

# Introduction to Research in Computer Science

Lewis University

Spring 2022

# Outline

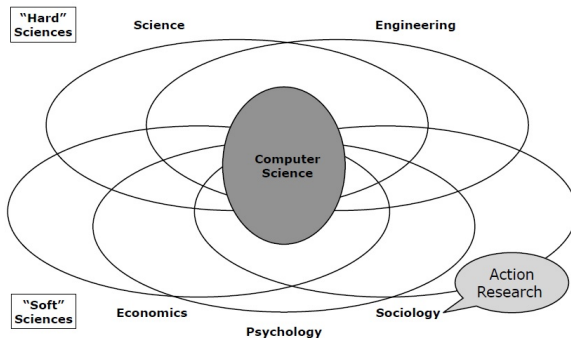
- 1 Course Objectives
- 2 Course Administration
- 3 What is Computer Science?
- 4 Areas of Research in Computer Science
- 5 Algorithms
- 6 Distributed systems
- 7 Building Reliable Software
- 8 Software Verification (Correctness)
- 9 Correctness vs Efficiency

# Course Objectives

- Gain an appreciation for the diverse areas encompassing modern computer science
- Understand different approaches to research in computer science, including proofs, observational studies, and implementation-driven research
- Be able to read and ask questions of a computer science research paper
- Present and answer questions about a technical paper
- Explore formal verification as a research area in computer science

- Method of Delivery: Synchronous (live) and Asynchronous (Pre-recording)
- Assessment:
  - Class Contribution (Discussions on BB): 15%
  - Assignments (2 to 3): 35%
  - Course Project (Presentation 15%, Project 35%): 50%
- All assignments and Project (Report and Presentation) should be done in Latex
- Compulsory Readings: Weekly reading, how to read scientific paper links
- Project Proposal's due date: Week #3

# What is Computer Science?



# What is Computer Science?

- Some combination of:
  - Science
  - Engineering
  - Mathematics
- More precisely
  - “Computer Science” is a name of a field
  - Computer science is a the scientific endeavor relating to computing

# Areas of Research in Computer Science

- Abundant-data applications, algorithms, and architectures
- Artificial intelligence and robotics
- Verification, proofs, and automated debugging of hardware designs, software, networking protocols, mathematical theorems, etc
- Bio-informatics and other uses of CS in biology, biomedical engineering, and medicine
- Databases, data centers, information retrieval, and natural-language processing

# Areas of Research in Computer Science

- Emerging technologies for computing hardware, communication, and sensing
- Human-computer interaction
- Large-scale networking
- Limits of computation and communication
- Multimedia
- Programming languages and environments
- Security of computer systems and support for digital democracy



A skilled programmer must have good insight into algorithms.

At bachelor level you were offered courses on basic algorithms: searching, sorting, pattern recognition, graph problems, ...

You learned how to detect such subproblems within your programs, and solve them effectively.

You are trained in algorithmic thought for uniprocessor programs (e.g. divide-and-conquer, greedy, memorization).

Consider the following template C code posted on BB.

What algorithm is being implemented here? Write a program in the language of your choice to test this algorithm.

There are three desirable properties for a good algorithm. ....

What are three desirable properties for a good algorithm?

Designing correct, efficient and implementable algorithms for real-world problem requires access to two distinct bodies of knowledge:

- *Techniques*.....
- *Resources*.....
- **Lesson:** There is a fundamental difference between algorithms, which always produce a correct result, and heuristics, which may usually do a good job but without providing guarantee.

What are three desirable properties for a good algorithm?

Designing correct, efficient and implementable algorithms for real-world problem requires access to two distinct bodies of knowledge:

- *Techniques*.....
- *Resources*.....
- **Lesson:** There is a fundamental difference between algorithms, which always produce a correct result, and heuristics, which may usually do a good job but without providing guarantee.

What are three desirable properties for a good algorithm?

Designing correct, efficient and implementable algorithms for real-world problem requires access to two distinct bodies of knowledge:

- *Techniques*.....
- *Resources*.....
- **Lesson:** There is a fundamental difference between algorithms, which always produce a correct result, and heuristics, which may usually do a good job but without providing guarantee.

# Algorithms: Analysis - Big O notation

Complexity measures state how resource consumption (messages, time, space) grows in relation to input size.

For example, if an algorithm has a worst-case message complexity of  $O(n^2)$ , then for an input of size  $n$ , the algorithm in the worst case takes *in the order of*  $n^2$  messages.

Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}_{>0}$ .

$f = O(g)$  if, for some  $C > 0$ ,  $f(n) \leq C \cdot g(n)$  for all  $n \in \mathbb{N}$ .

$f = \Theta(g)$  if  $f = O(g)$  and  $g = O(f)$ .

# Algorithms: Analysis - Big O notation

Complexity measures state how resource consumption (messages, time, space) grows in relation to input size.

For example, if an algorithm has a worst-case message complexity of  $O(n^2)$ , then for an input of size  $n$ , the algorithm in the worst case takes *in the order of*  $n^2$  messages.

Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}_{>0}$ .

$f = O(g)$  if, for some  $C > 0$ ,  $f(n) \leq C \cdot g(n)$  for all  $n \in \mathbb{N}$ .

$f = \Theta(g)$  if  $f = O(g)$  and  $g = O(f)$ .

# Distributed systems

A **distributed system** is an interconnected collection of autonomous processes.

Motivation:

- information exchange
- resource sharing
- parallelization to increase performance
- replication to increase reliability
- multicore programming



# Distributed versus uniprocessor

Distributed systems differ from uniprocessor systems in three aspects.

- *Lack of knowledge on the global state*: A process has no up-to-date knowledge on the local states of other processes.  
**Example**: termination and deadlock detection become an issue.
- *Lack of a global time frame*: No total order on events by their temporal occurrence.  
**Example**: mutual exclusion becomes an issue.
- *Nondeterminism*: Execution of processes is nondeterministic, so running a system twice can give different results.  
**Example**: race conditions

# Distributed versus uniprocessor

Distributed systems differ from uniprocessor systems in three aspects.

- *Lack of knowledge on the global state*: A process has no up-to-date knowledge on the local states of other processes.

**Example**: termination and deadlock detection become an issue.

- *Lack of a global time frame*: No total order on events by their temporal occurrence.

**Example**: mutual exclusion becomes an issue.

- *Nondeterminism*: Execution of processes is nondeterministic, so running a system twice can give different results.

**Example**: race conditions

# Distributed versus uniprocessor

Distributed systems differ from uniprocessor systems in three aspects.

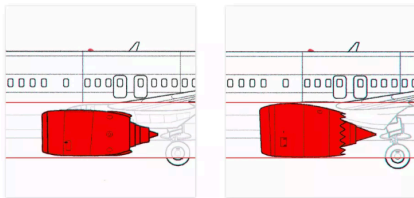
- *Lack of knowledge on the global state*: A process has no up-to-date knowledge on the local states of other processes.  
**Example**: termination and deadlock detection become an issue.
- *Lack of a global time frame*: No total order on events by their temporal occurrence.  
**Example**: mutual exclusion becomes an issue.
- *Nondeterminism*: Execution of processes is nondeterministic, so running a system twice can give different results.  
**Example**: race conditions

# Building Reliable Software

- Suppose you run a software company
- Support you've sunk 30+ person-years into developing the “next big thing”:
  - ★ Boeing Dreamliner2 flight controller
  - ★ Autonomous vehicle control software for Tesla
  - ★ Gene therapy DNA tailoring algorithms
  - ★ Super-efficient green-energy power grid controller
- How do you avoid disasters?
  - ★ Turns out software endangers lives

# Boeing 737 Max Crashes

- Involved in two crashes
  - ✦ Lion Air Flight 610 on October 29, 2018 — 189 dead
  - ✦ Ethiopian Airlines Flight 302 on March 10, 2019 — 157 dead
- The crash is attributed to design errors including flight control software
  - ✦ The position of larger engines on 737 Max generated additional lift



Engine placement on the third-generation 737 NG (left) versus the MAX (right).

# Boeing 737 Max Crashes

- Manoeuvring Characteristics Augmentation System (MCAS)
  - ✦ Software to sense angle of attack (AoA) from a sensor and automatically compensate
- Crashes due to AoA sensor data but also due to MCAS software
- Every time MCAS was switched on and off again, it acted like first time pitching nose lower
  - ✦ incorrect spec not including history
- Max 0.8 degrees pitch during testing, which was changed to 2.4 after
  - ✦ Executing conditions not reflective of testing
- MCAS completely ignored that pilots were desperately pulling back on the yoke
  - ✦ Incorrect spec not considering environment

# Not an isolated incident

- NASA's Mars Climate Orbiter
  - ✦ A sub contractor on the engineering team failed to make a simple conversion from English units to metric
  - ✦ \$125 million
- Ariane 5 Flight 501
  - ✦ The software had tried to cram a 64-bit number into a 16-bit space.
  - ✦ Crashed both the primary and the backup computer
  - ✦ \$500 million payload lost + \$XXX to fix the flaw.
- Hawaii Sends Out a State-Wide False Alarm About a Missile Strike
  - ✦ there were “troubling” design flaws in the Hawaii Emergency Management Agency's alert origination software.
- The Equifax social security hack
  - ✦ 143 million of their consumer records (names, SSN, credit card numbers) were stolen by attackers.

# Approaches to Validation

- Social
  - ✦ Code reviews
  - ✦ Extreme/pair programming
- Methodological
  - ✦ Design patterns
  - ✦ Test-driven development
  - ✦ Version control
  - ✦ Bug Tracking
- Technological
  - ✦ Static analysis
  - ✦ Fuzzers
- Mathematical
  - ✦ Sound Type Systems
  - ✦ Formal verification



Less formal: Techniques may miss problems in programs

All of these methods should be used!

Even the most formal can still have holes:

- did you prove the right thing?
- do your assumptions match reality?

