S = (\text{SKIP}, \quad \begin{array}{c}
\{ \forall i \in [1..n-1] \ a(i), a(i+1) := a(i+1), a(i), \text{ if } a(i) > a(i+1) \} \end{array} ) \]

Abstract execution model: No control flow, free processors select atomic assignments asynchronously
Program: scheduling, processes, location, communication infrastructure, language
Example: C++/PVM PC-cluster (Parallel Virtual Machine) / Erlang
Solution: Specification requirements are satisfied by program properties (synthesis and formal verification)
The notion of the state space makes it possible to define the semantical meaning of a problem independently of any program (functional approach).

The generalized concept of a problem is applicable for cases in which termination is not required but the behaviour of the specified system is restricted by safety and progress properties.

The solution of a problem may be a sequential program, a parallel one, or even a program built up from both sequential and parallel components.

synthesis of a solution for asynchronous operations (reduce), parallel elementwise processing (map), solutions based on process networks, pipeline (function composition)
Eötvös Loránd University of Sciences, cca. 30 000 stuents, leading research university of Hungary, 8 facultues

Faculty of Informatics, cca. 3000 students in CS, in teacher of informatics, in geoinformatics. Computer Science program since 1972,

applications: 40% of the annual budget, leader of EIT ICT Labs node, double degree joinr European masters, strong industrial (joint R&D Labs) and international partnership (e.g. CEEPUS network with Novi Sad, Maribor, Cluj, Ljubljana, Plovdiv)

Computer Science: 550 + 120 new bachelor students (three specializations) selected from 1200+ applicants, 120+ new master students per year (three specializations), 20+ new PhD students per year, 60+ international students,

Zoltán Horváth, hz@inf.elte.hu
48-60 ECTS mathematics in BSc curricula (depending on specialization item strong specialization in Software Technology (formal model for sequential programming (15 ECTS), 3 semesters of Programming languages (15 ECTS): C++, Ada, functional programming, Software engineering, Object oriented programming, etc.),

Software Technology Lab: R&D projects are part of master program (16+20 ECTS), 6 projects with cca. 70 master students in teams,

design of parallel and distributed software both at bachelor and at master level: theory, practice, programming assignments, computer lab tests, oral exams.
Parallelism and Functional Programming

- Sparkle-T - proof tool for temporal properties (e.g. invariants) of Clean programs - with Máté Tejfel and Tamás Kozsik
- DClean - a coordination language for type safe distributed cooperation of Clean programs - with Viktória Zsók, Zoltán Hernyák
- Static analysis - to support code comprehension at industrial level in Erlang - with Melinda Tóth, István Bozó and others
- Paraphrase - property preserving transformations of Erlang programs to enable parallel multicore execution - with Tamás Kozsik, Melinda Tóth, István Bozó, Judit Kőszegi, Dániel Horpácsi, Viktória Fördős and others
- DSL - functional programming for CPS - Rea language (based on Erlang and on Sacla)