# Lecture 3 - January 17

## Introduction, Math Review

***Model-Based Development***
***TLA+***
***Logical vs. Programming Operators***

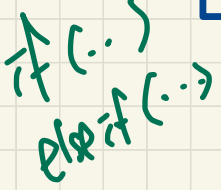
## Announcement

- `Lab1` released
  + tutorial videos
  + problems to solve

# Software Development Process

Lab1 choice {...} or {...}

↳ non-deterministic op.




**REQUIREMENT**

- Natural Language
  (incomplete, ambiguous, contradicting)
- Requirement Elicitation

**DESIGN**

- Blueprints
- Not necessarily executable & testable

3342: Event-B model
4315: TLA+ (Pluscal dialect)

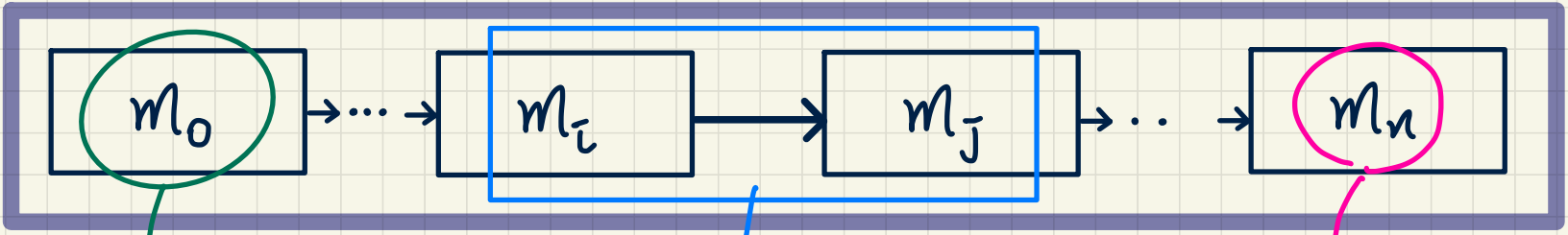
**IMPLEMENTATION**

- API Given
- Efficient (data structures & algorithms)
- Unit Tests

**RELEASE**

- Customer's Acceptance
- Recall?

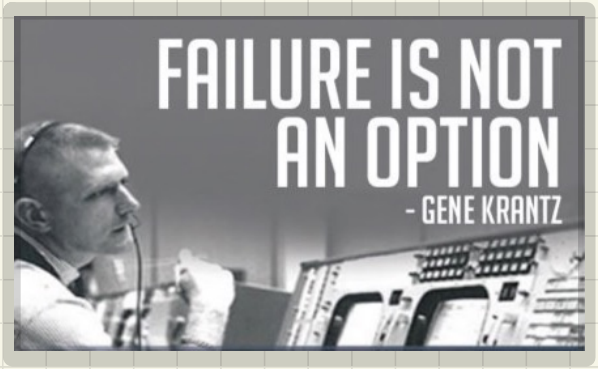
# Correct by Construction




simplest, most abstract model

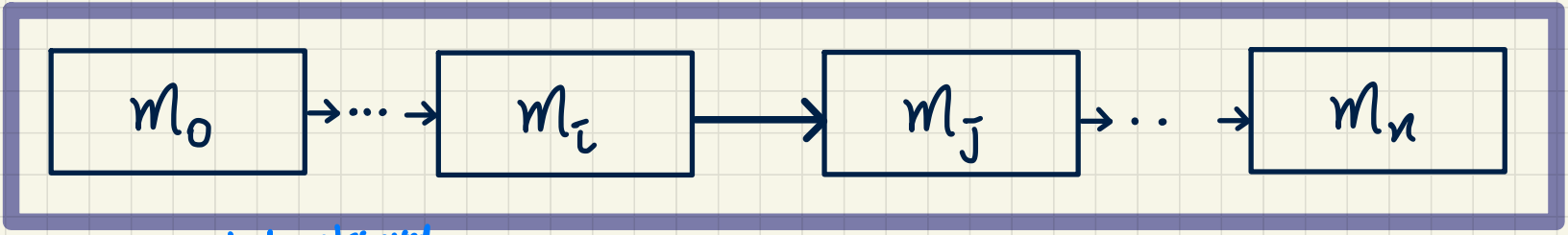
- $m_i$ is more abstract than $m_j$
- $m_j$ is more concrete than $m_i$.

