EECS4315 Mission-Critical Systems

Lecture Notes

Winter 2023

Jackie Wang

Lecture 1 - January 10

Syllabus & Introduction

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

formal methods

theorem proving (3342)

manual, semi-automated

- manual ↓ step by step via relevant inference rules by hands
- semi-automated ↓ human intervention for the prover to complete

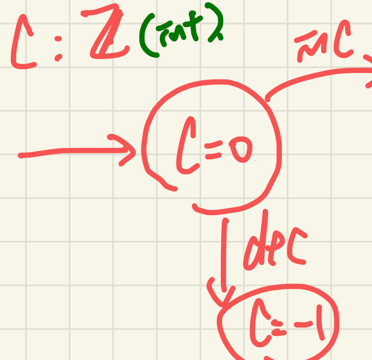
- no state ex. concern
- challenge: discharge POs.

model checking (4315)

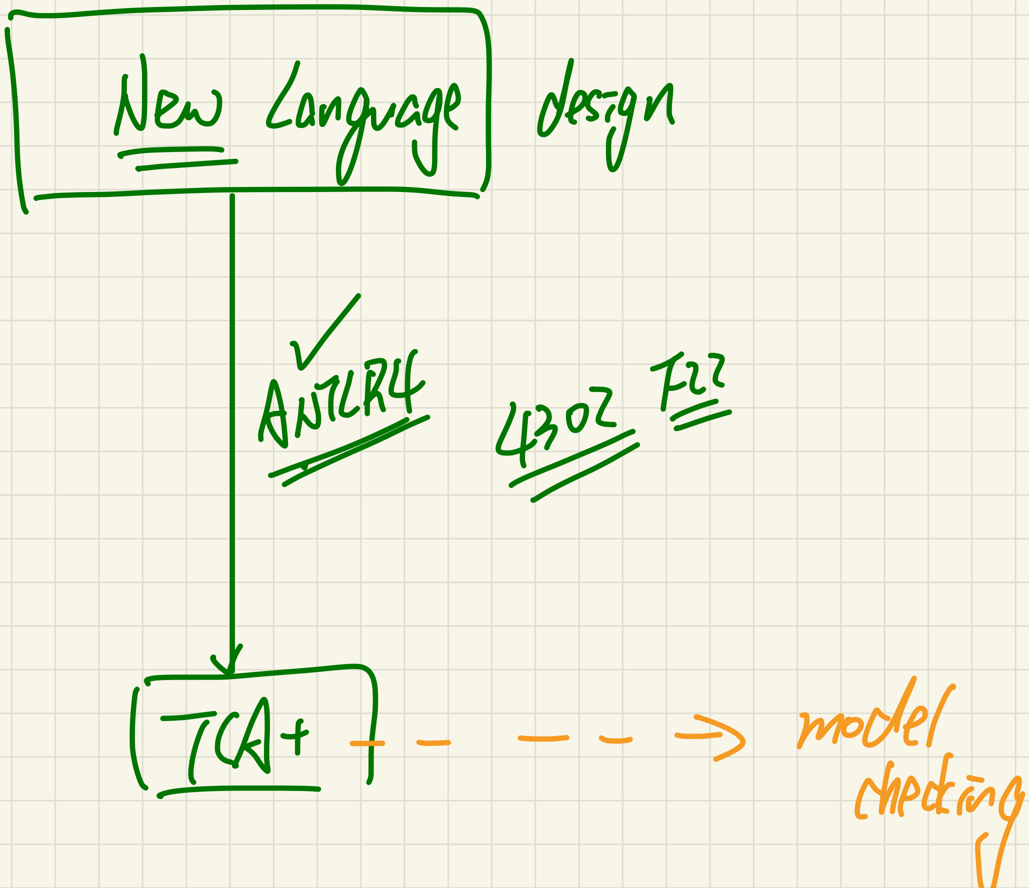
automated → based on your specification, a graph is generated.

BUT suffers from state explosion problem

c : Z (int)



if property is violated ⇒ a witness (counter ex. trace) is gen-


New Language design
ANTLR4
4302 F22
TLA+
model checking.

Logic covered

untimed.

