EECS4315 Mission-Critical Systems

Lecture Notes

Winter 2025

Jackie Wang

Lecture 1 - January 6

Syllabus & Introduction

Safety-Critical Systems
Verification vs. Validation
Theorem Proving vs. Model Checking
TLA+

Course Learning Outcomes (CLOs)

CLO1 Explain the importance of safety-, mission-, business-, and security-critical systems.

CLO2 Demonstrate knowledge of the importance of good software engineering practices for critical systems.

CLO3 Use rigorous software engineering methods to develop dependable software applications that are accompanied by certification evidence for their safety and correctness.

CLO4 Demonstrate knowledge of the method and tools using deductive approaches (such as theorem proving).

CLO5 Demonstrate knowledge of methods and tools for algorithmic approaches (such as model checking, bounded satisfiability) etc.

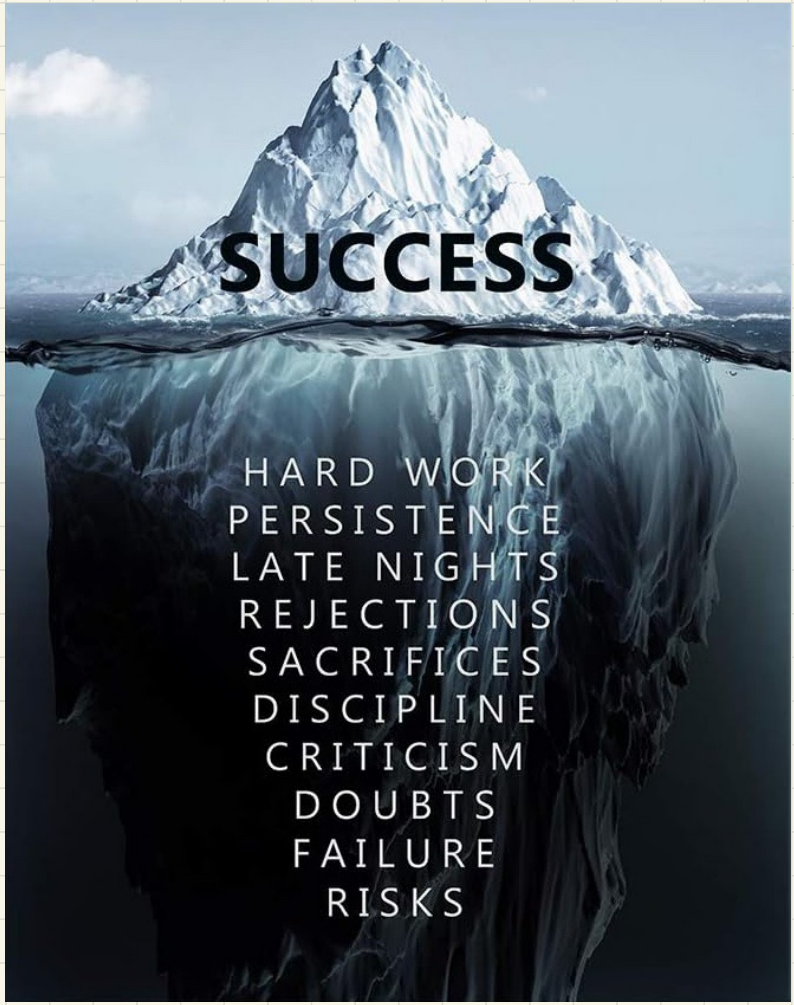
CLO6 Demonstrate knowledge of the theory underlying deductive and algorithmic approaches.

CLO7 Use industrial strength tools associated with the methods on large systems.

Exam
- ↳ 3 hours
- ↳ cumulative
- ↳ allowed to use a data sheet
- ↳ mostly short answer questions (no multiple choices)

Programming Tests
- ① submitted "code" should compile
- ↳ in-person
- ↳ TLA+ toolbox (Lab1) Pluscal (partial marks possible)
- ② syntax guide available during test.
- ↳ similar to lab exercises

Written Tests
- ↳ mostly multiple choice (one or more correct answers).
- ↳ not cumulative
- ↳ eClass (in-person)

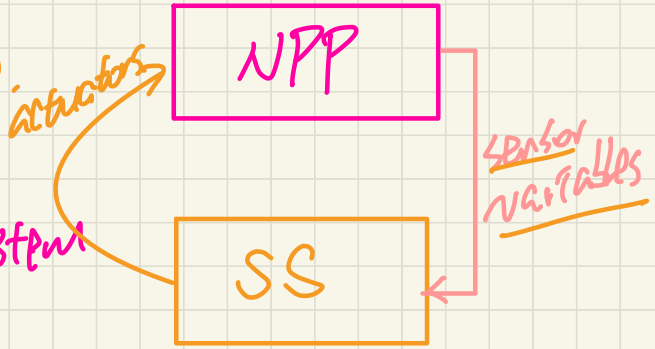


General Tips about Success

Source: https://a.co/d/aQ13fR1

Safety Critical Systems (SCS)

- ↳ auto-pilot / auto driving
- ↳ traffic light / train gate
- ↳ air bag deployment.
- ↳ elevator / escalator
- ↳ pacemaker (embedded system)
  - ↳ pacemaker challenge (McMaster Uni.)
- ↳ nuclear power plant
  - ↳ shutdown system



Lecture 2 - January 8

Introduction

Lab1 Guidance
Verification vs. Validation
Mission- vs. Safety-Critical Systems

Announcements/Reminders

- Lab1 released
- Scheduled lab session tomorrow at 9am
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- Trial attendance check via iClicker today!
- Slides on Math Review (Predicates) posted
- Notes template posted
- Monday lecture venue (R N203) unchanged

Acceptance Criteria

(1) Validation of **Requirements** ✓ → 4312.

Are we building the right product?

(a) **precise** → no scope of different interpretations by different engineers.

(b) **complete**

↳ all input scenarios should have some well-defined response by com. system

↳ • natural language (customer) poses big challenge.

• mathematics

↳ 3312: Event-b.

4315: TLA+ ↳ transition system.

solution: implement a compiler to turn them into the same semantic domain.

(2) Verification of **Implementation**

Are we building the product right?

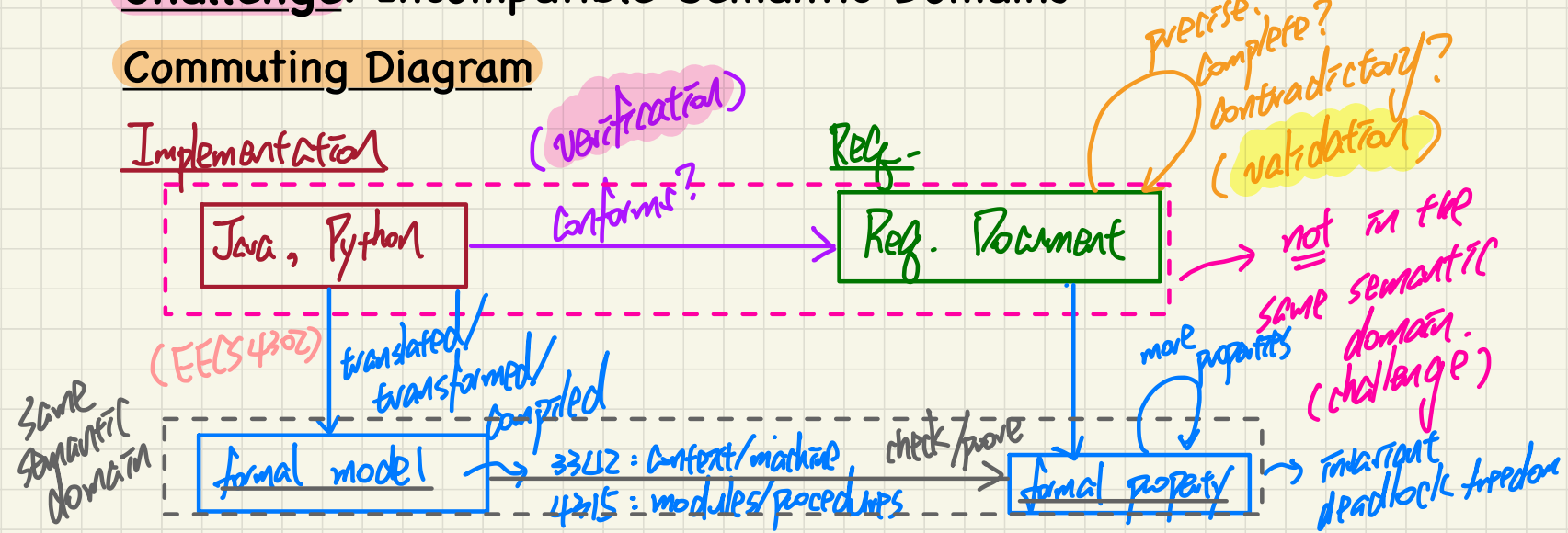
↳ **Req** vs. **Programs** belong to different domains

Goals: Verification vs. Validation

- Implementation (Java, Python)
- Requirements Document (Natural Language)
- Validity: Ambiguity, Incompleteness, Contradiction
- Compiler Technology (e.g., ANTLR4 @ EECS4302)

Challenge: Incompatible Semantic Domains

Commuting Diagram



Mission-Critical vs. Safety-Critical

Safety critical

When defining safety critical it is beneficial to look at the definition of each word independently. Safety typically refers to being free from danger, injury, or loss. In the commercial and military industries this applies most directly to human life. Critical refers to a task that must be successfully completed to ensure that a larger, more complex operation succeeds. Failure to complete this task compromises the integrity of the entire operation. Therefore a safety-critical application for an RTOS implies that execution failure or faulty execution by the operating system could result in injury or loss of human life.

Safety-critical systems demand software that has been developed using a well-defined, mature software development process focused on producing quality software. For this very reason the DO-178B specification was created. DO-178B defines the guidelines for development of aviation software in the USA. Developed by the Radio Technical Commission for Aeronautics (RTCA), the DO-178B standard is a set of guidelines for the production of software for airborne systems. There are multiple criticality levels for this software (A, B, C, D, and E).

These levels correspond to the consequences of a software failure:

- Level A is catastrophic
- Level B is hazardous/severe
- Level C is major
- Level D is minor
- Level E is no effect

safety critical

mission-critical

Safety-critical software is typically DO-178B level A or B. At these higher levels of software criticality the software objectives defined by DO-178B must be reviewed by an independent party and undergo more rigorous testing. Typical safety-critical applications include both military and commercial flight, and engine controls.

Mission critical

A mission refers to an operation or task that is assigned by a higher authority. Therefore a mission-critical application for an RTOS implies that a failure by the operating system will prevent a task or operation from being performed, possibly preventing successful completion of the operation as a whole.

Mission-critical systems must also be developed using well-defined, mature software development processes. Therefore they also are subjected to the rigors of DO-178B. However, unlike safety-critical applications, mission-critical software is typically DO-178B level C or D. Mission-critical systems only need to meet the lower criticality levels set forth by the DO-178B specification.

Generally mission-critical applications include navigation systems, avionics display systems, and mission command and control.

Source: http://pdf.cloud.opensystemsmedia.com/advancedtca-systems.com/SBS.Jan04.pdf

(C1) System S is mission-critical.

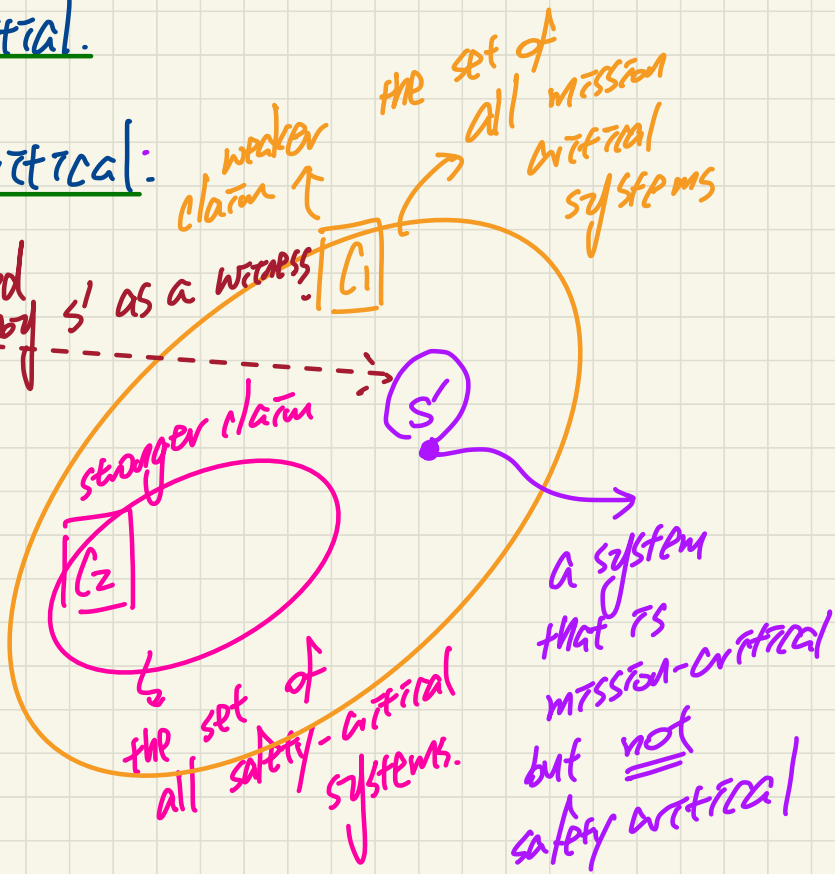
(C2) System S is safety-critical:

(1) $C_1 \Rightarrow C_2$ — not true in general

(2) $C_2 \Rightarrow C_1$

(3) $C_1 \Leftrightarrow C_2$

C_2 is a stronger claim than C_1 (fewer systems satisfying C_2 than systems satisfying C_1)



Lecture 3 - January 13

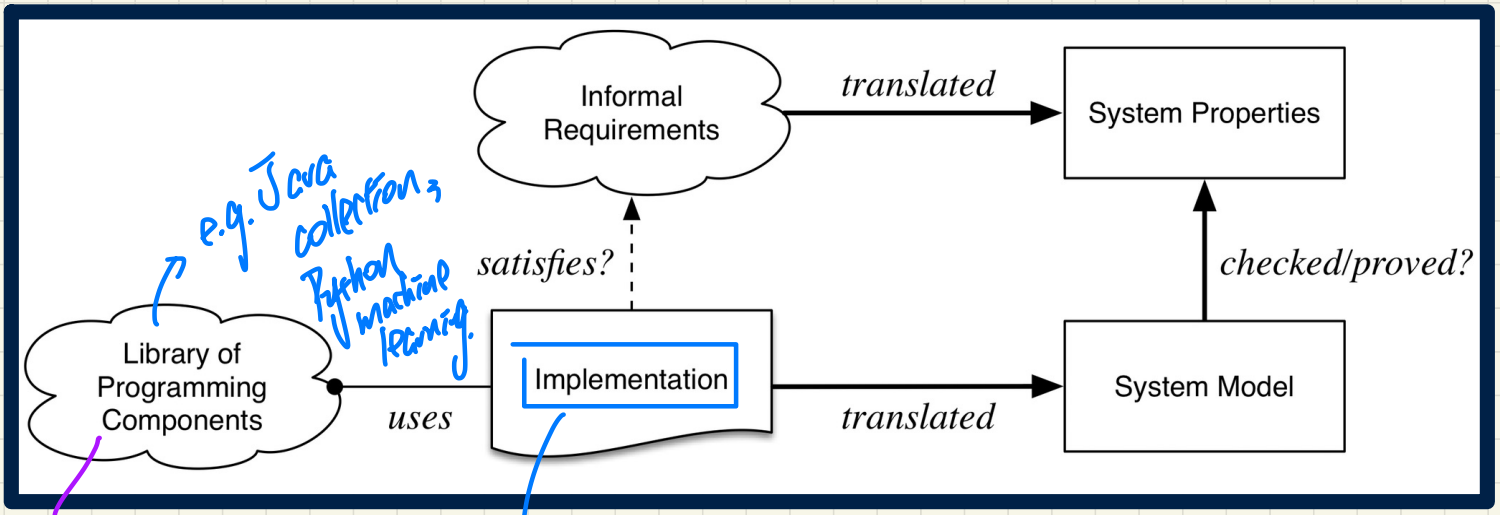
Introduction

Software Development Process
Assurance Cases
Correct by Construction
State Space
Counter Problem: Theorem Proving

Announcements/Reminders

- Lab1 released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu

Building the product right? Building the right product?



e.g. Java collection, Python machine learning.

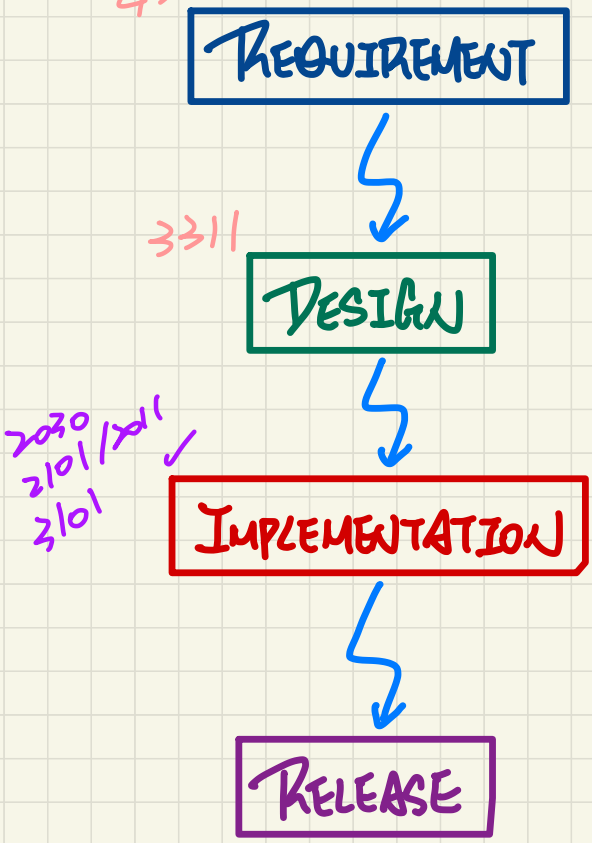
practically, not feasible to verify this component formally.
↳ instead: unit testing

is not necessarily built from scratch
⇒ library classes might be used

("don't re-invent the wheel" unless you really have to)

Software Development Process

the earlier to reveal a defect/bug, the less costly it is



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

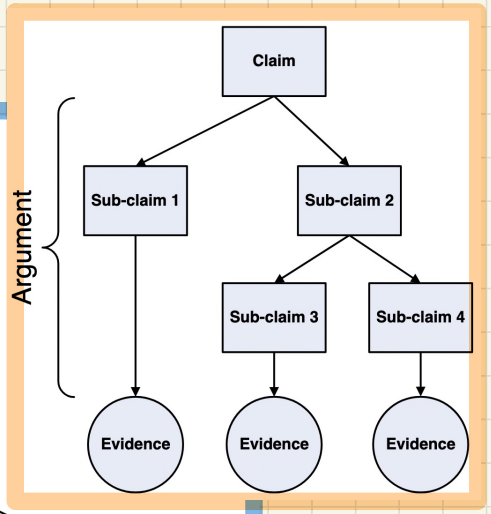
- Blueprints
- Not necessarily executable & testable

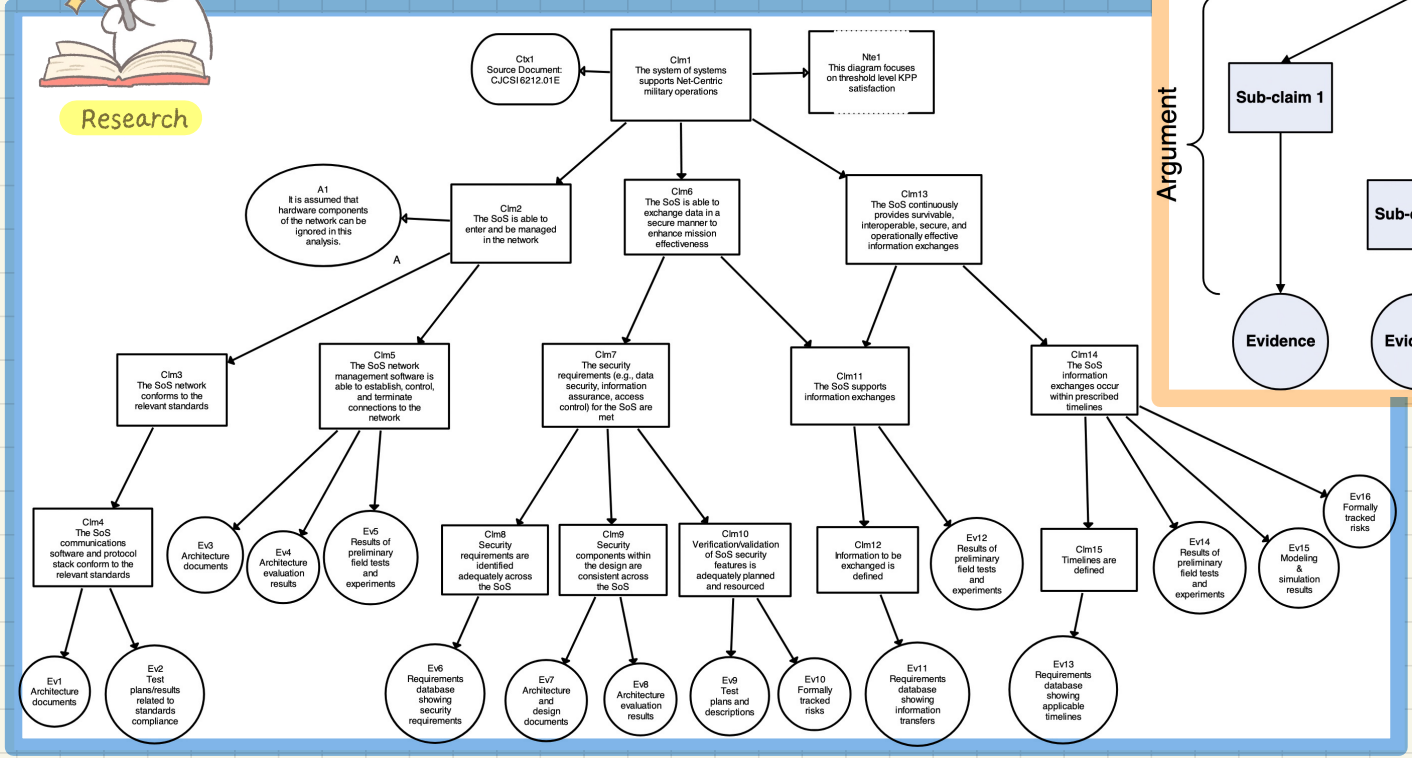
- API Given
- Efficient (data structures & algorithms)
- Unit Tests

- Customer's Acceptance
- Recall?

Certifying Systems: Assurance Cases

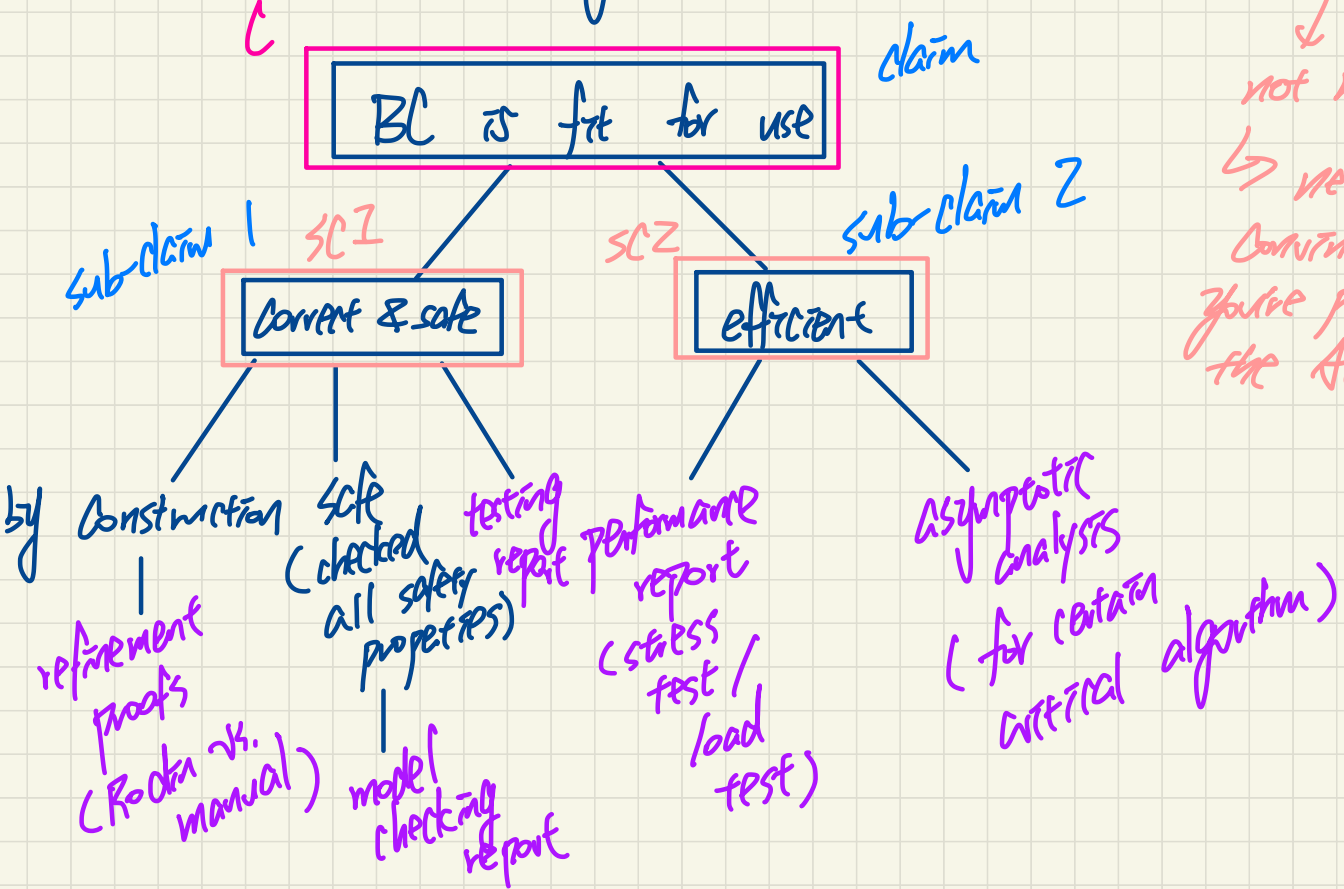
Research on "Assurance Cases" if interested!





Source: https://resources.sei.cmu.edu/asset_files/whitepaper/2009_019_001_29066.pdf

Assurance Case for Bridge Controller



$SC1 \wedge SC2 \Rightarrow C$

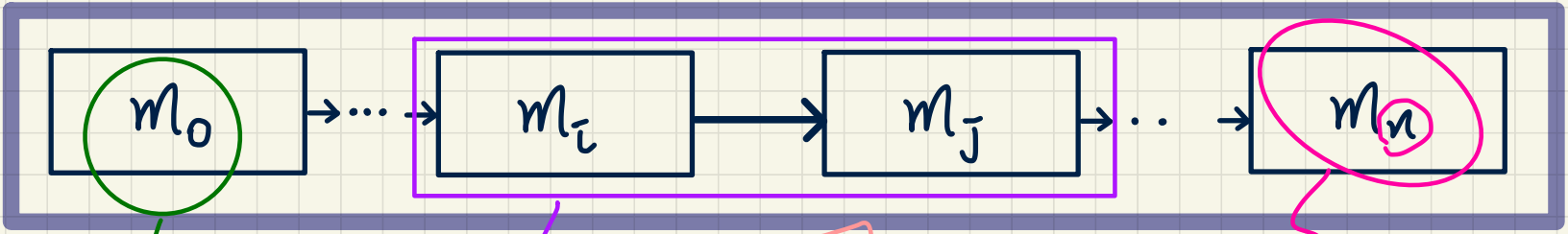
↓

not necessarily

↳ need to convince whoever you're presenting the AC to.

Correct by Construction

No no: a single model containing all details

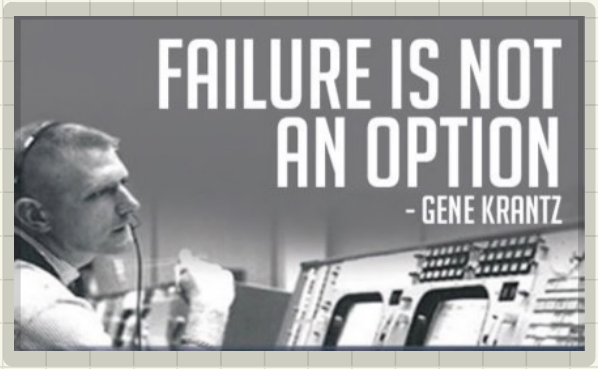


most abstract model

m_i is refined by m_j

↳ added with more details, variables.

most concrete model (closest to final implementation)

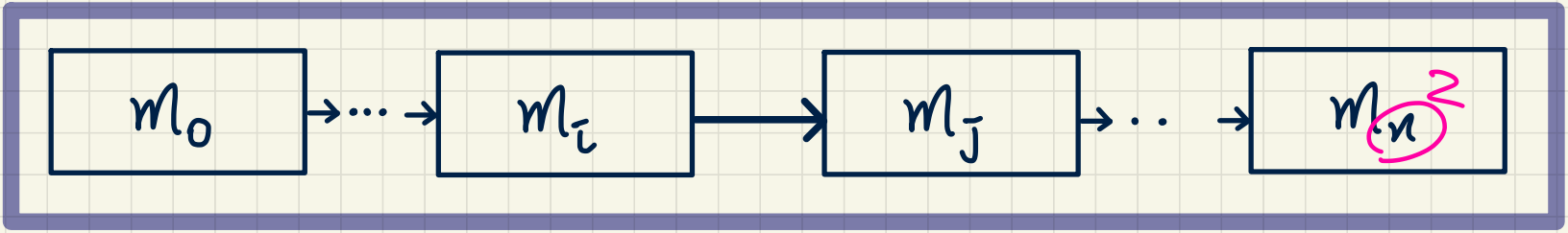


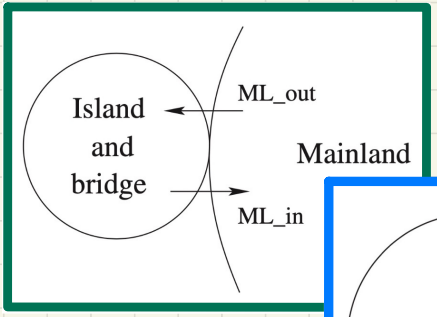
Proof obligations

1. deadlock freedom
2. invariant establishment & maintenance
3. livelock freedom (variant)

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

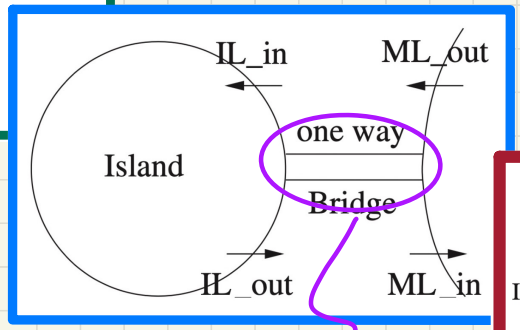
Correct by Construction: Bridge Controller System





m0

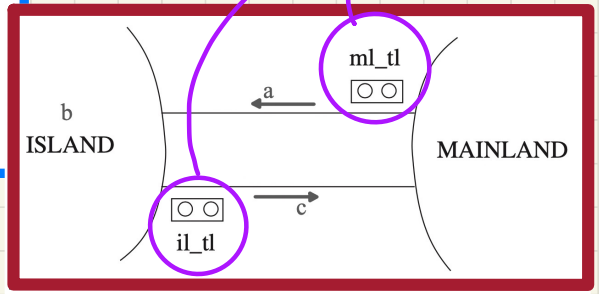
initial abstraction



m1

added concreteness

further added concreteness



m2

state explosion problem.

State Space of a Model

e.g. $C : \{1, 2, 3\}$ — type of constant

[valuations] of $C = |\{1,2,3\}| = 3$

Definition: The state space of a model is the set of all possible valuations of its declared constants and variables, subject to declared constraints.

the more valuations of variables/constants, the more computationally demanding it's to check properties

DFS on graph

upper bound

lower bound

Say an initial model of a bank system with two constants and a variable:

$c \in \mathbb{N}1 \land L \in \mathbb{N}1 \land accounts \in String \pfun \mathbb{Z}$ — set of all possible p.f. /* typing constraint */

$\forall id \bullet id \in \mathrm{dom}(accounts) \Rightarrow -c \leq accounts(id) \leq L$ /* desired property */

Q1. Give some example configurations of this initial model's state space.

$\langle C = 100,\ L = 500,\ accounts = \{\} \rangle$

$\langle C = 5000,\ L = 10000,\ accounts = \{\text{"bill"} \mapsto 6000\} \rangle$

$C = 200$, $L = 600$, owns = {"bill", "steve"}

Q2. How large exactly is this initial model's state space? infinite!

$C \in \mathbb{N}1$ → card is ∞

$L \in \mathbb{N}1$

$accounts \in String \pfun \mathbb{Z}$ — large # valuations, infinite

To be model checkable, need to restrict the state space. balance: interval.

Exercise: Theorem Proving vs. Model Checking

Variable: state space

An integer counter c

Safety Constraints:

MIN_VALUE <= c <= MAX_VALUE

Unconditional Update:

init: initializes c as zero assuming: 0 >= min.

Conditional Updates:

inc: increments c when ?? c < max

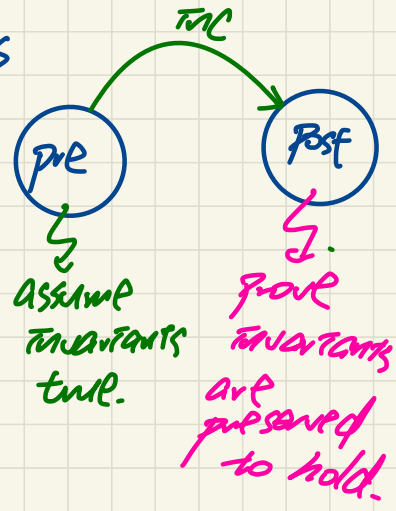
dec: decrements c when ?? c > min

Theorem Proving: Deductive Approach via Inference Rules

1. Formulate the model
2. Discharge proof obligations.

TMC / INV0_1 / INV
TMC / INV0_2 / INV

2. Proof Obligations



machine (dynamic)

1. CONSTANTS
   MIN, MAX
   AXIOMS
   MIN ∈ ℕ
   MAX ∈ ℕ
   MIN ≤ MAX

context (static)

VARIABLES
c

INVARIANTS
typing constraint
INV0_1 : c ∈ ℕ
INV0_2 : MIN ≤ c ∧ c ≤ MAX
safety constraint

EVENTS
TMC ✓
¬(c < MAX) → disabled
guard
enabling cond.
when c < MAX begin c := c+1 end

dec
when c > MIN begin c := c-1 end

Lecture 4 - January 15

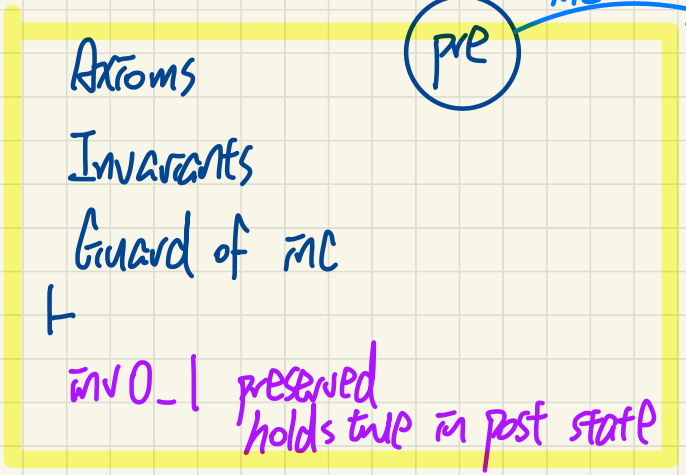
Introduction, Math Review

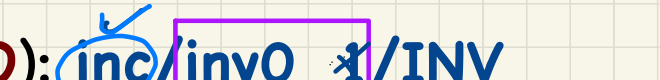
Counter Problem: Model Checking
Reachability Graph
Commutativity vs. Short-Circuit Eval.