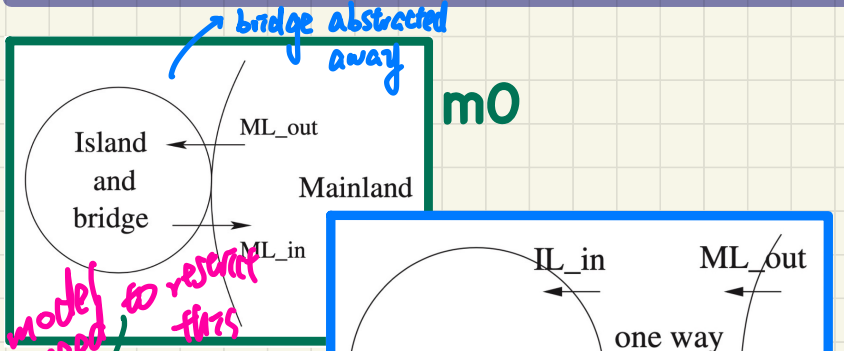
most sophisticated, most concrete model (closest to call) → variables

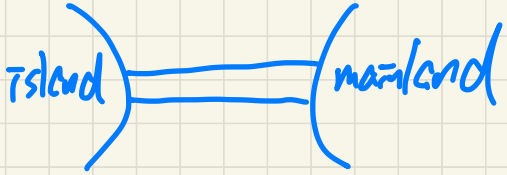



Source: https://audiobookstore.com/audiobooks/failure-is-not-an-option-1.aspx

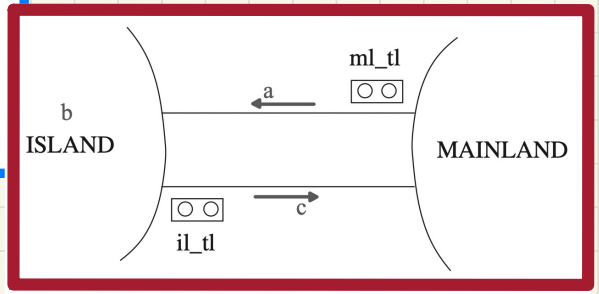
# Correct by Construction: Bridge Controller System