EECS 3342: $\boxed{\text{proposition / predicates}}$

EECS 4315: temporal logic

relative notion of time

- eventuall P holds
- infinitely often P holds

LTL - linear temporal logic

CTL - computation tree logic

Lecture 2 - January 12

Introduction

Safety- vs. Mission-Critical Systems
Formal Methods
Industrial Standards
Verification vs. Validation

SCS

→ auto-pilot / auto-driving

traffic light / train gate

air bag deployment

elevator / escalator

impulse dector / pacemaker

nuclear power plant / shutdown system

OPG

↳ Ontario Power Gen.

↳ Darlington

Shutdown

Systems.

Precise math.

EECS4312 → NAT

↳ no scope of multiple interpretations.

↳ code: may be precise but too low level!

Complete

↳ no missing cases.

(Java) Implemen. - - - → Property

e.g. sensor_val < T

formulate

formulate

3342: Rodin

4315: TLA+

Spec — invariant refinement → predicate

(P1) System $\mathcal{S}$ is mission critical

(P2) System $\mathcal{S}$ is safety critical

(1) $P_1 \equiv P_2$

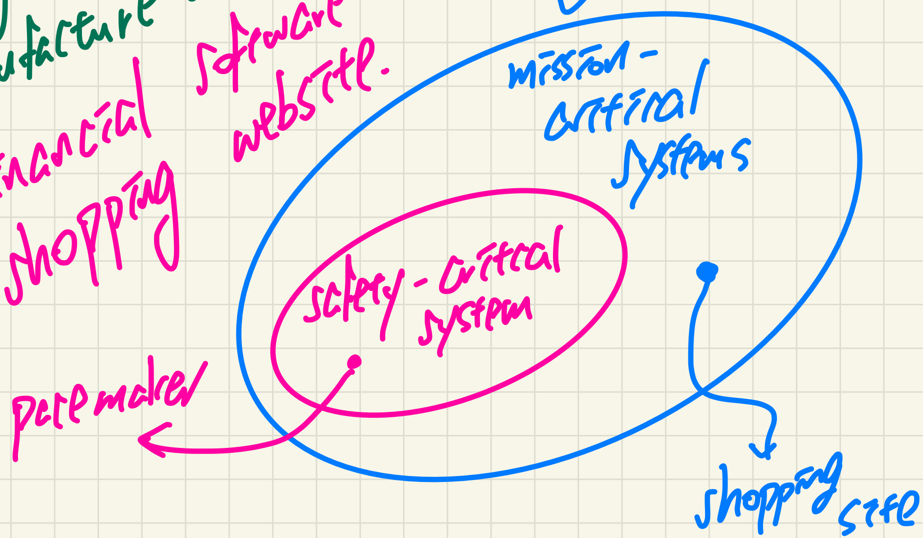
(2) $P_1 \Rightarrow P_2$ — not true in general

(3) $P_2 \Rightarrow P_1$ — always the case

e.g. car gas emission

e.g. manufacture a working iPad

e.g. financial software shopping website.



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.

mission-critical

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).

3342: theorem proving
4315: model checking.

These levels correspond to the consequences of a software failure:

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

safety-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.

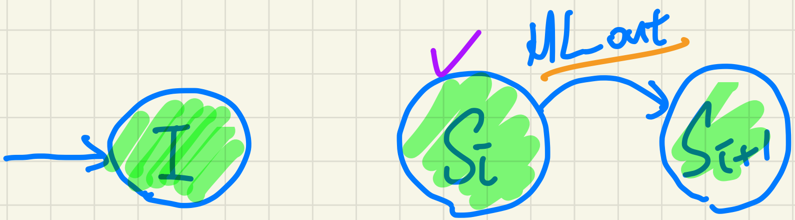
Generally mission-critical applications include navigation systems, avionics display systems, and mission command and control.

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

Safety (invariant) property

I

(mathematical induction).
↳ weak



assume I satisfied in S_i

prove I satisfied in S_{i+1}

according to before-after predicate of ML_out.

(verification) — have we built the product right?

(validation) — have we built the right product?