Announcements/Reminders

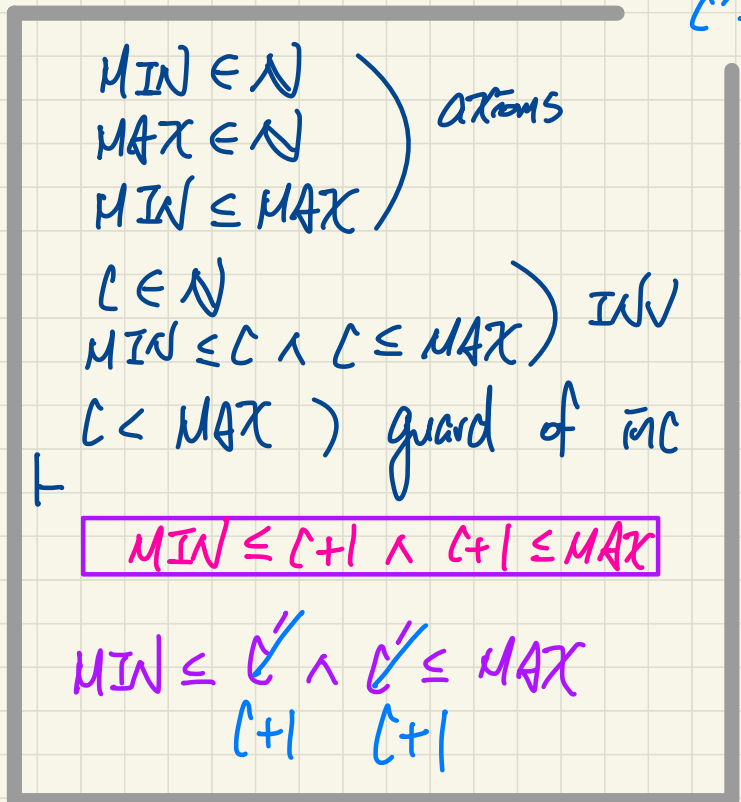
- Lab1 released
- TA contact information (on-demand for labs) on eClass
- I will attend tomorrow's scheduled lab session
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu

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





$c' = c + 1$



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

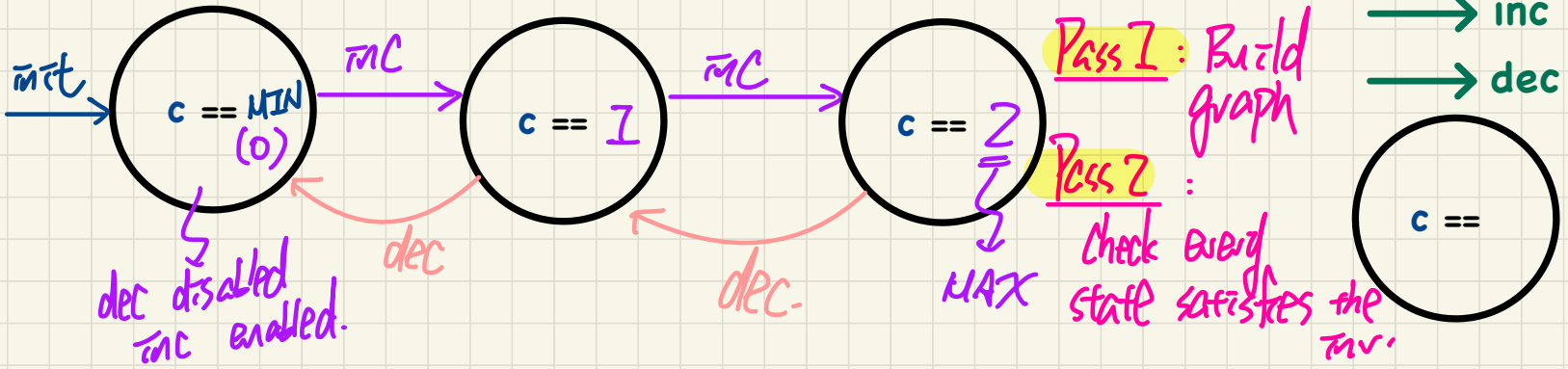
MIN_VALUE <= c <= MAX_VALUE

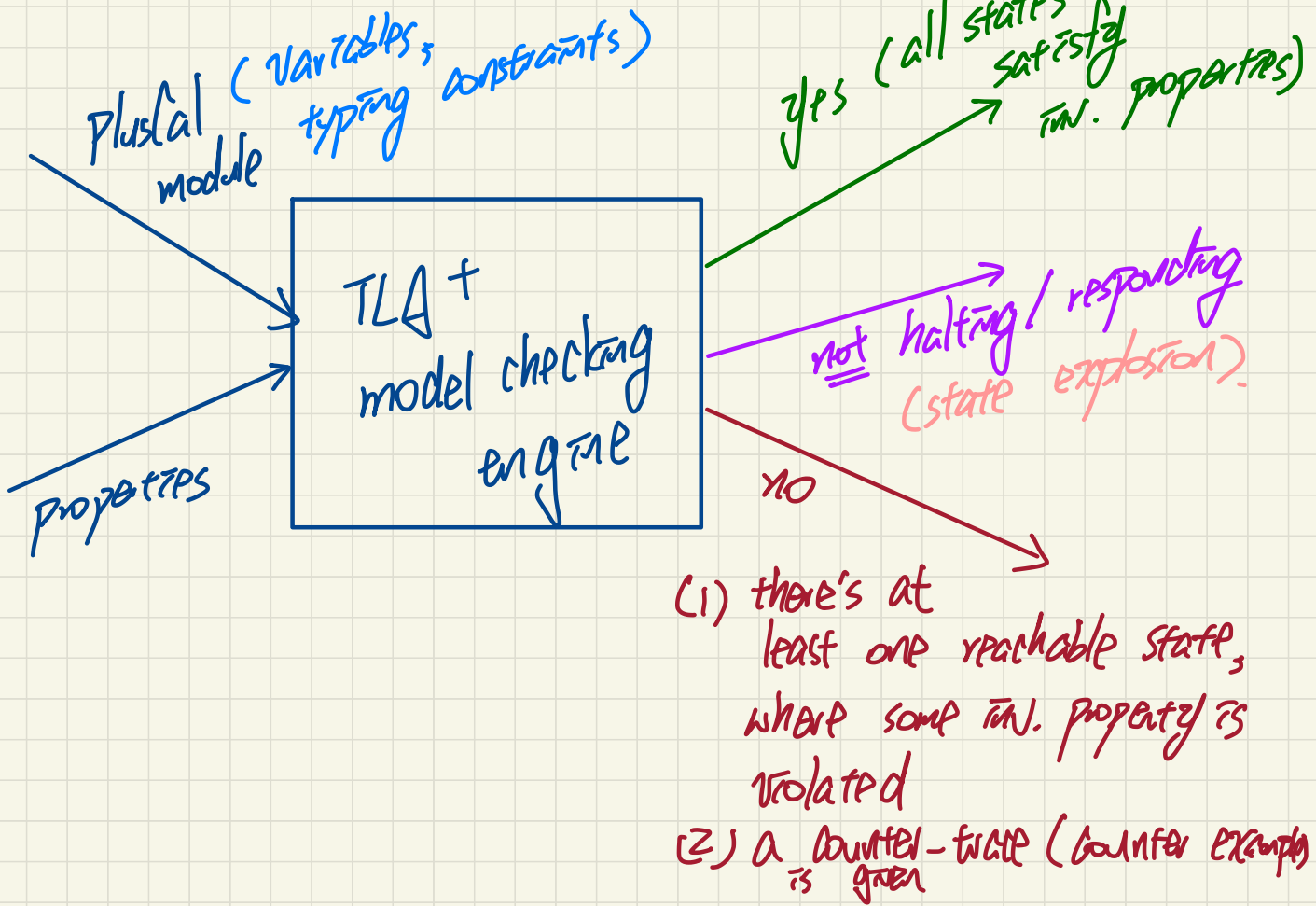
state space should be: (1) finite (2) reasonably small

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

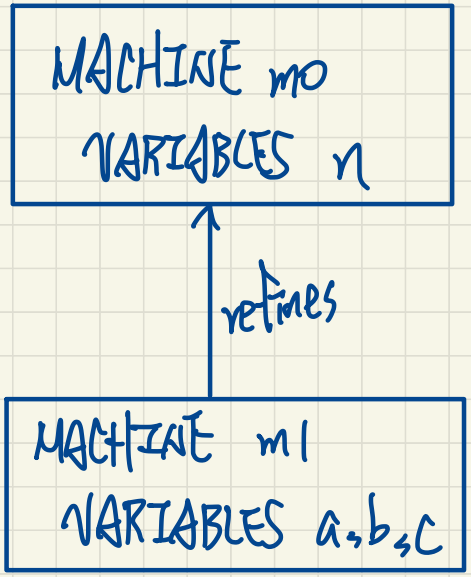
TLA + (***Temporal Logic of Actions***) *is a* ***high-level language*** *for modeling programs and systems–especially concurrent and distributed ones. It's based on the idea that the best way to describe things precisely is with* ***simple mathematics***.
TLA+ and its tools are useful for eliminating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

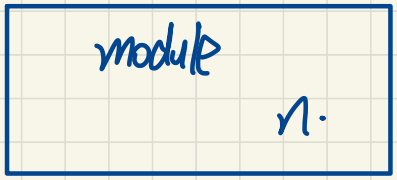
Rodin (refinement tool)



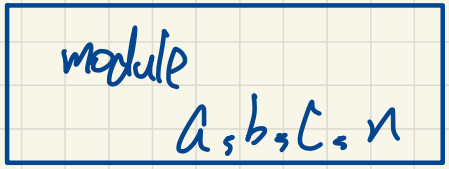
4315

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.

Lecture 5 - January 20

Math Review

Formulating the Model Checking Problem
Describing Implications
Theorems of Propositional Logic

Announcements/Reminders

- **Lab1** due this Thursday (Jan 23)
- **TA contact information** (on-demand for labs) on eClass
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu

Model Checking Problem

(1) Given module m and some invariant property P:

$$\forall s \cdot s \in RG(m) \Rightarrow P(s)$$

$RG(m)$: all reachable states of the system being modeled.

(We don't care about unreachable states.)

(2) When there's a counter-example: $\exists s \cdot s \in RG(m) \wedge \neg P(s)$

witness of inv. violation

counter trace: init $\Rightarrow$ ○ ⇝ (s)

a.length == 10

11 → SCE will not eval. a[i]

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

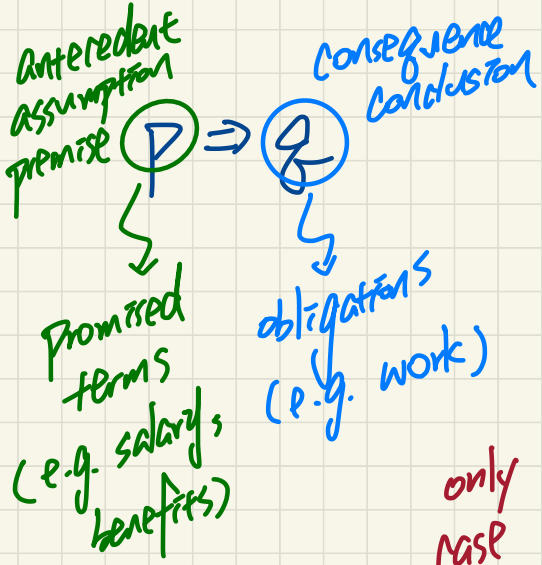
(2) i >= 0 && a[i] >= 10 && i < a.length (i = -1, i = 11)

(A1) Neither (1) nor (2) works all the time.

(A2) For (1), i >= 0 checked too late. e.g. i == -1 will trigger AIOBE.

For (2), i < a.length checked too late e.g. i == 11 will trigger AIOBE.

Implication ≈ Whether a Contract is Honoured



$P \Rightarrow q$ true if the contract is honoured (not breached)

			$P \Rightarrow q$
(C1)	P = T	q = T	T
(C2)	P = T	q = F	F
(C3)	P = F	q = T	T
(C4)	P = F	q = F	T

only case where ⇒ is false

only case where ⇒ is true non-trivially

Describing p ⇒ q

p: snow storm

q: cancel class

q if p, p is sufficient for q

p	q	p ⇒ q
true	true	true
true	false	false
false	true	true
false	false	true

when p is not T, don't care what q is.

given that p is T, q is also T

p is T but q does not follow that, so ⇒ is false

q unless ¬p

enumerates the two scenarios for p ⇒ q to be T

① q

② ¬p (p = false)

p	q	p ⇒ q
true	true	true
true	false	false
false	true	true
false	false	true

¬q ∧ p

p only if q, q is necessary for p

p	q	p ⇒ q
true	true	true
true	false	false
false	true	true
false	false	true

knowing that p is true, the only way to make p ⇒ q true is when q is true as well.

p ⇒ q ≡ ¬p ∨ q

Q. Which of the following expressions are equivalent to $p \Rightarrow q$

- [x] q if p
- [ ] q only if p → $q \Rightarrow p$

To Prove

$P \iff Q$

if, only if

Need to prove

(1) $P \Leftarrow Q$

$Q \Rightarrow P$ P if Q

(2) $P \Rightarrow Q$

P only if Q

Example: Inverse, Converse, Contrapositive

$$x > 0 \wedge x \leq 23 \Rightarrow y \geq 23 \vee y < 46$$

De Morgan

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

Contrapositive

$$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$$

Inverse

$$\neg(x > 0 \wedge x \leq 23) \Rightarrow \neg(y \geq 23 \vee y < 46)$$

equational style of proofs

$\equiv$ { De Morgan } justification

$$x \leq 0 \vee x > 23 \Rightarrow y < 23 \wedge y \geq 46$$

Converse

$$y \geq 23 \vee y < 46 \Rightarrow x > 0 \wedge x \leq 23$$

Contrapositive : converse of inverse

$$y < 23 \wedge y \geq 46 \Rightarrow x \leq 0 \vee x > 23$$

Identity

$0 + a = a$

$1 * a = a$

$T \Rightarrow p \equiv p$

$T \wedge p \equiv p$

$F \vee p \equiv p$

Zero

$F \Rightarrow p \equiv T$

$F \wedge p \equiv F$

$T \vee p \equiv T$

when the zero of $\Rightarrow$ is the antecedent, the value of consequence is fixed.

logical basis of short-circuit evaluation.

Precedence

$\neg$

$\wedge$

$\vee$

$\Rightarrow$

$\equiv$

$\neg p \wedge q \vee r$

$\mathbb{N}$

non-negative

natural numbers

↳ 0, 1, 2, 3 ...

$\mathbb{Z}$

integers

$-\infty, \dots, 0, \dots, +\infty$

Rodin

∀

(!) $x : P(x) \Rightarrow Q(x)$

(#) $x : P(x) \ \& \ Q(x)$

∃

TLA+/PlusCal

Range.

∀ (\A) x (\in) Nat (,) y \in Int (:) $P(x)$

\E x \in Nat , y \in Int (:) $P(x)$

∧

Lecture 6 - January 22

Math Review

∀ vs. ∃: Syntax, Meaning, Examples
∀ vs. ∃: Proof Strategies
Switching between ∀ and ∃

Announcements/Reminders

- **Lab1** due tomorrow (Jan 23)
- **Lab2** to be released next week (by Monday's class)
- **TA contact information** (on-demand for labs) on eClass
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu

Quantification

ProgTest/Labs : ASCII syntax

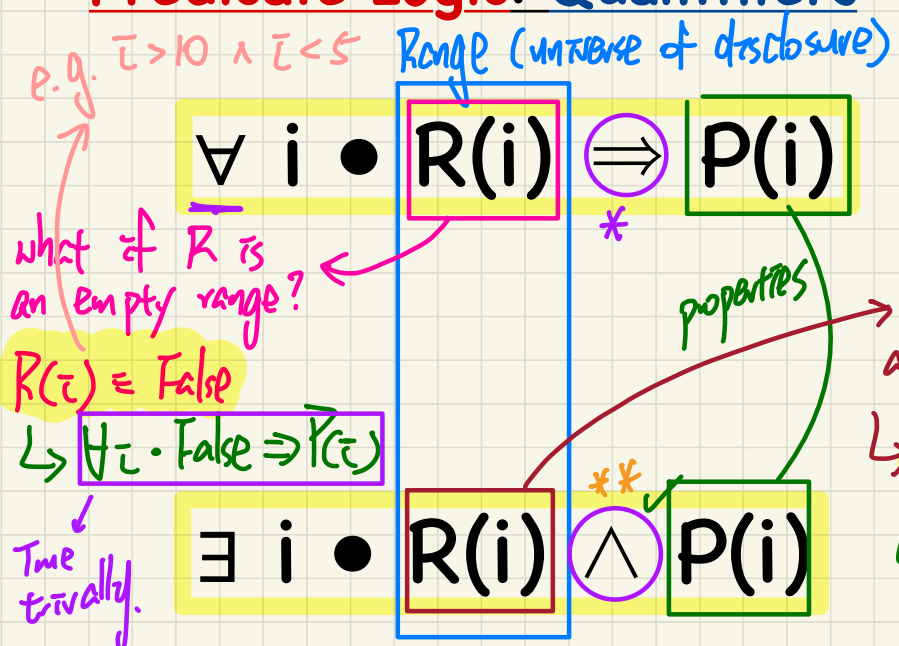
Lectures/Exam : math syntax on paper

↓

may also be asked to write in Pluscal syntax

Predicate Logic: Quantifiers

- syntax
- base cases in programming



* ① $\bar{i}$ is in the range (i.e., R($\bar{i}$) ≡ True), then P($\bar{i}$) determines if $\Rightarrow$ is True.

② $\bar{i}$ is not in the range (R($\bar{i}$) ≡ False), no need to consider P($\bar{i}$).

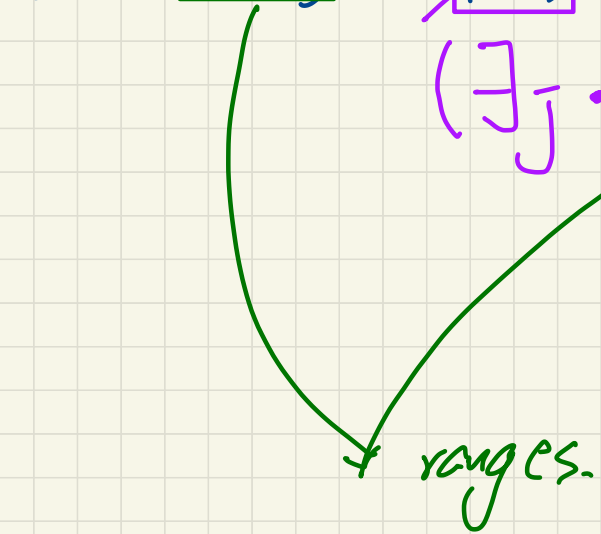
** $\bar{i}$ should be in the range, and P($\bar{i}$) should be true.

```
boolean allPos(int[] ns){
  if(ns.length == 0){
    return true;
  }
}
```
cannot find a witness violating pos.

```
boolean somePos(int[] ns){
  if(ns.length == 0){
    return false;
  }
}
```
cannot find a witness satisfying pos.

Nested Quantifications

$$\forall i \cdot R(i) \Rightarrow (\exists j \cdot Q(i,j) \wedge S(i,j))$$



Logical Quantifiers: Examples

Monday: Examples on Nested Quantification

$\forall\, i \bullet i \in \mathbb{N} \Rightarrow i \geq 0$ (T)

$0, 1, 2, \ldots, +\infty$

(F) $\forall\, i \bullet i \in \mathbb{Z} \Rightarrow i \geq 0$ $\quad T \Rightarrow F \equiv F$

$-\infty, \ldots, -3, -2, -1, \ldots \quad -2 \geq 0$ (F)

(F) $\forall\, i, j \bullet i \in \mathbb{Z} \wedge j \in \mathbb{Z} \Rightarrow i < j \vee i > j$

$2\ 2 \qquad 2 \in \mathbb{Z} \wedge 2 \in \mathbb{Z} \qquad 2 < 2 \vee 2 > 2$

(T) $\Rightarrow$ (F) $\equiv$ (F).

(T) $\exists\, i \bullet i \in \mathbb{N} \wedge i \geq 0$

$0 \qquad 0 \in \mathbb{N} \wedge 0 \geq 0 \quad T \wedge T \equiv T$.

$\exists\, i \bullet i \in \mathbb{Z} \wedge i \geq 0$

$0 \in \mathbb{Z} \wedge 0 \geq 0$

(T) $\exists\, i, j \bullet i \in \mathbb{Z} \wedge j \in \mathbb{Z} \wedge (i < j \vee i > j)$

$3, 2 \qquad 3 \in \mathbb{Z} \wedge 2 \in \mathbb{Z} \wedge (3 < 2 \vee 3 > 2)$

(T) (T)

Logical Quantifiers: Proof Strategies

How to prove $\forall\ i \bullet R(i) \Rightarrow P(i)$?

① Assume $R(\bar{i})$. Show $P(\bar{i})$ so as to consider all values in the range

② Prove $\neg R(\bar{i})$
↳ $R(\bar{i}) \equiv$ Fake (empty range)

How to prove $\exists\ i \bullet R(i) \wedge P(i)$?

① Give a (satisfaction) witness $\bar{i}$: $R(\bar{i})$ and $P(\bar{i})$

$T \wedge T \equiv T$

How to disprove $\forall\ i \bullet R(i) \Rightarrow P(i)$?

think about the counter-example(s) are given by a model checker

① Give a (violation) witness $\bar{i}$: $R(\bar{i})$ but $\neg P(\bar{i})$

$T \Rightarrow F \equiv F$

How to disprove $\exists\ i \bullet R(i) \wedge P(i)$?

(1) Show $\neg R(\bar{i})$ (empty range ⇝ no way to find a sat. witness)

(2) Show $R(\bar{i}) \wedge \neg P(\bar{i})$ for all $\bar{i}$ that satisfies R. (i.e. none of the values in the range can be valid sat. witness)

Prove/Disprove Logical Quantifications

- Prove or disprove: $\forall x \bullet (x \in \mathbb{Z} \land 1 \le x \le 10) \Rightarrow x > 0$.

↳ non-empty range: $1, 2, \ldots, 10$ ↳ each > 0

- Prove or disprove: $\forall x \bullet (x \in \mathbb{Z} \land 1 \le x \le 10) \Rightarrow x > 1$.

witness: 1 $\quad 1 \in \mathbb{Z} \land 1 \le 1 \le 10 \Rightarrow 1 > 1$ (T) (F) $\equiv$ (F).

- Prove or disprove: $\exists x \bullet (x \in \mathbb{Z} \land 1 \le x \le 10) \land x > 1$.

witness: 2 $\quad 2 \in \mathbb{Z} \land 1 \le 2 \le 10 \land 2 > 1$ (T) (T) $\equiv$ (T).

- Prove or disprove that $\exists x \bullet (x \in \mathbb{Z} \land 1 \le x \le 10) \land x > 10$?

each values in the range $1, 2, \ldots, 10$ > 10 is (F) (no valid witness can be found).

General Forms of $\forall$ and $\exists$

a single predicate P

$$\forall x \cdot \boxed{x \in \mathbb{Z} \wedge 1 \leq x \leq 10 \Rightarrow x > 1}$$

$$\forall x \cdot \boxed{P(x)}$$

$$\exists x \cdot P(x)$$

Axioms

$$\forall x \cdot P(x) \equiv \neg \exists x \cdot \neg P(x)$$

$$\exists x \cdot P(x) \equiv \neg \forall x \cdot \neg P(x)$$

Logical Quantifications: Conversions

R(x): x ∈ 4315_class
P(x): x receives A+

* $(\forall X \bullet R(X) \Rightarrow P(X))$ $\equiv$ ** $\neg(\exists X \bullet R(X) \wedge \neg P(X))$

Equational Style of Proof

* $\forall x \cdot R(x) \Rightarrow P(x)$ $\Theta(x)$

$\equiv$ { Axiom: $\forall x \cdot \Theta(x) \equiv \neg(\exists x \cdot \neg\Theta(x))$ }

$\neg\exists x \cdot \neg(R(x) \Rightarrow P(x))$

$\equiv$ { Theorem: $p \Rightarrow q \equiv \neg p \vee q$ }

$\neg\exists x \cdot \neg(\neg R(x) \vee P(x))$

$\equiv$ { De Morgn: $\neg(p \vee q) \equiv \neg p \wedge \neg q$, Double Neg. $\neg\neg p \equiv p$ }

** $\neg\exists x \cdot R(x) \wedge \neg P(x)$

$(\exists X \bullet R(X) \wedge P(X)) \Leftrightarrow \neg(\forall X \bullet R(X) \Rightarrow \neg P(X))$

De Morgan

Lecture 7 - January 27

Lab1 Review (Part 1), Lab2 Preview

Finite Reachability Graph with Cycles
Atomic Updates with a Single Label
Algo. Contracts: Pre- & Post-condition

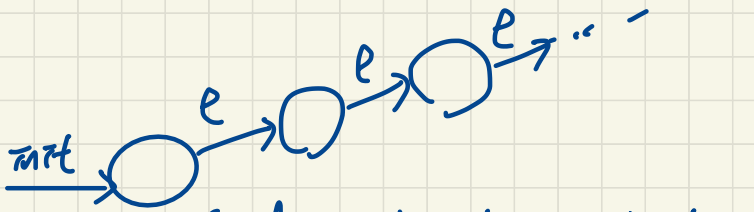
Announcements/Reminders

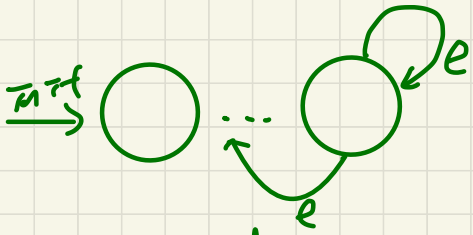
- **Lab1** solution released
- **Lab2** released
- **TA contact information** (on-demand for labs) on eClass
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu

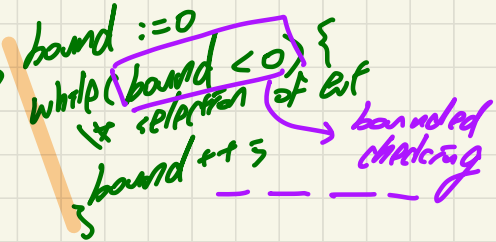
1. Invariant vs. Variant 2. Liveness/Progress 3. Req. on using Variant

Lab1 Solution Discussion: Roadmap

- **Finite** Reachability Graph from **Unbounded** Event Interleavings
- **Atomic** vs. **Non-Atomic** Updates
- Encoding System Variant







for checking to terminate

Promela

e
begin
atomically
end

multiple labelled atoms expanded

not the case for TLA+

vs.

while (true) {

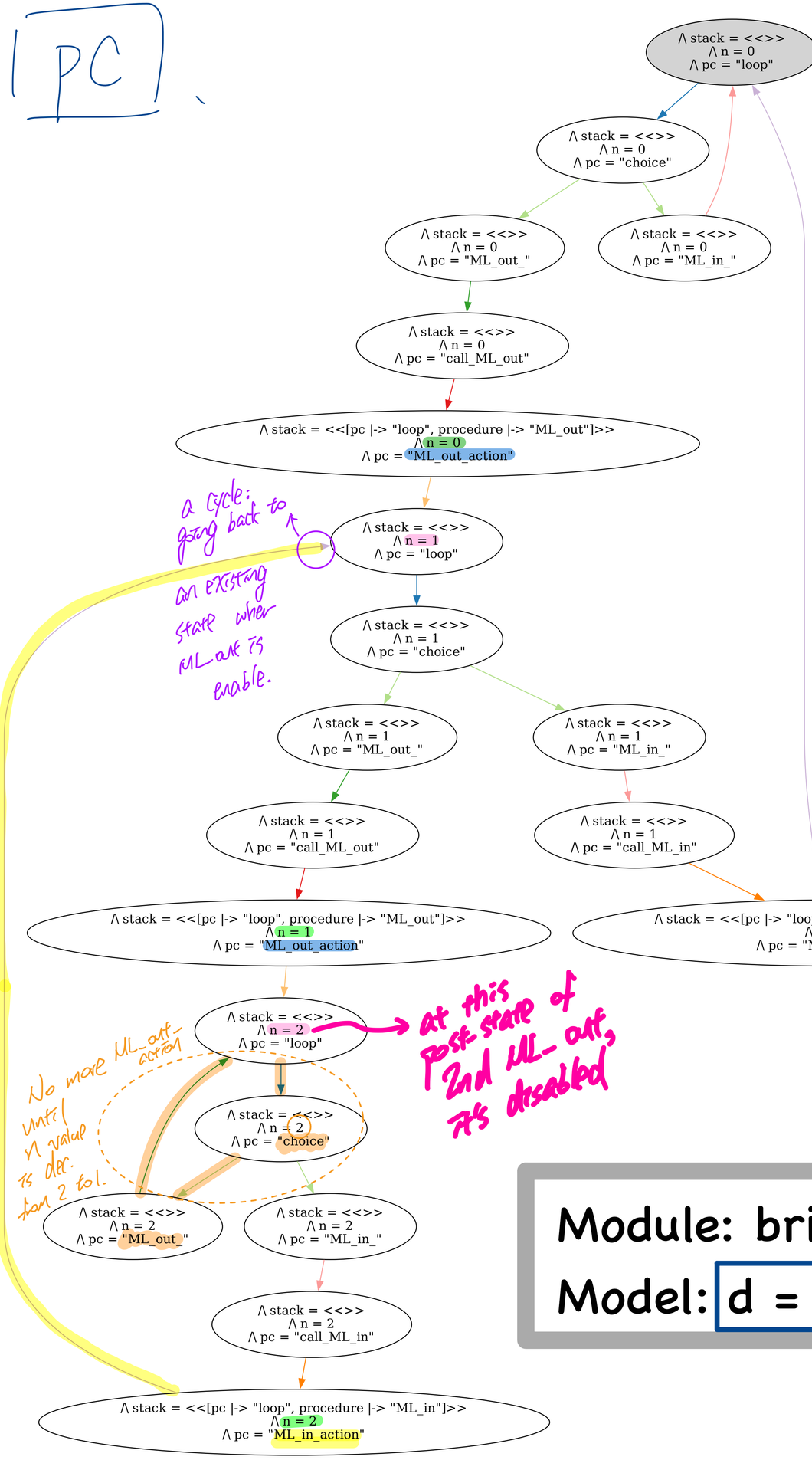
assuming that guards of events are such that the RG remains finite

would not cause model checking not to terminate (i.e., infinite)

by having cycles

no event transitions that are always enabled to reach to new states

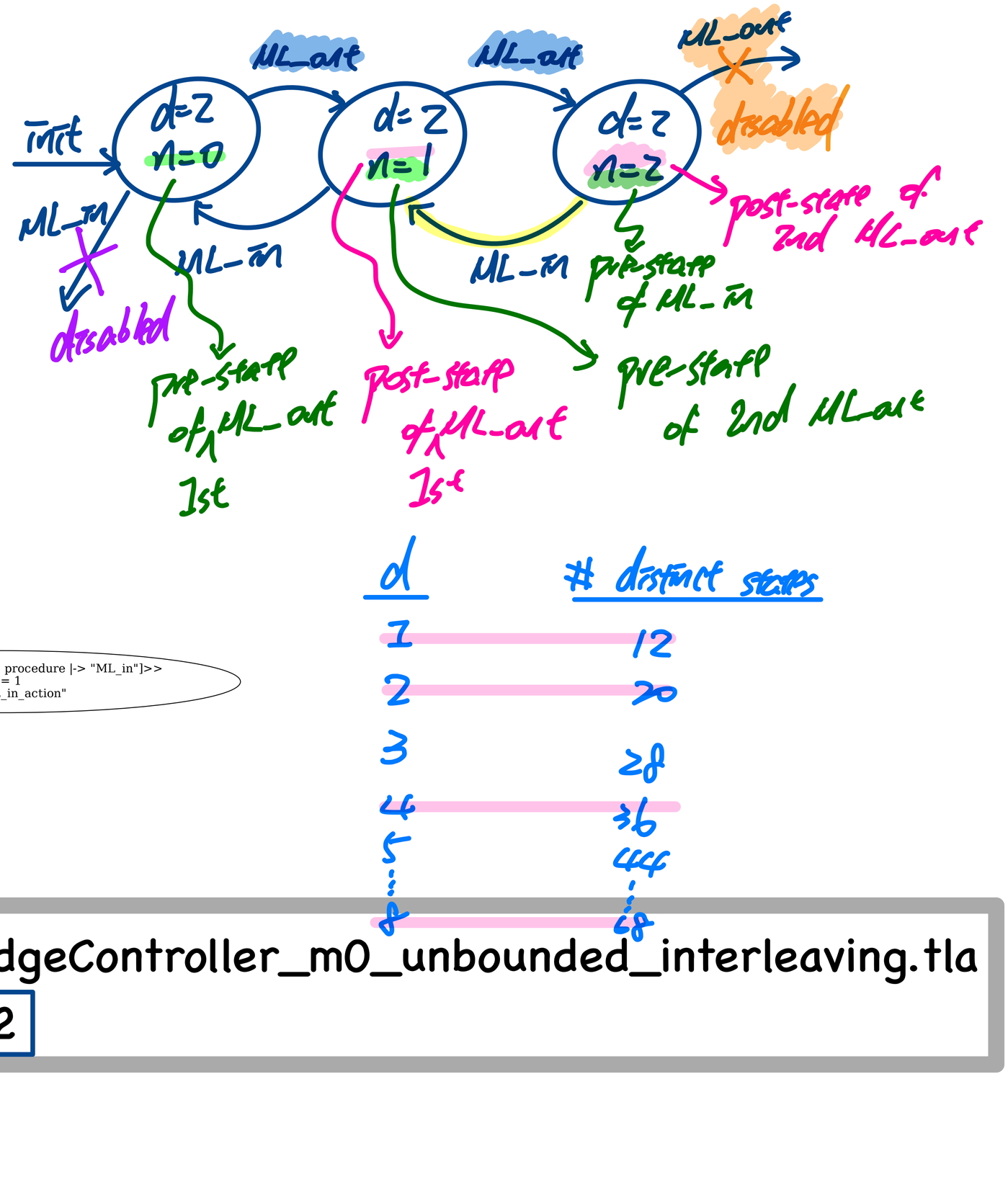
PC



Next State Actions

ML_out_ | ML_out_action | call_ML_out | choice | ML_in_ | ML_in_action | call_ML_in | loop

bridgeController_m0_unbounded_interleaving.tla



d	# distinct states
1	12
2	20
3	28
4	36
5	44
⋮	⋮
8	68

Module: bridgeController_m0_unbounded_interleaving.tla

Model: d = 2

1. When defining modules, try to be as "correct" as possible.

2. Write "good" invariants/assertions to verify.

↳ TLC checker ✓ → set small bounds on constants and inspect the state graph to make sure the system works as expected

↳ TLC checker ✗ → error trace → e.g. e should be enabled but not the case

↳ need to study this with RA to fix the bug

Lab1 Solution Discussion: Atomic vs. Non-Atomic Updates

Multiple Labels for Procedure Actions

```
--algorithm bridgeController_m1 {
  variable n = 0, a = 0, b = 0, c = 0;

  procedure ML_out() {
    ML_out_action_abstract: n := n + 1;
    ML_out_action_concrete: a := a + 1;
    return;
  }
```

act 1 expr. atomically

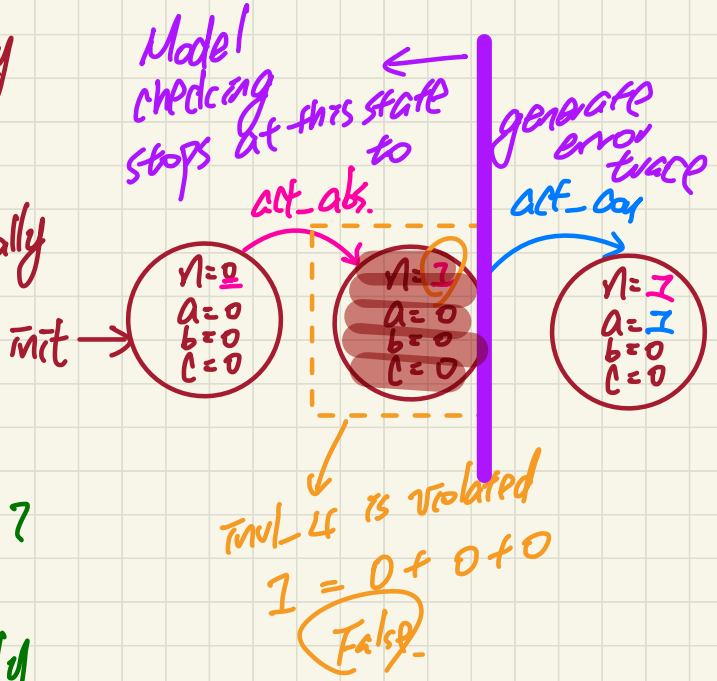
act 2 expr. atomically

Single Labels for Procedure Actions

```
--algorithm bridgeController_m1 {
  variable n = 0, a = 0, b = 0, c = 0;

  procedure ML_out() {
    ML_out_action_abstract: n := n + 1;
                            a := a + 1;
                            return;
  }
```

act 1 & act 2 expr. atomically

inv1_4: a + b + c = n



Lecture 8 - January 29

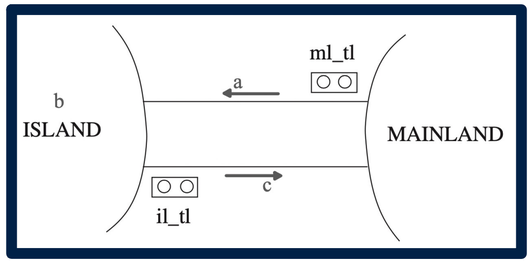
Lab1 Review (Part 2)

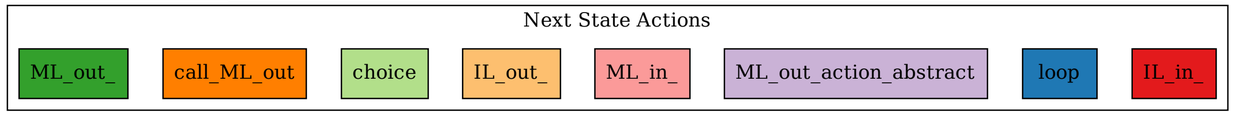
Identifying Atomicity in State Graph
Recall (from EECS3342): System Variant
Encoding & Checking Variant in TLA+

Announcements/Reminders

- **Lab1** solution released
- **Lab2** released
- **TA contact information** (on-demand for labs) on eClass
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu

bridgeController_m1_no_variant.tla





Multiple Labels for Procedure Actions

```
--algorithm bridgeController_m1 {
  variable n = 0, a = 0, b = 0, c = 0;

  procedure ML_out() {
    ML_out_action_abstract: n := n + 1;
    ML_out_action_concrete: a := a + 1;
    return;
  }
```


TLC Errors

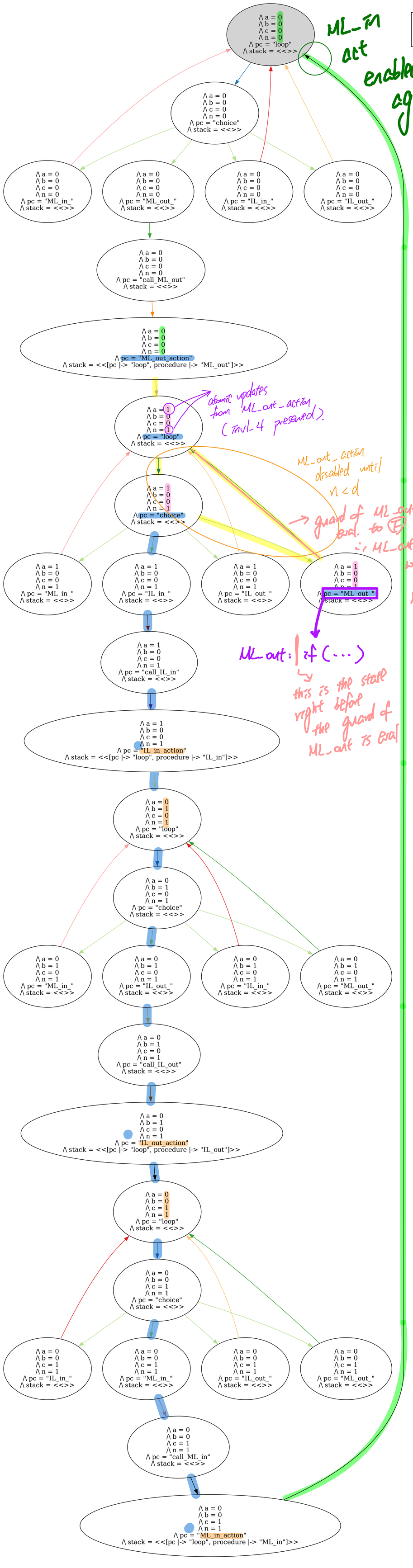
Model_1_d1

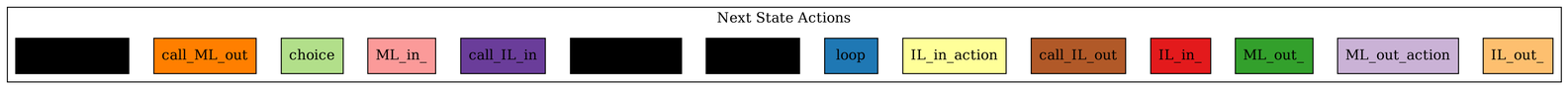
Invariant inv1_4 is violated.

