CPSC 416 Distributed Systems

Winter 2022 Term 2 (March 7, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Logistics

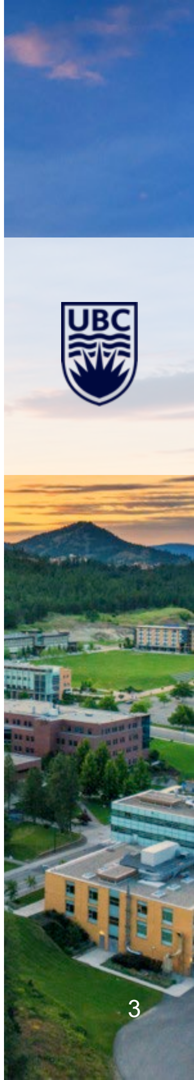
Deadlines

Project 3 Released. Late Deadline: April 13, 2023. Report Grades Released.

Project 4 Released. Initially Due: March 13, 2023

Project 5 Released Due: April 13, 2023

All project work is due April 13, 2023. Late projects are scaled by 75%.



Deadlines

Alternate Path 1 & 2: Review in progress

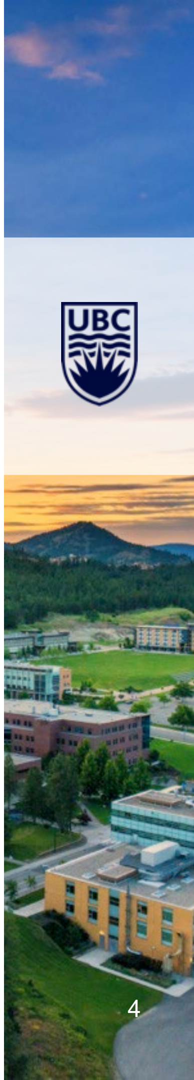
- Piazza private threads need TLC
    - **Weekly updates due each Monday @ 23:59 PT**

Instructor Office Hours:

- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 14:00-15:00

TA Office Hours:

- Eric: Friday 9-11 am (in-person and Zoom)
- Japraj: Wednesday 3-5 pm (Zoom)
- Yennis: Thursday 2-4 (Zoom), Friday 2-4 (in-person)

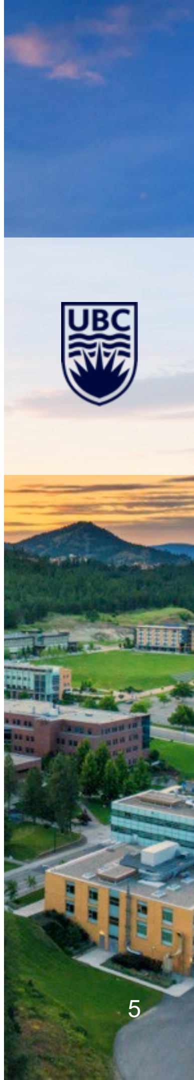


Readings

Required:

Recommended:

- Automatic verification of finite-state concurrent systems using temporal logic specifications.
- Abstracting the Geniuses Away from Failure Testing
- fpaxos/fpaxos-tlaplus: TLA+ specification of Flexible Paxos (github.com)

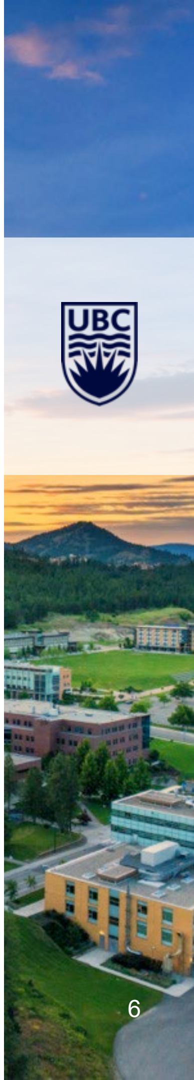


Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?