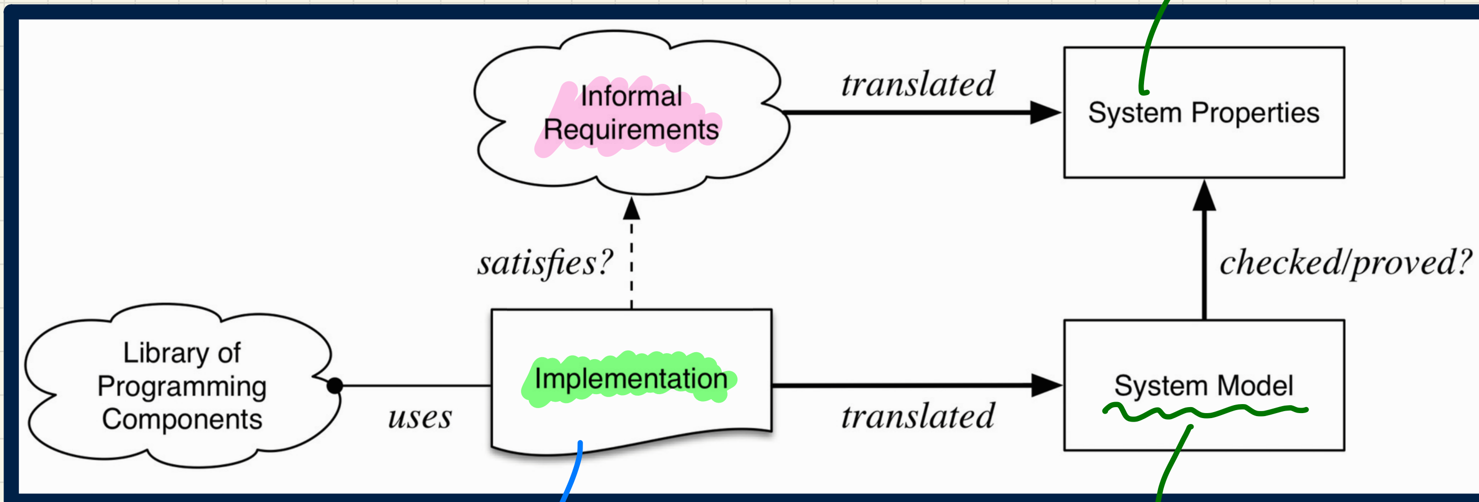
Implicit assumption:
1. requirements clarified
2. design decisions made

was the right process followed to build

req. elicitation

make sure built what the customers want

Building the product right?