Error-Trace Exploration

Error-Trace

Name	Value
<Initial predicate>	State (num = 1)
<loop line 114, col 9 to line 116, col 44 of module brid	State (num = 2)
<choice line 118, col 11 to line 123, col 46 of module	State (num = 3)
<ML_out_ line 125, col 12 to line 129, col 47 of modul	State (num = 4)
<call_ML_out line 131, col 16 to line 136, col 44 of mo	State (num = 5)
<ML_out_action_abstract line 74, col 27 to line 77, col	State (num = 6)
a	0
b	0
c	0
n	1
pc	"ML_out_action_concrete"
stack	<<[pc |-> "loop", procedure |-> "ML_out"]>>





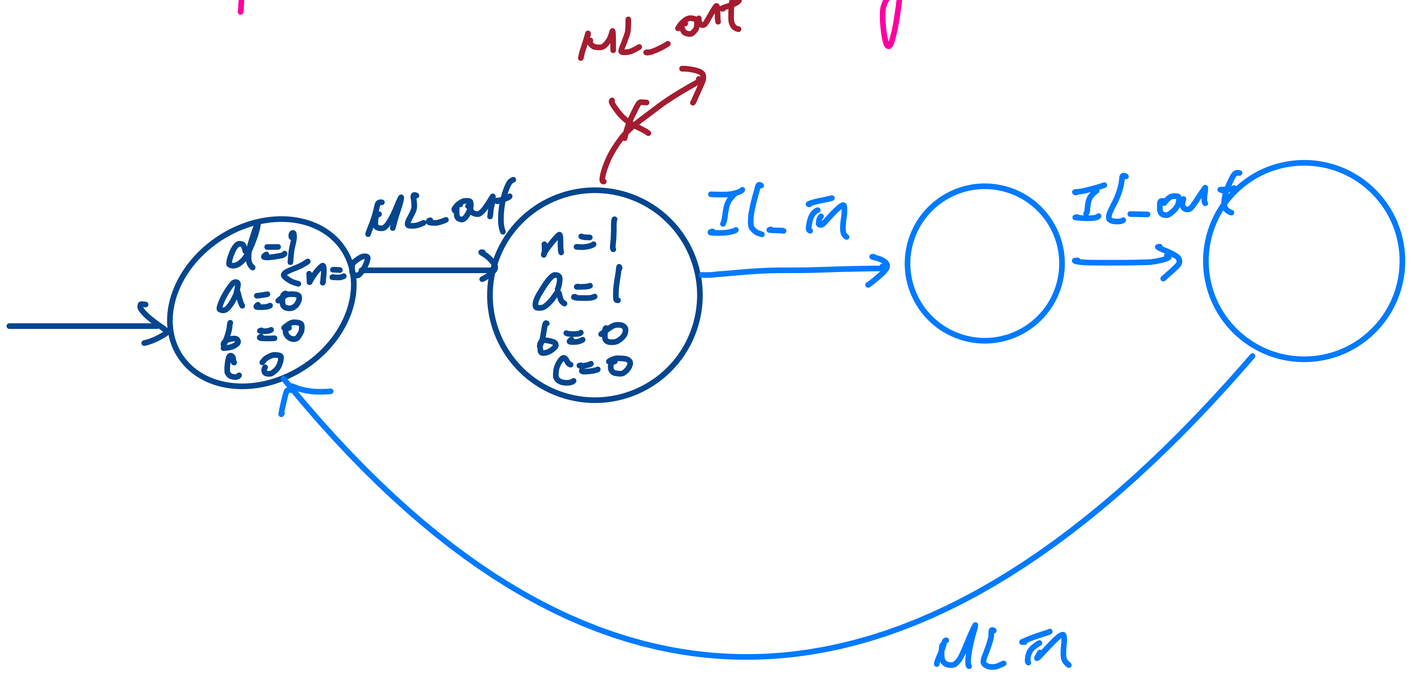
Single Labels for Procedure Actions

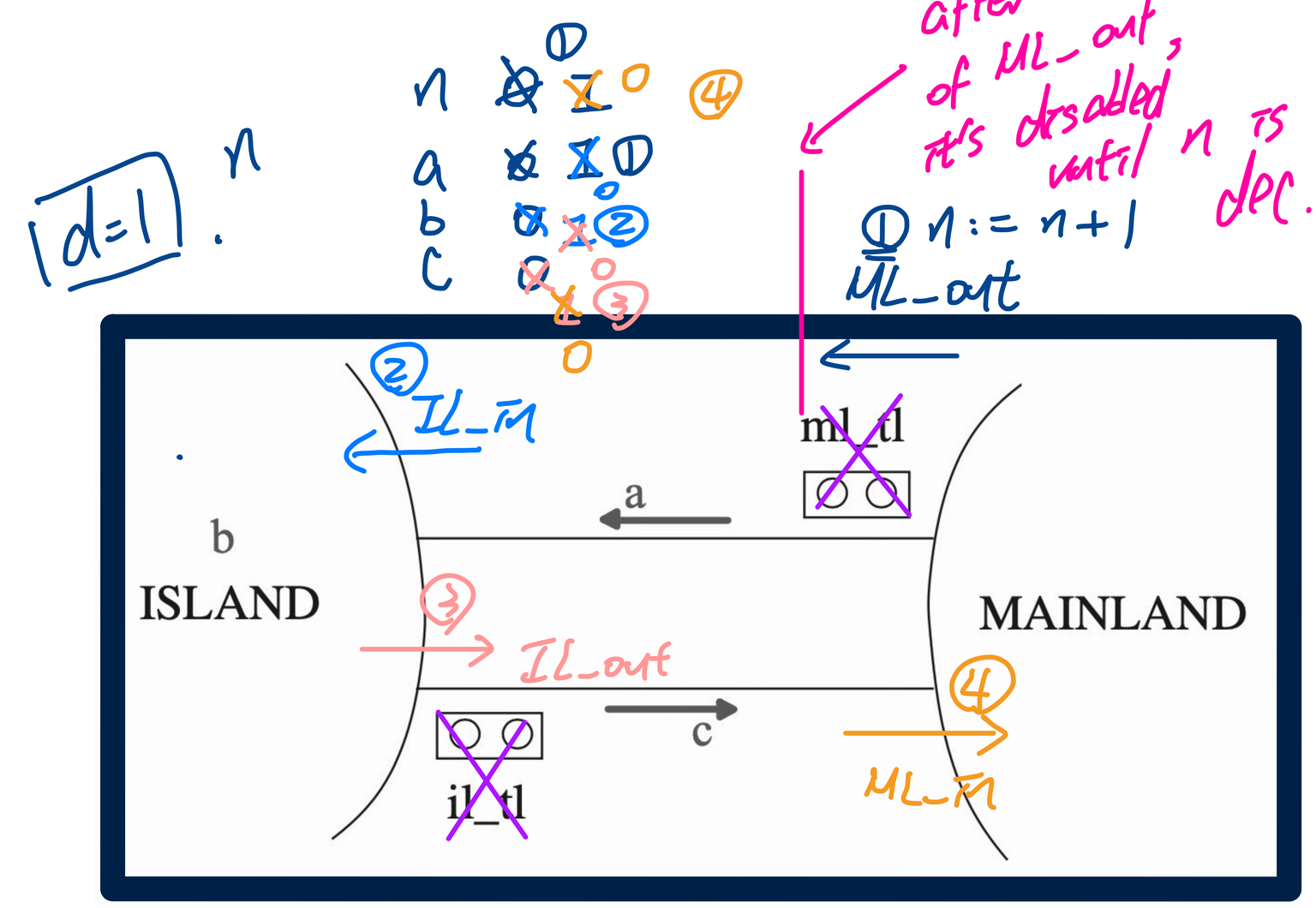
```
--algorithm bridgeController_m1 {
  variable n = 0, a = 0, b = 0, c = 0;

  procedure ML_out() {
    ML_out_action abstract: n := n + 1;
                            a := a + 1;
                            return;
  }
```

bridgeController_m1_no_variant.tla

Exercise

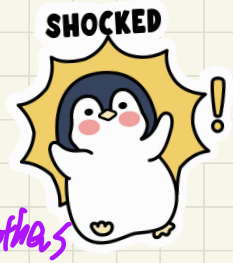
1. Gen. RG d = 2
2. Conceptual sketch of how ML_out is enabled/disabled
3. Inspect the RG and verify that.





system's progress is "locked" in an active way

Livelock Caused by New Events Diverging



↳ certain events happening indef., preventing others from happening.

An alternative m1 (for demonstration)

constants: d

axioms:
axm0_1 : $d \in \mathbb{N}$
axm0_2 : $d > 0$

variables: a, b, c

invariants:
inv1_1 : $a \in \mathbb{Z}$
inv1_2 : $b \in \mathbb{Z}$
inv1_3 : $c \in \mathbb{Z}$

old events

ML_out
when
$a + b < d$
$c = 0$
then
$a := a + 1$
end

ML_in
when
$c > 0$
then
$c := c - 1$
end

new events

IL_in
begin
$a := a - 1$
$b := b + 1$
end

IL_out
begin
$b := b - 1$
$c := c + 1$
end

(unconditional)

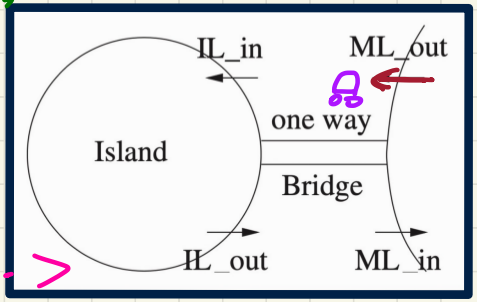
solution: have a measure on imposing an upper bound on the # of interleavings of new events

while(1) {

} no

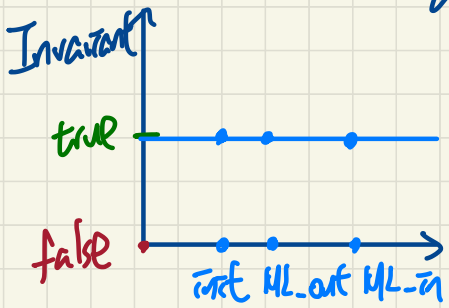
Abstract transitions: <init, ML_out, skip, skip, skip, skip, ... >
m1
new events occurring indefinitely.

Concrete transitions: <init, ML_out, IL_in, IL_out, IL_in, IL_out, ... >
a possible trace that's problematic.

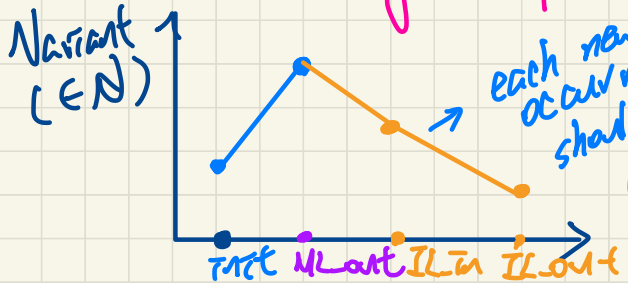


Invariant vs. Variant

Invariant : Boolean expression that should always hold (after each event occurrence)



Variant : Integer expression that may change after event occurrence.



each new event occurrence should strictly decrease the value of variant ⇒ # new events in interleaving is finite ⇒ no livelock

Use of a Variant to Measure New Events Converging

fixed

variables: a, b, c

invariants:
- inv1_1 : $a \in \mathbb{N}$
- inv1_2 : $b \in \mathbb{N}$
- inv1_3 : $c \in \mathbb{N}$
- inv1_4 : $a + b + c = n$
- inv1_5 : $a = 0 \vee c = 0$

ML_out
when
$a + b < d$
$c = 0$
then
$a := a + 1$
end

ML_in
when
$c > 0$
then
$c := c - 1$
end

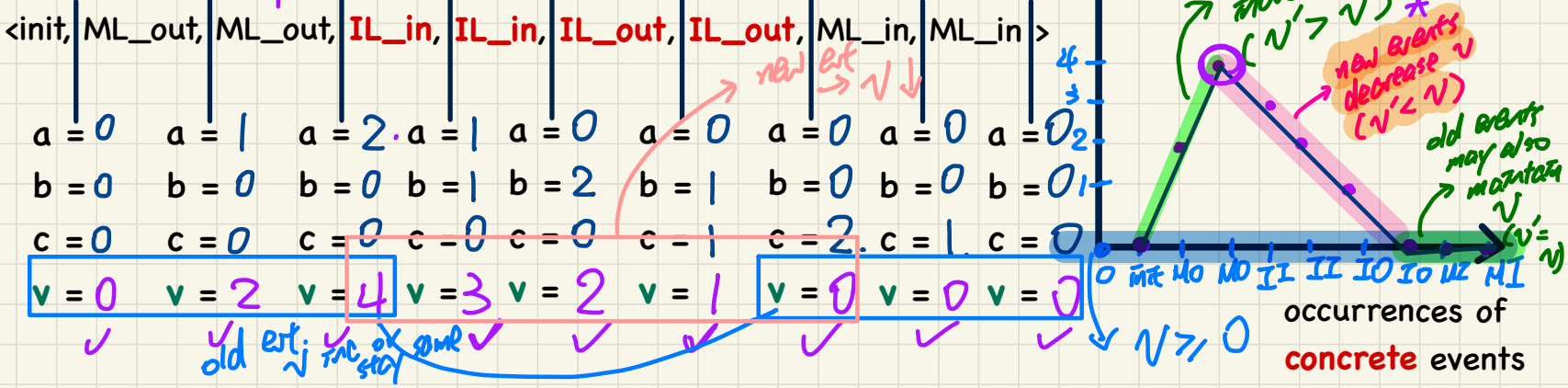
IL_in
when
$a > 0$
then
$a := a - 1$
$b := b + 1$
end

IL_out
when
$b > 0$
$a = 0$
then
$b := b - 1$
$c := c + 1$
end

* Given that the starting value of V of the first new event is finite,

Variants for New Events: 2 · a + b

the # of interleaved new events is also finite.



PO of Convergence/Non-Divergence/Livelock Freedom

Variants for New Events: 2 · a + b

Variant Stays Non-Negative

$A(c)$
$I(c, v)$
$J(c, v, w)$
$H(c, w)$
$\vdash$ $V(c, w) \in \mathbb{N}$

guard of new event

NAT

IL_in/NAT

new event

$v \geq 0$

$d \in \mathbb{N}$
$d > 0$
$n \in \mathbb{N}$ $n \leq d$

$a \in \mathbb{N}$ $b \in \mathbb{N}$ $c \in \mathbb{N}$ $a > 0$ $a = c \vee c = 0$ $a + b + c = n$

$\vdash 2 \cdot a + b \in \mathbb{N}$

* $v' < v$ $(b+1)$

$2 \cdot a' + b' < 2 \cdot a + b$

$(a-1)$

variant: V(c, w)

v

$V(c, w)$

v

$V(c, F(c, w))$

IL_in/NAT

new event

occurrences of **new** events

A New Event Occurrence Decreases Variant

$A(c)$
$I(c, v)$
$J(c, v, w)$
$H(c, w)$
$\vdash$ $V(c, F(c, w)) < V(c, w)$

$a' = a - 1$ $\wedge$ $b' = b + 1$

effect of new event

v v

VAR

IL_in/VAR

new event

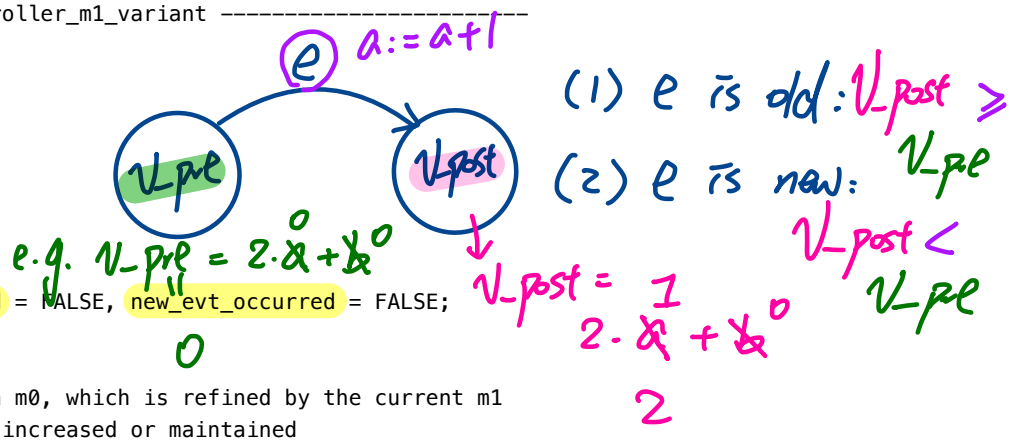
v's value strictly decreased.

$d \in \mathbb{N}$
$d > 0$
$n \in \mathbb{N}$ $n \leq d$

$a \in \mathbb{N}$ $b \in \mathbb{N}$ $c \in \mathbb{N}$ $a > 0$ $a = c \vee c = 0$ $a + b + c = n$

* $\vdash 2 \cdot (a-1) + (b+1) < 2 \cdot a + b$

```
------------------------ MODULE bridgeController_m1_variant ------------------------
EXTENDS Integers, Naturals, Sequences, TLC
CONSTANT d
ASSUME /\ d \in Nat
       /\ d > 0
(*
--algorithm bridgeController_m1 {
  variable
    n = 0, a = 0, b = 0, c = 0,
    V_pre = 0, V_post = 0, old_evt_occurred = FALSE, new_evt_occurred = FALSE;

  (*
     Old events: ones that already exist in m0, which is refined by the current m1
     Value of the system variant is always increased or maintained
       by each occurrence of an old event.
  *)
  procedure ML_out() {
    ML_out_action: n := n + 1;
                   a := a + 1;
                   return;
  }

  procedure ML_in() {
    ML_in_action: n := n - 1;
                  c := c - 1;
                  return;
  }

  (*
     New events: ones that do not exist in m0, which is refined by the current m1
     Value of the system variant is always decreased
       by each occurrence of a new event event.
  *)
  procedure IL_in() {
    IL_in_action: a := a - 1;
                  b := b + 1;
                  return;
  }

  procedure IL_out() {
    IL_out_action: b := b - 1;
                   c := c + 1;
                   return;
  }

  {
    loop: while (TRUE) {
      (* Without the first two updates resetting the event log,
         when new_evt_occurred == true, after the next line,
         V_pre == V_post, which will violate the VAR variant constraint.
      *)
      update_variant_pre: new_evt_occurred := FALSE;
                          old_evt_occurred := FALSE;
                          V_pre := 2 * a + b;
      choice: either {
                ML_out: if ( (n < d) /\ (a + b < d) /\ (c = 0)) {
                  call ML_out: call ML_out();
                  update_evt_log_ml_out: new_evt_occurred := FALSE;
                                         old_evt_occurred := TRUE;
                                         V_post := 2 * a + b;
                };
              }
              or {
```



captures the pre-state value of V for the next event
↳ old or new.

atomic updates

```
      ML_in: if ( (n > 0) /\ (c > 0) ) {
        call_ML_in: call ML_in();
        update_evt_log_ml_in: new_evt_occurred := FALSE;
                              old_evt_occurred := TRUE;
                              V_post := 2 * a + b;
      };
    }
    or {
      IL_in: if ( a > 0 ) {
        call_IL_in: call IL_in();
        update_evt_log_il_in: new_evt_occurred := TRUE;
                              old_evt_occurred := FALSE;
                              V_post := 2 * a + b;
      };
    }
    or {
      IL_out: if ( (b > 0) /\ (a = 0) ) {
        call_IL_out: call IL_out();
        update_evt_log_il_out: new_evt_occurred := TRUE;
                               old_evt_occurred := FALSE;
                               V_post := 2 * a + b;
      };
    };
  }
 }
}
*)
\* BEGIN TRANSLATION (chksum(pcal) = "ce02e87c" /\ chksum(tla) = "5f2f5c21")
...
\* END TRANSLATION

\* checking invariants
inv1_1 == a \in Nat
inv1_2 == b \in Nat
inv1_3 == c \in Nat
inv1_4 == a + b + c = n
inv1_5 == (a = 0) \/ (c = 0)

\* checking variants
variants == 2 * a + b >= 0
event_log_consistent == ~(/\ old_evt_occurred = TRUE /\ new_evt_occurred = TRUE)
variant_not_decreased == (old_evt_occurred = TRUE => V_post >= V_pre)
variant_decreased == (new_evt_occurred = TRUE => V_post < V_pre)

\* checking deadlock freedom
guard_ML_out == /\ (n < d)
                /\ (a + b < d)
                /\ (c = 0)
guard_ML_in == /\ (n > 0)
               /\ (c > 0)
guard_IL_in == a > 0
guard_IL_out == /\ (b > 0)
                /\ (a = 0)
deadlockfree == guard_ML_out \/ guard_ML_in \/ guard_IL_in \/ guard_IL_out
================================================================================
```

Lecture 9 - February 3

ProgTest1 *Guide*, Math Review

Implementation Correctness
Completeness of Contracts:
Pre-condition vs. Post-condition

Announcements/Reminders

- ProgTest1 guide released
- Mockup Test scheduled during lab on Thursday, Feb. 6
- Lab1 solution released
- Lab2 released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

Correct Algorithm and Complete Postcondition (1.1)

```
------- MODULE ExampleModule -------
EXTENDS Integers, Sequences, TLC
CONSTANT input
-- algorithm SomeAlgo {
  variables
    output = ..., ...
  {
    \* Preconditions
    assert Q;

    (* Implementation in PlusCal *)
    Imp.

    \* Postcondition 1
    assert R1;
    \* Postcondition 2
    assert R2;
  }
}
```

Contracts

assess: (1) given valid input, your Q must pass. (2) given invalid input, your Q must fail.

(pre-state)

(post-state) may involve both pre- and post-state values.

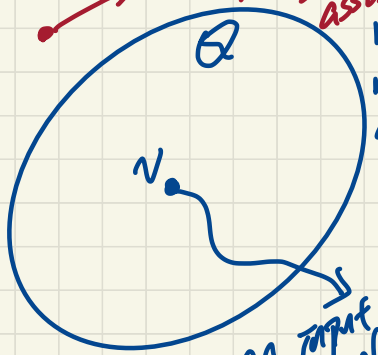
Q complete?

– a predicate which states the assumption on input value(s) made by alg.

to be passed to the alg

an input value sat. Q is not assumed

Q must be mapped, by alg., to the correct output val.



an input value satisfying Q

Correct Algorithm and Complete Postcondition (1.2)

```
------- MODULE ExampleModule -------
EXTENDS Integers, Sequences, TLC
CONSTANT inputSeq, inputVal
-- algorithm BinarySearch {
  variables
    output = FALSE, ...
  {
    \* Preconditions
    assert Q;

    (* Implementation in PlusCal *)
    Imp.

    \* Postcondition 1
    assert R1;
    \* Postcondition 2
    assert R2;
  }
}
```

Q complete?

Extend?

inputSeq is sorted

(1) Passing <1, 2, 3, 4>

should not trigger any precond errors.

(2) Passing <1, 3, 2, -2>

should trigger some precond. error.

Correct Algorithm and Complete Postcondition (2.1)

```
------- MODULE ExampleModule -------
EXTENDS Integers, Sequences, TLC
CONSTANT input
-- algorithm SomeAlgo {
  variables
    output = ..., ...
  {
    \* Preconditions
    assert Q;

    (* Implementation in PlusCal *)
    Imp.

    \* Postcondition 1
    assert R1;
    \* Postcondition 2
    assert R2;
  }
}
```

2 General

Specify one or more predicate assertions relating the input and output variables.

→ Postconditions do not need to adapt for different input values

Imp. correct?

Given that input satisfies precond Q,
does the alg. terminate in a state where the output has the expected val.

Two ways for asserting the output:

1. Specific: alg(input1) = expO1 (actual output)
   alg(input2) = expO2

drawback: assert expO1 has to be changed according to the specific input.

Correct Algorithm and Complete Postcondition (2.2)

```
------- MODULE ExampleModule -------
EXTENDS Integers, Sequences, TLC
CONSTANT inputSeq, inputVal
(* algorithm BinarySearch {
  variables
    output = FALSE, ...
  {
    \* Preconditions
    assert /* inputSeq is sorted */;

    (* Implementation in PlusCal *)
    Imp.

    \* Postcondition 1
    assert /* inputSeq unchanged */;
    \* Postcondition 2
    assert /* output computed correctly */;
  }
}
```

output = inputVal ∈ inputSeq.

Imp. correct?

[1] Specific

sorting

set input = <2,4,1,3>

assert output = <1,2,3,4>

search

inputSeq = <1,2,3,4>

inputVal = 5

assert output = FALSE

[2] General

↓

specify the "complete" set of postconditions to be checked.

Correct Algorithm and Complete Postcondition (3.1)

```
------- MODULE ExampleModule -------
EXTENDS Integers, Sequences, TLC
CONSTANT input
(* algorithm SomeAlgo {
  variables
    output = ..., ...
  {
    \* Preconditions
    assert Q;

    (* Implementation in PlusCal *)
    Imp.

    \* Postcondition 1
    assert R1;
    \* Postcondition 2
    assert R2;
  }
}
```

say you're required to specify each one of them.

R1 and R2 complete?

(1) if the kinds of imp. errors are captured by either one of them

(2) if they account for all possible ways that the imp. can go wrong.

Correct Algorithm and Complete Postcondition (3.2)

```
------- MODULE ExampleModule -------
EXTENDS Integers, Sequences, TLC
CONSTANT inputSeq, inputVal
(* algorithm BinarySearch {
  variables
    output = FALSE, ...
  {
    \* Preconditions
    assert /* inputSeq is sorted */;

    (* Implementation in PlusCal *)
    Imp.

    \* Postcondition 1
    assert R1;
    \* Postcondition 2
    assert R2;
  }
}
```

unchanged input

correct output

answers submitted by you.

R1 and R2 complete?

(1) Assess quality of your R1?

Replace by a faulty imp. that always removes the 1st item in input.

(2) Assess quality of your R2?

Replace by a faulty imp. which (a) does not modify input (b) always returns false

Predicate Logic: Exercise 1

Consider the following predicate:

$$\forall\, x, y \bullet x \in \mathbb{N} \land y \in \mathbb{N} \Rightarrow x * y > 0$$

→ not a theorem.

Choose **all** statements that are **correct**.

1. It is a theorem, provable by (5, 4). ✗
2. It is a theorem, provable by (2, 3). ✗
3. ✓ It is not a theorem, witnessed by (5, 0).
4. It is not a theorem, witnessed by (12, -2).
5. It is not a theorem, witnessed by (12, 13).

$5 \in \mathbb{N} \land 0 \in \mathbb{N} \Rightarrow 5 * 0 > 0$

T ⇒ F

Lecture 10 - February 5

Math Review Exercises, Model Checking

Nested Quantification
Model Checking Intro: $\vdash$ ***vs.*** $\models$
State Graph vs. (Computation) Paths

Announcements/Reminders

- ProgTest1 guide released
- Mockup Test scheduled in tomorrow's lab session
- Lab1 solution released
- Lab2 released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

Predicate Logic: Exercise 1

Consider the following predicate:

$$\forall\, x, y \bullet x \in \mathbb{N} \land y \in \mathbb{N} \Rightarrow x * y > 0$$

not a theorem.

to disprove give a witness of violation

Choose **all** statements that are **correct**.

- ✗ 1. It is a theorem, provable by (5, 4).
- ✗ 2. It is a theorem, provable by (2, 3).
- ✓ 3. It is not a theorem, witnessed by (5, 0).
- ✗ 4. It is not a theorem, witnessed by (12, -2).
- ✗ 5. It is not a theorem, witnessed by (12, 13).

$5 \in \mathbb{N} \land 0 \in \mathbb{N} \Rightarrow 5 * 0 > 0$ (T ⇒ F)

$12 \in \mathbb{N} \land 13 \in \mathbb{N} \Rightarrow 12 * 13 > 0$ (T ⇒ T)

$12 \in \mathbb{N} \land -2 \in \mathbb{N} \Rightarrow 12 * -2 > 0$ (F ⇒ T)

Predicate Logic: Exercise 2

Consider the following predicate:

$$\exists x, y \bullet x \in \mathbb{N} \land y \in \mathbb{N} \land x * y > 0$$

a theorem

to prove, show a witness of satisfaction.

Choose **all** statements that are **correct**.

1. ✓ It is a theorem, provable by (5, 4). $5 \in \mathbb{N} \land 4 \in \mathbb{N} \land 5 * 4 > 0$ T
2. ✓ It is a theorem, provable by (2, 3).
3. ✗ It is a theorem, provable by (-2, -3). $-2 \in \mathbb{N}$ $-3 \in \mathbb{N}$ $-2 * -3 > 0$ F
4. ✗ It is not a theorem, witnessed by (5, 0).
5. ✗ It is not a theorem, witnessed by (12, -2).
6. ✗ It is not a theorem, witnessed by (12, 13).

Nested Logical Quantifiers

$\forall i \bullet i \in \mathbb{Z} \Rightarrow (\exists j \bullet j \in \mathbb{N} \wedge i + j = 0)$

$\forall j \cdot j \in \mathbb{N} \wedge (i+j) \neq 0$

$\Leftrightarrow$

(F) To disprove it: choose some i s.t. $\neg(\exists j \cdot j \in \mathbb{N} \wedge i + j = 0)$

Witness: choose $i = 2$, $\neg\exists j \cdot j \in \mathbb{N} \wedge 2 + j = 0$

$\forall i \bullet \underline{i \in \mathbb{N}} \Rightarrow (\exists j \bullet j \in \mathbb{Z} \wedge i + j = 0)$ P

can't be negative.

(T) To prove it: argue that all possible values of i satisfy P

$i \in \mathbb{N} \Rightarrow i \geq 0$ for each i, choose $j = -i \in \mathbb{Z}$

$\exists i \bullet i \in \mathbb{N} \wedge (\forall j \bullet j \in \mathbb{Z} \Rightarrow i \cdot j > 0)$ P

show $\neg P$

(F) $i \in \mathbb{N} \Rightarrow i \geq 0$ witness: choose $j = 0$ or $j < 0$

$\exists i \bullet i \in \mathbb{N} \wedge (\forall j \bullet j \in \mathbb{Z} \Rightarrow i \cdot j \geq 0)$

(T) Witness: choose $i = 0$ $\Rightarrow$ each $j \in \underline{\mathbb{Z}}$ is s.t. $0 \cdot j \geq 0$

Use of Model Checking in Industry

Pentium FDIV bug: https://en.wikipedia.org/wiki/Pentium_FDIV_bug

*The Pentium FDIV bug is a hardware bug affecting the **floating-point unit (FPU)** of the early Intel Pentium processors. Because of the bug, the processor would return incorrect binary floating point results when dividing certain pairs of high-precision numbers.*

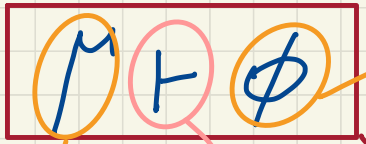
*In December 1994, Intel **recalled** the defective processors ... In its 1994 annual report, Intel said it incurred "**a $475 million pre-tax charge** ... to recover replacement and write-off of these microprocessors."*

*In the aftermath of the **bug** and subsequent **recall**, there was a marked increase in the use of formal verification of hardware floating point operations across the **semiconductor industry**. Prompted by the discovery of the bug, a technique ... called "word-level **model checking**" was developed in 1996. Intel went on to use **formal verification** extensively in the development of later CPU architectures. In the development of the Pentium 4, symbolic trajectory evaluation and **theorem proving** were used to **find a number of bugs that could have led to a similar recall incident** had they gone undetected.*

instruction spec.

Formal Verification: Proof Based vs. Check Based

3342 4315

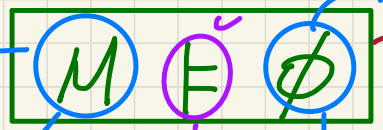


system formula written as predicates (axioms, invariants, guards, before-after predicates)

a series of applications of inference rules.

undecidable problem

proofs in general are done:
(1) manually
(2) interactively (guide the prover)



automated RG
(a) build the
(b) traverse the graph

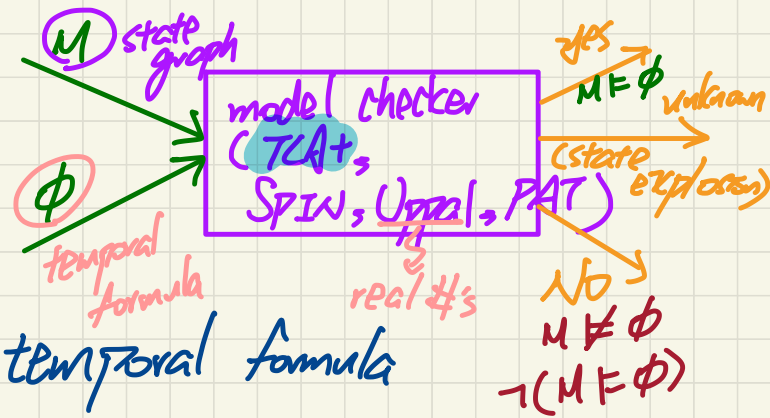
invariant
temporal logic
- LTL
- CTL

system descriptions (formulated as a labelled transition system)

"models"/ satisfies

LTS (a graph with:
(a) finite # of states & transitions
and
(b) infinite # of possible paths

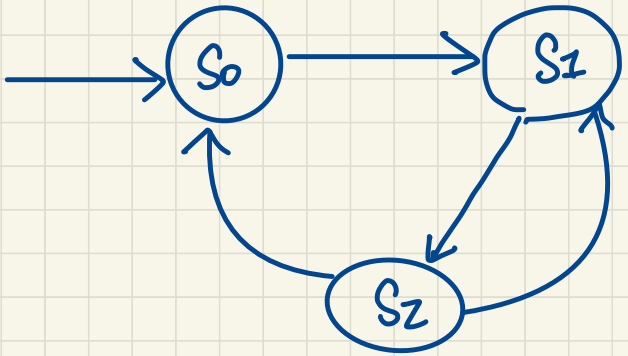
Temporal Logic



- Syntax: structural rules for writing compound temporal formula
  - → syntax tree
  - ≒ context-free grammar

- semantics: given a temporal formula ϕ that's syntactically correct, what's its meaning?
  - (1) how to express
  - (2) how to check if a formula is satisfies ($M \stackrel{?}{\models} \phi$)
  - (3) when the check fails, how to interpret & find error traces?

State Graph vs. (Computation) Path



Finite # states : S_0, S_1, S_2

# of paths is infinite :
there's at least one cycle.

(P1) $S_0 \to S_1 \to S_2 \to S_0 \to \cdots$

(P2) $S_0 \to S_1 \to S_2 \to S_1 \to S_2 \to S_1 \to \cdots$

trace.