Today’s Failure

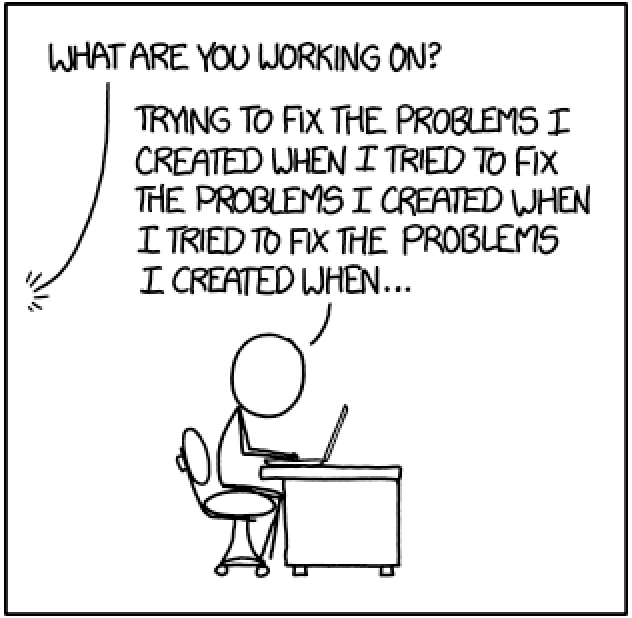
Today's Failure: Software Development

Distributed Systems Software Challenges:

- Not having a solid model
- Not *thinking about failure*
- Assuming explicit linearity

Biggest mistakes:

- Overlooking possible sources of error
- Writing code *before* you have a model
- Not using validation tools
- Assuming your testing is adequate



Lesson Goals

Formal Verification & TLA+

Introduction to Formal Verification

Introduction to TLA+

Formal Verification

Step 1: Create a formal model of the system of interest

- Hardware
- Protocols
    - Computer Bus
    - Communications
- Software
    - Mostly *concurrent* software

[Source]

Formal Verification

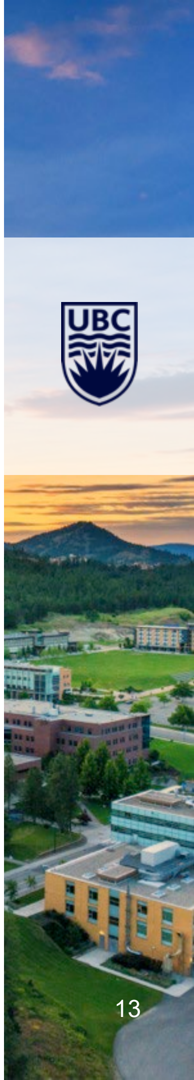
Step 2: Create a *formal specification*

- Identify specific properties required
- Identify ground truths ("invariants")

Formal Verification

Step 3: Validate your model

- Theorem proving (could be interactive)
- Model checking



Model Checking

Specifications are *Formulas*

- Formulas described using Temporal Logic

Programs are *Models*

- Abstracted as Finite State Machines

Interpretation ⊨ Formula

Given M is a set of interpretations and ϕ is a set of formulas the relationships are:

$$M \models \phi$$

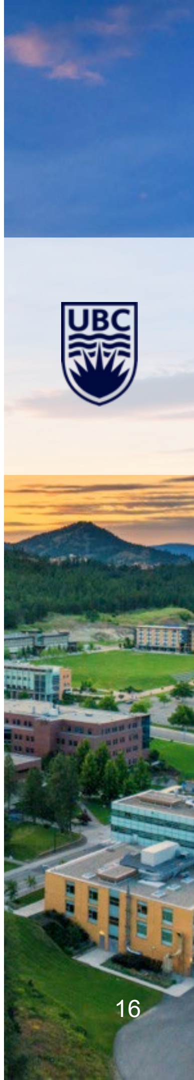
Or M entails ϕ. Thus, if one is true, so is the other.

Alternatively: M models ϕ

Interpretation $\models$ Formula

Questions:

- For a fixed ϕ is $M \models \phi$ true for all M
  - Is ϕ valid?
  - Can prove using a theorem prover (such as Isabelle)
- For a fixed ϕ is $M \models \phi$ true for some M
  - Satisfiable
- Given a fixed class of M what ϕs make $M \models \phi$ true?
  - Research
- For a fixed M and ϕ is it true that $M \models \phi$
  - Model checking



Model Checking

Many tasks can be cast as model checking

Interpretations M	$\models$	Formulas ϕ	Task
Token sequences	$\models$	Grammars	Parsing
Database tables	$\models$	SQL Queries	Query execution
Email texts	$\models$	Spam rules	Spam detection
Letter sequences	$\models$	Dictionary	Spell checking
Audio data	$\models$	Acoustic/lang. model	Speech recognition
Finite State Machines	$\models$	Temporal logic	Specification checking

Model Checking Examples

Model Checking has been used to:

- Check Microsoft Windows device drivers for bugs
  - The “Static Driver Verifier” tool
- The SPIN tool (http://spinroot.com):
  - http://spinroot.com/spin/success.html
  - Flood control barrier control software
  - Call processing software at Lucent
  - Parts of Mars Science Laboratory
- PEPA (Performance Evaluation Process Algebra)
  - Multiprocessor systems
  - Biological systems

Models for Model Checking

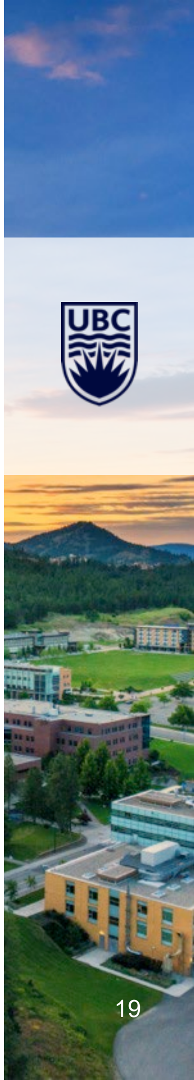
A model of some system has:

- A finite set of states
- A subset of states considered as the initial states
- A transition relation which, given a state, describes all states that can be reached “in one time step”.

Good for

- Software, sequential and concurrent
- Digital hardware
- Communication protocols

Refinements of this setup can handle: Infinite state spaces, Continuous state spaces, Continuous time, Probabilistic Transitions. Good for hybrid (i.e., discrete and continuous) and control systems



Models for Model Checking

Models are always abstractions of reality.

- We must choose what to model and what not to model
- There will limitations forced by the formalism
  - e.g., here we are limited to finite state models
- There will be things we do not understand sufficiently to model
  - e.g., people

Model Checking: Specifications

We are interested in specifying behaviours of systems over time.

- Use Temporal Logic

Specifications are built from:

1. Primitive properties of individual states e.g., “is on”, “is off”, “is active”, “is reading”;
2. Propositional connectives ∧, ∨, ¬, →; and
3. Temporal connectives:

   e.g., At all times, the system is not simultaneously reading and writing.

   If a request signal is asserted at some time, a corresponding grant signal will be asserted within 10 time units.

Model Checking: Specifications

The exact set of temporal connectives differs across temporal logics.

Logics can differ in how they treat time:

- Linear time vs. Branching time

These differ in reasoning about non-determinism.

Non-determinism

In general, system descriptions are non-deterministic.

A system is non-deterministic when, from some state there are multiple alternative next states to which the system could transition.

Non-determinism is good for:

- Modelling alternative inputs to the system from its environment (External non-determinism)
- Under-specifying the model, allowing it to capture many possible system implementations (Internal non-determinism)

Linear versus Branching Time

Linear Time

- Considers paths (sequences of states)
- If system is non-deterministic, many paths for each initial state
- Questions of the form:
  - For all paths, does some path property hold?
  - Does there exist a path such that some path property holds?

LTL Syntax

LTL = Linear(-time) Temporal Logic

Assume some set Atom of atomic propositions Syntax of LTL formulas ϕ: $\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid X\phi \mid F\phi \mid G\phi \mid \phi U \phi$ where $p \in$ Atom. Pronunciation: ▸ $X\phi$ — neXt ϕ ▸ $F\phi$ — Future ϕ ▸ $G\phi$ — Globally ϕ ▸ $\phi U \psi$ — ϕ Until ψ Other common connectives: W (weak until), R (release). Precedence high-to-low: $(X, F, G, \neg),(U),(\wedge, \vee), \rightarrow$

Linear versus Branching Time

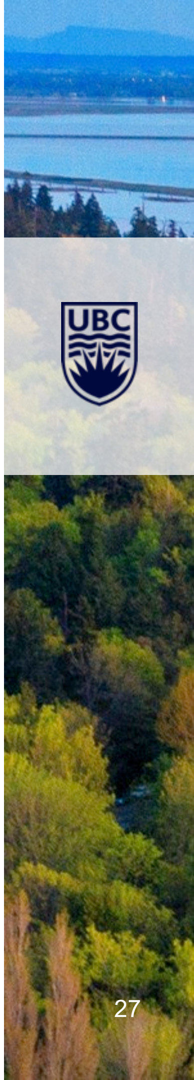
Branching Time

- Considers tree of possible future states from each initial state
- If system is non-deterministic from some state, tree forks
- Questions can become more complex, e.g.,
  - For all states reachable from an initial state, does there exist an onwards path to a state satisfying some property?
- Most-basic branching-time logic (CTL) is complementary to most-basic linear-time logic (LTL)
- Richer branching-time logic (CTL$*$) incorporates CTL and LTL

LTL – Informal Semantics

LTL formulas are evaluated at a position i along a path π through the system (a path is a sequence of states connected by transitions)

- An atomic p holds if p is true for the state at position i.
- The propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$ have their usual meanings.
- Meaning of LTL connectives:
  - $X\phi$ holds if ϕ holds at the next position;
  - $F\phi$ holds if there exists a future position where ϕ holds;
  - $G\phi$ holds if, for all future positions, ϕ holds;
  - $\phi U\psi$ holds if there is a future position where ψ holds, and ϕ holds for all positions prior to that.



