Lecture 4 - January 15

Introduction, Math Review

Counter Problem: Model Checking
Reachability Graph
Commutativity vs. Short-Circuit Eval.

Announcements/Reminders

- Lab1 released
- TA contact information (on-demand for labs) on eClass
- I will attend tomorrow's scheduled lab session
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu

Formulate Proof Obligation (PO): inc/inv0_1/INV

(inv0_1 → inv0_2)

pre —inc→ post

inc ↓ action: before-after predicate

$c' = c + 1$

Axioms
Invariants
Guard of inc
⊢
inv0_1 preserved
holds true in post state

$MIN \in \mathbb{N}$
$MAX \in \mathbb{N}$
$MIN \leq MAX$) axioms

$c \in \mathbb{N}$
$MIN \leq c \wedge c \leq MAX$) INV

$c < MAX$) guard of inc

⊢

$MIN \leq c+1 \wedge c+1 \leq MAX$

$MIN \leq c' \wedge c' \leq MAX$ (with c' replaced by $c+1$ in each place)

Proof

$MIN \in \mathbb{N}$
$MAX \in \mathbb{N}$
$MIN \leq MAX$
$C \in \mathbb{N}$
$MIN \leq C \wedge C \leq MAX$
$C < MAX$
$\vdash$
$MIN \leq C+1 \wedge C+1 \leq MAX$

MON, ARI ↳ AND_L

$MIN \leq C$
$C \leq MAX$
$C < MAX$
$\vdash$
$MIN \leq C+1 \wedge C+1 \leq MAX$

AND_R, MON

$MIN \leq C \vdash MIN \leq C+1$ ..

$C < MAX \vdash C+1 \leq MAX$ ..

(R1) When PO is discharged/proved, any combination of MIN and MAX

(R2) No concern of state explosion (e.g. MAX−MIN is huge)