An infinite sequence of states models the time

temporal logic

Lecture 11 - February 10

Lab2 Solution Walkthrough, Model Checking

Generalizing rounds, Function, Macro
Postconditions: getAllSuffixes
LTL Grammar: Top-Down Derivation

Announcements/Reminders

- **ProgTest1** this Thursday during the lab session
  + Please arrange your commute accordingly.
  + Test will only be canceled if the university is closed.
- **Practice Test** questions and solutions released
- **Lab1** and **Lab2** solutions released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

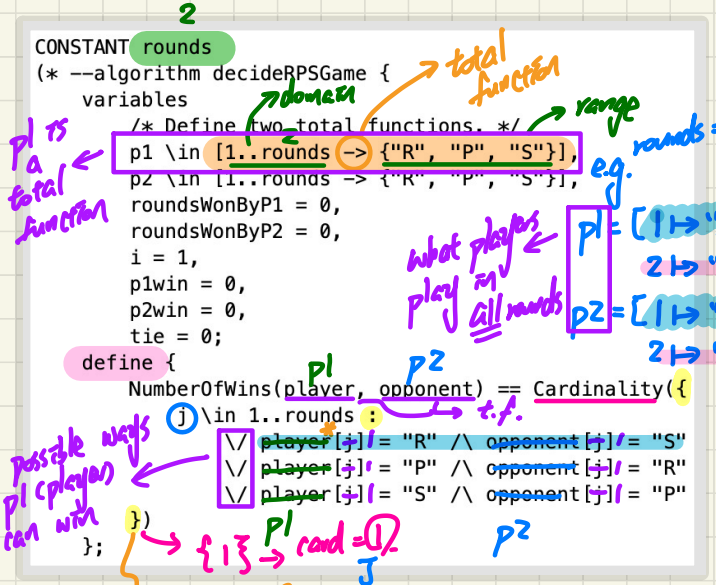
Lab2 Solution:
decideRPSGameResult
(Generalizing # of Rounds)

```
(*
--algorithm decideRPSGameResult {
variable
    p1r1 \in {"R", "P", "S"},
    p2r1 \in {"R", "P", "S"},
    numOfRoundsWonByPlayer1 = 0,
    numOfRoundsWonByPlayer2 = 0,
    p1win = FALSE,
    p2win = FALSE,
    tie = FALSE,
    i = 1,
    rounds = 2;
{
    while(i <= rounds){
        if (p1r1 = "R" /\ p2r1 = "P"){
            numOfRoundsWonByPlayer2 := numOfRoundsWonByPlayer2 + 1;
        };
        else if (p1r1 = "R" /\ p2r1 = "S"){
            numOfRoundsWonByPlayer1 := numOfRoundsWonByPlayer1 + 1;
        };
        ... /* Consider cases where p1r1 = "P" and p1r1 = "S" */

        \* Get prepared for the next round.

        choose_p1_next: with (x \in {"R", "P", "S"}) {
            p1r1 := x;
        };
        choose_p2_next: with (x \in {"R", "P", "S"}) {
            p2r1 := x;
        };
        i := i + 1;
    };

    if (numOfRoundsWonByPlayer1 = numOfRoundsWonByPlayer2){
            tie := TRUE;
    };
    else if (numOfRoundsWonByPlayer1 > numOfRoundsWonByPlayer2){
            p1win := TRUE;
    };
    else {
        p2win := TRUE;
    };
}
*)
```

initial non-deterministic choices for players in the 1st round

non-deterministic choices for players in the next round

Lab2 Solution:
decideRPSGameResult (Postcondition)




```
    {
        while (i <= rounds) {
            if (p1[i] = "R") {
                if (p2[i] = "P") {
                    roundsWonByP2 := roundsWonByP2 + 1;
                } else if (p2[i] = "S") {
                    roundsWonByP1 := roundsWonByP1 + 1;
                };
            }
            /* Consider p1[i] = "P" or p1[i] = "S"
            ...
            i := i + 1;
        };

        /* Set p1win, p2win, tie w.r.t. roundsWonByP1, roundsWonByP2.
        ...

        \* Assert consistency among variables.
        assert
            /\ roundsWonByP1 > roundsWonByP2 => (
                /\ p1win = 1
                /\ p2win = 0
                /\ tie = 0)
            /* Also consider _ < _ and _ = _
            ...

        \* Verify the correctness of win decisions.
        assert NumberOfWins(p1, p2) = roundsWonByP1;
        assert NumberOfWins(p2, p1) = roundsWonByP2;
    }
} *)
```

final value of prg. variable

calling the macro

Lab2 Solution: getAllSuffixes_v3 (1)

```
---------------------------- MODULE getAllSuffixes_v3 ----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v3 {
    variable result = input, postfixSoFar = <<>>, i = Len(input) - 1;
    {
        postfixSoFar := <<input[Len(input)]>>;
        result[Len(input)] := postfixSoFar;
        while (i > 0) {
            postfixSoFar := <<input[i]>> \o postfixSoFar;
            result[i] := postfixSoFar;
            i := i - 1;
        };
        assert \A j \in 1..Len(input): Len(result[j]) = Len(input) - j + 1;
        assert \A j \in 1..Len(result): (\A k \in 1..Len(result[j]): result[j][k] = input[j - 1 + k]);
    }
}
*)
```

concatenation

index 1 2 3

input: [23, 46, 69]

result:

[[23, 46, 69], 1

[46, 69], 2

[69]] 3

Len(r[j]) = Len(input) + 1 - j

j + Len(r[j]) = Len(input) + 1

j	Len(result[j])		Len(input)
1	3	4	3
2	2	4	3
3	1	4	3

Postcondition

(1) Understand the quantification expression on some concrete example.

(2) Practice writing from scratch.

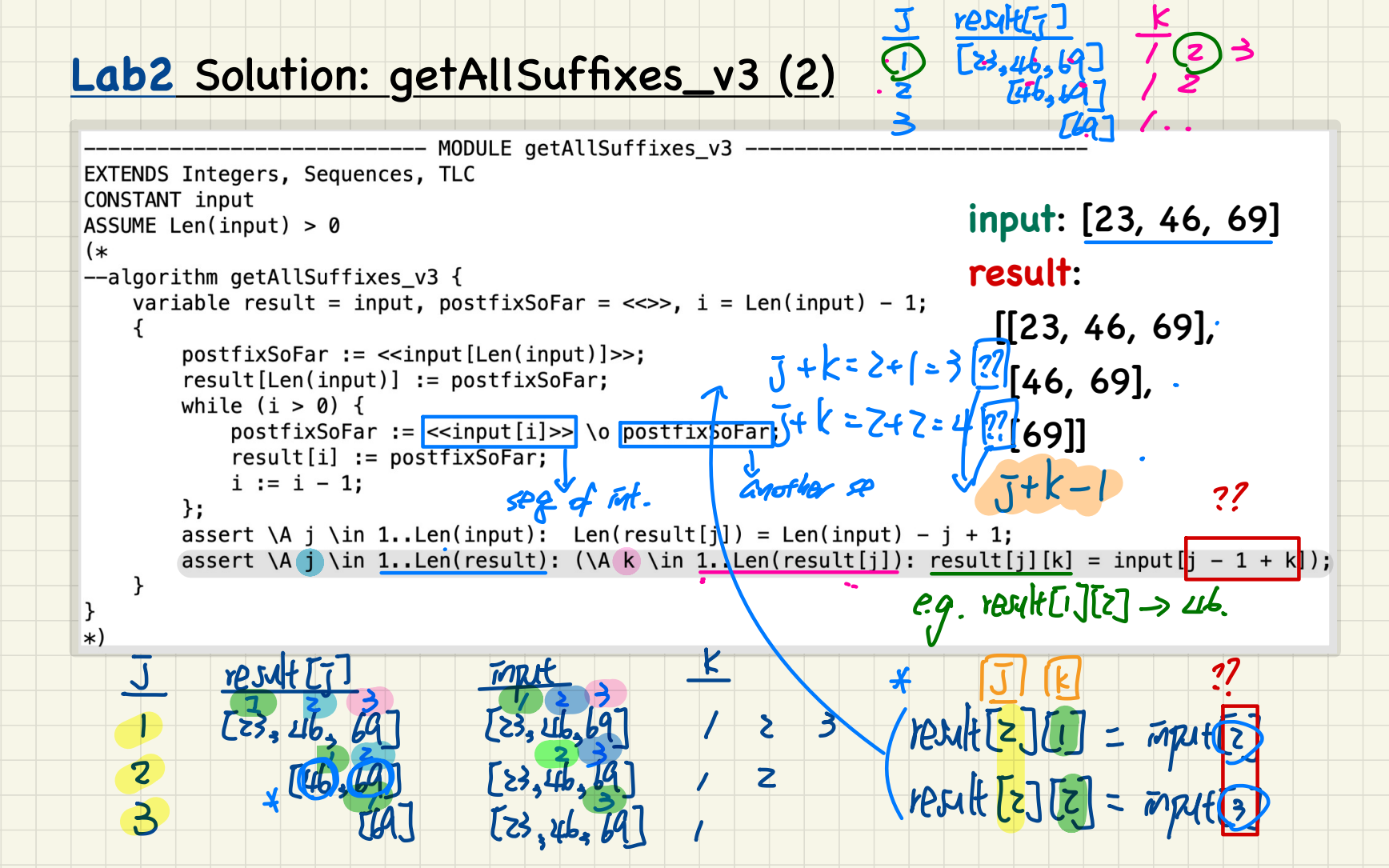
Lab2 Solution: getAllSuffixes_v3 (2)

```
---------------------------- MODULE getAllSuffixes_v3 ----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v3 {
    variable result = input, postfixSoFar = <<>>, i = Len(input) - 1;
    {
        postfixSoFar := <<input[Len(input)]>>;
        result[Len(input)] := postfixSoFar;
        while (i > 0) {
            postfixSoFar := <<input[i]>> \o postfixSoFar;
            result[i] := postfixSoFar;
            i := i - 1;
        };
        assert \A j \in 1..Len(input):  Len(result[j]) = Len(input) - j + 1;
        assert \A j \in 1..Len(result): (\A k \in 1..Len(result[j]): result[j][k] = input[j - 1 + k]);
    }
}
*)
```

input: [23, 46, 69]

result:

[[23, 46, 69],
[46, 69],
[69]]



LTL Syntax: Context-Free Grammar

$$
\begin{array}{lll}
\phi & ::= & \top \quad [\,\textit{true}\,] \\
& | & \bot \quad [\,\textit{false}\,] \\
& | & p \quad [\,\texttt{propositional atom}\,] \\
& | & (\neg\phi) \quad [\,\texttt{logical negation}\,] \\
& | & (\phi \wedge \phi) \quad [\,\texttt{logical conjunction}\,] \\
& | & (\phi \vee \phi) \quad [\,\texttt{logical disjunction}\,] \\
& | & (\phi \Rightarrow \phi) \quad [\,\texttt{logical implication}\,] \\
& | & (\mathbf{X}\,\phi) \quad [\,\texttt{neXt state}\,] \\
& | & (\mathbf{F}\,\phi) \quad [\,\texttt{some Future state}\,] \\
& | & (\mathbf{G}\,\phi) \quad [\,\texttt{all future states (Globally)}\,] \\
& | & (\phi\,\mathbf{U}\,\phi) \quad [\,\texttt{Until}\,] \\
& | & (\phi\,\mathbf{W}\,\phi) \quad [\,\texttt{Weak-until}\,] \\
& | & (\phi\,\mathbf{R}\,\phi) \quad [\,\texttt{Release}\,]
\end{array}
$$

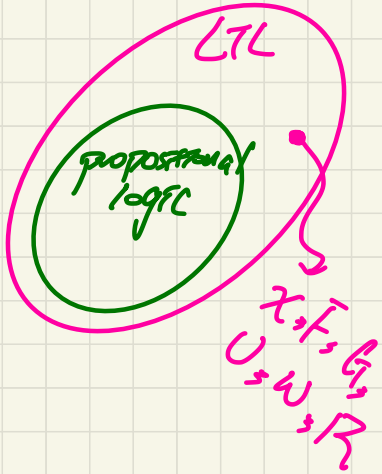
base cases

recursive cases

propositional logic

implicitly use $\forall$ and/or $\exists$

LTL



$$
\begin{array}{lll}
\phi & ::= & \top \quad \text{[true]} \\
& | & \bot \quad \text{[false]} \\
& | & p \quad \text{[propositional atom]} \\
& | & (\neg\phi) \quad \text{[logical negation]} \\
& | & (\phi \wedge \phi) \quad \text{[logical conjunction]} \\
& | & (\phi \vee \phi) \quad \text{[logical disjunction]} \\
& | & (\phi \Rightarrow \phi) \quad \text{[logical implication]} \\
& | & (\mathbf{X}\phi) \quad \text{[neXt state]} \\
& | & (\mathbf{F}\phi) \quad \text{[some Future state]} \\
& | & (\mathbf{G}\phi) \quad \text{[all future states (Globally)]} \\
& | & (\phi \mathbf{U} \phi) \quad \text{[Until]} \\
& | & (\phi \mathbf{W} \phi) \quad \text{[Weak-until]} \\
& | & (\phi \mathbf{R} \phi) \quad \text{[Release]}
\end{array}
$$

start symbol

(top-down) derivation:

$p \wedge \chi_\top$

syntactically simple

$$
\begin{aligned}
\phi &= \phi \wedge \phi \\
&= p \wedge \phi \\
&= p \wedge \chi\phi \\
&= p \wedge \chi_\top
\end{aligned}
$$

Lecture 12 - February 24

Model Checking

Operator Precedence
Parse Trees, LMDs, RMDs

Announcements/Reminders

- **ProgTest1** grading started on SUN, Feb 23
  + Expected to get raw results from TAs by MON, Mar 3
- **Lab3** to be released
- **WrittenTest1** guide to be released
- This week's office hour: 3pm, Wed
- TA contact information (on-demand for labs) on eClass

Operator Precedence

(...)

In the absence of pathentheses, what's the order of evaluation?

tightest

/* Unary Temporal Operator */

X, F, G

next, future, global

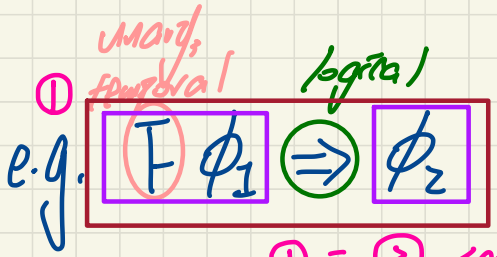
/* Binary Temporal Operator */

U, W, R

until, weak until, release

/* Logical Operator */

$\neg$, $\wedge$, $\vee$, $\Rightarrow$



① ≡ ② same meaning

② ≠ ③ not semantically equivalent.

vs.

② $(F\,\phi_1) \Rightarrow \phi_2$

vs.

③ $F(\phi_1 \Rightarrow \phi_2)$

Letter	symbol
x	○
F	◇
G	□

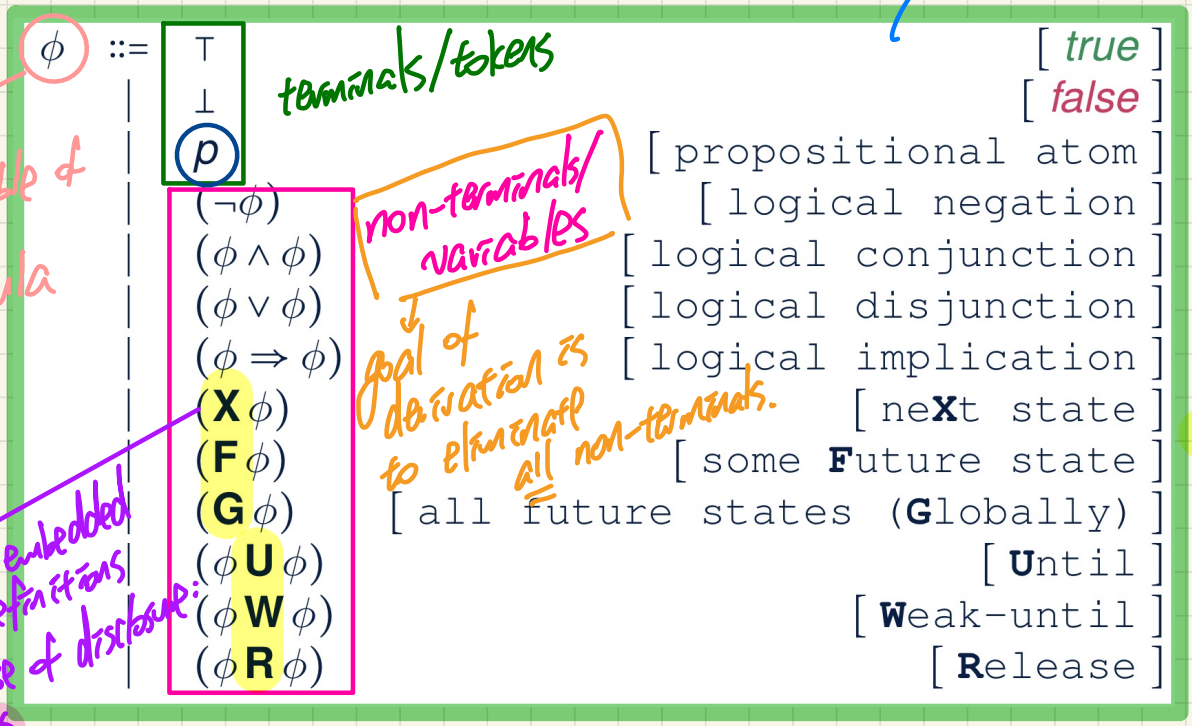
(1) $\dfrac{x\,\phi}{\text{III}}$

○ϕ

(2) $\dfrac{F\,G\,\phi}{\text{III}}$

◇□ϕ

Parsing: Some Practical Knowledge



Assumption: Operator **precedence** considered first before the CFG.

Interpreting a Formula: Parse Trees (1)

$$\phi ::= \top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid (\mathbf{X}\phi) \mid (\mathbf{F}\phi) \mid (\mathbf{G}\phi) \mid (\phi\,\mathbf{U}\,\phi) \mid (\phi\,\mathbf{W}\,\phi) \mid (\phi\,\mathbf{R}\,\phi)$$

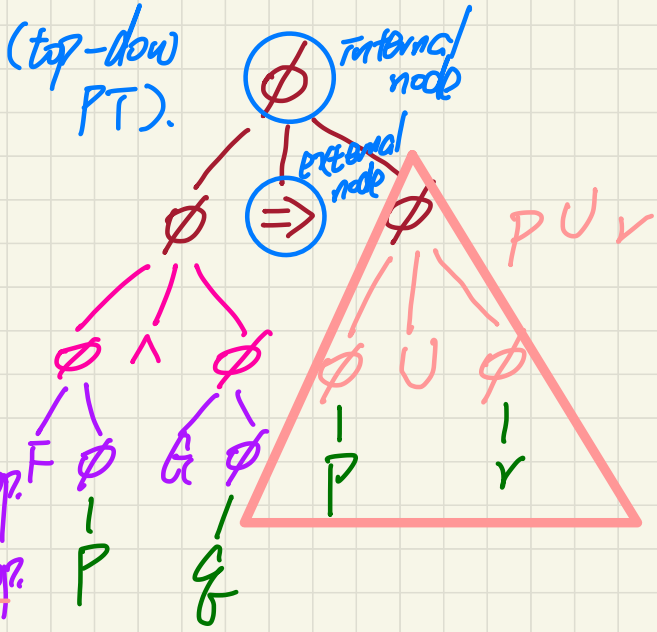
[true] [false] [propositional atom] [logical negation] [logical conjunction] [logical disjunction] [logical implication] [neXt state] [some Future state] [all future states (Globally)] [Until] [Weak-until] [Release]

not a keyword

Can be: p, q, r, ... (>0)

$$\mathbf{F}\,p \wedge \mathbf{G}\,q \Rightarrow p\,\mathbf{U}\,r$$

(top-down PT).



Before considering the CFG, first use operator precedence to group terms

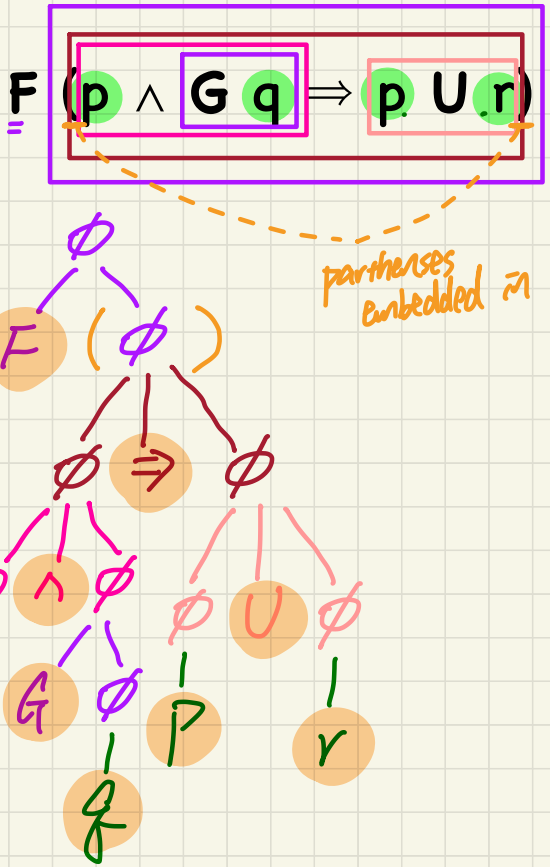
Unary Temp. op.

Binary Temp. op.

Logical op.

Interpreting a Formula: Parse Trees (2)

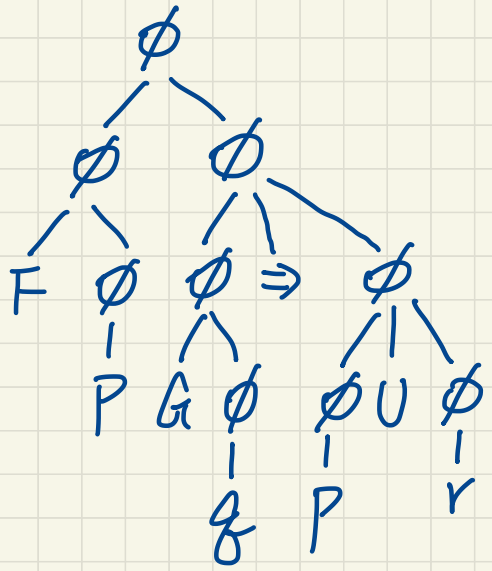
ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\,\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\,\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\,\phi)$	[all future states (**G**lobally)]
	|	$(\phi\,\mathbf{U}\,\phi)$	[**U**ntil]
	|	$(\phi\,\mathbf{W}\,\phi)$	[**W**eak-until]
	|	$(\phi\,\mathbf{R}\,\phi)$	[**R**elease]



Interpreting a Formula: Parse Trees (3)

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\,\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\,\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\,\phi)$	[all future states (**G**lobally)]
	|	$(\phi\,\mathbf{U}\,\phi)$	[**U**ntil]
	|	$(\phi\,\mathbf{W}\,\phi)$	[**W**eak-until]
	|	$(\phi\,\mathbf{R}\,\phi)$	[**R**elease]

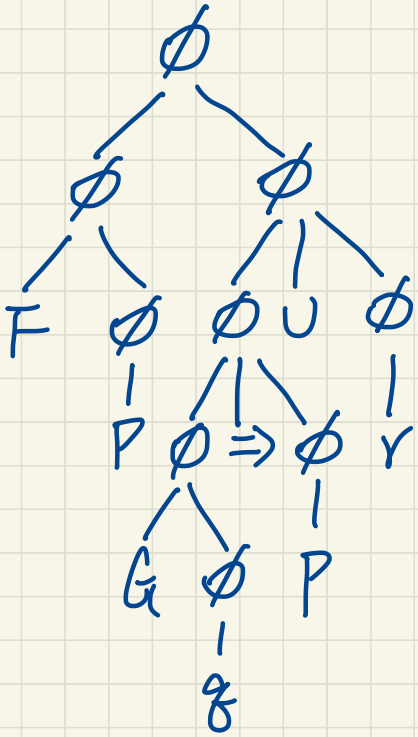
F p ∧ (G q ⇒ p U r)



Interpreting a Formula: Parse Trees (4)

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\phi)$	[all future states (**G**lobally)]
	|	$(\phi \mathbf{U} \phi)$	[**U**ntil]
	|	$(\phi \mathbf{W} \phi)$	[**W**eak-until]
	|	$(\phi \mathbf{R} \phi)$	[**R**elease]

$$\mathbf{F}\ p \wedge ((\mathbf{G}\ q \Rightarrow p)\ \mathbf{U}\ r)$$



Interpreting a Formula: LMD (1)

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\phi)$	[all future states (**G**lobally)]
	|	$(\phi \mathbf{U} \phi)$	[**U**ntil]
	|	$(\phi \mathbf{W} \phi)$	[**W**eak-until]
	|	$(\phi \mathbf{R} \phi)$	[**R**elease]

$\mathbf{F}\,p \wedge \mathbf{G}\,q \Rightarrow p\ \mathbf{U}\ r$

LMD

ϕ → left-most non-terminal

① $\phi \Rightarrow \phi$ — partially completed formula after 1st derivation

is derived to

② $\phi \wedge \phi \Rightarrow \phi$

③ $\mathbf{F}\phi \wedge \phi \Rightarrow \phi$

④ $\mathbf{F}p \wedge \phi \Rightarrow \phi$

⑤ $\mathbf{F}p \wedge \mathbf{G}\phi \Rightarrow \phi$

⑥ $\mathbf{F}p \wedge \mathbf{G}q \Rightarrow \phi$

⑦ $\mathbf{F}p \wedge \mathbf{G}q \Rightarrow \phi\ \mathbf{U}\ \phi$

⑧ $\mathbf{F}p \wedge \mathbf{G}q \Rightarrow p\ \mathbf{U}\ \phi$

⑨ $\mathbf{F}p \wedge \mathbf{G}q \Rightarrow p\ \mathbf{U}\ r$

Interpreting a Formula: LMD (2)

$$\mathbf{F}\,(p \wedge \mathbf{G}\,q \Rightarrow p\ \mathbf{U}\ r)$$

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\,\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\,\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\,\phi)$	[all future states (**G**lobally)]
	|	$(\phi\ \mathbf{U}\ \phi)$	[**U**ntil]
	|	$(\phi\ \mathbf{W}\ \phi)$	[**W**eak-until]
	|	$(\phi\ \mathbf{R}\ \phi)$	[**R**elease]

implicit parentheses

① ϕ

② $\mathbf{F}(\phi)$

③ $\mathbf{F}(\phi \Rightarrow \phi)$

④ $\mathbf{F}(\phi \wedge \phi \Rightarrow \phi)$

⑤ $\mathbf{F}(p \wedge \phi \Rightarrow \phi)$

⑥ $\mathbf{F}(p \wedge \mathbf{G}\,\phi \Rightarrow \phi)$

⑦ $\mathbf{F}(p \wedge \mathbf{G}\,q \Rightarrow \phi)$

⑧ $\mathbf{F}(p \wedge \mathbf{G}\,q \Rightarrow \phi\ \mathbf{U}\ \phi)$

⑨ $\mathbf{F}(p \wedge \mathbf{G}\,q \Rightarrow p\ \mathbf{U}\ \phi)$

$\mathbf{F}(p \wedge \mathbf{G}\,q \Rightarrow p\ \mathbf{U}\ r)$

Interpreting a Formula: LMD (3)

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\phi)$	[all future states (**G**lobally)]
	|	$(\phi\,\mathbf{U}\,\phi)$	[**U**ntil]
	|	$(\phi\,\mathbf{W}\,\phi)$	[**W**eak-until]
	|	$(\phi\,\mathbf{R}\,\phi)$	[**R**elease]

$$\mathbf{F}\ p \wedge (\mathbf{G}\ q \Rightarrow p\ \mathbf{U}\ r)$$

ϕ

① $\Rightarrow \phi \wedge \phi$

② $\Rightarrow \mathbf{F}\,\phi \wedge \phi$

③ $\Rightarrow \mathbf{F}\,p \wedge \phi$

④ $\Rightarrow \mathbf{F}\,p \wedge (\phi \Rightarrow \phi)$

⑤ $\Rightarrow \mathbf{F}\,p \wedge (\mathbf{G}\,\phi \Rightarrow \phi)$

⑥ $\Rightarrow \mathbf{F}\,p \wedge (\mathbf{G}\,q \Rightarrow \phi)$

⑦ $\Rightarrow \mathbf{F}\,p \wedge (\mathbf{G}\,q \Rightarrow \phi\ \mathbf{U}\ \phi)$

⑧ $\Rightarrow \mathbf{F}\,p \wedge (\mathbf{G}\,q \Rightarrow p\ \mathbf{U}\ \phi)$

⑨ $\Rightarrow \mathbf{F}\,p \wedge (\mathbf{G}\,q \Rightarrow p\ \mathbf{U}\ r)$

Interpreting a Formula: LMD (4)

$$\phi ::= \top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid (\mathbf{X}\phi) \mid (\mathbf{F}\phi) \mid (\mathbf{G}\phi) \mid (\phi\,\mathbf{U}\,\phi) \mid (\phi\,\mathbf{W}\,\phi) \mid (\phi\,\mathbf{R}\,\phi)$$

- $\top$ [*true*]
- $\bot$ [*false*]
- p [propositional atom]
- $(\neg\phi)$ [logical negation]
- $(\phi \wedge \phi)$ [logical conjunction]
- $(\phi \vee \phi)$ [logical disjunction]
- $(\phi \Rightarrow \phi)$ [logical implication]
- $(\mathbf{X}\phi)$ [neXt state]
- $(\mathbf{F}\phi)$ [some Future state]
- $(\mathbf{G}\phi)$ [all future states (Globally)]
- $(\phi\,\mathbf{U}\,\phi)$ [Until]
- $(\phi\,\mathbf{W}\,\phi)$ [Weak-until]
- $(\phi\,\mathbf{R}\,\phi)$ [Release]

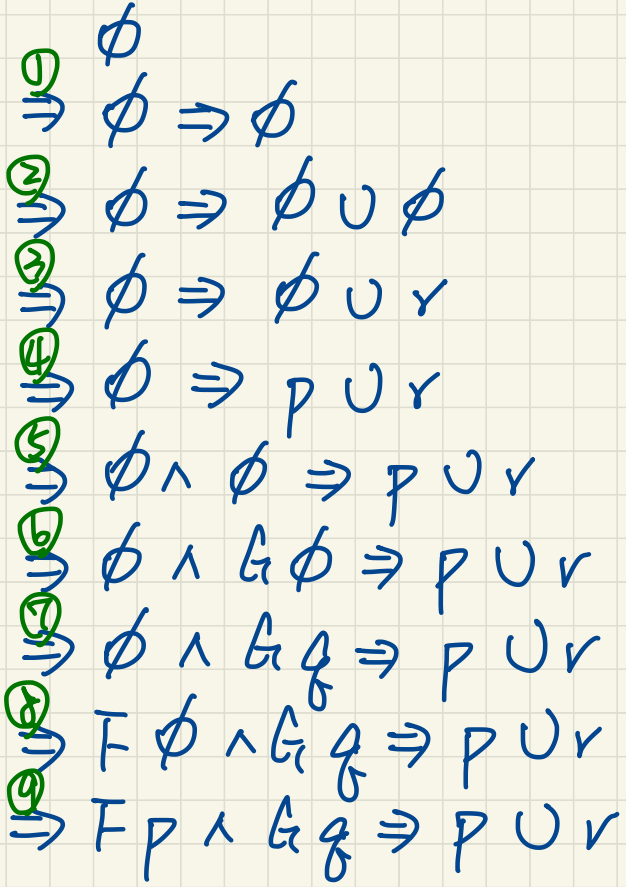
$$\mathbf{F}\ p \wedge ((\mathbf{G}\ q \Rightarrow p)\ \mathbf{U}\ r)$$

1. ϕ
2. $\Rightarrow \phi \wedge \phi$
3. $\Rightarrow \mathbf{F}\,\phi \wedge \phi$
4. $\Rightarrow \mathbf{F}\,p \wedge \phi$
5. $\Rightarrow \mathbf{F}\,p \wedge (\phi\ \mathbf{U}\ \phi)$
6. $\Rightarrow \mathbf{F}\,p \wedge ((\phi \Rightarrow \phi)\ \mathbf{U}\ \phi)$
7. $\Rightarrow \mathbf{F}\,p \wedge ((\mathbf{G}\,\phi \Rightarrow \phi)\ \mathbf{U}\ \phi)$
8. $\Rightarrow \mathbf{F}\,p \wedge ((\mathbf{G}\,q \Rightarrow \phi)\ \mathbf{U}\ \phi)$
9. $\Rightarrow \mathbf{F}\,p \wedge ((\mathbf{G}\,q \Rightarrow p)\ \mathbf{U}\ \phi)$
10. $\Rightarrow \mathbf{F}\,p \wedge ((\mathbf{G}\,q \Rightarrow p)\ \mathbf{U}\ r)$

Interpreting a Formula: RMD (1)

F p ∧ G q ⇒ p U r

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\phi)$	[all future states (**G**lobally)]
	|	$(\phi \mathbf{U} \phi)$	[**U**ntil]
	|	$(\phi \mathbf{W} \phi)$	[**W**eak-until]
	|	$(\phi \mathbf{R} \phi)$	[**R**elease]



Interpreting a Formula: RMD (2)

$\mathbf{F}\,(p \wedge \mathbf{G}\,q \Rightarrow p\ \mathbf{U}\ r)$

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\,\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\,\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\,\phi)$	[all future states (**G**lobally)]
	|	$(\phi\,\mathbf{U}\,\phi)$	[**U**ntil]
	|	$(\phi\,\mathbf{W}\,\phi)$	[**W**eak-until]
	|	$(\phi\,\mathbf{R}\,\phi)$	[**R**elease]

ϕ

① $\Rightarrow F(\phi)$

② $\Rightarrow F(\phi \Rightarrow \phi)$

③ $\Rightarrow F(\phi \Rightarrow \phi \cup \phi)$

④ $\Rightarrow F(\phi \Rightarrow \phi \cup r)$

⑤ $\Rightarrow F(\phi \Rightarrow p \cup r)$

⑥ $\Rightarrow F(\phi \wedge \phi \Rightarrow p \cup r)$

⑦ $\Rightarrow F(\phi \wedge G\phi \Rightarrow p \cup r)$

⑧ $\Rightarrow F(\phi \wedge Gq \Rightarrow p \cup r)$

⑨ $\Rightarrow F(p \wedge Gq \Rightarrow p \cup r)$

Interpreting a Formula: RMD (3)

$$\mathbf{F}\ p \wedge (\mathbf{G}\ q \Rightarrow p\ \mathbf{U}\ r)$$

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\phi)$	[all future states (**G**lobally)]
	|	$(\phi\ \mathbf{U}\ \phi)$	[**U**ntil]
	|	$(\phi\ \mathbf{W}\ \phi)$	[**W**eak-until]
	|	$(\phi\ \mathbf{R}\ \phi)$	[**R**elease]

ϕ

① $\Rightarrow \phi \wedge \phi$

② $\Rightarrow \phi \wedge (\phi \Rightarrow \phi)$

③ $\Rightarrow \phi \wedge (\phi \Rightarrow \phi \cup \phi)$

④ $\Rightarrow \phi \wedge (\phi \Rightarrow \phi \cup r)$

⑤ $\Rightarrow \phi \wedge (\phi \Rightarrow p \cup r)$

⑥ $\Rightarrow \phi \wedge (G\phi \Rightarrow p \cup r)$

⑦ $\Rightarrow \phi \wedge (G\,q \Rightarrow p \cup r)$

⑧ $\Rightarrow F\phi \wedge (G\,q \Rightarrow p \cup r)$

⑨ $\Rightarrow F\,p \wedge (G\,q \Rightarrow p \cup r)$

Interpreting a Formula: RMD (4)

$F\ p \wedge ((G\ q \Rightarrow p)\ U\ r)$

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional atom]
	|	$(\neg\phi)$	[logical negation]
	|	$(\phi \wedge \phi)$	[logical conjunction]
	|	$(\phi \vee \phi)$	[logical disjunction]
	|	$(\phi \Rightarrow \phi)$	[logical implication]
	|	$(\mathbf{X}\phi)$	[ne**X**t state]
	|	$(\mathbf{F}\phi)$	[some **F**uture state]
	|	$(\mathbf{G}\phi)$	[all future states (**G**lobally)]
	|	$(\phi\ \mathbf{U}\ \phi)$	[**U**ntil]
	|	$(\phi\ \mathbf{W}\ \phi)$	[**W**eak-until]
	|	$(\phi\ \mathbf{R}\ \phi)$	[**R**elease]

ϕ

① $\Rightarrow \phi \wedge (\phi)$

② $\Rightarrow \phi \wedge (\phi\ U\ \phi)$

③ $\Rightarrow \phi \wedge (\phi\ U\ r)$

④ $\Rightarrow \phi \wedge ((\phi \Rightarrow \phi)\ U\ r)$

⑤ $\Rightarrow \phi \wedge ((\phi \Rightarrow p)\ U\ r)$

⑥ $\Rightarrow \phi \wedge ((G\ \phi \Rightarrow p)\ U\ r)$

⑦ $\Rightarrow \phi \wedge ((G\ q) \Rightarrow p)\ U\ r)$

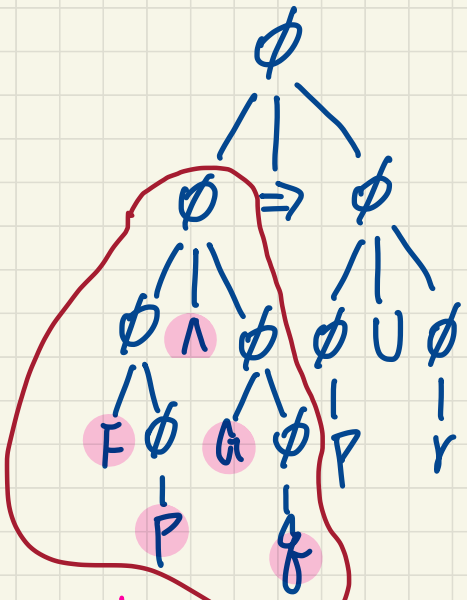
⑧ $\Rightarrow F\phi \wedge ((G\ q) \Rightarrow p)\ U\ r)$

⑨ $\Rightarrow F\ p \wedge ((G\ q) \Rightarrow p)\ U\ r)$

Interpreting a Formula: PT vs. LMD vs. RMD

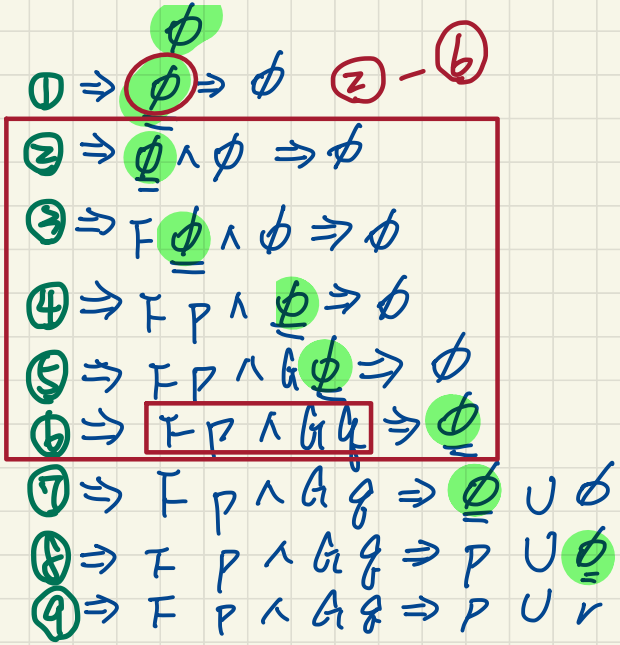
$\mathbf{F\,p \wedge G\,q \Rightarrow p\,U\,r} \in L(g)$

Parse Tree

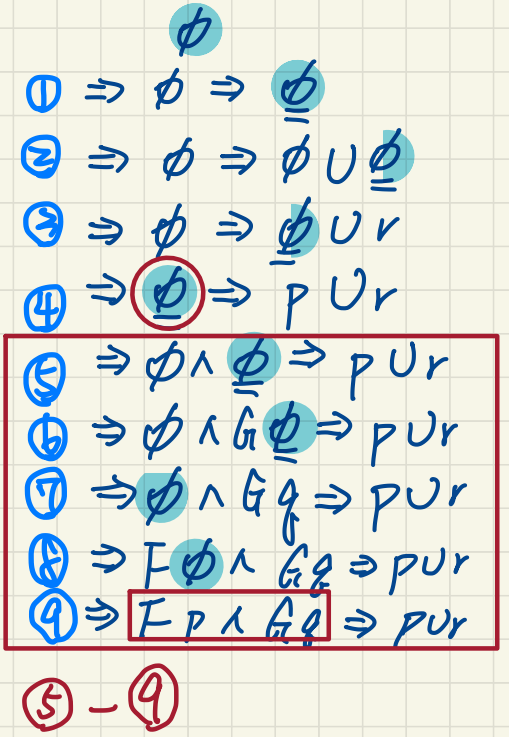


subformula/subtree: $Fp \wedge Gq$

LMD



RMD



Deriving Subformulas from a Parse Tree

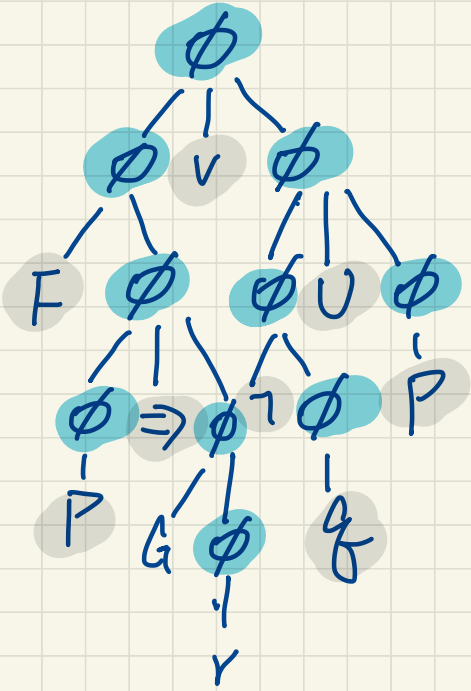
Enumerate all subformulas of:

$$\mathbf{F}\ (p \Rightarrow \mathbf{G}\ r) \vee ((\neg\ q)\ \mathbf{U}\ p)$$

PT.