predicates

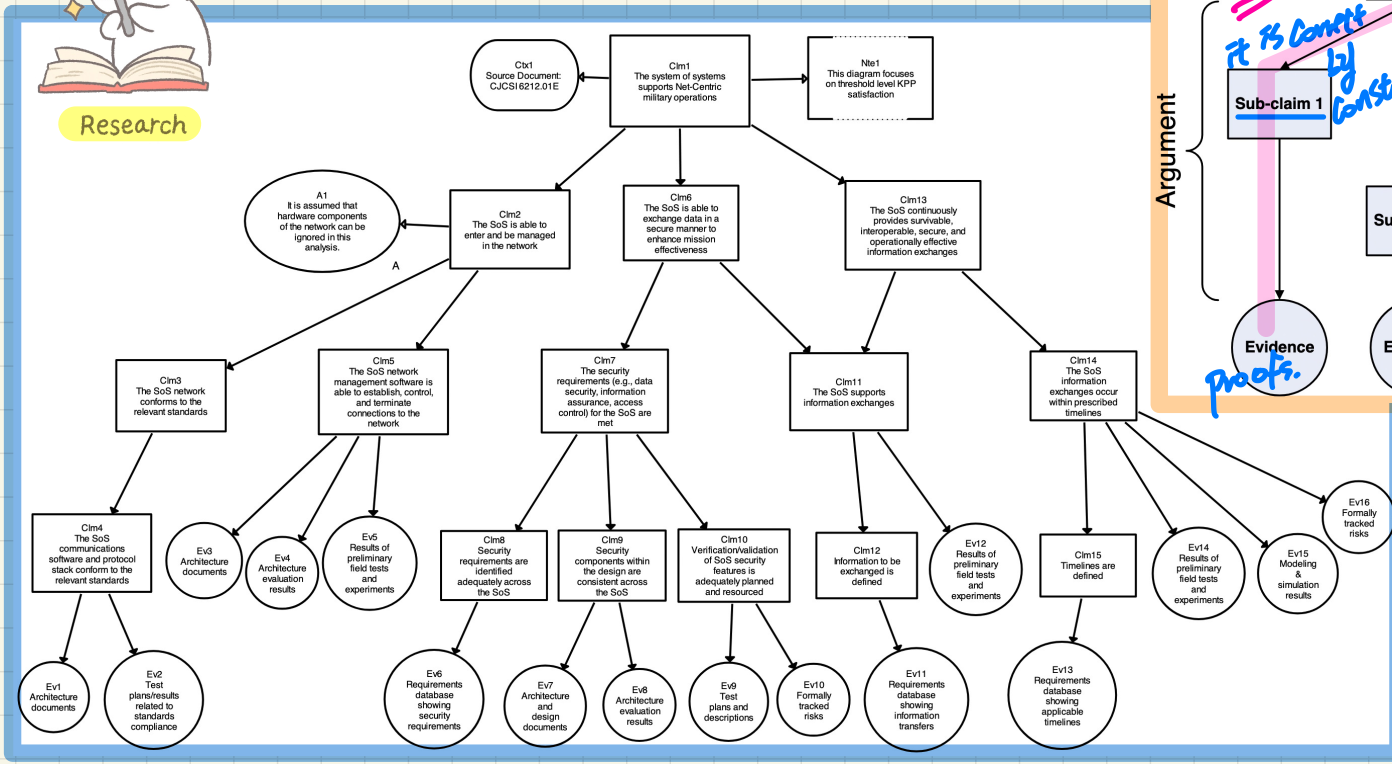
abstract state machine

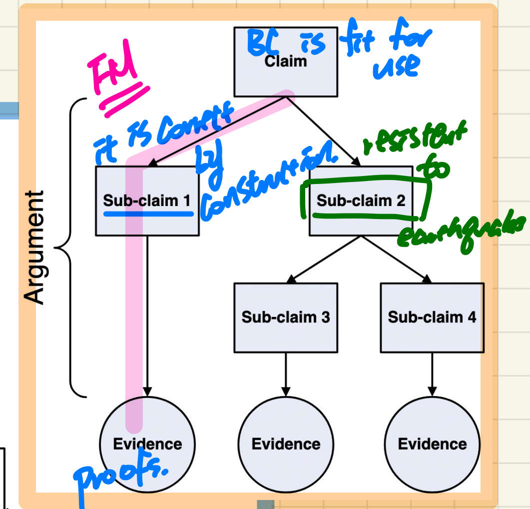
1. depends on other components
2. those components should be certified!

Certifying Systems: Assurance Cases

Research

Research on "Assurance Cases" if interested!





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

Lecture 3 - January 17

Introduction, Math Review

Model-Based Development
TLA+
Logical vs. Programming Operators

Announcement

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

Software Development Process

[Lab] choice {...} or {...}

non-deterministic op.

if (..)
elseif (..)

REQUIREMENT

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

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

DESIGN

- Blueprints
- Not necessarily executable & testable

IMPLEMENTATION

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

RELEASE

- Customer's Acceptance
- Recall?