What is TLA+

Given a model for a distributed system:

- Describe a model
- Verify the expected behaviour

Uses:

- Distributed Database
- Network Protocols

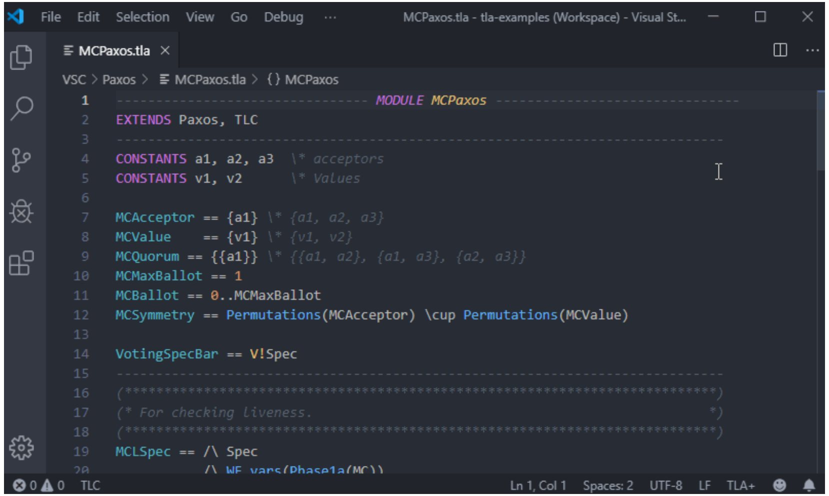
Benefit:

- Allows you to *reason* about the entire system
- Explore different scenarios
- Use “search spaces” to expand test effectivenss

Why Use TLA+

Benefits

- Allows formal verification
  - Formal, precise system description
  - Permits automated reasoning
- Early error detection
- Higher confidence in the system
- Flexibility
  - Expressive language
  - Can model complex systems
  - Modular
  - Easy management



TLA+ versus Other Formal Methods

Alternatives:

- Model Checking
- Theorem Proving
- Abstract interpretation
- Petri Nets
- Z Notation

Advantages of TLA+:

- Easy to learn/simple
- Expressive
- Scalable
- Versatile

TLA+ Syntax & Semantics

Uses mathematical notation

- Precise
- Human-readable

Specifications are express using:

- State variables: state of system at any given time
- State transitions: mutation of state based on inputs
- Temporal properties: invariant conditions that must be true over time

Specifications are composable: allows modular decomposition

TLA+ toolset:

- TLA+ toolbox
- TLC model checker
- PlusCal language

TLA+ Case Studies

[Amazon Web Services]

[Azure: Cosmos DB Service]

[Train Management]

[Uber: Driver/Rider matching]

TLA+ Resources & Learning Materials

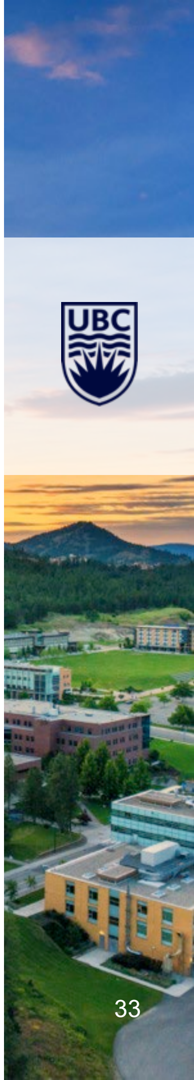
[TLA+ Homepage]

[TLA+ Video Course]

[TLA+ Google Group]

[TLA+ Examples on Github]

[TLA+ Toolbox]



Lesson Review

What we covered

Introduction to Formal Verification

Introduction to TLA+

Questions?


UBC
THE UNIVERSITY OF BRITISH COLUMBIA
THE UNIVERSITY OF BRITISH