How many subtrees in PT?

10 subtrees



Lecture 13 - February 26

Model Checking

Subformula
Labeled Transition System (LTS)
Paths, Path Suffixes

Announcements/Reminders

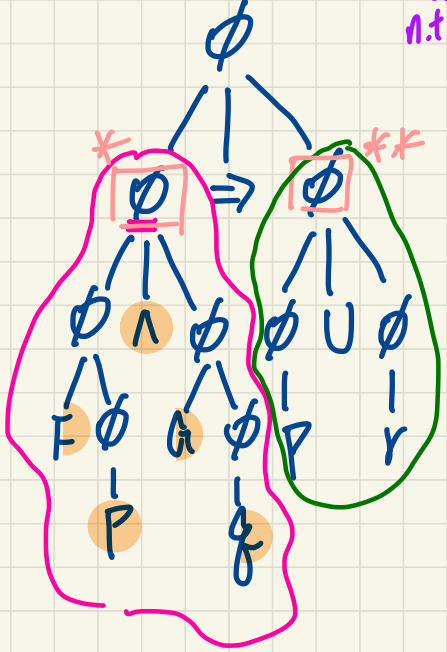
- **WrittenTest1** guide & examples by the end of **Friday**
  + All lectures materials up to and including today
  + **Lab1** and **Lab2** (solutions & in-class discussion)
  + **Review Q&A** (Zoom): 7:30pm on Monday, Mar 3
- **Lab3** to be released next Wednesday
- Tomorrow's lab (9 to 10): office hour for your **WT1**
- This week's office hour: 3pm, Wed
- TA contact information (on-demand for labs) on eClass

$\phi \overset{①}{\Rightarrow} \boxed{\phi} \Rightarrow \phi \overset{②}{\Rightarrow} \phi \wedge \phi \Rightarrow \phi \overset{③}{\Rightarrow} \cdots \overset{⑥}{\Rightarrow} Fp \wedge Gq \Rightarrow \phi \overset{⑦}{\Rightarrow} \cdots$

Interpreting a Formula: PT vs. LMD vs. RMD

$\mathbf{F\,p \wedge G\,q \Rightarrow p\,U\,r} \in L(g)$

Parse Tree



*LMD

picked as left-most n.t. by ②

ϕ

① $\Rightarrow \boxed{\phi} \Rightarrow \phi$ ②-⑥

② $\Rightarrow \phi \wedge \phi \Rightarrow \phi$

③ $\Rightarrow F\phi \wedge \phi \Rightarrow \phi$

④ $\Rightarrow Fp \wedge \phi \Rightarrow \phi$

⑤ $\Rightarrow Fp \wedge G\phi \Rightarrow \phi$

⑥ $\Rightarrow Fp \wedge Gq \Rightarrow \phi$

⑦ $\Rightarrow Fp \wedge Gq \Rightarrow \phi\,U\,\phi$

⑧ $\Rightarrow Fp \wedge Gq \Rightarrow p\,U\,\phi$

⑨ $\Rightarrow Fp \wedge Gq \Rightarrow p\,U\,r$

RMD

picked as right-most n.t. by ⑤.

ϕ

① $\Rightarrow \phi \Rightarrow \phi$

② $\Rightarrow \phi \Rightarrow \phi\,U\,\phi$

③ $\Rightarrow \phi \Rightarrow \phi\,U\,r$

④ $\Rightarrow \boxed{\phi} \Rightarrow p\,U\,r$

⑤ $\Rightarrow \phi \wedge \phi \Rightarrow p\,U\,r$

⑥ $\Rightarrow \phi \wedge G\phi \Rightarrow p\,U\,r$

⑦ $\Rightarrow \phi \wedge Gq \Rightarrow p\,U\,r$

⑧ $\Rightarrow F\phi \wedge Gq \Rightarrow p\,U\,r$

⑨ $\Rightarrow \boxed{Fp \wedge Gq} \Rightarrow p\,U\,r$

⑤-⑨

Deriving Subformulas from a Parse Tree

Enumerate all subformulas of:

$$F\,(p \Rightarrow G\,r) \vee ((\neg\, q)\ U\ p)$$

PT. How many subtrees in PT?

Q1. = 10 subtrees

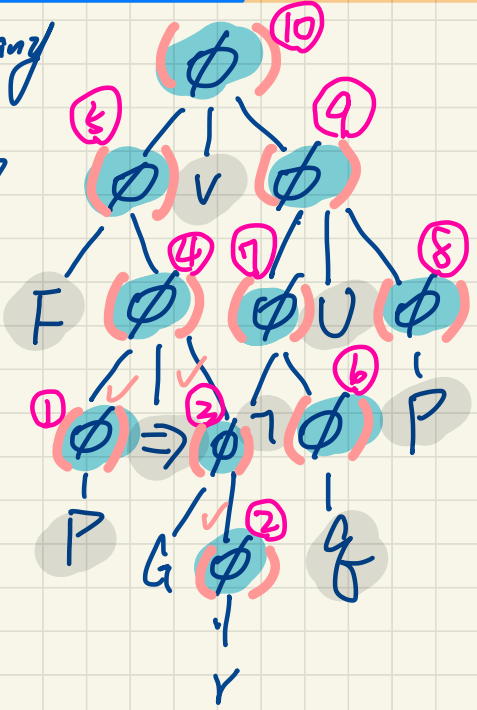
Q2. How many subformulas?

10

Q3. How many distinct subformulas?

9

(∵ ① and ⑧ are the same)



① p
② r
③ G(r)
④ (p) ⇒ (G(r))
⑤ F((p) ⇒ (G(r)))
⑥ q
⑦ ¬(q)
⑧ p
⑨ (¬(q)) U (p)
⑩ (⑤) ∨ (⑨)

formal model (specified as a LTS).

Labelled Transition System (LTS)

$M = (S, \longrightarrow, L)$, given P

- S: a finite # of states
- $\longrightarrow$: transition relation (a set of pairs; cannot be a function: a state may be transitioned into more than one states)
- L: * Labelling Function
- P: a set of atomic propositions (which evaluates to T or F)

$\longrightarrow \in S \leftrightarrow S$: set of all possible relations between S and S.

$L \in S \rightarrow \mathbb{P}(P)$ **

** (total function)

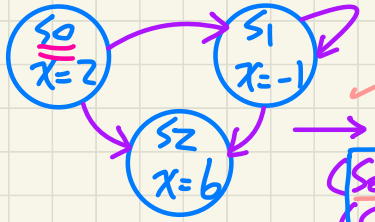
$\{\emptyset, \{x>0\}, \{x>4\}, \{x>0, x>4\}\}$

$S \rightarrow S'$

$\text{ran}(\{s_0\} \lhd \longrightarrow) = \{(s_0, s_1), (s_0, s_2)\}$

$= \{s_1, s_2\}$

e.g. $P = \{x>0, x>4\}$



$\longrightarrow = \{(s_0, s_1), (s_0, s_2), (s_1, s_1), (s_1, s_2)\}$

$L(s_0) = \{x>0\}$

$L(s_1) = \emptyset$ {}

$L(s_2) = \{x>0, x>4\}$

Q. Formulate deadlock freedom:

From any state, it is always possible to make progress.

$\forall s \cdot s \in S \Rightarrow (\exists s' \cdot s' \in S \wedge$ ① $(s, s') \in \longrightarrow$ ② $s' \in \longrightarrow[\{s\}]$ ③ $s' \in \text{ran}(\{s\} \lhd \longrightarrow)$ ④ use range restriction $)$

Labelled Transition System (LTS)

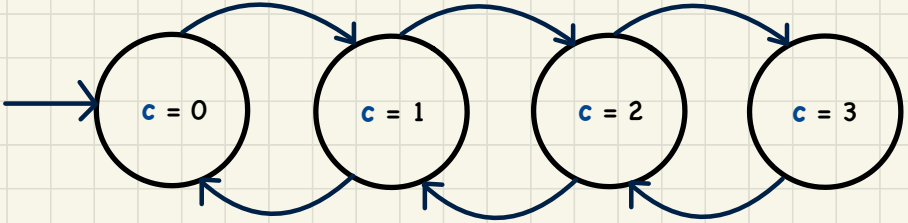
Exercises Consider the system with a counter c with the following assumption:

$$0 \leq c \leq 3$$

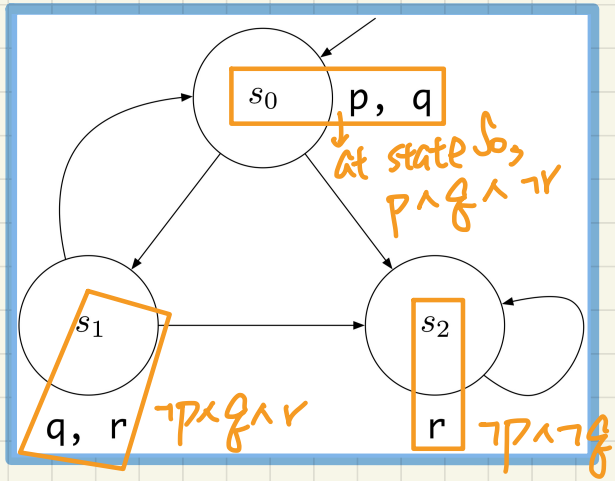
Say c is initialized 0 and may be incremented (via a transition *inc*, enabled when $c < 3$) or decremented (via a transition *dec*, enabled when $c > 0$).

- **Draw** a ***state graph*** of this system.
- **Formulate** the state graph as an ***LTS*** (via a triple $(S, \longrightarrow, L)$).

  Assume: Set P of atoms is: $\{ c \geq 1, c \leq 1 \}$



Labelled Transition System (LTS): Formulation & Paths



at state s_0, $p \wedge q \wedge \neg r$

$\neg p \wedge q \wedge r$

$\neg p \wedge \neg q \wedge r$

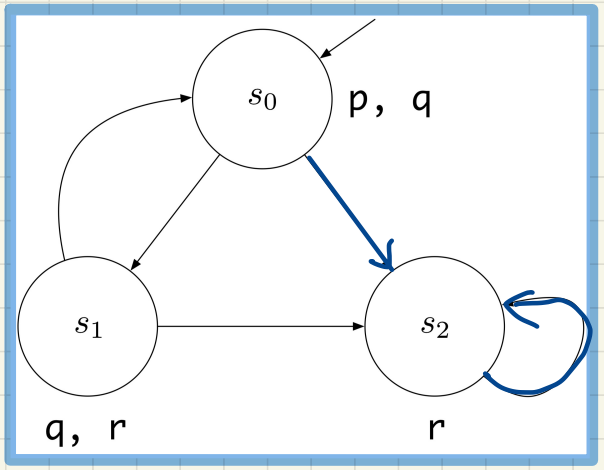
Assume: $P = \{p, q, r\}$

$M = (S, \longrightarrow, L)$

$S = \{s_0, s_1, s_2\}$

$\longrightarrow = \{(s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_2), (s_2, s_2)\}$

$L = \{(s_0, \{p, q\}), (s_1, \{q, r\}), (s_2, \{r\})\}$



name of state

Path : $s_0 \to s_2 \to s_2 \to s_2 \to \cdots$

$\pi =$

notions of path components:

s_1 s_2 s_3 s_4

indices

Path Suffix

π^0 X

$\pi^1 = \pi$

$\pi^2 = s_2 \to s_2 \to \cdots$

$\pi = s_1 \to s_2 \to s_3 \to s_4 \to s_5$

$(\pi^2)^3 = s_4 \to s_5 \to \cdots$

$= \pi^4$

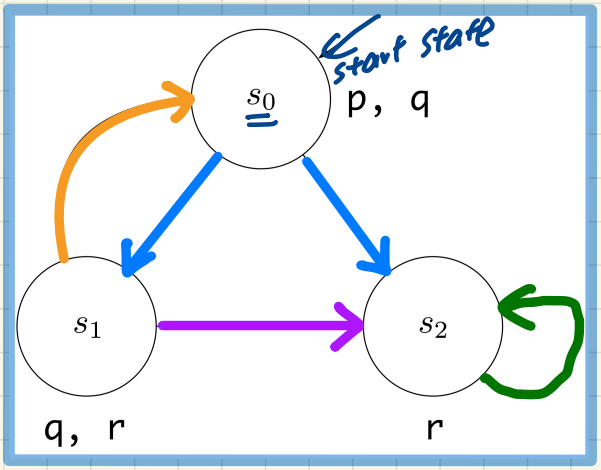
Lecture 14 - March 3

Model Checking

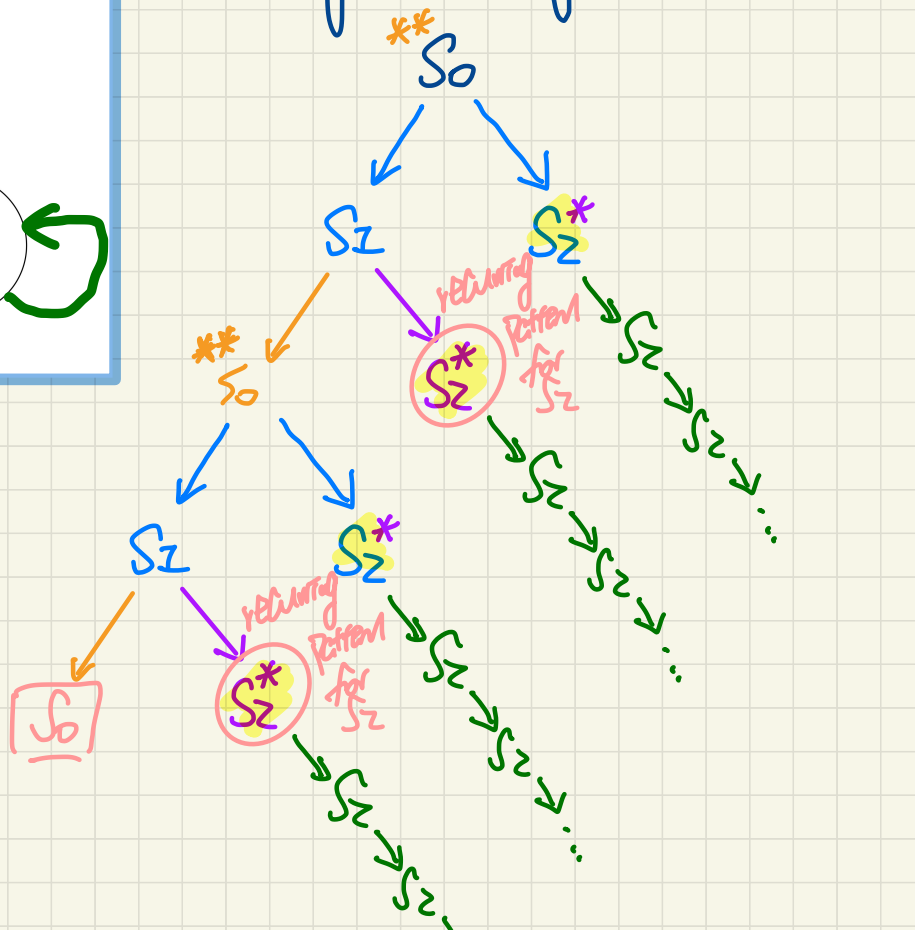
Unfolding/Unwinding Paths
Satisfaction Relations: Path vs. Model
Formulations: X, F, G

Announcements/Reminders

- **ProgTest1** results to be released (by end of Friday)
- **WrittenTest1** guide & examples released
  + **Review Q&A** (Zoom): 7:30pm on Monday, Mar 3
- **Lab3** to be released after WrittenTest1
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

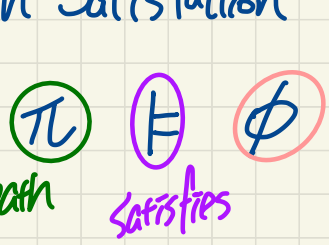


Unfolding/Unwinding (transitions)

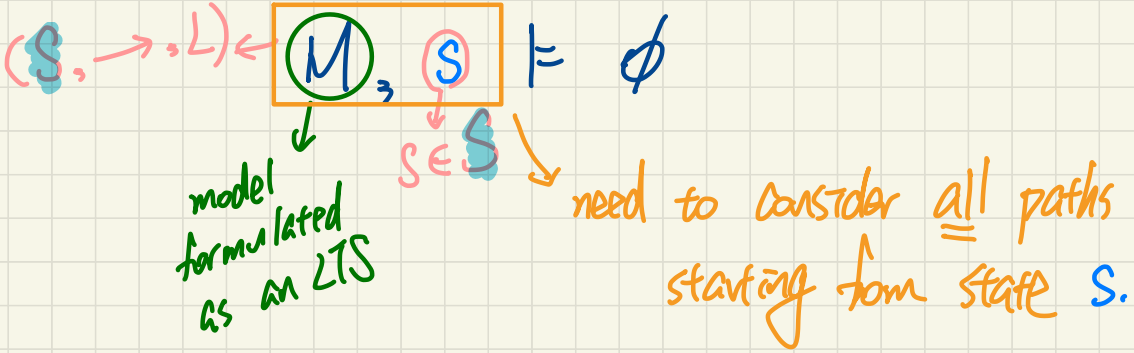


Satisfaction Relations

(1) Path Satisfaction

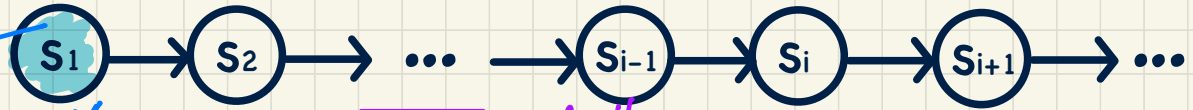


(2) Model Satisfaction



Path Satisfaction: Logical Operations

A path satisfies a proposition
if its initial state ("current state") satisfies it.



first state of π

base cases

$\pi \models p \Leftrightarrow p \in \mathcal{L}(s_1)$ — set of all propositional atoms satisfied in the 1st state (s_1) of π; $p \in \mathcal{L}(s_i)$

$\pi \models \top$ (whether the 1st state of π satisfies $\top$)

$\pi \models \bot$

recursive cases

$\pi \models \neg\phi \Leftrightarrow \neg(\pi \models \phi)$

$\pi \models \phi_1 \wedge \phi_2 \Leftrightarrow \pi \models \phi_1 \wedge \pi \models \phi_2$

$\pi \models \phi_1 \vee \phi_2 \Leftrightarrow \pi \models \phi_1 \vee \pi \models \phi_2$

$\pi \models \phi_1 \Rightarrow \phi_2 \Leftarrow \pi \models \phi_1 \Rightarrow \pi \models \phi_2$

* $\forall i \cdot i \,\%\, 2 = 0 \wedge i > 0 \Rightarrow \pi^i \models p$ (2, 4, 6, ...)

π^i: path starting from the ith component of π

$\forall i \cdot i \in \mathbb{N} \wedge i > 0 \Rightarrow \pi^{2i} \models p$

* Q: Express that **all** the even-numbered states satisfies a proposition p.

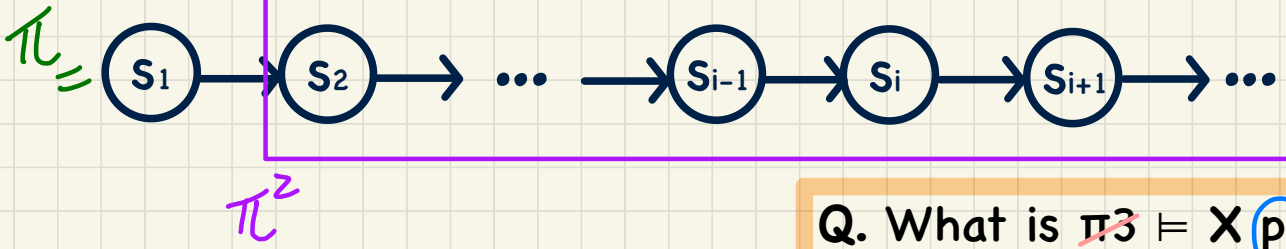
($s_2, s_4, s_6, \ldots$)

Path Satisfaction: Temporal Operations (1)

A path satisfies $X\phi$ (next op.) → nested, arbitrarily complicated LTL formula.

if the next state (of the "current state") satisfies it.

$\pi \models X\phi$ $\quad$ $\pi^2 \models \phi$



$\pi = s_1 \to s_2 \to s_3 \to s_4 \to s_5 \to \cdots$ $\quad$ π^3 $\quad$ $(\pi^3)^2$ $\quad$ π^4

Q. What is $\pi^3 \models X\,p$ checking? (p: prop. atom)

Formulation (over a path)

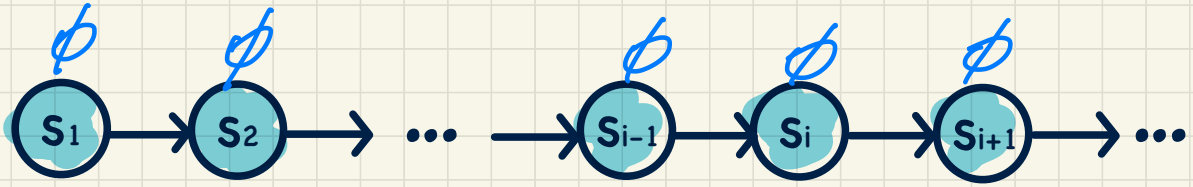
$$\pi \models X\phi \iff \pi^2 \models \phi$$

π^2: a path starting from the 2nd component of ϕ

$$\pi^3 \models Xp \iff (\pi^3)^2 \models p \iff \pi^4 \models p \iff p \in L(s_4)$$

Path Satisfaction: Temporal Operations (2)

A path satisfies $\mathbf{G}\phi$ (Global) → Henceforth, ϕ is true.
if the every state satisfies it.



Formulation (over a path)

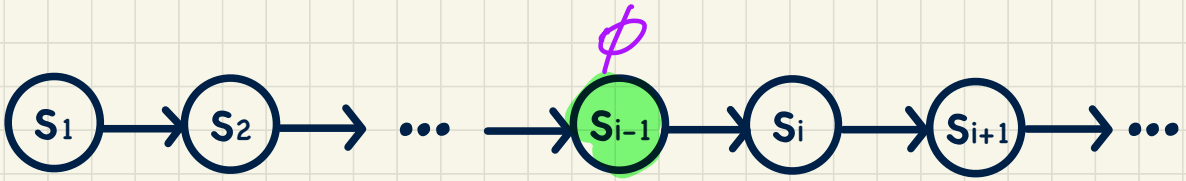
$$\pi \vDash \mathbf{G}\phi \iff \forall i \cdot i \geq 1 \Rightarrow \pi^i \vDash \phi$$

□

$i \in \mathbb{N}$ X
$\because i = 0 \Rightarrow \pi^0$ undefined.

Path Satisfaction: Temporal Operations (3)

A path satisfies $\mathbf{F}\phi$ (Future) ⟶ Eventually, ϕ is true.
if some future state satisfies it.

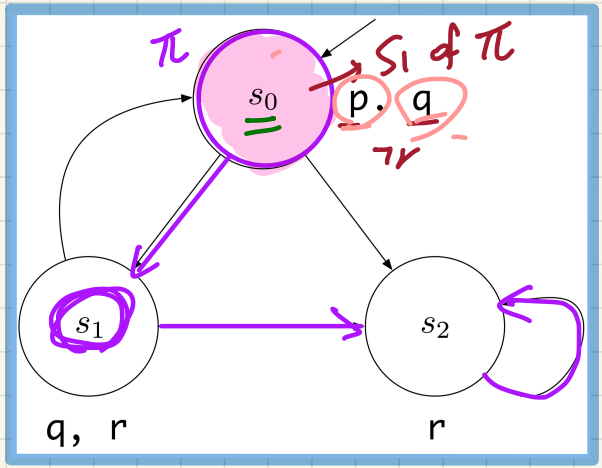


Formulation (over a path)

$$\pi \vDash \boxed{\mathbf{F}}\phi \iff \exists i \cdot i \geq 1 \wedge \pi^i \vDash \phi$$

$\boxed{\mathbf{F}} \equiv \Diamond$

Model vs. Path Satisfaction: Exercises (1.1)



Recall: $\pi \models p \Leftrightarrow p \in L(s_1)$

Say: $\pi = s_0 \to s_1 \to s_2 \to s_2 \to \dots$

- $\pi \models \top$ (T)
- $\pi \not\models \bot$ (T) $\quad \pi \not\models \bot \equiv \neg(\pi \models \bot)$
- $\pi \models p \wedge q \equiv \pi \models p \wedge \pi \models q \equiv$ (T) $\quad p \in L(s_0)$, $q \in L(s_0)$
- $\pi \models p \vee q$ (T)
- $\pi \models p \Rightarrow q \equiv \pi \models p \Rightarrow \pi \models q \equiv$ (T)
- $\pi \models r \equiv r \in L(s_0)$ (F)
- $\pi \models r \Rightarrow p \wedge q \wedge r \equiv$

$\pi^2 \models \top$

$\pi^2 \models \bot$

$\pi^2 \models r \Rightarrow p \wedge q \wedge r$

Exercise: What if we change the LHS to π^2?

$\pi \models r \Rightarrow \pi \models p \wedge q \wedge r$ (F) (T)

Review Q & A - Mar. 3

Written Test 1

LTS: Deadlock Freedom

Q. Formulate **deadlock freedom**:

From any state, it is always possible to make progress.

→ for reactive systems, for each state, it's always possible to make some progress transition

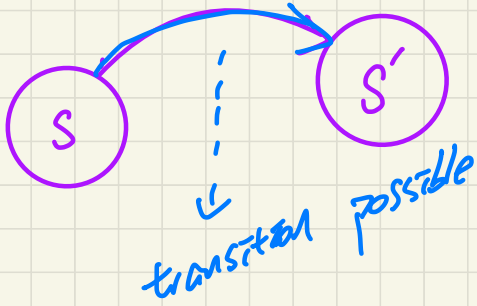
$\forall s \cdot s \in S \Rightarrow (\exists s' \cdot s' \in S \wedge$ ① $(s, s') \in \longrightarrow$ ② $s' \in \longrightarrow[\{s\}])$

② $s' \in \mathrm{ran}(\{s\} \lhd \longrightarrow)$

④ use range restriction

$\mathcal{L}(S_1) =$

$\mathcal{L}(S_2)$

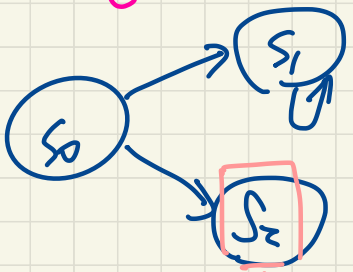


$\longrightarrow = \{ \ldots (s, s'), \ldots \}$

$\longrightarrow = \{ (S_0, S_1), (S_0, S_2), (S_1, S_1) \}$

$s' \in \mathrm{ran}(\{S_0\} \lhd \longrightarrow)$

relation



another relation whose domain is just $\{S_0\}$

$\{(S_0, S_1), (S_0, S_2)\}$

no progress can be made.

$\mathrm{ran}(\{(S_0, S_1), (S_0, S_2)\}) = \{S_1, S_2\}$

Lecture 15 - March 5

Model Checking

Model Satisfaction
Path vs. Model Satisfactions: X,G, F

Announcements/Reminders

- **ProgTest1** results to be released (by end of Friday)
- **WrittenTest1** guide & examples released
- **Review Q&A** materials posted
- **Lab3** to be released after WrittenTest1
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

Model Satisfaction

Given:

- Model $M = (S, \rightarrow, L)$
- State $s \in S$
- LTL Formula ϕ

$M, s \models \phi$ **iff** for **every** path π of M starting at s, $\pi \models \phi$.

Formulation (over all paths)

$s \models \phi \Leftrightarrow \forall \pi \cdot \pi$ starts with $s \wedge s \in S \Rightarrow \pi \models \phi$

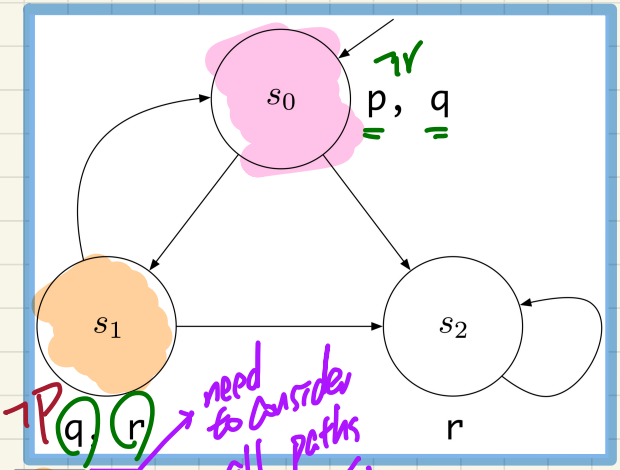
(model sat (where M is clear in the context); path sat.; $\pi = s \rightarrow \cdots$)

How to prove vs. disprove $M, s \models \phi$?

(1) To prove $s \models \phi$, need to show that every path (pattern) π, $\pi \models \phi$

(2) To disprove $s \models \phi$, provide a witness path π, $\neg(\pi \models \phi)$ ($\pi \not\models \phi$)

Model vs. Path Satisfaction: Exercises (1.2)



$s \models p \Leftrightarrow$ all π starting at s, $\pi \models p$

$s_0 \in S$ model sat. (all paths starting s_0)

s_0	$\models$	$\top$
s_0	$\not\models$	$\bot$
s_0	$\models$	$p \wedge q$
s_0	$\models$	$p \vee q$
s_0	$\models$	$p \Rightarrow q$
s_0	$\models$	r
s_0	$\models$	$r \Rightarrow p \wedge q \wedge r$

all such path π has $\pi^1 = s_0 \to \ldots$

only need to consider s_0 to evaluate props. formula.

$\pi = s_0 \to s_1 \to \ldots$

when eval. prop.

$s_1 \models p$ vs. $\pi_1 \models p$

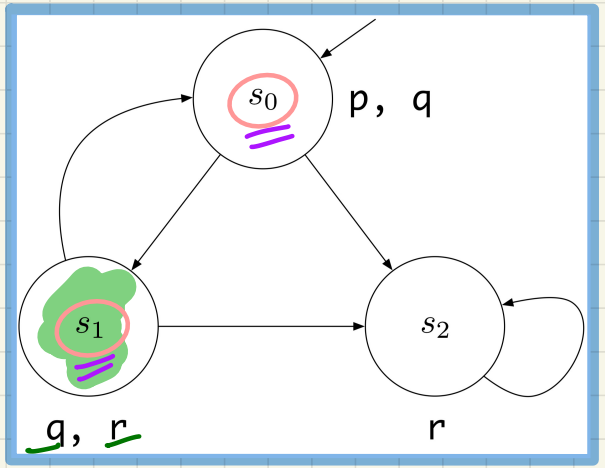
as $\pi^1 : s_0 \models p$

s_1	$\models$	$\top$
s_1	$\not\models$	$\bot$
s_1	$\models$	$p \wedge q$
s_1	$\models$	$p \vee q$
s_1	$\models$	$p \Rightarrow q$
s_1	$\models$	$r \Rightarrow p \wedge q \wedge r$

need to consider all paths starting s_1

Exercise: What if we change the LHS to s_1?

Model vs. Path Satisfaction: Exercises (2.1)



Recall: $\pi \models \mathbf{X}\,\phi \Leftrightarrow \pi^2 \models \phi$

Say: $\pi = s_0 \to s_1 \to s_2 \to s_2 \to \ldots$ (π^2)

path sct.

$\pi \models \mathbf{X}\,\top \Leftrightarrow \pi^2 \models \top$ (T)

$\pi \not\models \mathbf{X}\,\bot$ (T)

$\pi \models \mathbf{X}\,(q \wedge r)$ (T)

$\pi \models \mathbf{X}\,q \wedge r$ (F) $\Leftrightarrow$ $\pi \models \mathbf{X}\,q$ ($\pi^2 \models q$) $\wedge$ $\pi \models r$ (F)

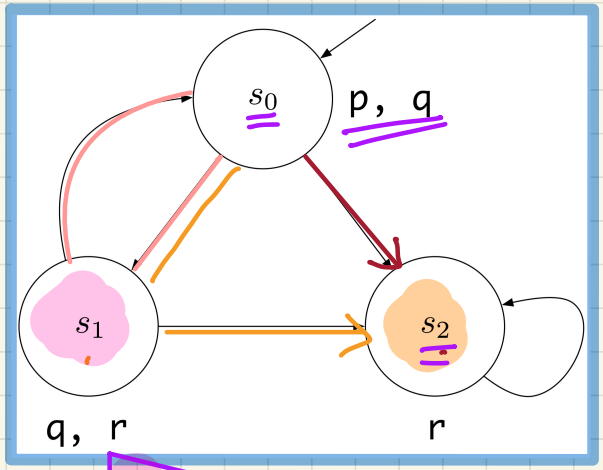
$\pi \models \mathbf{X}\,(q \Rightarrow r)$ (T)

$\pi \models \mathbf{X}\,q \Rightarrow r$ (F) $\Leftrightarrow$ $\pi \models \mathbf{X}\,q$ (T, $\pi^2 \models q$) $\Rightarrow$ $\pi \models r$ (F)

Exercise: What if we change the LHS to π^2?