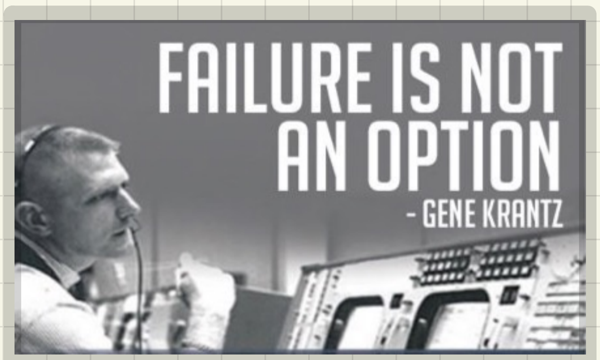
Correct by Construction

$$m_0 \to \cdots \to m_i \to m_j \to \cdots \to m_n$$

simplest, most abstract model

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

most sophisticated; most concrete model (closest to code) ↳ variables

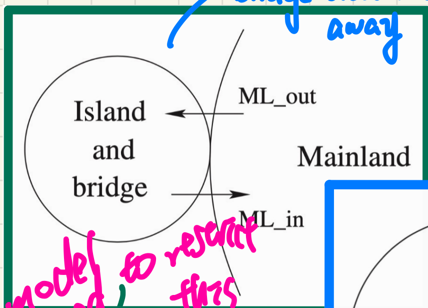


Source: https://audiobookstore.com/audiobooks/failure-is-not-an-option-1.aspx

Correct by Construction: Bridge Controller System

$m_0 \to \cdots \to m_i \to m_j \to \cdots \to m_n$

bridge abstracted away



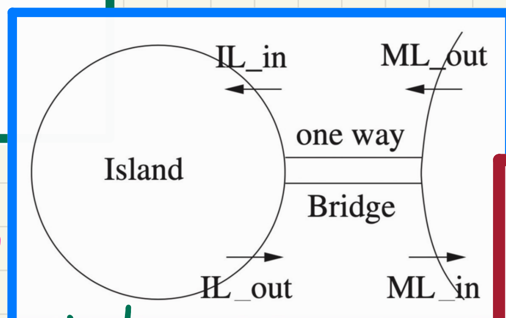
m0

model to restrict this to a specific value

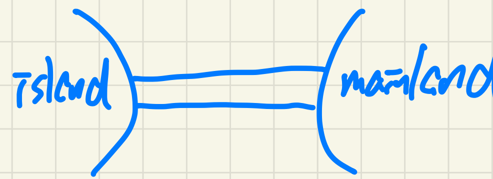
to check, need

$d : \mathbb{N}$

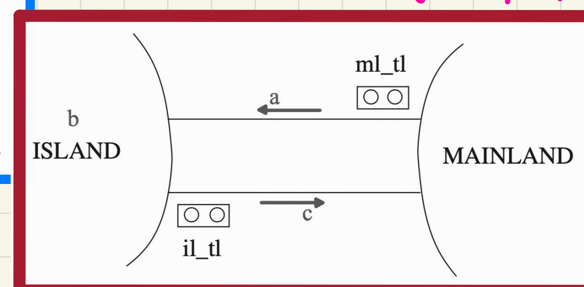
limit on # cars on bridge-island complex