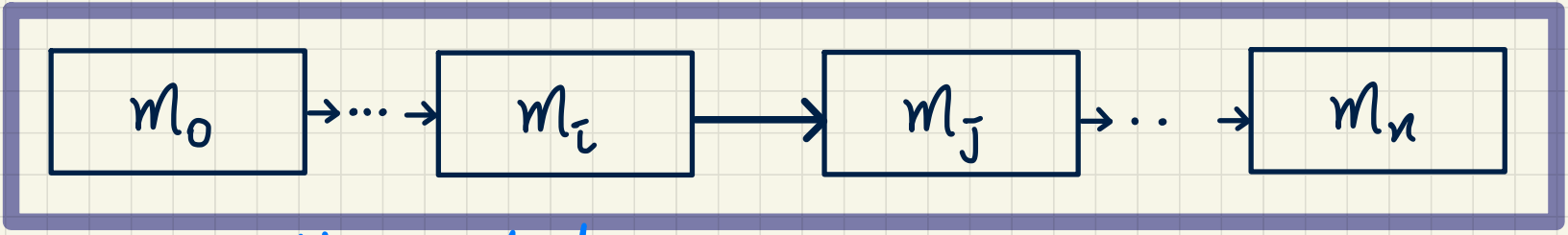




1. bridge 2. traffic light 3. flow in both directions

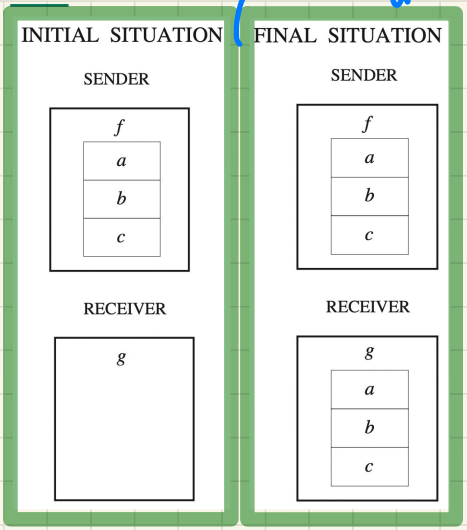


# Correct by Construction: File Transfer Protocol

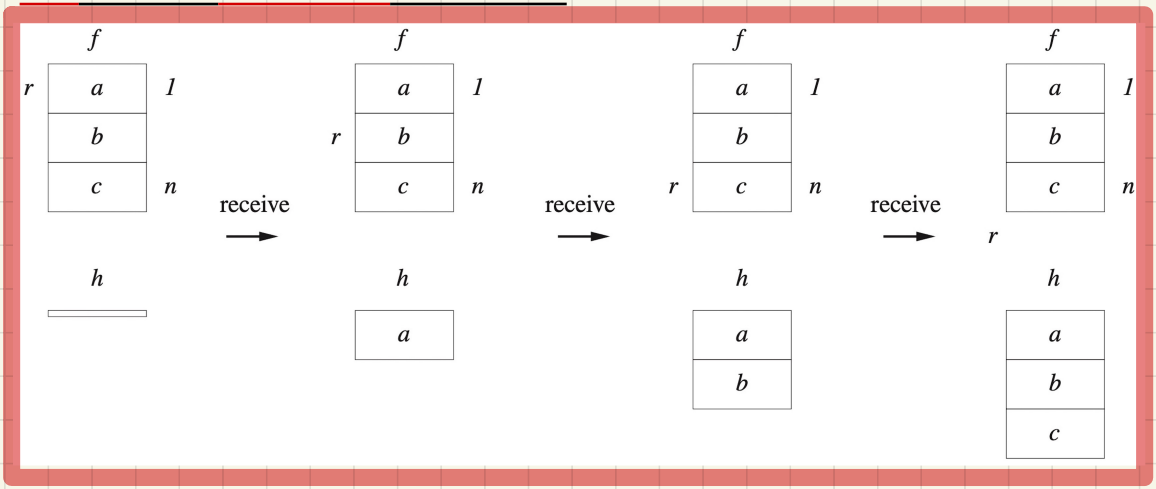


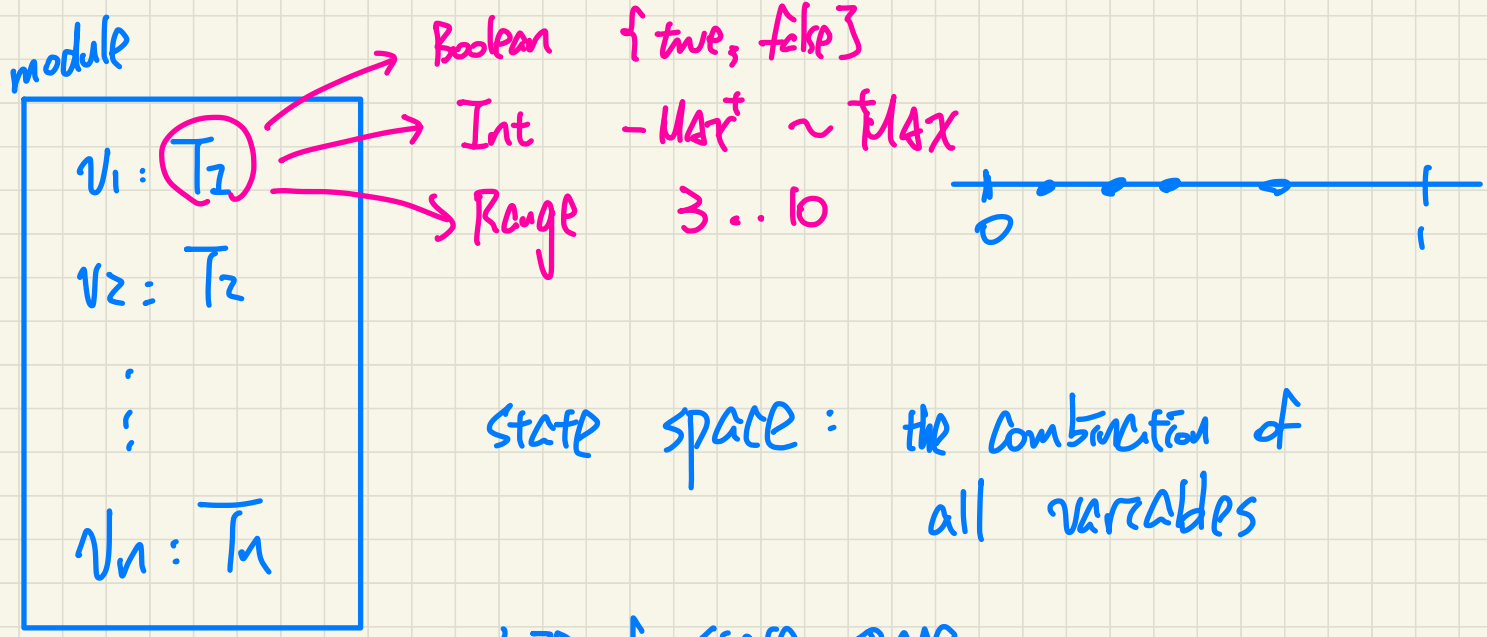

## m0

delay in transmission abstracted away