Exercise

Model vs. Path Satisfaction: Exercises (2.2)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

model sat. (consider all paths starting from s_0)

$s_0 \models X\ \top$ (T) satisfied by s_1 and s_2?

$s_0 \not\models X\ \bot$ (T)

$s_0 \models X\ (q \wedge r)$ (F) witness: $s_0 \to s_2 \to s_2 \to \ldots$ only satisfies r.

$s_0 \models X\ q \wedge r$ (F) (1) witness: $s_0 \to s_2^* \to \ldots$ not sat. q (2) $s_0^* \to s_2 \to$ not sat. r.

$s_0 \models X\ (q \Rightarrow r)$ (T)

$s_0 \models X\ q \Rightarrow r$ (T) F

witness: $s_0 \to s_2^* \to s_2 \to \ldots$ not sat. q

only need to consider s_1 and s_2

(1) $s_0 \to s_1 \to s_0 \to s_1 \to \ldots$

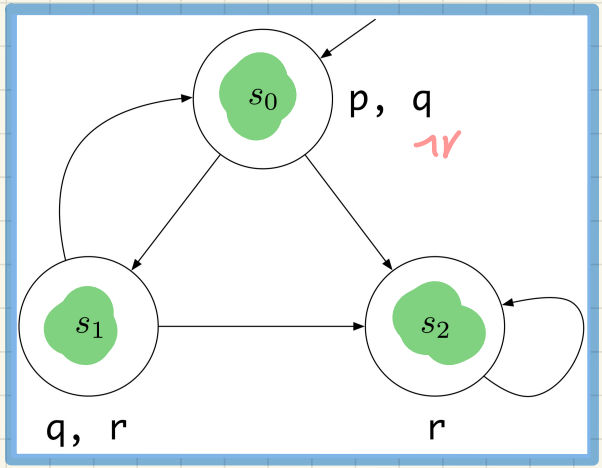
(2) $s_0 \to s_2 \to s_2 \to \ldots$

(3) $s_0 \to s_1 \to s_0 \to \ldots \to s_1 \to s_2 \to s_2 \to \ldots$

next states of paths starting from s_0

Exercise: What if we change the LHS to s_1?

Model vs. Path Satisfaction: Exercises (3.1)



$\pi \models \mathbf{G}\,\phi \Leftrightarrow \forall\, i \bullet i \geq 1 \Rightarrow \pi^i \models \phi$

Say: $\pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$

↳ reachable = $\{s_0, s_1, s_2\}$

$\pi \models \mathbf{G}\,\top$ (T)

$\pi \not\models \mathbf{G}\,\bot$ (T)

$\pi \models \mathbf{G}\,\neg(p \wedge r)$ (T) — $\neg p \vee \neg r$; e.g. in s_0, $\neg r$ is true.

$\pi \models \mathbf{G}\,r$ (F) → witness: $s_0, \neg r$

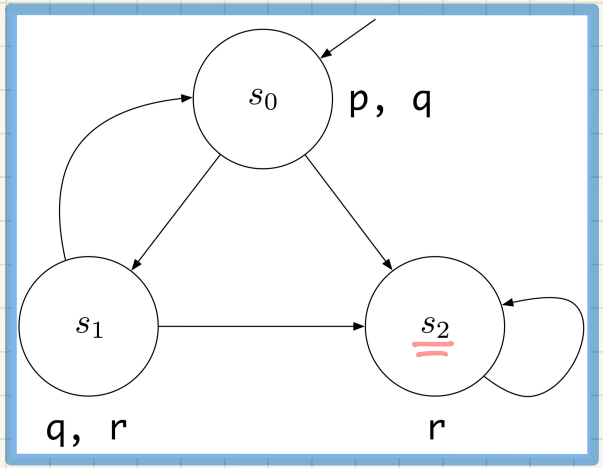
$\pi^2 \models \mathbf{G}\,r$

reachable = $\{s_1, s_2\}$

check these on reachable states of paths.

Exercise: What if we change the LHS to π^2?

Model vs. Path Satisfaction: Exercises (3.2)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

model sat. (all paths starting from s_0)

$s_0 \models G\ \top$ (T)

$s_0 \not\models G\ \bot$ (T)

$s_0 \models G\ \neg(p \wedge r)$ (T) $\neg p \vee \neg r$ ∴ $\neg p \vee \neg r$ sat. in all states.

$s_0 \models G\ r$ (F) witness: $s_0 \to s_1 \to s_0 \to$ ↓ not sat.

$s_2 \models G\ r$ (T)

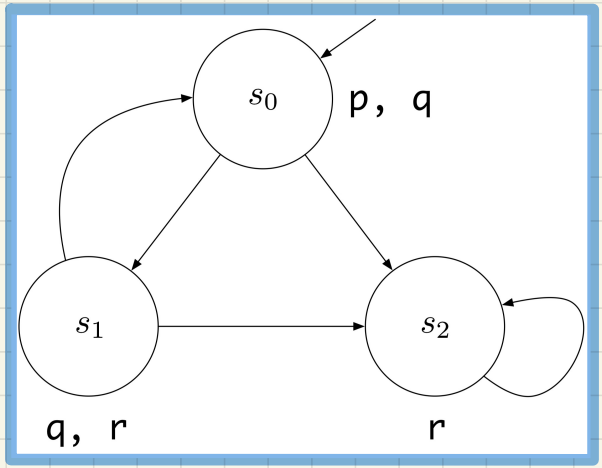
s_2 is the only reachable state for all paths starting at s_2

Reachable states from paths starting at s_0 = $\{s_0, s_1, s_2\}$

$s_1 \models G\ \top$ (T) witness: $\neg\top$ $s_1 \to s_0 \to s_1$ reachable = $\{s_0, s_1, s_2\}$

Exercise: What if we change the LHS to s_1?

Model vs. Path Satisfaction: Exercises (4.1)



$\pi \models \mathbf{F}\,\phi \Leftrightarrow \exists\, i \bullet i \geq 1 \wedge \pi^i \models \phi$

Say: $\pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$

↳ reaches $\{s_0, s_1, s_2\}$

$\pi \models \mathbf{F}$	$\top$	T	
$\pi \not\models \mathbf{F}$	$\bot$	T	
$\pi \models \mathbf{F}$	$\neg(p \wedge r)$ ($\neg p \vee \neg r$)	T	witness state: s_0
$\pi \models \mathbf{F}$	r	T	witness: s_2
$\pi \models \mathbf{F}$	$(q \wedge r)$	T	witness: s_1

Exercise: What if we change the LHS to π^2?

Lecture 16 - March 10

Model Checking

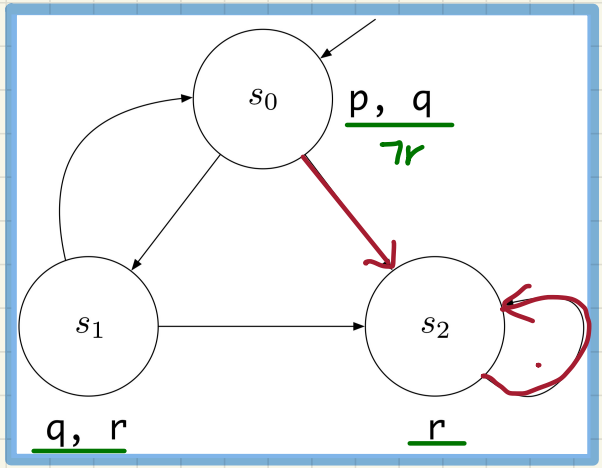
Nesting Temporal Operators: FGϕ

Exercise: Gϕ vs. FGϕ

Announcements/Reminders

- **ProgTest1** results & feedback released
  + Submit a regrading request if necessary.
- **WT1** results to be released by the end of Friday
- **Lab3** released
- Guide for **ProgTest2** to be released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

Model vs. Path Satisfaction: Exercises (4.2)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

$s_0 \in S$

$s_0 \models \mathbf{F}\ \top$ (T)

$s_0 \not\models \mathbf{F}\ \bot$ (T)

$s_0 \models \mathbf{F}\ \neg(p \wedge r)$ (T) → all paths starting from s_0 have a future state satisfying $\neg p \vee \neg r$

this is the case: all states satisfy $\neg p \vee \neg r$

$s_0 \models \mathbf{F}\ r$ (T) → all paths starting from s_0 have a future state satisfying r.

$s_0 \models \mathbf{F}\ (q \wedge r)$ (F)

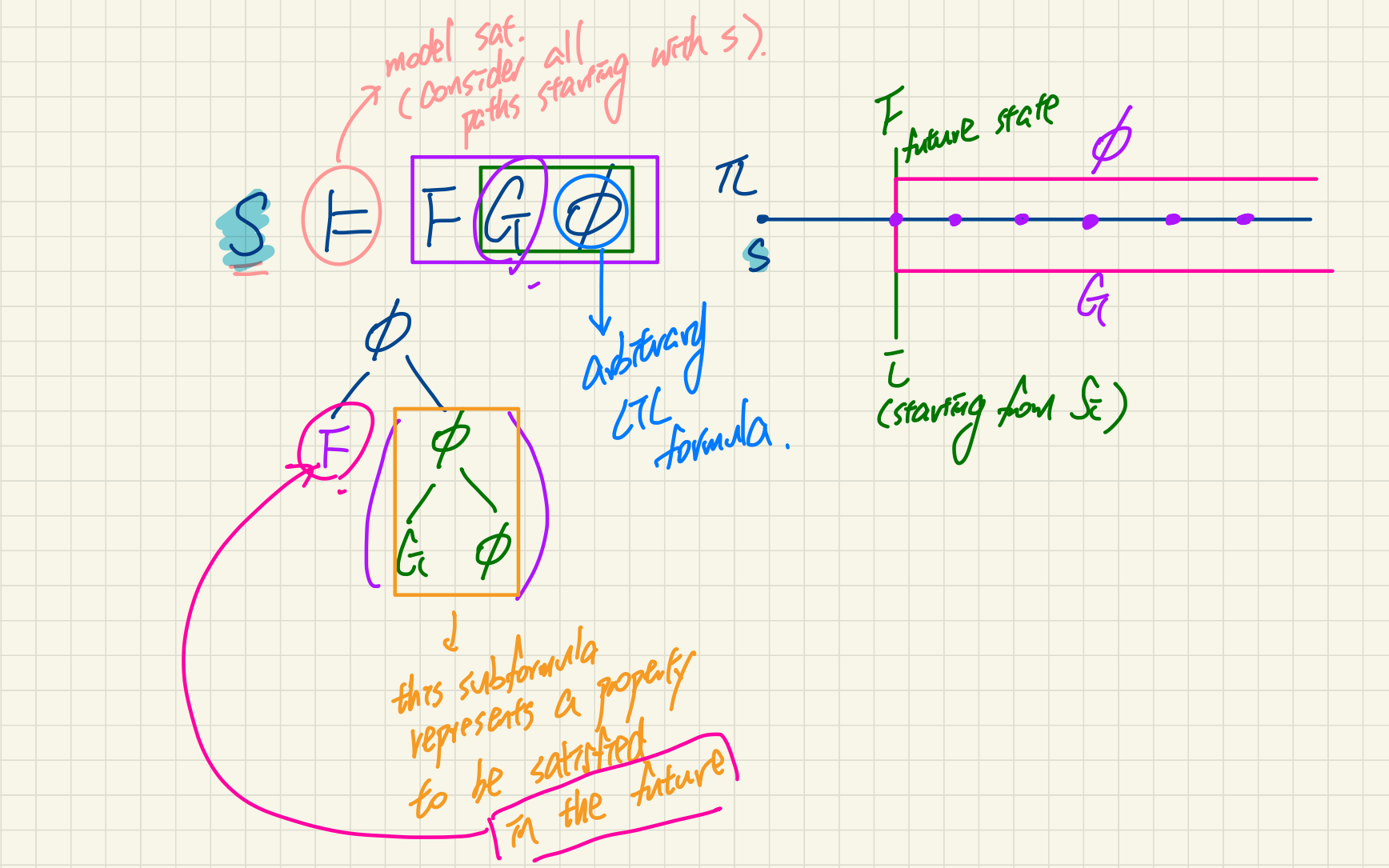
Witness path: $s_0 \to s_2 \to s_2$ ($\neg r$, $\neg q$)

in all such paths s_1 or s_2 is reachable, and both of them sat. r.

$s_0 \models \mathbf{F}(q \wedge p)$ (T)

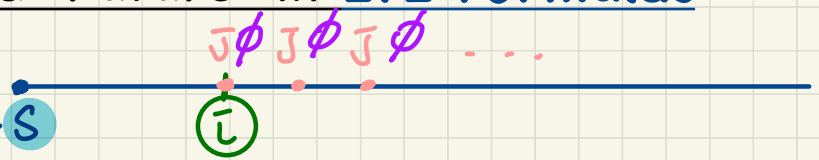
↳ all paths starting with a sat. $q \wedge p$ right away.

Exercise: What if we change the LHS to s_1?


model sat.
(consider all paths starting with s).
s
⊨
F G ϕ
arbitrary LTL formula.
π
s
F
future state
ϕ
G
i
(starting from s_i)
ϕ
F
ϕ
G ϕ
this subformula represents a property to be satisfied in the future

Nesting "Global" and "Future" in LTL Formulas

$s \models FG\,\phi$ ($\models$ (model sat.))



Each path starting with s is s.t. eventually, ϕ holds continuously.

∀ ∃ ∀ (F ✓, G)

Q. Formulate the above nested pattern of LTL operator.

$s \models FG\phi \overset{*}{\Leftrightarrow} \forall \pi \cdot \pi = s \to \cdots \Rightarrow$

$\left(\exists^{**}_{i} \cdot i \geq 1 \wedge \left(\forall^{***}_{j} \cdot \boxed{j \geq i} \Rightarrow (\pi^{j} \models \phi) \right) \right)$

(model sat. ; path sat.)

Q. How to **prove** the above nested pattern of LTL operators?

*① Consider all paths starting at s

**② Find a witness i ***③ each state at index i, i+1, i+2, . . . satisfies ϕ

Q. How to **disprove** the above nested pattern of LTL operators?

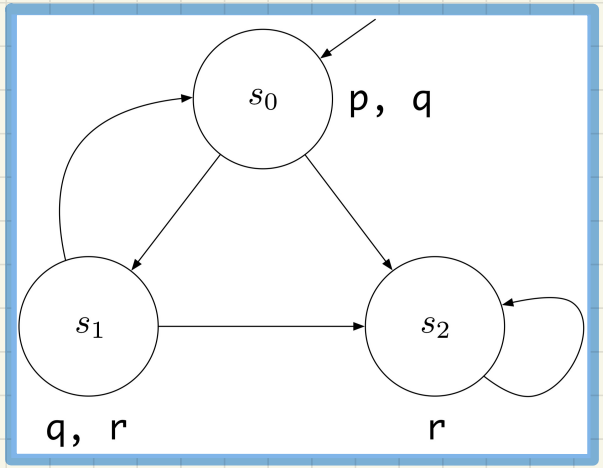
disproving ∃ / disproving ∀

*① Find a witness $\pi = s \to \cdots$ disproving ③ (G)

**② For all possible future states at indices $i=1, 2, 3, \ldots$

With respect to each i, there is a j, s.t. $j \geq i$ and $\pi^{j} \not\models \phi$

Model Satisfaction: Exercises (5.1)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

$s_0 \models FG\ r$ (F)

Witness: there's no such a future point starting from which G r is true

$s_0 \models FG\ (p \vee q)$ (T)

$s_0 \to s_1 \to s_0 \to s_2 \to \ldots$ can never be avoided but r is not sat. in s_0

Witness $s_0 \to s_2 \to s_2 \to \cdots$

↳ not sat. $p \vee q$ (there's no such future point after which $p \vee q$ is sat.)

$s_0 \models FG\ (p \vee r)$ (T)

$p \vee r$ is sat. by all states, so for all paths starting from s_0, choose s_0 as the future point.

Exercise: What if we change the LHS to s_2?

Q. $F_1: S \models G\phi$

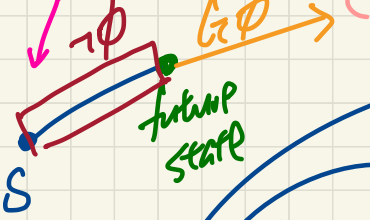
$F_2: S \models FG\phi$

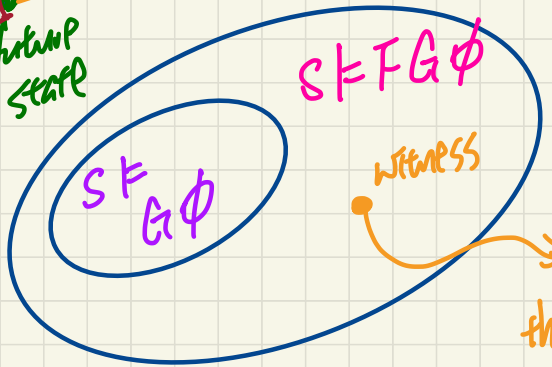
(1) $F_1 \Rightarrow F_2$

(2) $F_2 \Rightarrow F_1$

(3) $F_1 \Leftrightarrow F_2$

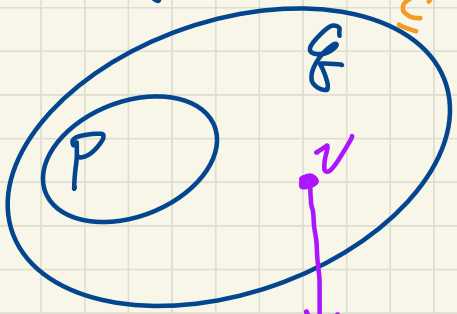
$S \models G\phi \Rightarrow S \models FG\phi$ (not vice versa)





a path that satisfies $FG\phi$ but not $G\phi$

$p \Rightarrow q$ ✓ → set of satisfying values of p is a subset ($\subseteq$) of that of q.



If p cannot be equivalent to q, we should be able to find a value v that ① sat. q ② not sat. p

Lecture 17 - March 17

Model Checking

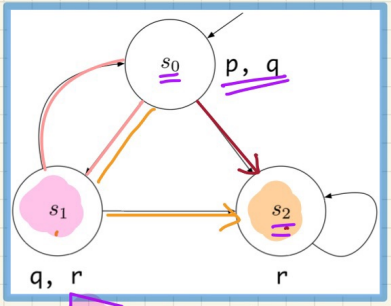
Parsing Property
Exercise: Fϕ ⇒ FGϕ
Nesting Temporal Operators: GFϕ

Announcements/Reminders

- **ProgTest2** guide & example questions released
- **WrittenTest2** potential shift of date?
- **ProgTest1** results & feedback released
  + Submit a regrading request if necessary.
- **WT1** results & feedback released
- **Lab3** due today
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

Correction: Exercise 2.2 from March 5

Model vs. Path Satisfaction: Exercises (2.2)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

model sat. (consider all paths starting from s_0)

$s_0 \models X\ \top$ ⊤ satisfied by s_1 and s_2?

$s_0 \not\models X\ \bot$ ⊤

$s_0 \models X\ (q \wedge r)$ F witness: $s_0 \to s_2 \to s_2 \to \ldots$ only satisfies r.

$s_0 \models X\ q \wedge r$ F witness: $s_0 \to s_2 \to \ldots$ not sat. q; $s_0 \to s_2 \to$ not sat. r.

$s_0 \models X\ (q \Rightarrow r)$ ⊤

$s_0 \models X\ q \Rightarrow r$ ⊤ only need to consider s_1 and s_2

F witness: $s_0 \to s_2 \to s_2 \to \ldots$ not sat. q

(1) $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$

(2) $s_0 \to s_2 \to s_2 \to \cdots$

(3) $s_0 \to s_1 \to s_0 \to \cdots \to s_1 \to s_2 \to s_2$

next states of paths starting from s_0

Exercise: What if we change the LHS to s_1?

false.

what about $s_0 \to s_1 \to \ldots$?

s_0: r?

s_1: q?

YES

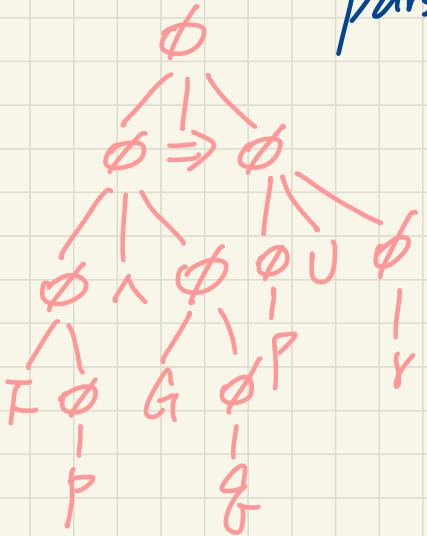
$T \Rightarrow F$

F!

Given two LTL formula strings f_1 and f_2

$Fp \wedge Gq \Rightarrow p \,U\, r$ $(Fp) \wedge (Gq) \Rightarrow (p \,U\, r)$

(1) If $f_1 \neq f_2$ (different spellings), but f_1 and f_2 have the same parse tree, f_1 and f_2 are considered as semantically equivalent.

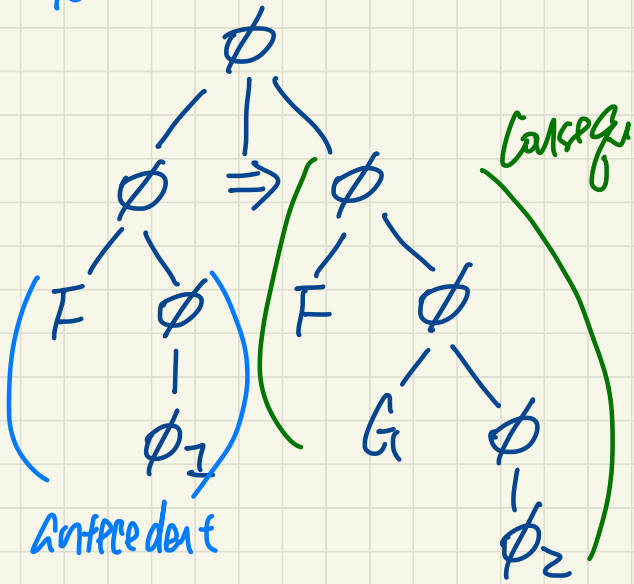


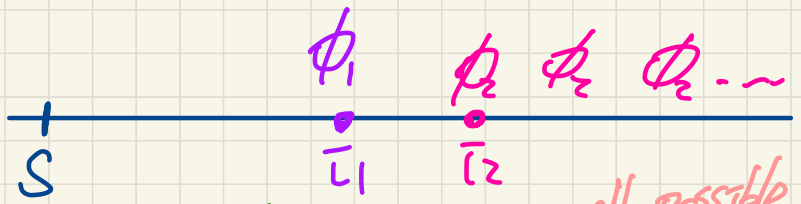
(2) If $f_1 = f_2$, but f_1 and f_2 have distinct PTs, this means the grammar is ambiguous. (e.g., "dangling else").

$S \models F\phi_1 \Rightarrow FG\phi_2$

Consider all paths $\pi = s \to \cdots$

If at some future point $\bar{i}_1$, ϕ_1 is true,
then at some future point $\bar{i}_2$ $G\phi_2$ is true.





all possible.

① $\bar{i}_1 < \bar{i}_2$
② $\bar{i}_1 > \bar{i}_2$
③ $\bar{i}_1 = \bar{i}_2$

Nesting "Global" and "Future" in LTL Formulas

$s \models F\phi_1 \Rightarrow FG\phi_2$

Each path π starting with s is s.t. if eventually $\phi 1$ holds on π, then $\phi 2$ eventually holds on π continuously.

Q. **Formulate** the above nested pattern of LTL operators.

$$\forall \pi \cdot \pi = s \rightarrow \cdots \Rightarrow \Big((\exists i_1 \cdot i_1 \geq 1 \wedge \pi^{i_1} \models \phi_1) \Rightarrow (\exists i_2 \cdot i_2 \geq 1 \wedge (\forall j \cdot j \geq i_2 \Rightarrow \pi^j \models \phi_2)) \Big)$$

c. $\_ \Rightarrow T$

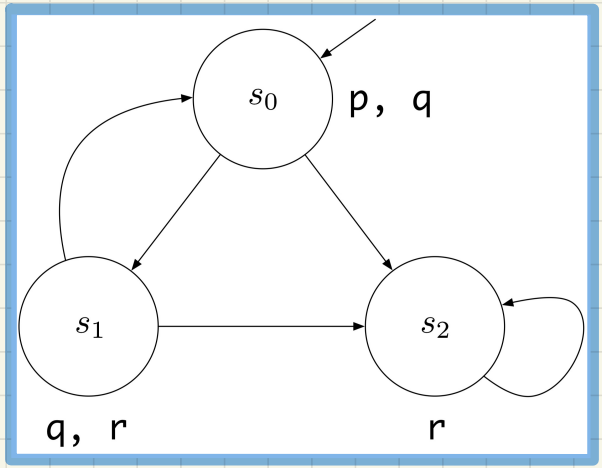
Q. How to **prove** the above nested pattern of LTL operators?

① Consider all path patterns starting with s ② for each path a. $T \Rightarrow T$ b. $F \Rightarrow \_$

Q. How to **disprove** the above nested pattern of LTL operators?

① Show a witness path π ② for π, show: $T \Rightarrow F$

Model Satisfaction: Exercises (5.2)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

$s_0 \models F(\neg q \wedge r) \Rightarrow FG\ r$ (T)

① $s_0 \to s_1 \to s_0 \to s_2 \to \cdots$
↳ reachable states: s_0 and s_1 (¬q)
↳ $F(\neg q \wedge r)$ is false
↳ false $\Rightarrow$ _ ≡ True

$s_0 \models F(\neg q \vee r) \Rightarrow FG\ r$ (F.)

② $s_0 \to s_2 \to s_2 \to \cdots$
↳ s_2 satisfies r, and the remaining path all reach s_2
↳ FG r is True
↳ _ ⇒ True ≡ True.

Witness: $s_0 \to s_1 \to s_0 \to s_2 \to \cdots$
$F(\neg q \vee r)$: established by s_1
FG r: always violated by s_0
↳ T ⇒ F ≡ F.

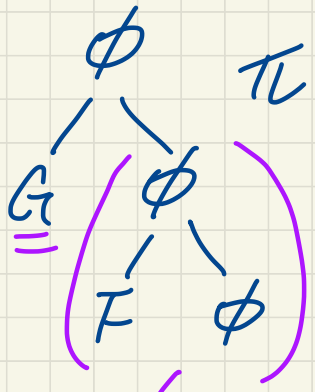
③ $s_0 \to s_1 \to s_0 \to \cdots \to s_2 \to s_2$

④ $s_0 \to s_2 \to \cdots \to s_0 \to s_2$

Exercise: What if we change the LHS to s_2?

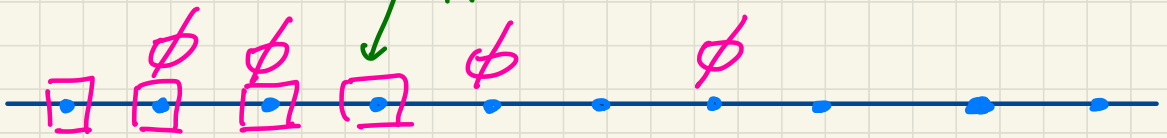
Exercise: compare $G\phi$ vs. $GF\phi$

$S \models GF\phi$



this property is globally true

↳ for each state, it's always possible to find a future point that satisfies ϕ.



↳ $GF\phi$ is false if there is a state s_i from which no future point can be found that satisfies ϕ

$G\neg\phi$

Nesting "Global" and "Future" in LTL Formulas

$s \overset{*}{\models} \mathbf{GF}\ \phi$ → infinitely often, ϕ is true.

Each path starting with s is s.t. continuously, ϕ eventually holds.

Q. Formulate the above nested pattern of LTL operator.

$* \ \forall \pi \cdot \pi = s \rightarrow \cdots \Rightarrow$

$\left(\forall i \cdot i \geq 1 \Rightarrow \left(\exists j \cdot j \geq i \wedge \pi^j \models \phi \right) \right)$

Q. How to prove the above nested pattern of LTL operators?

Q. How to disprove the above nested pattern of LTL operators?

Exercice 1

(1) $GF\phi \Rightarrow G\phi$

(2) $G\phi \Rightarrow GF\phi$

(3) $GF\phi \Leftrightarrow G\phi$

Exercise 2

(1) $FG\phi \Rightarrow GF\phi$

(2) $GF\phi \Rightarrow FG\phi$

(3) $FG\phi \Leftrightarrow GF\phi$

Lecture 18 - March 19

Model Checking

Nested Temporal Operator: GFϕ

Exercise: Gϕ vs. GFϕ

Exercise: FGϕ vs. GFϕ

Announcements/Reminders

- **ProgTest2** focuses on Lab2 (no Lab3).
- **WrittenTest2** date remains unchanged.
- **ProgTest1** results & feedback released
  + Submit a regrading request if necessary.
- **WT1** results & feedback released
- **Lab3** solution released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

Nesting "Global" and "Future" in LTL Formulas

$s \models \mathbf{GF}\ \phi$ → infinitely often, ϕ is true.

Each path starting with s is s.t. continuously, ϕ eventually holds.

Q. Formulate the above nested pattern of LTL operator.

$$* \ \forall \pi \cdot \pi = s \rightarrow \cdots \Rightarrow$$
$$(** \forall i \cdot i \geq 1 \Rightarrow (*** \exists j \cdot j \geq i \wedge \pi^j \models \phi))$$

Q. How to **prove** the above nested pattern of LTL operators?

* (1) consider all path patterns π

** (2) consider all states t at index i in π

*** (3) find a witness state at index j, such that $\pi^j \models \phi$

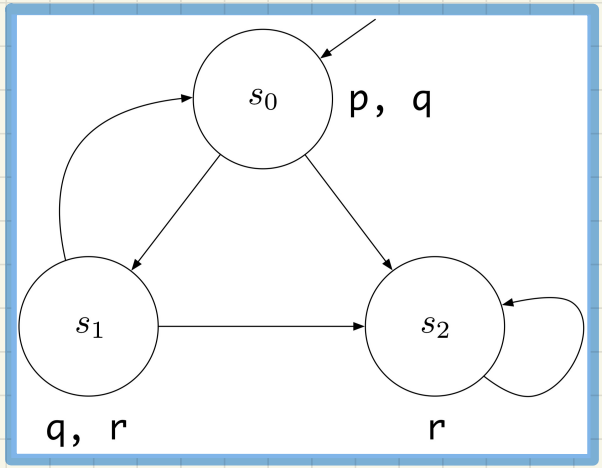
Q. How to **disprove** the above nested pattern of LTL operators?

* (1) Give a witness path π.

** (2) Give a witness state at index i in π it is: $\pi^j \not\models \phi$

*** (3) all possible j, s.t. $j \geq i$

Model Satisfaction: Exercises (6.1)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

$s_0 \models \mathbf{GF}\ p$ False (2) $s_2 \models G \neg p$

Witness path: $s_0 \to s_2 \to s_2$

(1) starting from s_2, there's no future point that satisfies p

$s_0 \models \mathbf{GF}\ (p \vee q)$ False

Witness path: $s_0 \to \boxed{s_2 \to s_2 \to \cdots}$ $\neg(p \vee q)$

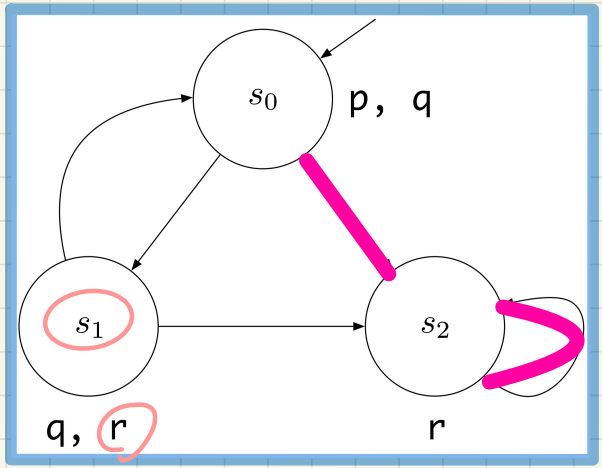
$s_0 \models GF(p \vee r)$ True

↳ in fact, $s_0 \models G(p \vee r)$

Property: $G\phi \Rightarrow GF\phi$ (not vice versa)

Exercise: What if we change the LHS to s_2?

Model Satisfaction: Exercises (6.2)



may try
(1) F ⇒ —
(2) — ⇒ T

$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

$s_0 \models \mathbf{GF}\ p \Rightarrow \mathbf{GF}\ r$

only need to consider: $s_0 \to s_1 \to s_2 \to s_1 \to$

(1) Assume the antecedent: GF p

(2) In this path $s_0 \to s_1 \to s_2 \to s_1 \to \cdots$ consequent is true.

r

$s0 \models \mathbf{GF}\ r \Rightarrow \mathbf{GF}\ p$ F

Witness path: $s_0 \to s_2 \to s_2 \to s_2 \to$

GF r is true but GF p is false

T ⇒ F ≡ F

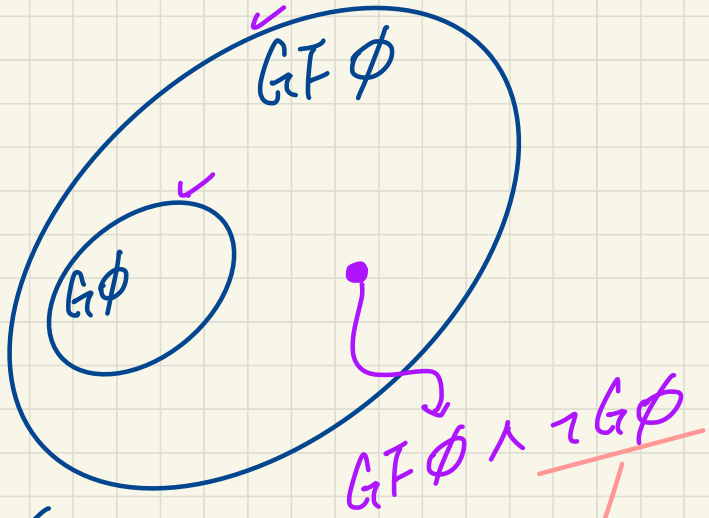
Exercise: What if we change the LHS to s_2?

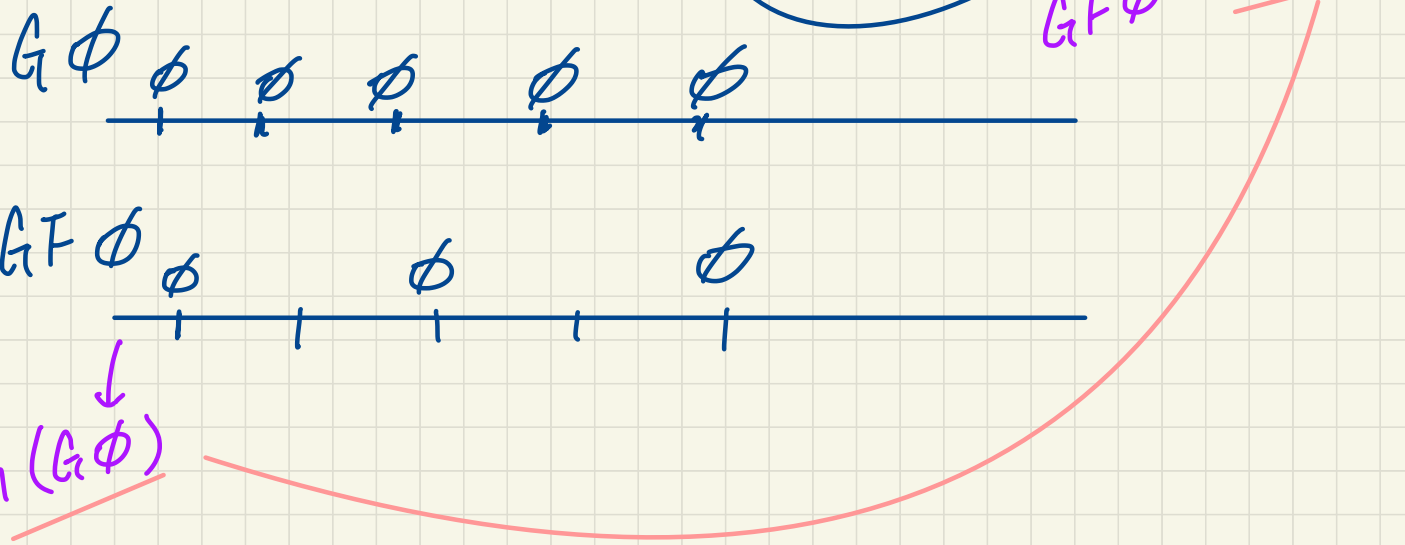
Exercice 1

(1) $GF\phi \Rightarrow G\phi$

(2) $G\phi \Rightarrow GF\phi$

(3) $GF\phi \Leftrightarrow G\phi$



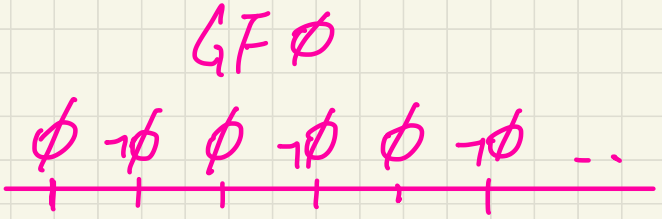


Exercise 2

(1) $FG\phi \Rightarrow GF\phi$

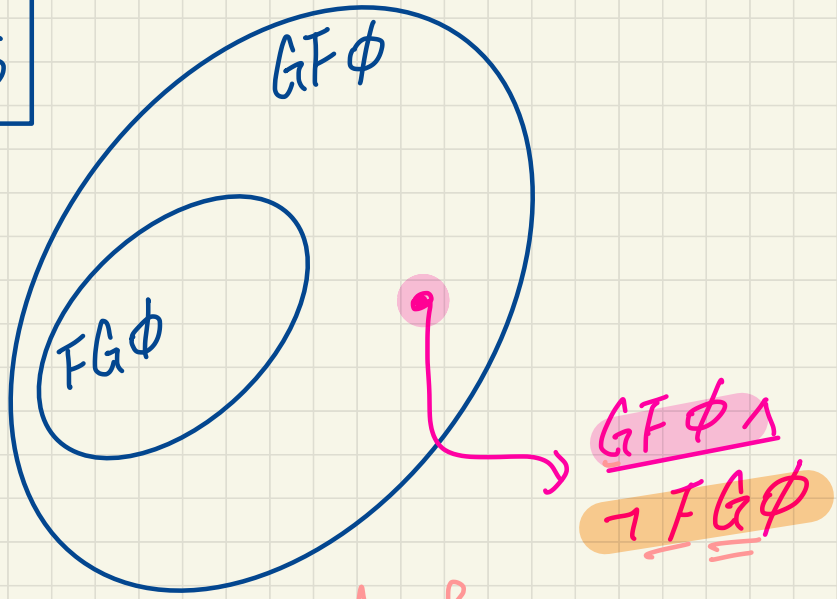
(2) $GF\phi \Rightarrow FG\phi$

(3) $FG\phi \Leftrightarrow GF\phi$



ϕ is satisfied in every other state

↳ $GF\phi$ ✓



$FG\phi$ ✗

↳ there's no such a future point.