More formal: eliminate *with certainty* as many problems as possible.



# Software: first failure cause of computing systems

Size: from some (tens) of thousands of code lines to some millions of code lines

Development effort:

0,1-0,5 person.year / KLOC (large software)  
5-10 person.year / KLOC (critical software)

Share of the effort devoted to fault removal:

45-75%

Fault density:

10-200 faults / KLOC created during development



- static analysis
- proof
- model-checking
- *testing*

0,01-10 faults / KLOC residual in operation



# The Quest for Software Correctness

Speech@50-years Celebration CWI Amsterdam

“It is fair to state, that in this digital era correct systems for information processing are more valuable than gold.”



Henk Barendregt

# The Importance of Software Correctness

Rapidly increasing **integration of ICT** in different applications

- embedded systems
- communication protocols
- transportation systems

⇒ reliability increasingly depends on software!

Defects can be **fatal** and extremely **costly**

- products subject to mass-production
- safety-critical systems

# What is System Verification?

## Folklore “definition”

System verification amounts to check whether a system fulfills the qualitative requirements that have been identified

## Verification $\neq$ validation

- Verification = “check that we are building the thing **right**”
- Validation = “check that we are building the **right** thing”

# Software Verification Techniques

## Peer reviewing

- static technique: manual code inspection, no software execution
- detects between 31 and 93% of defects with median of about 60%
- subtle errors (concurrency and algorithm defects) hard to catch

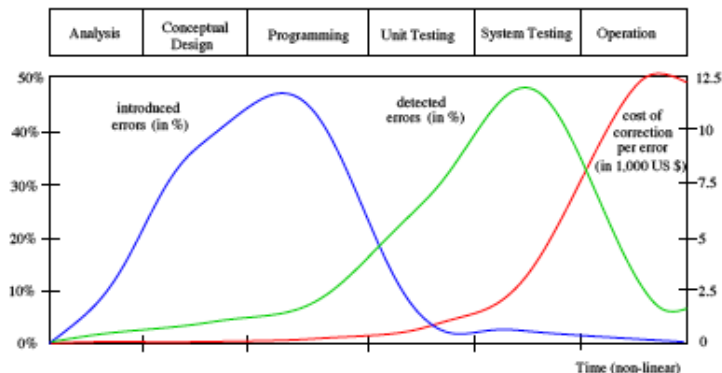
## Testing

- dynamic technique in which software is executed

## Some figures

- 30% to 50% of software project costs devoted to testing
- more time and effort is spent on validation than on construction
- accepted defect density: about 1 defects per 1,000 code lines

# Bug Hunting: the Sooner, the Better



# Formal Methods

## Intuitive description

Formal methods are the

“applied mathematics for modelling and analysing ICT systems”

Formal methods offer a large potential for:

- obtaining an **early integration** of verification in the design process
- providing **more effective** verification techniques (higher coverage)
- **reducing** the verification time

## Usage of formal methods

Highly recommended by IEC, FAA, and NASA for safety-critical software

# Formal Verification Techniques for Property $P$

## Deductive methods

- method: provide a formal **proof** that  $P$  holds
- tool: theorem prover/proof assistant or proof checker
- applicable if: system has form of a mathematical theory

## Model checking

- method: **systematic check** on  $P$  in all states
- tool: model checker (SPIN, NUSMV, UPPAAL, ...)
- applicable if: system generates (finite) behavioural model

## Model-based simulation or testing

- method: test for  $P$  by **exploring possible behaviours**
- applicable if: system defines an executable model



# Simulation and Testing

## Basic procedure:

- take a model (simulation) or a realisation (testing)
- stimulate it with certain inputs, i.e., the tests
- observe reaction and check whether this is “desired”

## Important drawbacks:

- number of possible behaviours is very large (or even infinite)
- unexplored behaviours may contain the fatal bug

## About testing ...

testing/simulation can show the presence of errors, **not** their absence

# Milestones in Formal Verification

- **Mathematical program correctness** (Turing, 1949)
- **Syntax-based technique for sequential programs** (Hoare, 1969)
  - for a given input, does a computer program generate the correct output?
  - based on compositional proof rules expressed in predicate logic
- **Syntax-based technique for concurrent programs** (Pnueli, 1977)
  - handles properties referring to states during the computation
  - based on proof rules expressed in temporal logic
- **Automated verification of concurrent programs**
  - model-based instead of proof-rule based approach
  - does the concurrent program satisfy a given (logical) property?

# Which Approach Should We Consider?

Correctness vs Efficiency: What Comes first?

Are your Undergraduate Knowledge of Algorithms still relevant to the analysis of distributed systems?

Urgent Need to have a **formal framework** for describing Algorithms:  
Uniprocessor or distributed systems

In this course, we will analyze **correctness proofs = Formal methods** as the main support for your introduction to research in computer science.

Possible Themes for research initiation: Verification, proofs, and automated debugging of hardware designs, software, networking protocols, formal methods for complex systems, and mathematical theorems.