## m1: more concrete than m0






state space: the combination of all variables

size of state space:

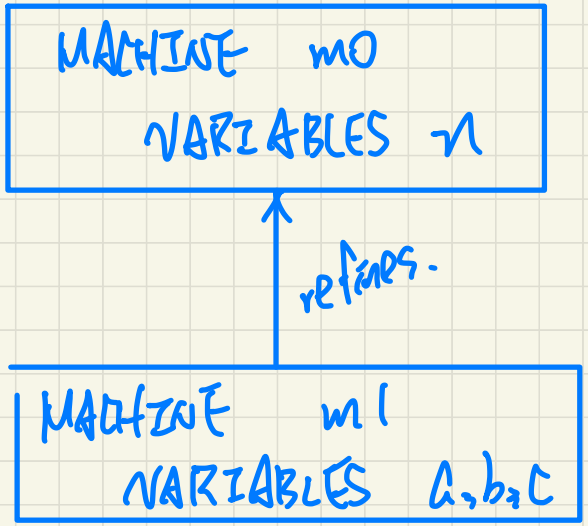
$$|T_1| \times |T_2| \times \cdots \times |T_n|$$

model checkers in general do not support verification on real numbers.

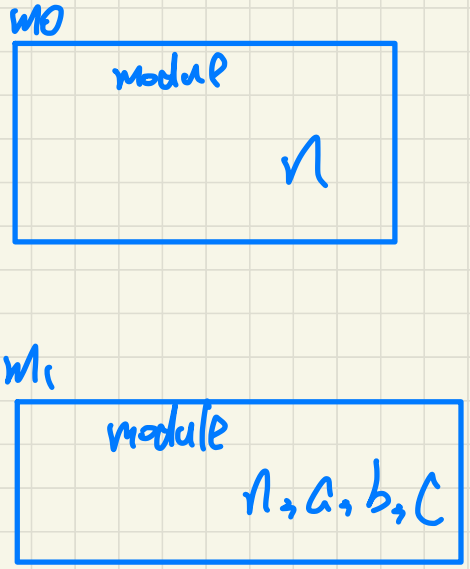
e.g. $R = 0..1$ finite (discrete) range: 2 possibilities.