Lecture 19 - March 24

Model Checking

Temporal Operators: U, W, R
Formulating English Sentences in LTL

Announcements/Reminders

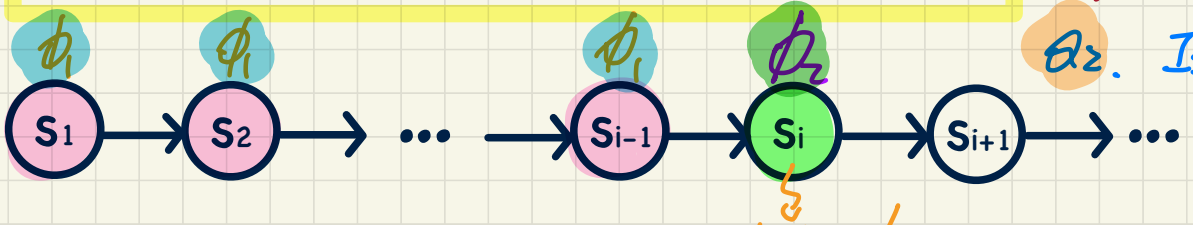
Wed
↳ 20 min
↳ tips.

- **WrittenTest2** this Thursday
- **Lab4** to be released
- Office Hour this week: 3pm on Wed, Thu
- TA contact information (on-demand for labs) on eClass

Path Satisfaction: Temporal Operations (4)

$\pi \models \phi_1 \, U \, \phi_2$ (U: until)

There is some future state satisfies ϕ_2, and until then, all states satisfy ϕ_1 .



whether ϕ_1 is true here not, does not matter.

Q1. Is it ok that $G\,\phi_1$ but $G\neg\phi_2$?

$G\neg\phi_2 = \neg F\phi_2$

No. → $\phi_1 \, U \, \phi_2$ → false

Q2. Is it ok that $G\,\phi_1$ and $F\,\phi_2$?

True → $\phi_1 \, U \, \phi_2$ → true

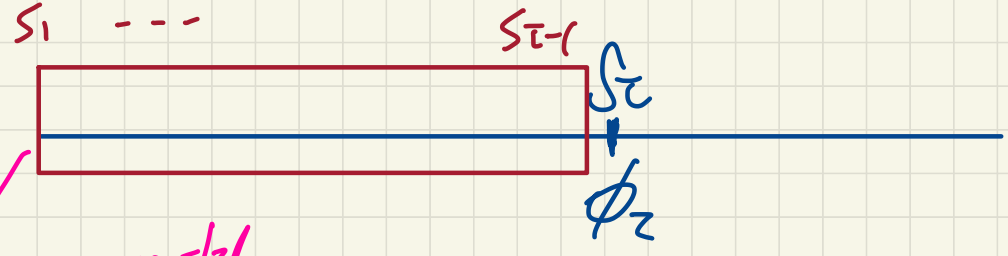
$\phi_1, \phi_1, \phi_1, \phi_1 \ldots, \phi_2$ → $F\phi_2$

Formulation (over a path)

$$\pi \models \phi_1 \, U \, \phi_2 \iff \left(\exists \bar{i} \cdot \bar{i} \geq 1 \wedge \left(\pi^{\bar{i}} \models \phi_2 \right) \wedge \left(\forall j \cdot 1 \leq j \leq \bar{i}-1 \Rightarrow \pi^{j} \models \phi_1 \right) \right)$$

ϕ_1, ϕ_2: LTL formula

Given a path : $F\phi_1 \wedge F\phi_2$



not necessarily
$G\phi_1$ from s_1 to s_{i-1},
given only $F\phi_1$.

$\pi \vDash \phi_1 \, \mathcal{U} \, \phi_2 \iff \left(\exists i \cdot i \geq 1 \wedge \left(\pi^i \vDash \phi_2 \wedge \left(\forall j \cdot 1 \leq j \leq i-1 \Rightarrow \pi^j \vDash \phi_1 \right) \right) \right)$

ϕ_1 < TK < formula

$\exists i \cdot i \geq 1 \wedge \left(\pi^{\textcircled{1}\,i} \vDash \phi_2 \wedge \left(\forall j \cdot 1 \leq j \leq i-1 \Rightarrow \pi^j \vDash \phi_1 \right) \right)$

$1 \leq j \leq 0$

$\forall j \cdot \text{false} \Rightarrow \pi^j \vDash \phi_1$

true

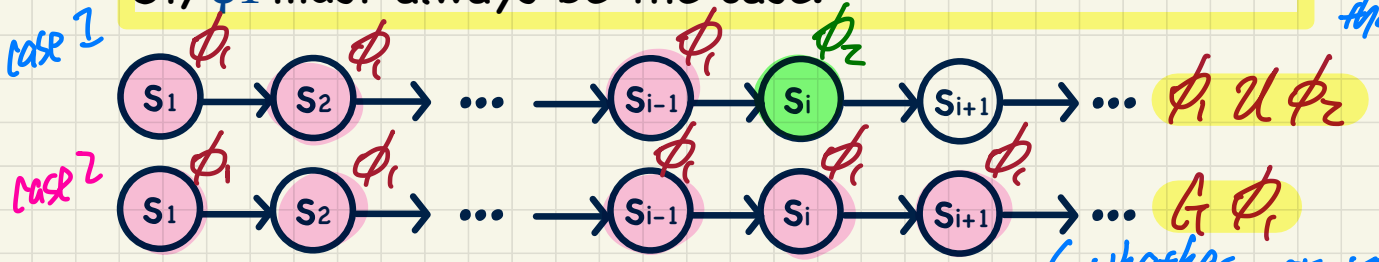
Path Satisfaction: Temporal Operations (5)

weak until

$\pi \models \phi_1 \, W \, \phi_2$

If there is ever a future state that satisfies ϕ_2, then until then, all states satisfy ϕ_1.

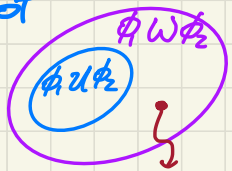
Or, ϕ_1 must always be the case.



Q1. Is it ok that $G \, \phi_1$ but $G \neg \phi_2$?
ok → $\phi_1 \, W \, \phi_2$

Q2. Is it ok that $G \, \phi_1$ and $F \, \phi_2$? (TMP.)
↳ ok.

(whether or not $F \, \phi_2$ does not matter)



Formulation (over a path)

$\phi_1 \, W \, \phi_2 \iff \phi_1 \, U \, \phi_2$

case 1

case 2 $G \, \phi_1$

$(\forall k \cdot k \geq 1 \Rightarrow \pi^k \models \phi_1)$

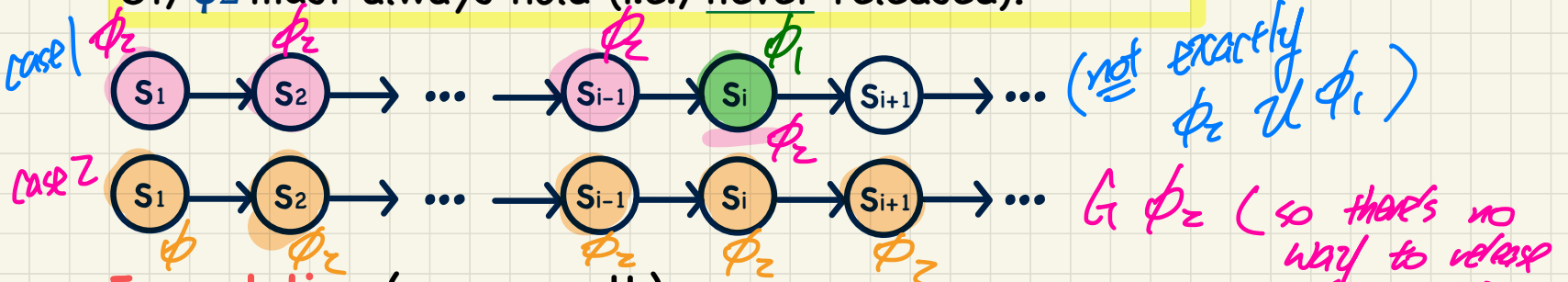
Path Satisfaction: Temporal Operations (6)

release

$\pi \models \phi 1 \; R \; \phi 2$

If there is ever a future state that satisfies $\phi 1$, then until then, all states satisfy $\phi 2$.

Or, $\phi 2$ must always hold (i.e., never released).



Formulation (over a path)

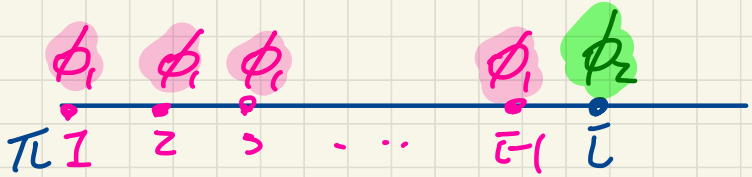
$$\pi \models \phi_1 \; R \; \phi_2 \iff \left(\exists i \cdot i \geq 1 \wedge \left(\pi^i \models \phi_1 \wedge \left(\forall j \cdot j \leq i \cdot \pi^j \models \phi_2\right)\right)\right) \vee \left(\forall k \cdot k \geq 1 \Rightarrow \pi^k \models \phi_2\right)$$

case 1 / case 2 / $G \; \phi_2$

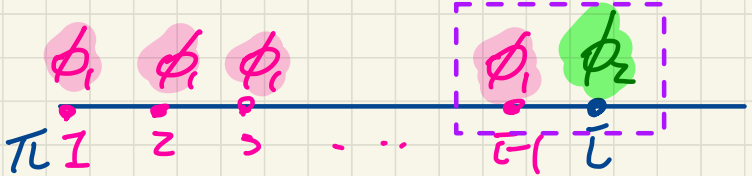
contrast: $\pi \models \phi_2 \; \mathcal{U} \; \phi_1$

U, W, R

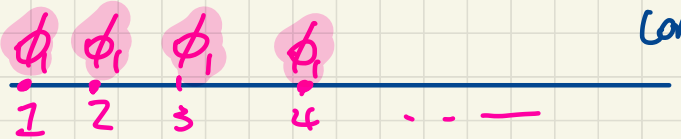
Until $\phi_1 \, U \, \phi_2$



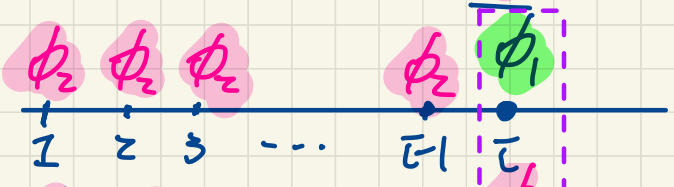
Weak Until $\phi_1 \, W \, \phi_2$



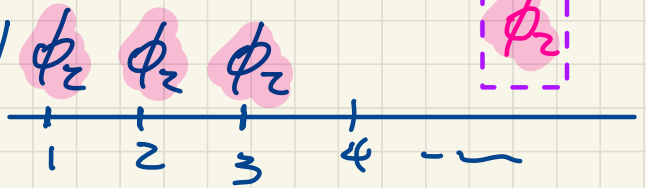
(or) V



Release $\phi_1 \, R \, \phi_2$



(or) V



Formulating Natural Language in LTL (1)

Natural Language:

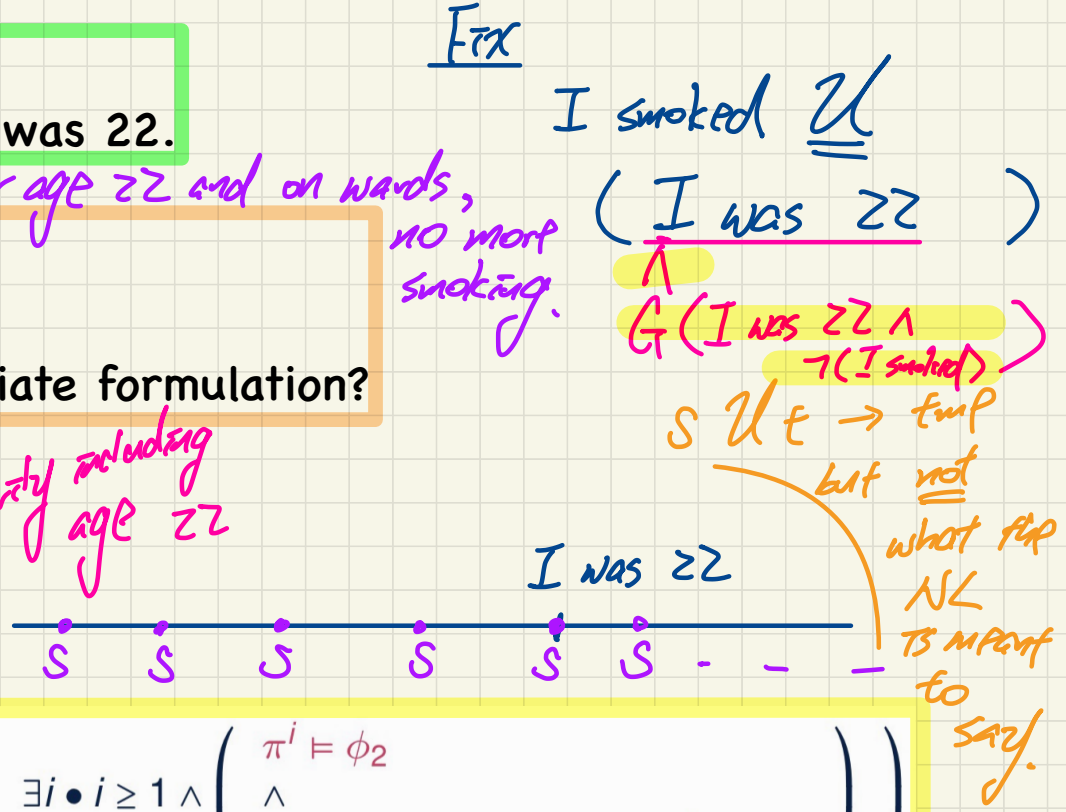
I had smoked until I was 22.

Atom t: I was 22

Atom s: I smoke

Q. Is s U t an appropriate formulation?

$$\pi \models \phi_1 \mathbf{U} \phi_2 \iff \left(\exists i \bullet i \geq 1 \wedge \left(\begin{array}{l} \pi^i \models \phi_2 \\ \wedge \\ (\forall j \bullet 1 \leq j \leq i-1 \Rightarrow \pi^j \models \phi_1) \end{array} \right) \right)$$



Formulating Natural Language in LTL (2.1)

Natural Language:

It's impossible to reach a state where the system is started but not ready.

$G\,\phi \equiv \neg F \neg \phi$

$F\,\phi \equiv \neg G \neg \phi$

Assumed atoms:

- started
- ready

$\neg (F (started \wedge \neg ready))$

LTL Formulation

$\neg (F (started \wedge \neg ready))$

exercise?

$\equiv G (\neg (started \wedge \neg ready))$

$\equiv G (\neg started \vee ready)$ $started \Rightarrow ready$

Formulating Natural Language in LTL (2.2)

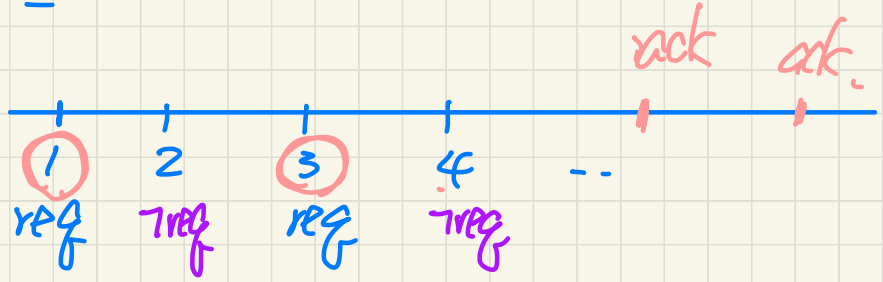
Natural Language:

Whenever a request is made, (G)
it will be acknowledged eventually. (F)

Assumed atoms:

- requested
- acknowledged

$$\underline{G}\ (\text{requested} \Rightarrow F\,\text{ack.})$$



LTL Formulation

exercise.

Formulating Natural Language in LTL (2.3)

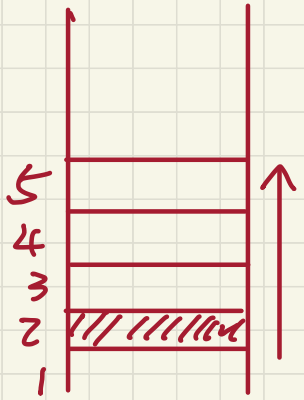
Natural Language:

An elevator traveling upwards at the 2nd floor
does not change its direction
when it has passengers wishing to to to the 5th floor.

Assumed atoms:

- floor2, floor5
- directionUp
- buttonPressed5

LTL Formulation



$$G\left(floor2 \wedge buttonPressed5 \Rightarrow X(G\ directionUp)\right)$$

$$(directionUp\ \mathcal{U}\ floor5)$$

↓ W ?

Lecture 20 - March 26

Model Checking

Exercises: U, W, R
Stronger vs. Weaker Assertions

Announcements/Reminders

- **WrittenTest2** this Thursday
- **Lab4** to be released
- Office Hour this week: 3pm on Wed, Thu
- TA contact information (on-demand for labs) on eClass

Formulating Natural Language in LTL (1)

Natural Language:

I had smoked until I was 22.

Atom t: I was 22

Atom s: I smoke

Q. Is s U t an appropriate formulation?

Ex

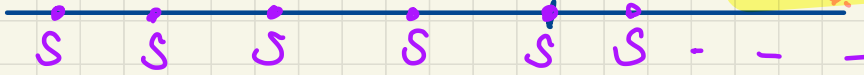
I smoked $\mathcal{U}$ (I was ≥2 ∧ $G\neg$ I smoke)

was ≥ ↳ for age ≥2 and onwards, no more smoking.

this global property starts to hold at the future point where I was ≥2

only insists smoking up to and not necessarily including age 22

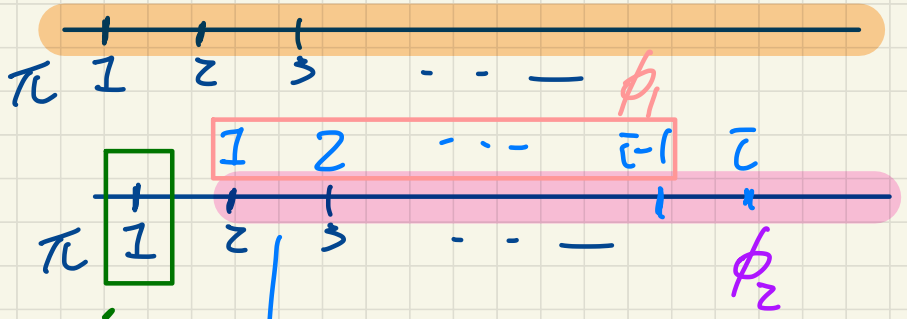
$s \mathcal{U} t \rightarrow$ fine but not what the NL is meant to say.



$$\pi \models \phi_1 \mathbf{U} \phi_2 \iff \left(\exists i \bullet i \geq 1 \wedge \left(\begin{array}{l} \pi^i \models \phi_2 \\ \wedge \\ (\forall j \bullet 1 \leq j \leq i-1 \Rightarrow \pi^j \models \phi_1) \end{array} \right) \right)$$

$\pi \vDash \phi_1 \mathcal{U} \phi_2$

$\pi^2 \vDash \phi_1 \mathcal{U} \phi_2$



not relevant for π^2

π^2

Formulating Natural Language in LTL (2.4)

Natural Language:

Whenever a process makes a request, it starts waiting.
As soon as no other process is in the critical region,
the process is granted access to the critical region.

Assumed atoms:

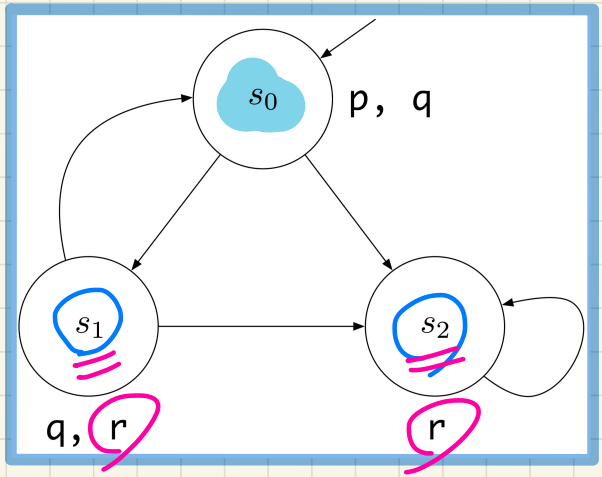
- requested
- waiting
- ~~granted~~
- noOneInCs

LTL Formulation

$$G(\text{requested} \Rightarrow (\text{noOneInCS}\ R\ \text{waiting}))$$

Q. Is starvation freedom guaranteed?

Model Satisfaction: Exercises (7.1)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

$s_0 \models p \; U \; r$ — True

$s_0 \models p \; W \; r$ — True

↳ ∵ $p \; U \; r$ is true.

$s_0 \models r \; R \; p$ — false.

Witness $s_0 \to s_1 \to s_0 \to s_1 \to \ldots$

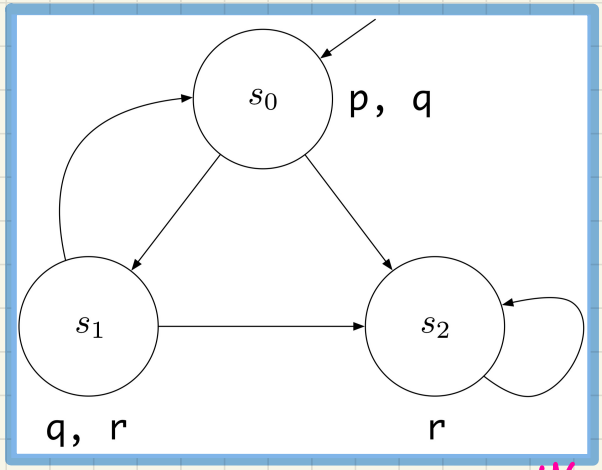
Cond. 2) s_1 is not sat. ∵ s_1 does not sat. p.

Cond. 1) s_1 is not sat. ∵ there's no such future point where $r \wedge p$ is sat.

$\pi \models r \; R \; p \Leftrightarrow$ (1) $(\exists i \cdot i \geq 1 \wedge (\pi^i \models r \wedge \forall j \cdot j < i \Rightarrow \pi^j \models p))$ ∨ (2) $G \, p$

Exercise: What if we change the LHS to s_2?

Model Satisfaction: Exercises (7.2)



$s \models \phi \Leftrightarrow$ all π starting at s, $\pi \models \phi$

$s_0 \models (p \vee r)\ \mathbf{U}\ (p \wedge r)$ — false

Witness: $s_0 \to s_2 \to s_2 \to \cdots$

$p \wedge r$ is never sat.

$s_0 \models (p \vee r)\ \mathbf{W}\ (p \wedge r)$ → true

(1) We know: $(p \vee r)\ \mathcal{U}\ (p \wedge r)$ is not sat.

(2) but, $\mathcal{G}(p \vee r)$

$(p \vee r)\ \mathbf{W}\ (p \wedge r)$ is sat.

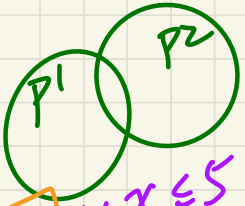
$s_0 \models (p \wedge r)\ \mathbf{R}\ (p \vee r)$

true — always true.

Exercise: What if we change the LHS to s_2?

Assertions: precondition, postcondition, invariant

satisfying values of an assertion



$x > 3 \rightarrow \{x \mid x > 3\} = \{4, 5, 6, 7, \ldots\}$

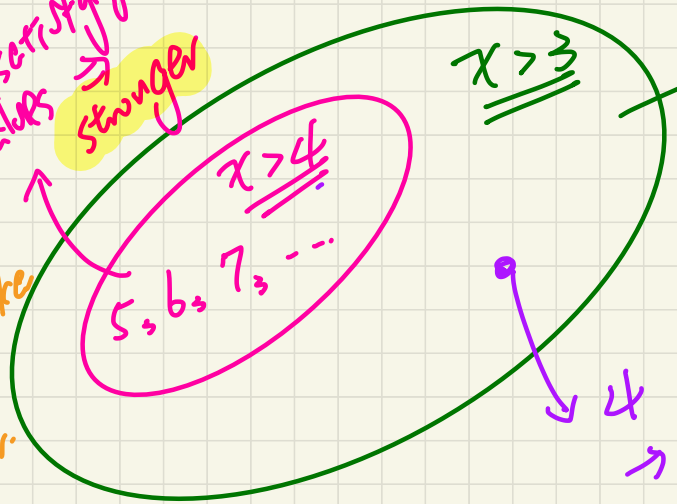
$x > 4 \rightarrow \{x \mid x > 4\} = \{5, 6, 7, \ldots\}$

P1: $1 \le x \le 5$

P2: $-2 \le x \le 3$

fewer satisfying values ⇒ stronger

cannot claim one is stronger/weaker than the other.



more satisfying values ⇒ weaker

$P1 \Rightarrow P2$

P1: stronger

P2: weaker

4 → $x > 3$ (T)

→ $x > 4$ (F)

Strongest assertion? false

Weakest assertion? true.

WTZ

- definitions: F, G, X

  ↳ prove vs. disprove them?

- path vs. model satisfaction.

≈ 9 questions.

Lecture 21 - March 31

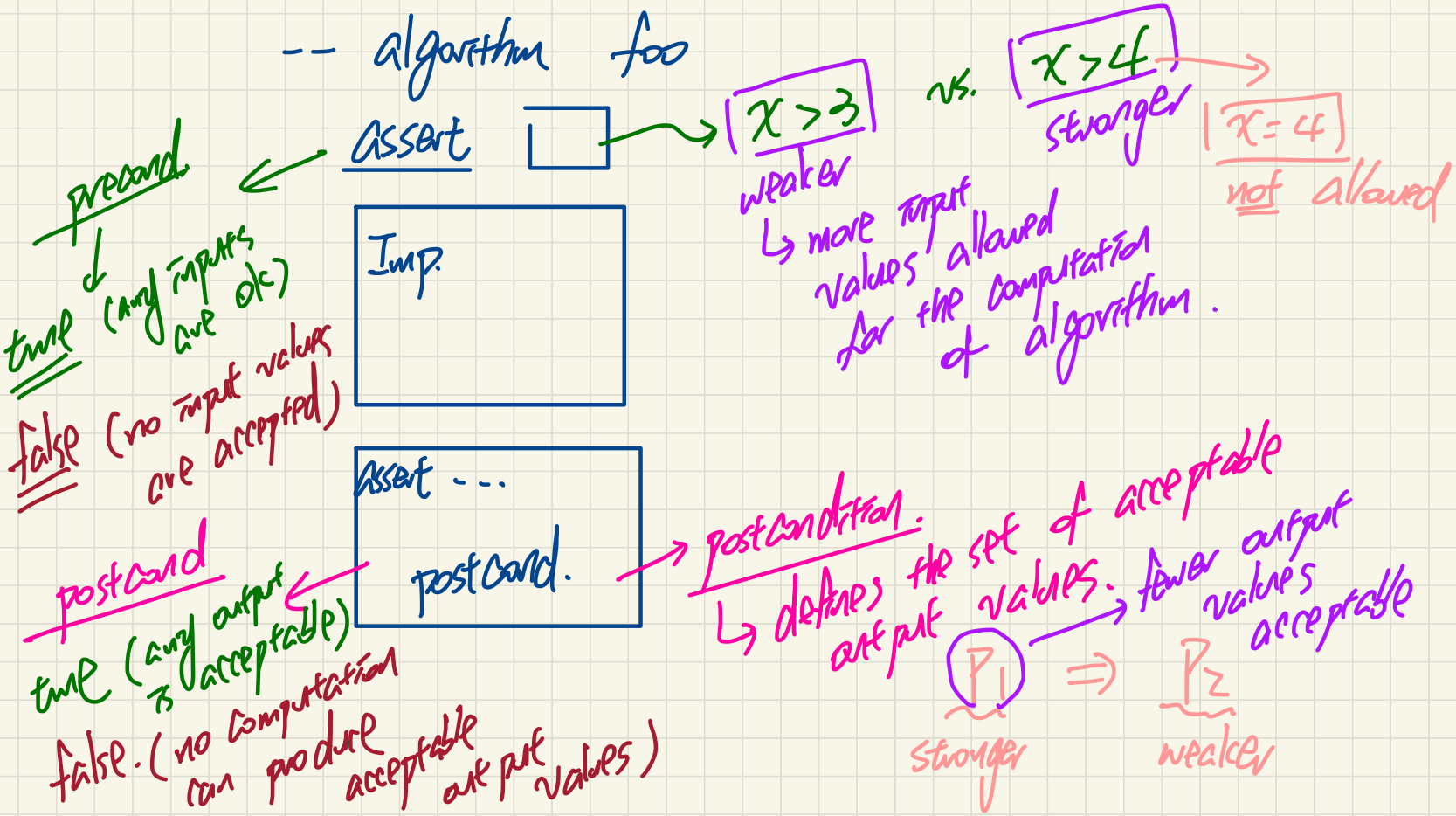
Program Verification

Weakest Precond: Predicate Transformer
wp Rules: Assignments, Conditionals

Announcements/Reminders

- **WrittenTest2** result released
- **Lab4** released
- Bonus opportunity: Final **Course Evaluation**
- Office Hour this week: 3pm on ~~Mon~~, Tue, Wed, Thu
- TA contact information (on-demand for labs) on eClass

Stronger vs. Weaker Assertions: Pre- vs. Post-Conditions



Program Correctness: Example (1)

```
--algorithm increment_by_9 {
  variable i;
  {
    (* precondition *)
    assert i > 3

    (* implementation *)
    i := i + 9;

    (* postcondition *)
    assert i > 13
  }
}
```

specification

precond. too weak: it allows $i = 4$

lead to postcond. violation.

$\{i > 3\}\ i := i + 9\ \{i > 13\}$

Correctness

1. Relative Concept
2. For input values sat. the precondition, executing the implementation will:

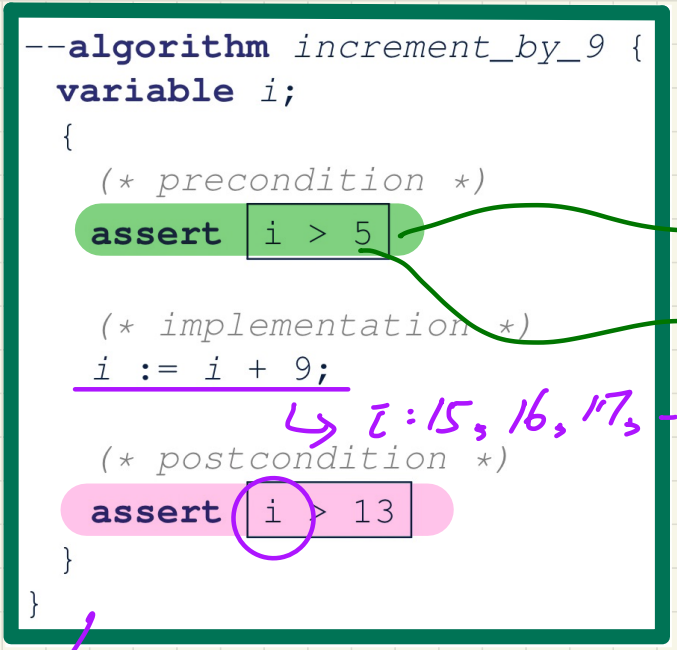
total correctness

(1) terminate

(2) upon termination, output/input values sat. postcondition.

partial correctness

Program Correctness: Example (2)



allowable input values: 6, 7, 8, 9, 10, ...

i : 15, 16, 17, ...

Is this precond. too strong?

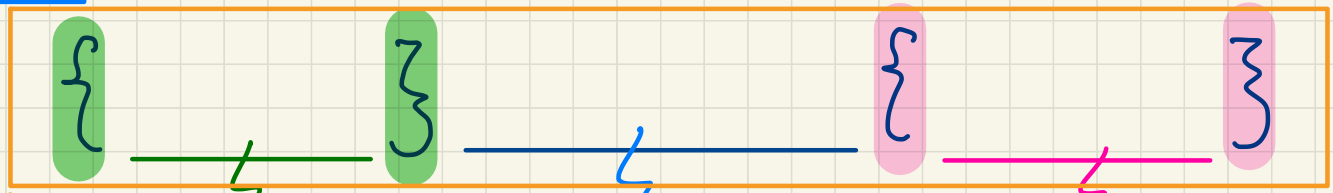
↳ i = 5

this disallowed input value actually would not cause any postcond. violation.

{i > 5} i := i + 9 {i > 13}

Tony Hoare (quick sort)

Hoare Triple: Syntax and Semantics



can be transformed into a predicate
↳ proved or disproved.

Hoare Triple : $\{Q\}\ S\ \{R\}$

↳ Starting in a pre-state satisfying Q, executing alg. S will

(1) terminate in a state Pol

(2) that post-state satisfies R Po2

Pol ∧ Po2 ⇒ total correctness.

Hoare Triple as a Predicate

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$

a predicate

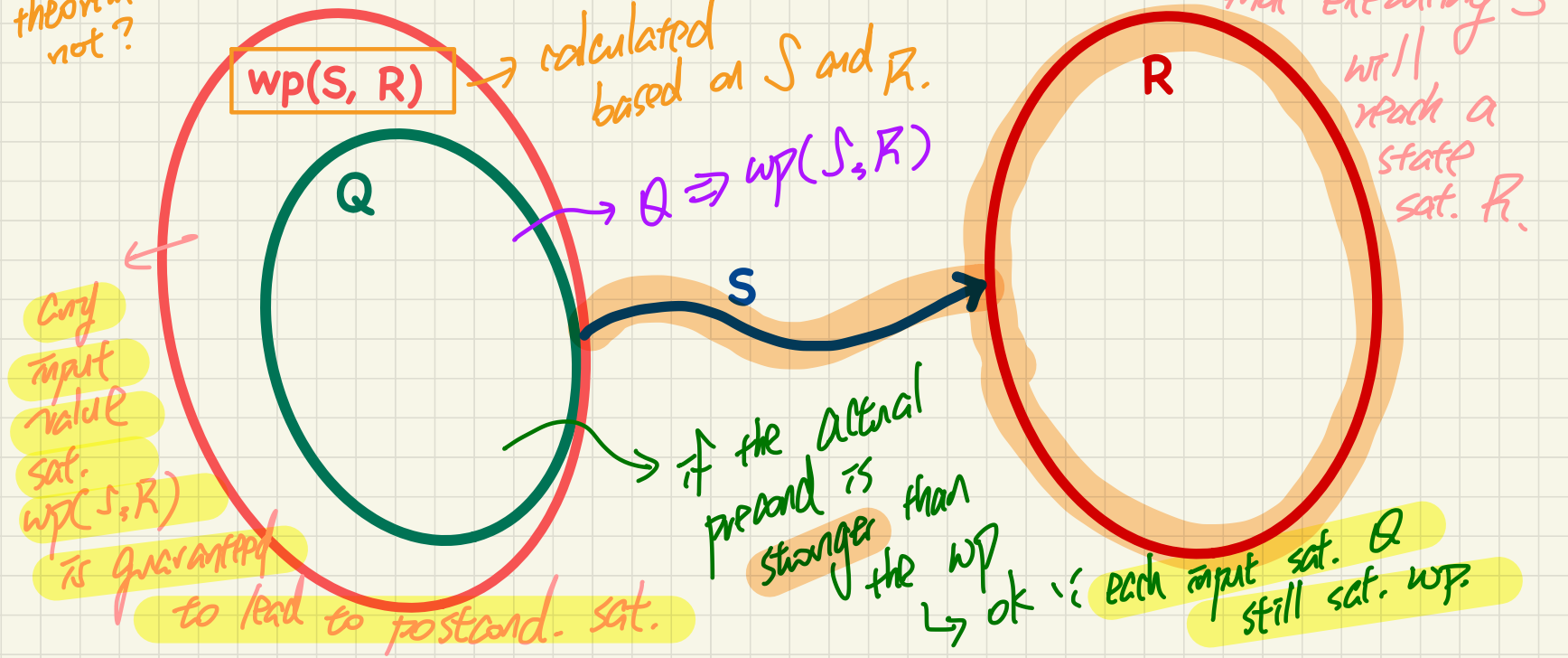
$wp(S, R)$: prog stmt. S, predicate R

predicate transformer

↳ weakest precondition satisfying which guarantees that executing S will reach a state sat. R.