Model Checking: Algorithmic Approach via Exhaustive Search

Invariant: 0 2 ✓ primarily Depth First Search

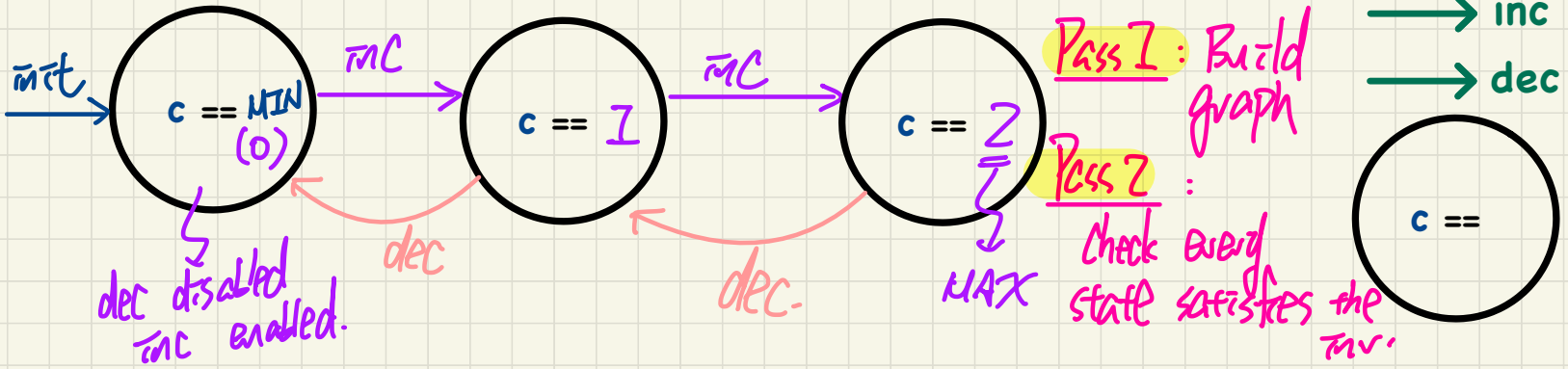
MIN_VALUE <= c <= MAX_VALUE

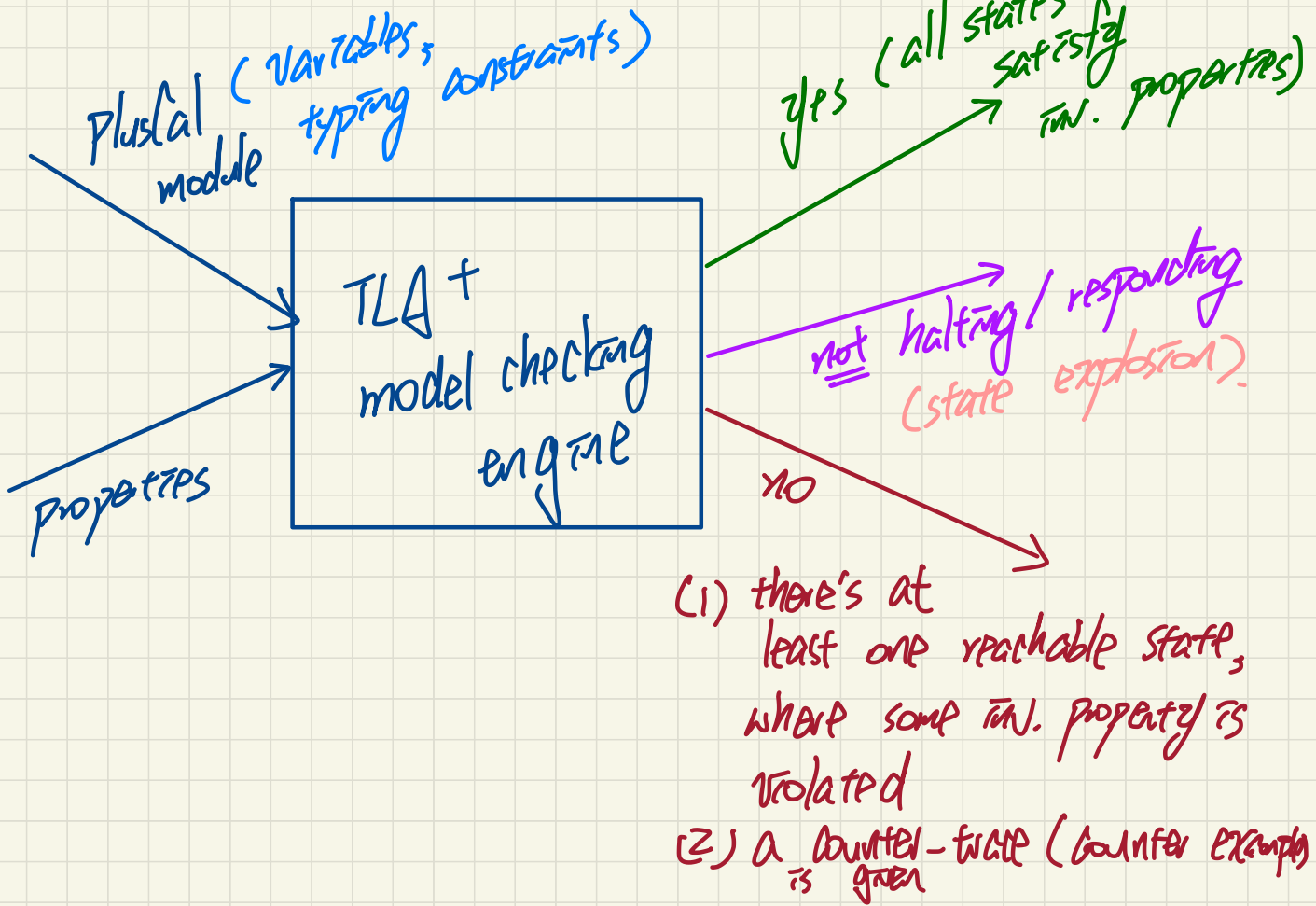
state space should be: (1) finite (2) reasonably small

Definition: A **reachability graph** includes **all** states reachable, via occurrences of **enabled** events, from the **initial** state.

Q: Given variables, the initial state, and the set of possible events, how can a RG be **automatically** generated?

To reach out as far as we can, in each reachable state, consider all transitions that are enabled.




PlusCal module
(variables, typing constraints)
properties
TLA+ model checking engine
yes (all states satisfy inv. properties)
not halting / responding (state explosion).
no
(1) there's at least one reachable state, where some inv. property is violated
(2) a counter-trace (counter example) is given

TLA+ Toolbox

Leslie Lamport
↳ LaTex

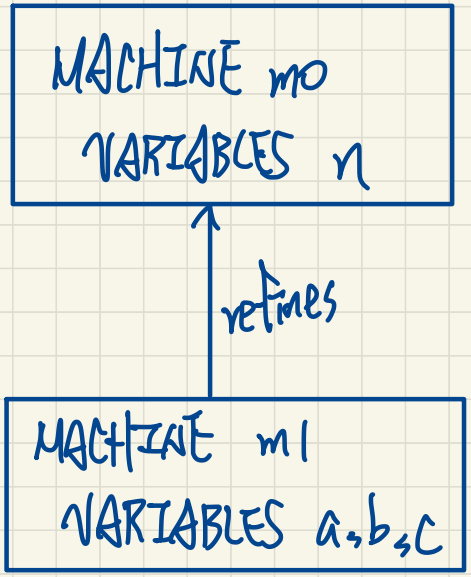
TLA + (***Temporal Logic of Actions***) *is a* ***high-level language*** *for modeling programs and systems–especially concurrent and distributed ones. It's based on the idea that the best way to describe things precisely is with* ***simple mathematics***.
TLA+ and its tools are useful for eliminating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

ProgTests

3342

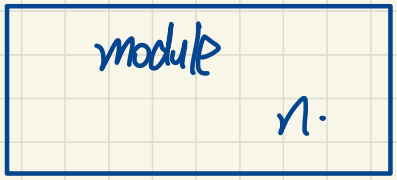
Rodin (refinement tool)



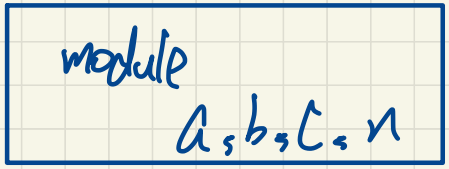
4315

TLA+ toolbox (no notion of refinement)

m0



m1



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

Commutativity (math)

$p \wedge q \equiv q \wedge p$

$p \vee q \equiv q \vee p$

Programming

$p \;\&\&\; q \not\equiv q \;\&\&\; p$

$p \;||\; q \not\equiv q \;||\; p$

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

$p \;||\; q$

$p \;\&\&\; q$: Evaluation left to right

↳ (C1) p evaluates (T) → still need to eval q

(C2) p evaluate (F) → skip eval. of q → overall (F)

(T)

(T).

(1) `i < a.length && a[i] >= 10 && i >= 0`

(2) `i >= 0 && a[i] >= 10 && i < a.length`

(Q1) Does it work always?

(Q2) YES → show
NO → give a counter-scenario where failure occurs.