e.g. $R = 0.0..1.0$ infinite → continuous

## Rodin ( refinement tool )




## TLA+ toolbox ( no notion of refinement)



# TLA+ Toolbox

*TLA+* (***Temporal Logic of Actions***) *is a* ***high-level language*** *for modeling programs and systems–especially concurrent and distributed ones. It's based on the idea that the best way to describe things precisely is with* ***simple mathematics***. *TLA+ and its tools are useful for eliminating fundamental* ***design errors***, *which are hard to find and expensive to correct in code.*

TLA+ is a language for modeling ***software*** above the code level and ***hardware*** above the circuit level.
It has an ***IDE*** (Integrated Development Environment) for writing models and running tools to check them. The tool most commonly used by engineers is the ***TLC model checker***, but there is also a proof checker.
TLA+ is based on mathematics and does not resemble any programming language. Most engineers will find *PlusCal*, described below, to be the easiest way to start using TLA+.

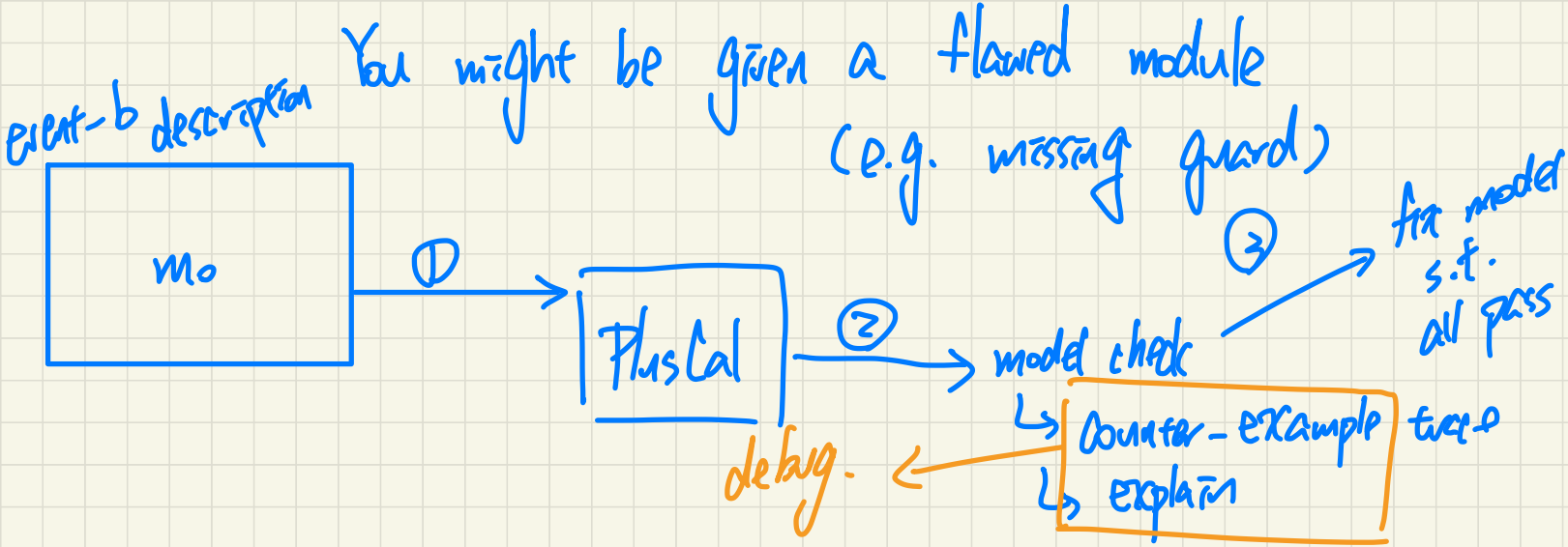
↓ C-like design language.

