

CS 515: Concurrent and Real-Time Systems

Stefan D. Bruda

Fall 2019



- Coordinates:
 - **Course Web page:** <http://cs.ubishops.ca/home/cs515> (also accessible following the obvious link from <http://bruda.ca>)
 - Instructor: Stefan Bruda
(<http://bruda.ca>, stefan@bruda.ca, Johnson 114B, ext. 2374)
 - Office hours?
- Textbook: Steve Schneider, Concurrent and Real-time Systems: The CSP Approach (Wiley 1999)
 - Electronic version available on-line, but be aware that pages and exercise numbers do not always match



- Coordinates:
 - **Course Web page:** <http://cs.ubishops.ca/home/cs515> (also accessible following the obvious link from <http://bruda.ca>)
 - Instructor: Stefan Bruda
(<http://bruda.ca>, stefan@bruda.ca, Johnson 114B, ext. 2374)
 - Office hours?
- Textbook: **Steve Schneider, Concurrent and Real-time Systems: The CSP Approach (Wiley 1999)**
 - Electronic version available on-line, but be aware that pages and exercise numbers do not always match
- Introduction in **formal methods**
 - Specification using a **process algebra**
 - Operational semantics (**transition systems**)
 - System verification (traces, failures, divergence)
 - Model-based testing
 - Specification using **temporal logic**
 - Basis of **model checking**
 - Timed specification and verification (if time permits)



- The historically mainstream method of program verification: **throw tests at the program and hope for the best**
 - Informal determination of what tests are meaningful
 - Can detect defects, but certainly **cannot guarantee any degree of correctness**
 - Still used nowadays, especially for application software
 - Extreme variant: let the user come up with and apply the tests (**beta versions**)



- The historically mainstream method of program verification: **throw tests at the program and hope for the best**
 - Informal determination of what tests are meaningful
 - Can detect defects, but certainly **cannot guarantee any degree of correctness**
 - Still used nowadays, especially for application software
 - Extreme variant: let the user come up with and apply the tests (**beta versions**)
- Alternate method: **deductively prove the program correct**
 - Program correctness is treated as a theorem
 - Proof done by hand
 - **Guarantees correctness**, takes **lots of time**, needs **experts**



- The historically mainstream method of program verification: **throw tests at the program and hope for the best**
 - Informal determination of what tests are meaningful
 - Can detect defects, but certainly **cannot guarantee any degree of correctness**
 - Still used nowadays, especially for application software
 - Extreme variant: let the user come up with and apply the tests (**beta versions**)
- Alternate method: **deductively prove the program correct**
 - Program correctness is treated as a theorem
 - Proof done by hand
 - **Guarantees correctness**, takes **lots of time**, needs **experts**
- Best method: **formal methods**
 - Test a system against a formal (mathematical) specification
 - Some effort to create the specification but the testing is **fully automated**
 - **Guarantees correctness**



Three Mile Island Nuclear Generating Station, Unit 2, 28 March 1979

- Cooling pump failure causes increased pressure
- Relief valve opened automatically; indicator light turns on in the control room
- Pressure drops, command to close the relief valve given automatically; indicator light turns off
- Problem: indicator light signals that current has been applied to the actuator, not that the valve is physically closed
- Mechanical problem prevents the valve to close; nobody knows!
- Faulty message by the indicator light confuses the operator, who fails to recognize the loss of coolant event



Three Mile Island Nuclear Generating Station, Unit 2, 28 March 1979

- Cooling pump failure causes increased pressure
- Relief valve opened automatically; indicator light turns on in the control room
- Pressure drops, command to close the relief valve given automatically; indicator light turns off
- Problem: indicator light signals that current has been applied to the actuator, not that the valve is physically closed
- Mechanical problem prevents the valve to close; nobody knows!
- Faulty message by the indicator light confuses the operator, who fails to recognize the loss of coolant event
- Core meltdown, one of the top 5 nuclear incidents ever recorded
- No formal verification of the system



The Pentium Microprocessor (successor of 80486), late 1994

- Unlike previous Intel CPUs the Pentium chips includes a floating-point unit (FPU)
- Speeds up computations with floating-point numbers
- All the Pentium chips built until late 1994 had errors in the on-chip FPU instructions for division
- Pentium's FPU incorrectly divides certain floating-point numbers
 - $4195835/3145727$ is 1.33382 according to math and 1.33374 according to said Pentium



The Pentium Microprocessor (successor of 80486), late 1994

- Unlike previous Intel CPUs the Pentium chips includes a floating-point unit (FPU)
- Speeds up computations with floating-point numbers
- All the Pentium chips built until late 1994 had errors in the on-chip FPU instructions for division
- Pentium's FPU incorrectly divides certain floating-point numbers
 - $4195835/3145727$ is 1.33382 according to math and 1.33374 according to said Pentium
- Widely publicized mistake, huge embarrassment for Intel
- **Joke of the day:**
 - Q:** Why did they call their new processor Pentium instead of 80586?
 - A:** Because they used the new processor to add 100 to 80486 and the result was 80585.999998
- Faulty design, never formally verified
- Causes Intel to introduce formal verification for all of its chips