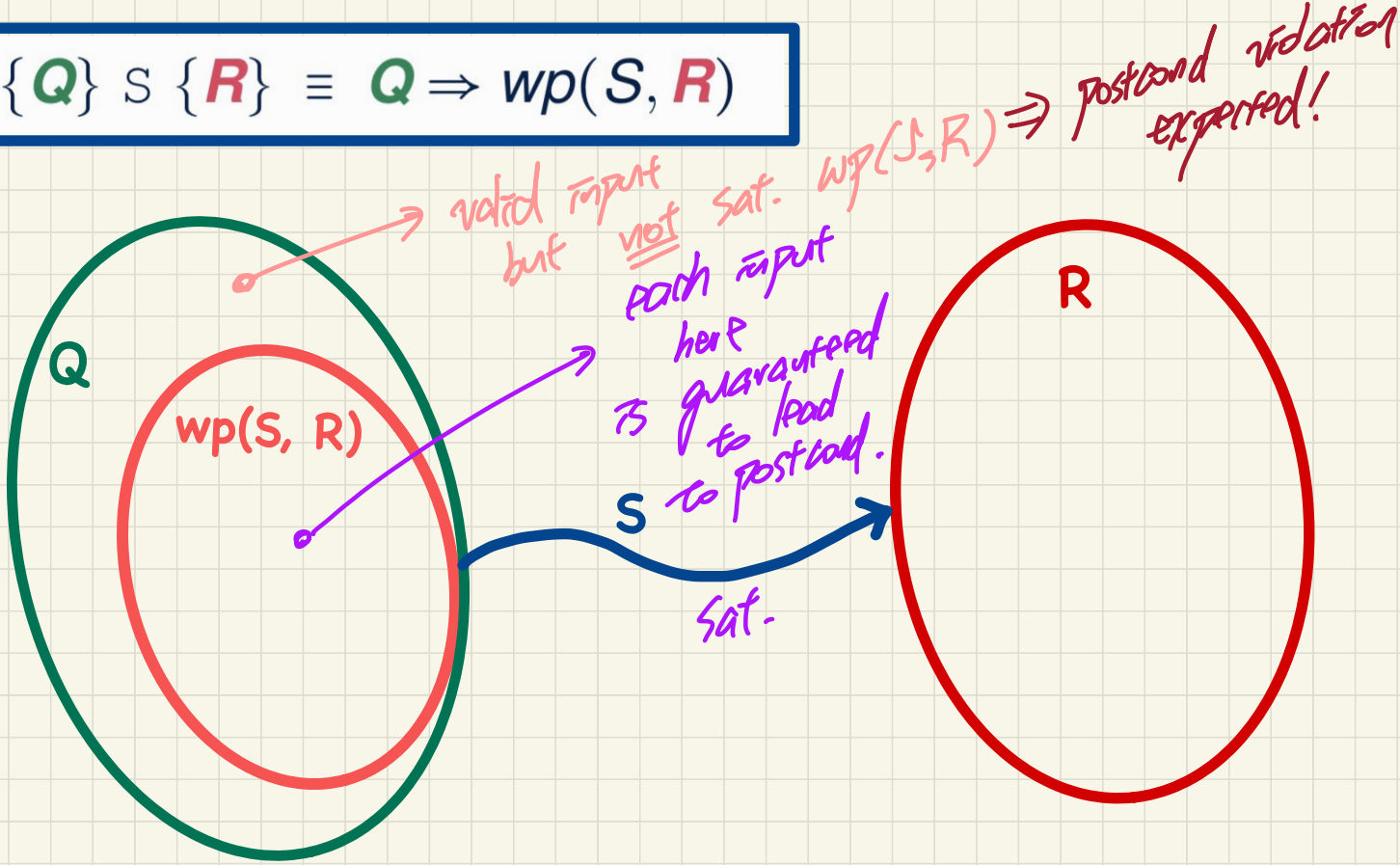
theorem or not?

no weaker than.



Hoare Triple: Incorrect Program

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$



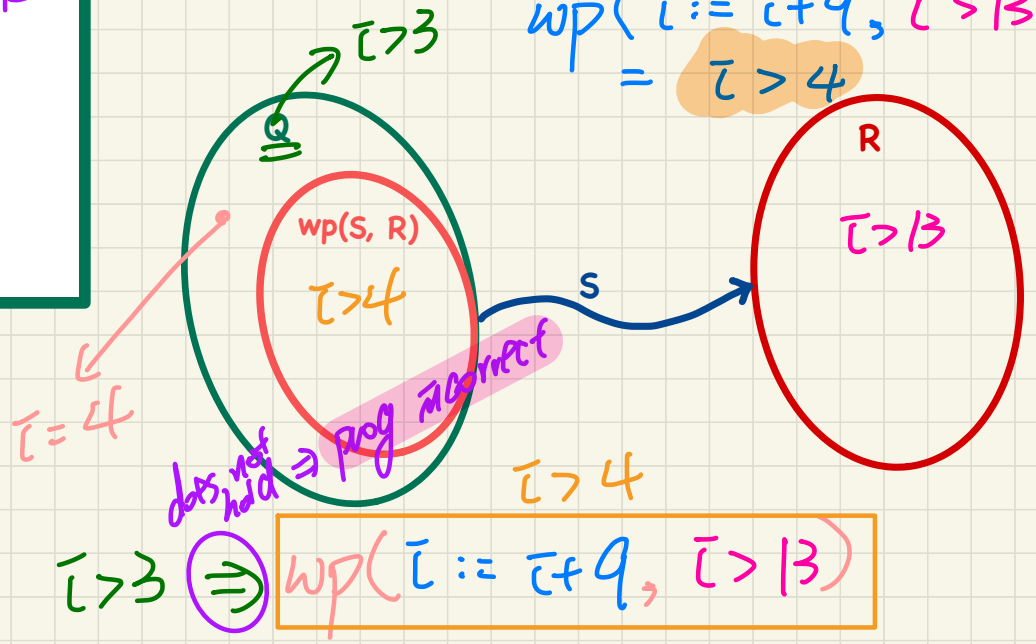
Program Correctness: Revisiting Example (1)

```
--algorithm increment_by_9 {
  variable i;
  {
    (* precondition *)
    assert i > 3;

    (* implementation *)
    i := i + 9;

    (* postcondition *)
    assert i > 13
  }
}
```

too weak

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$

$wp(i := i+9,\ i > 13) = i > 4$



Program Correctness: Revisiting Example (2)

```
--algorithm increment_by_9 {
  variable i;
  {
    (* precondition *)
    assert i > 5

    (* implementation *)
    i := i + 9;

    (* postcondition *)
    assert i > 13
  }
}
```

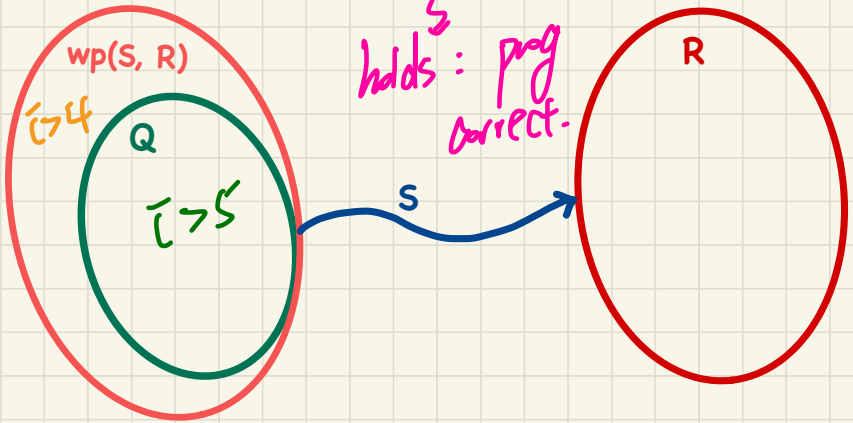
$wp(i := i+9, i>13)$

$i > 4$

$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$

$i > 5 \Rightarrow i > 4$

holds : prog correct.

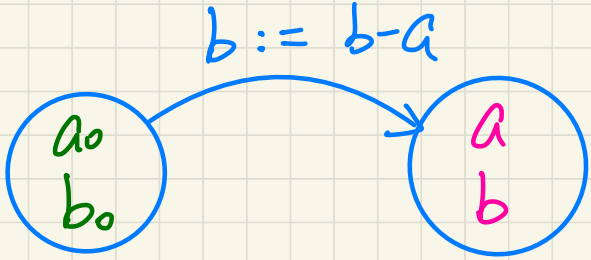


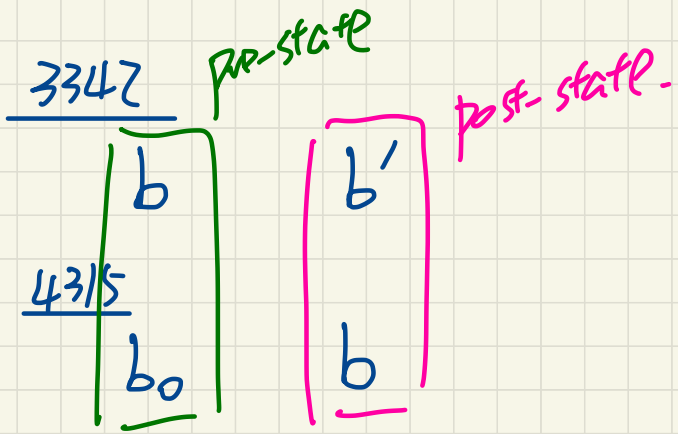
Expressing Pre-State vs. Post-State Values

$\{ b_0 > a_0 \}$ $b := b - a$ $\{ b = b_0 - a \}$

b: post-state value

b_0: pre-state value

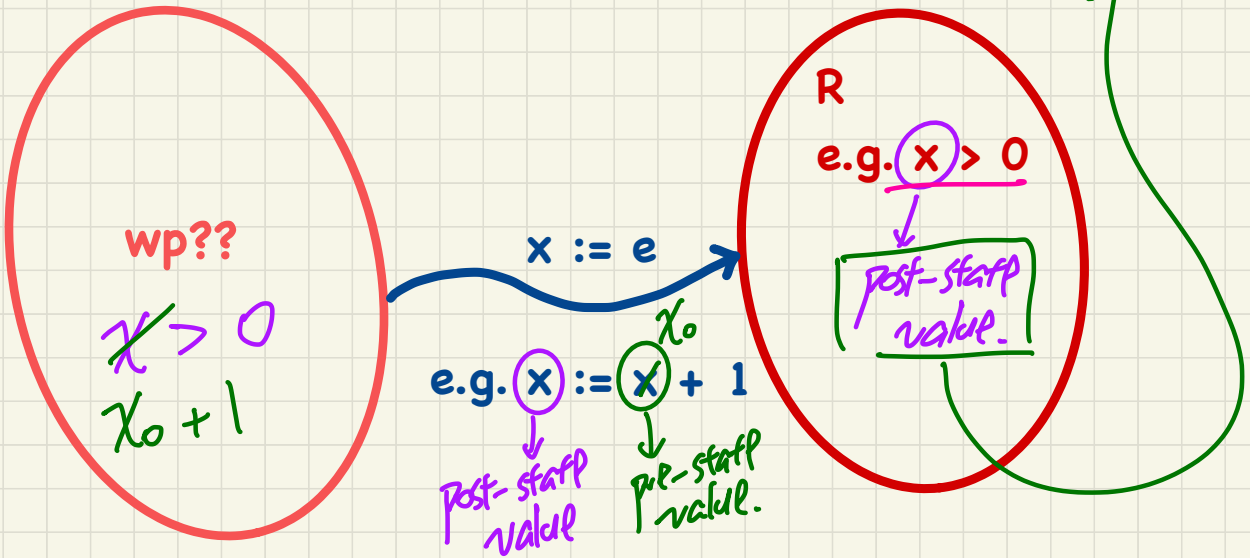




Rules of Weakest Precondition: Assignment

$wp(x := e, R) = R[x := e]$

$x_0 + 1$ $x > 0$



Correctness of Programs: Assignment (1)

What is the weakest precondition for a program `x := x + 1` to establish the postcondition $x > x_0$?

$$\{??\}\ \texttt{x := x + 1}\ \{x > x_0\}$$

$$wp(x := \underbrace{x_0 + 1}_{e}, x > x_0)$$

$$= \{ \text{wp rule of } := \}$$

$$x > x_0\ [x := x_0 + 1]$$

$$= x_0 + 1 > x_0$$

$$= 1 > 0 \equiv \boxed{\text{True}}$$

Correctness of Programs: Assignment (2)

Exercise

What is the weakest precondition for a program `x := x + 1` to establish the postcondition $x > x_0$?

$$\{??\}\ \texttt{x := x + 1}\ \{x = 23\}$$

Rules of Weakest Precondition: Conditionals

B ¬B

wp(if B then S1 else S2 end, R)

=

$B \Rightarrow wp(S_1, R)$

going for branch 1 guarantees that R is sat.

$\wedge$

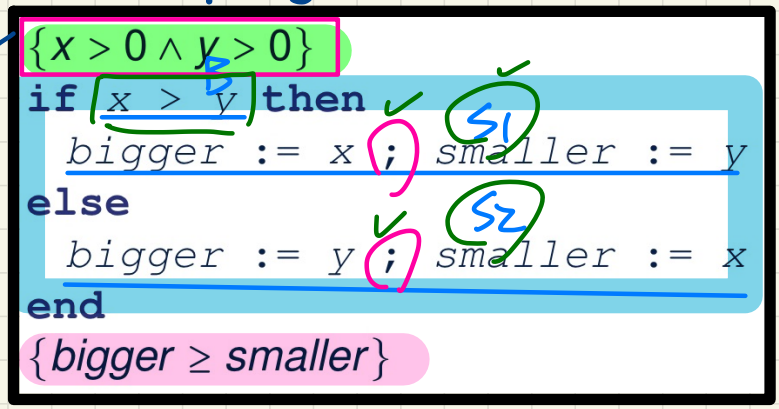
$\neg B \Rightarrow wp(S_2, R)$

going for branch 2 guarantees that R is sat.

both branches must est. the postcond. R.

Correctness of Programs: Conditionals

Is this program correct?



(Step 3) Prove or disprove

$x > 0 \wedge y > 0$

$\Rightarrow$

$wp(S, \text{bigger} \geq \text{smaller})$

(Step 1) Formulate Hoare Triple

$\{x > 0 \wedge y > 0\}$ **if** B **then** S_1 **else** S_2 **end** $\{\text{bigger} \geq \text{smaller}\}$

(S)

(Step 2) Calculate $wp(S, \underbrace{\text{bigger} \geq \text{smaller}}_{R})$

$\rightarrow$ $x > y \Rightarrow wp(S_1, R)$

$\neg(x > y) \Rightarrow wp(S_2, R)$

Lecture 22 - April 2

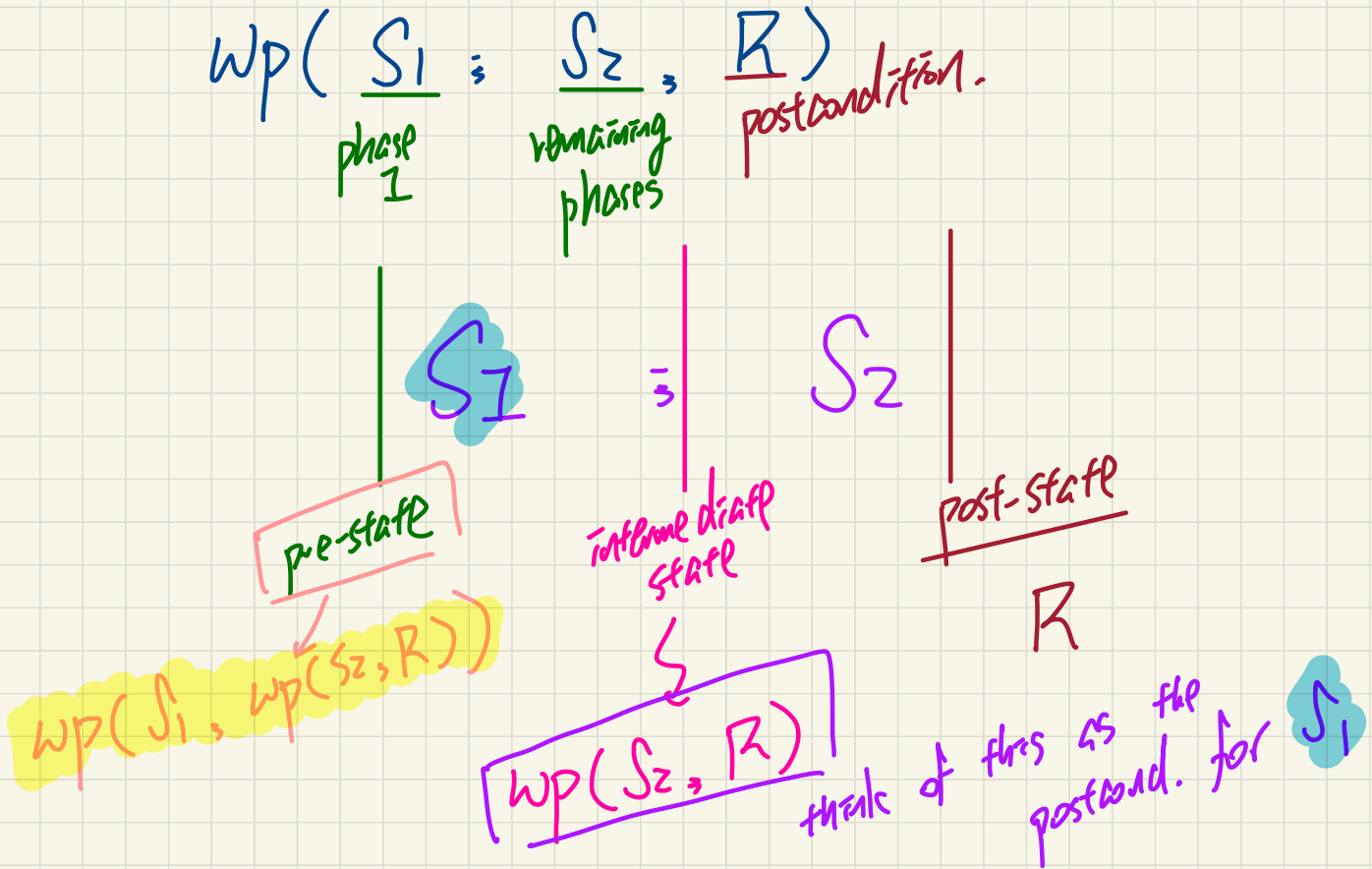
Program Verification

wp rule: Sequential Composition
Loop Invariant vs. Loop Variant
Correctness Conditions of Loops

Announcements/Reminders

- **Exam guide** released
- Some example questions to be released by April 7
- **WrittenTest2** result released
- **Lab4** released
- Bonus opportunity: Final **Course Evaluation**
- Office Hour this week: 3pm on Wed, Thu
- TA contact information (on-demand for labs) on eClass

wp Calculation for Sequential Composition



Correctness of Programs: Sequential Composition

Is $\{ \textit{True} \}$ `tmp := x; x := y; y := tmp` $\{ x > y \}$ correct?

Q S R

$\{Q\}\, S\, \{R\} \iff Q \Rightarrow wp(S, R)$?? 1.

(Step 2)

(Step 1) Calculate $wp(tmp := x;\ x := y;\ y := tmp,\ x > y)$

$= \{$ wp rule of ; $\}$

$wp(tmp := x,\ wp(x := y;\ y := tmp,\ x > y))$

$= \{$ wp rule of ; $\}$

$wp(tmp := x,\ wp(x := y,\ wp(y := tmp,\ x > y)))$

$= \{$ wp rule of := $\}$

$wp(tmp := x,\ wp(x := y,\ x > tmp))$

$= \{$ wp rule of := $\}$

$wp(tmp := x,\ y > tmp) = \{$ wp rule of := $\} = y > x$

$True \Rightarrow y > x$

$\equiv \{$ rule of $\Rightarrow \}$

$y > x$ ✗

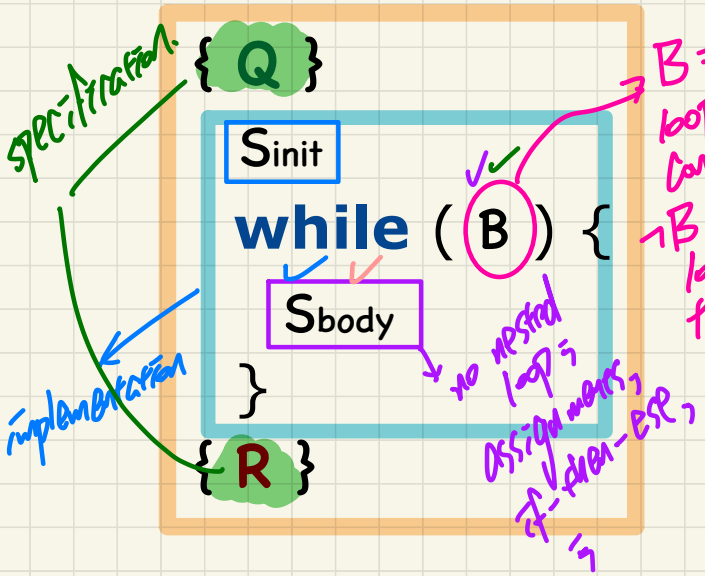
↳ counter ex

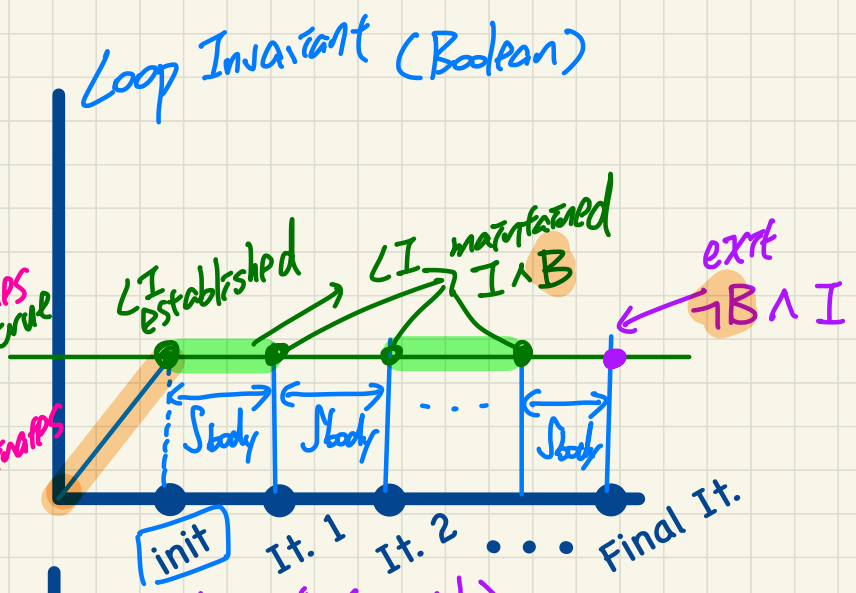
$y \leq x$

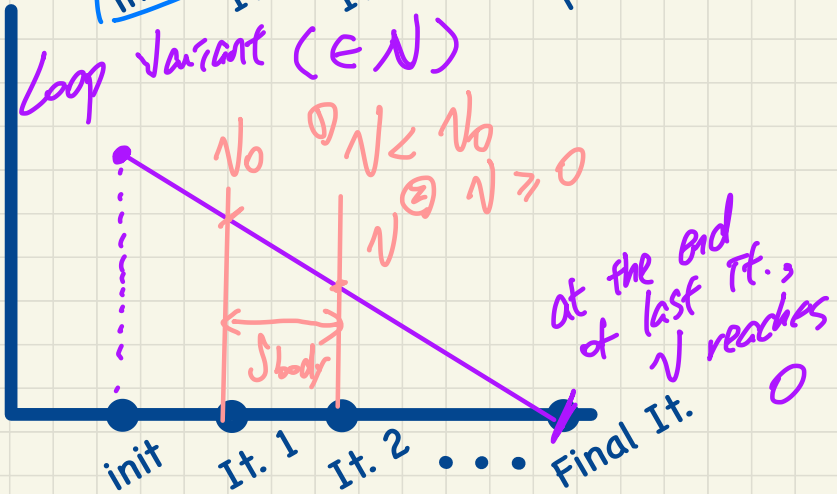
$x = 3$

$y = 2$

Correctness of Loops







Contracts of Loops

Syntax

```
CONSTANT ... (* input list *)
I(var_list) == ...
V(var_list) == ...
--algorithm MyAlgorithm {
  variables ..., variant_pre = 0, variant_post = 0
  {
    assert Q; (* Precondition *)
    S_init
    assert I(...); (* Is LI established? *)
    while( B ) {
      variant_pre := V(...);
      S_body
      variant_post := V(...);

      assert variant_post >= 0;
      assert variant_post < variant_pre;
      assert I(...); (* Is LI preserved? *)
    }
    assert R; (* Postcondition *)
  }
}
```

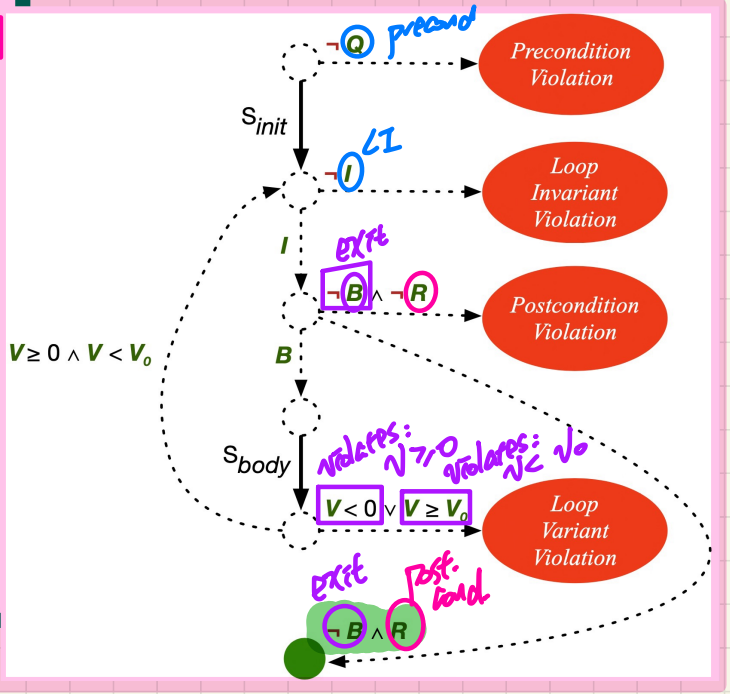
LI
LV

meta-variables to capture V. in pre- and post-state

before entering loop

1. $V_{post} < V_{pre}$
2. $V_{post} \in \mathbb{N}$

Runtime Checks



Contracts of Loops: Example

Assume: Q and R are true

```
1  I(i) == (1 <= i) /\ (i <= 6)
2  V(i) == 6 - i
3  --algorithm loop_invariant_test
4   variables i = 1, variant_pre = 0, variant_post = 0;
5   {
6    assert I(i);
7    while (i <= 5) {
8      variant_pre := V(i);
9      i := i + 1;
10     variant_post := V(i);
11     assert variant_post >= 0;
12     assert variant_post < variant_pre;
13     assert I(i);
14   } ;
15  }
```

Specification

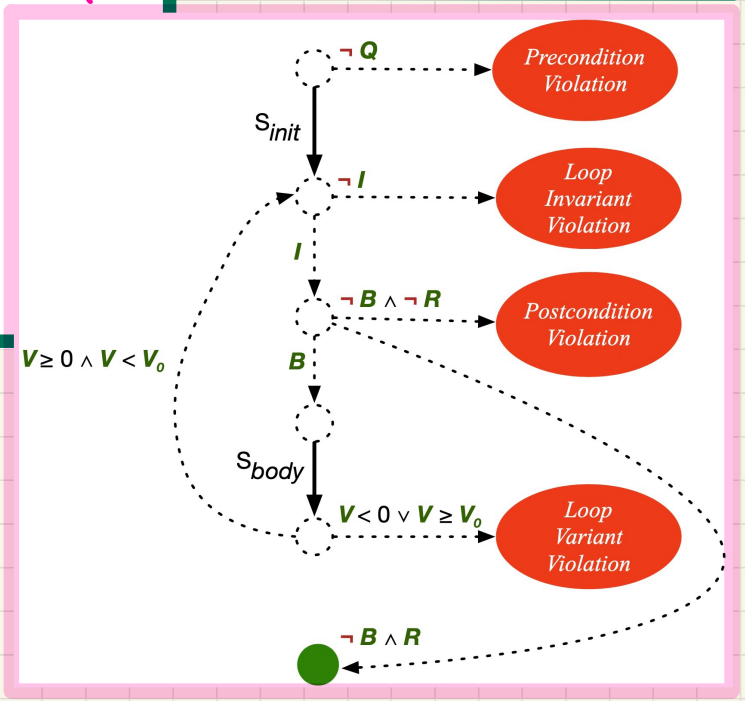
S_{init}

not part of S_{init}

S_{body}

end of iteration	i	I	Vpre	Vpost	B
	1	T	—	—	T
1	2	1	6-1=5	6-2=4	T
2	3		4	3	
3	4		3	2	
4	5		2	1	
5	6		1	0	F

Runtime Checks



Contracts of Loops: Violations

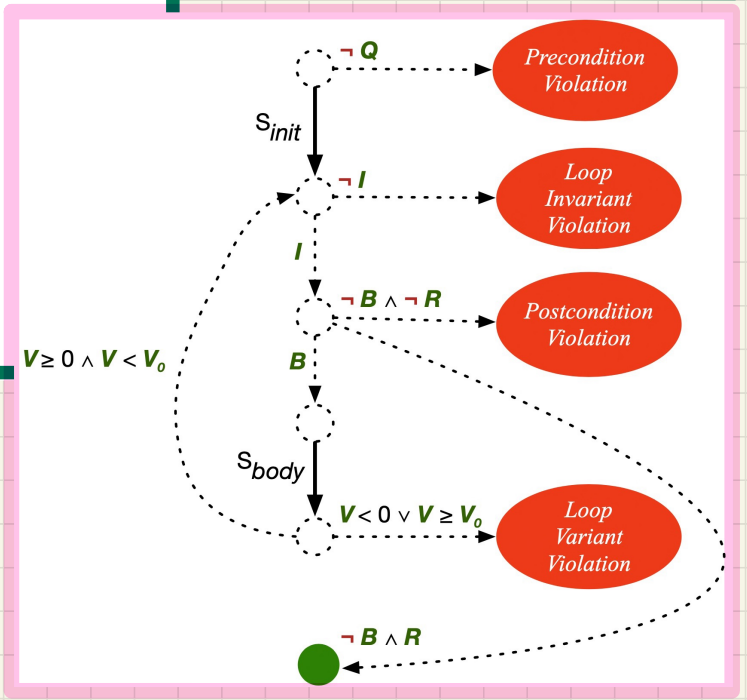
Exercise

Assume: Q and R are true

```
1  I(i) == (1 <= i) /\ (i <= 6)
2  V(i) == 6 - i
3  --algorithm loop_invariant_test
4   variables i = 1, variant_pre = 0, variant_post = 0;
5   {
6     assert I(i);
7     while (i <= 5) {
8       variant_pre := V(i);
9       i := i + 1;
10      variant_post := V(i);
11      assert variant_post >= 0;
12      assert variant_post < variant_pre;
13      assert I(i);
14    } ;
15  }
```

Specification

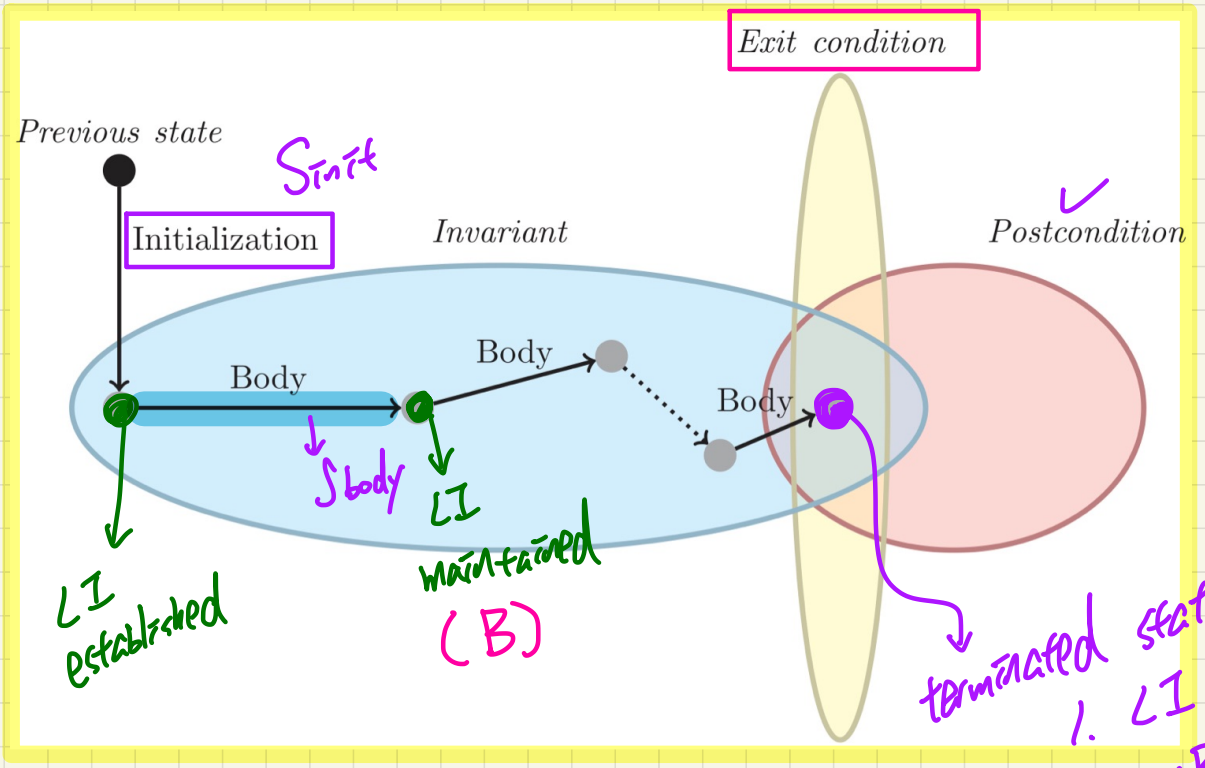
Runtime Checks



invariant: 1 <= i <= 5

variant: 5 - i

Contracts of Loops: Visualization



Correct Loops: Proof Obligations

- A loop is ***partially*** *correct* if:
  - Given precondition ***Q***, the initialization step S_{init} establishes ***LI*** *I*.
    $\{Q\}\ S_{init}\ \{LI\}$
  - At the end of S_{body}, if not yet to exit, ***LI*** *I* is maintained.
    $\{I \wedge B\}\ S_{body}\ \{I\}$
  - If ready to exit and ***LI*** *I* maintained, postcondition ***R*** is established.
    $\neg B \wedge LI \Rightarrow R$
- A loop ***terminates*** if:
  - Given ***LI*** *I*, and not yet to exit, S_{body} maintains ***LV*** *V* as non-negative.
    $\{LI \wedge B\}\ S_{body}\ \{V \geq 0\}$
  - Given ***LI*** *I*, and not yet to exit, S_{body} decrements ***LV*** *V*.
    $\{LI \wedge B\}\ S_{body}\ \{V < V_0\}$

```
{Q}
S_init
assert I(...);
while( B ) {
  variant_pre := V(...);
  S_body
  variant_post := V(...);
  assert variant_post >= 0;
  assert variant_post < variant_pre;
  assert I(...);
}
{R}
```

Correct Loops: Proof Obligations

Example

```
1  I(i) == (1 <= i) /\ (i <= 6)
2  V(i) == 6 - i
3  --algorithm loop_invariant_test
4   variables i = 1, variant_pre = 0, variant_post = 0;
5   {
6     assert I(i);
7     while (i <= 5) {
8       variant_pre := V(i);
9       i := i + 1;
10      variant_post := V(i);
11      assert variant_post >= 0;
12      assert variant_post < variant_pre;
13      assert I(i);
14    } ;
15  }
```

Specification

- A loop is ***partially*** *correct* if:
  - Given precondition ***Q***, the initialization step S_{init} establishes ***LI*** *I*.
  - At the end of S_{body}, if not yet to exit, ***LI*** *I* is maintained.
  - If ready to exit and ***LI*** *I* maintained, postcondition ***R*** is established.
- A loop ***terminates*** if:
  - Given ***LI*** *I*, and not yet to exit, S_{body} maintains ***LV*** *V* as non-negative.
  - Given ***LI*** *I*, and not yet to exit, S_{body} decrements ***LV*** *V*.

- No multiple choice questions
- definitions
- short answers
  - ↳ justification
  - ↳ assertions (Lab2) e.g., = algorithm (not as long as in RegTest).
  - ↳ proofs. (math review, LTL).
- qs booklet
- answer booklet

I hope you enjoyed learning with me 🤗

All the best to you !