# Prog Test

- data sheet
- PDF sheet

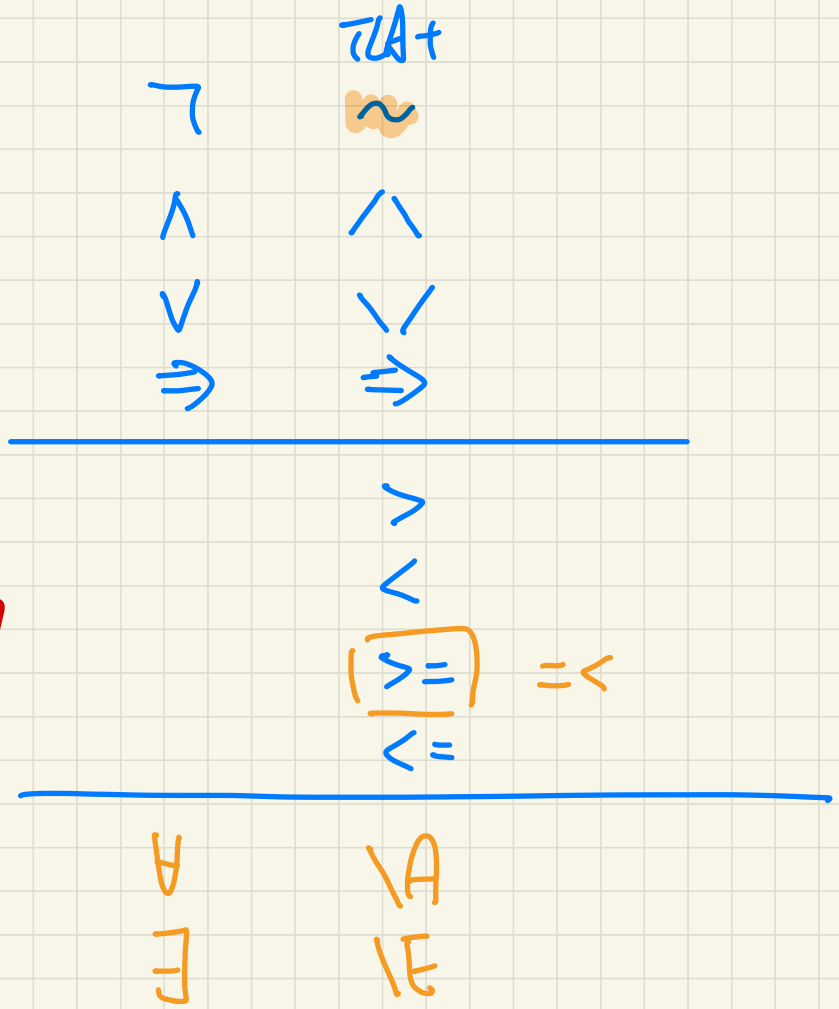
format: write PlusCal algorithms & properties
↳ inv. or temporal

You might be given a flawed module (e.g. missing guard)



# Lecture 1b

***Review on Math***



# Logical Operator vs. Programming Operator

| $p$ | $q$ | $p \wedge q$ | $p \vee q$ |
|---|---|---|---|
| true | true | true | true |
| true | false | false | true |
| false | true | false | true |
| false | false | false | false |

$((p \wedge q) \wedge r)$

Q. Are the $\wedge$ and $\vee$ operators equivalent to, respectively, && and || in Java?

e1 && e2

↳ if evaluated to (F)

e2 will not be evaluated ⇒ overall result : (F)

e1 || e2

↳ if evaluated to (T)

e2 will not be evaluated ⇒ result : (T)

precedence?

Array A : int[]

logic: well-definedness

Pc

i < A.length && A[i] >= 10 && i >= 0

(T)

A[-2]

i < 0

e.g. i == -2

ARRAY.