The Space Shuttle, 1981–2011

- 135 missions; second-longest-serving manned space vehicle
- Very thorough protocol for software changes
- Changing **one line** of code requires an average of **10 pages of documentation**
- Well-defined chain of responsibility
- All changes require extensive testing
- All changes must have a solid justification and are considered **a priori suspicious** (“what is not there cannot go wrong”)
- All but the most trivial changes required **formal proofs or correctness**



The Space Shuttle, 1981–2011

- 135 missions; second-longest-serving manned space vehicle
- Very thorough protocol for software changes
- Changing **one line** of code requires an average of **10 pages of documentation**
- Well-defined chain of responsibility
- All changes require extensive testing
- All changes must have a solid justification and are considered **a priori suspicious** (“what is not there cannot go wrong”)
- All but the most trivial changes required **formal proofs or correctness**
- **No software defect was ever found!**
- **Widely regarded as the most robust piece of software ever developed**
- **Price paid: Very slow development, huge development effort**



- A **process algebra** is like a programming language, but for describing the behaviour of a system rather than the system itself
 - Similar in spirit with a functional programming language
 - Like any programming language it has a **syntax** and a **semantics**
 - Semantics can be expressed in multiple ways
 - **Structural operational semantics** (SOS), best suited for describing the language but also supports verification
 - **Operational semantics**; best suited for automated verification: a process algebraic description “compiles” into a **transition system**



- A **process algebra** is like a programming language, but for describing the behaviour of a system rather than the system itself
 - Similar in spirit with a functional programming language
 - Like any programming language it has a **syntax** and a **semantics**
 - Semantics can be expressed in multiple ways
 - **Structural operational semantics** (SOS), best suited for describing the language but also supports verification
 - **Operational semantics**; best suited for automated verification: a process algebraic description “compiles” into a **transition system**
- Verification is based on the behaviour of a system S expressing the desired behaviour (**specification**) and a **system under test** I
 - The correctness of the system under test established based on an **implementation relation** (preorder): $I \sqsubseteq S$ or “ I implements S ”
 - This preorder induces an equivalence relation between processes: $I \equiv S$ iff $I \sqsubseteq S \wedge S \sqsubseteq I$
 - Several implementation relations can be defined, depending on what is deemed observable about processes
 - Some times convenient to define implementation relations based on intermediate processes (**tests**)
 - Algebraic formal methods in a nutshell: the study of various implementation relations



- Logical system specification is done using a formal logic
 - The good ol' Boolean logic is insufficient, so it is augmented with constructs that allow the specification of sequences of properties
 - Examples include “P will eventually be true”, “P is always true”, “P must remain true until Q becomes true”
 - The resulting formalism is called **temporal logic**
 - Temporal logic can be used to specify the properties of individual runs of a system under test (**linear time**)
 - Other kinds of temporal logic can be used to specify the properties of all the possible runs at once (**branching time**)
 - Both linear and branching time have advantages and disadvantages



- Logical system specification is done using a formal logic
 - The good ol' Boolean logic is insufficient, so it is augmented with constructs that allow the specification of sequences of properties
 - Examples include “P will eventually be true”, “P is always true”, “P must remain true until Q becomes true”
 - The resulting formalism is called **temporal logic**
 - Temporal logic can be used to specify the properties of individual runs of a system under test (**linear time**)
 - Other kinds of temporal logic can be used to specify the properties of all the possible runs at once (**branching time**)
 - Both linear and branching time have advantages and disadvantages
- Verification is based on a logical formula (specification) and a model of the behaviour of the system under test
 - Transition systems can be used to specify the latter, but to make things more interesting the traditional model is actually different (**Kripke structures**)
 - The system under test is verified against the specification using a **model checking** algorithm

SPECIFICATION AND VERIFICATION OF REAL-TIME SYSTEMS



- All the models enumerated earlier (process algebrae, temporal logic, transition systems, etc.) can be augmented to incorporate data on **real time** (as measures by a clock)
 - Real time can be **dense** (real values) or **discrete**
 - Real time introduces several extra issues, so considering it is not trivial (especially true for dense time)
- All the verification techniques mentioned earlier can then be augmented to account for real time information
 - We thus obtain **timed preorders**, **timed testing**, **timed model checking**, etc.
 - Note in passing: when talking about real time engineering types prefer the adjective “real-time” while math people prefer the adjective “timed”; they both refer to the same thing!
- Real time not expected to be covered in the course extensively (we will likely run out of... real time), but I hope to be able to provide a however short introduction



Ariane 5 Flight 501, 4 June 1996

- Brand new, heavier rocket; navigation software taken directly from Ariane 4
 - Software **formally verified** in the Ariane 4 setting
 - Flight path considerably different
 - **No new verification for the changed specification**
- Inertial parameters considerably higher for Ariane 5 (heavier), exceed the storage capacity of the program (**arithmetic overflow**)
- Main computer detects exception, shuts down
- Back-up computer fires up, detects the same exception, shuts down
- Flight path in shambles 37 seconds after launch, self-destruct activates
- **Down goes rocket and satellite for a grand total of \$370,000,000 in losses**
- **Oh the irony:** Arithmetic overflow can be handled in software; no such handler existed for this particular variable, because “overflow cannot happen here”
- **Oh the irony, take 2:** The subsystem that causes the fault was important for navigation in Ariane 4 but **was not even actively used in Ariane 5!**

FORMAL VERIFICATION MATTERS