m1

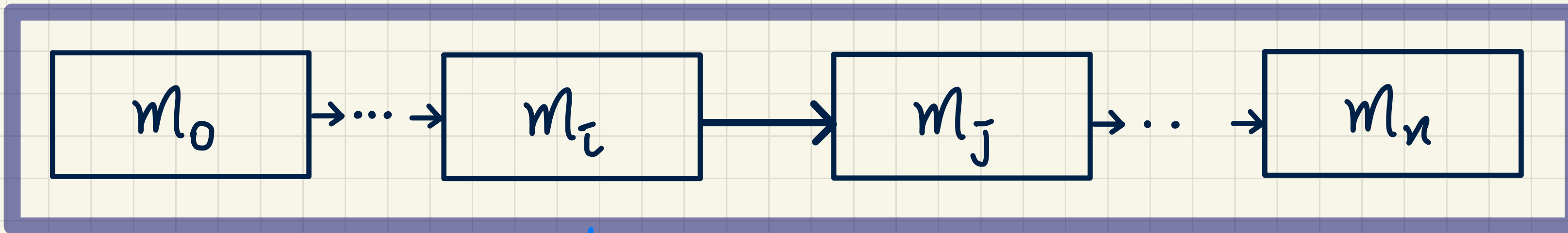


1. bridge
2. traffic light
3. flow in both directions



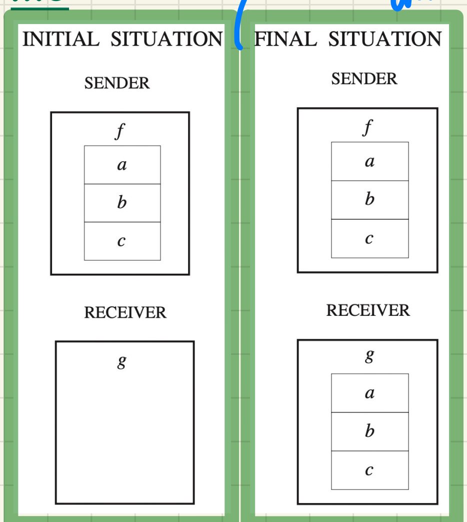
m2

Correct by Construction: File Transfer Protocol

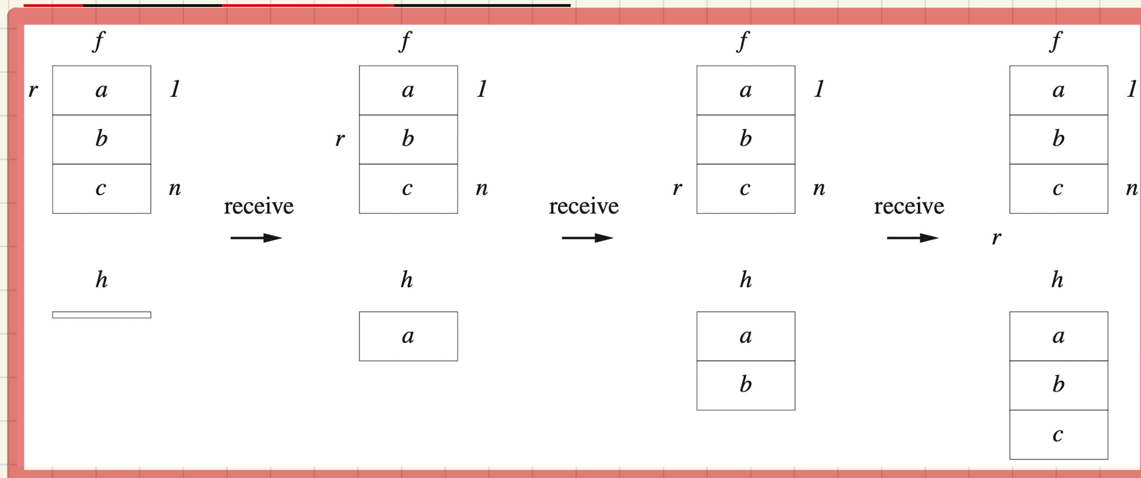


m0

delay in transmission abstracted away



m1: more concrete than m0



module

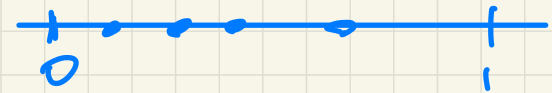
$v_1 : T_1$

- Boolean $\{true, false\}$
- Int $-MAX \sim +MAX$
- Range $3..10$

$v_2 : T_2$

$\vdots$

$v_n : T_n$



state space : the combination of all variables

size of state space :

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

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

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

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

Rodin (refinement tool)

MACHINE m0
VARIABLES n

refines.

MACHINE m1
VARIABLES a, b, c

TLA+ toolbox (no notion of refinement)

m0

module

n

m1

module

n, a, b, c

TLA+ Toolbox

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

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

C-like design language.

ProgTest

- data sheet
- PDF sheet

format: write PlusCal algorithms & properties

↳ inv. or temporal

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

event-b description

m0 $\xrightarrow{①}$ PlusCal $\xrightarrow{②}$ model check $\xrightarrow{③}$ fix model s.t. all pass

↳ counter-example trace

↳ explain

debug.

Lecture 1b

Review on Math

	TLA+
$\neg$	~
$\wedge$	/\
$\vee$	\/
$\Rightarrow$	=>

>

<

$\boxed{\text{>=}}$ =<

<=

$\forall$	\A
$\exists$	\E

Logical Operator vs. Programming Operator

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

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

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

precedence?

e1 && e2

if e1 evaluated to F

e2 will not be evaluated ⇒ overall result : F

e1 || e2

↳ if e1 evaluated to T

e2 will not be evaluated ⇒ result: T.

array a : int[ ]

logic: P_0 well-definedness

$\underbrace{i < a.\text{length}}_{T}$ && $a[i] >= 10$ && $i >= 0$

$a[-2]$

AIOBE.

$i < 0$

e.g. $\boxed{i == -2}$.

Lecture 4 - January 19

Math Review

Propositional Logic, Predicate Logic

Announcement

- Tuesday's lecture recording mossing!
- Lab1 released
  - + tutorial videos
  - + problems to solve

Implication ≈ Whether a Contract is Honoured

$p \Rightarrow q$

p → promised term (e.g. salary)

q → obligations (e.g. work).

$p \Rightarrow q$ T if the contract is not breached

(C1) $p = T$ $q = T$ T

(C2) $p = T$ $q = F$ F.

(C3) $p = F$ $q = T$ T

(C4) $p = F$ $q = F$. T.

Common base/trivial cases for justification.

Expressing Implications

p: snow storm
q: cancel class

q if p, p is sufficient for q

casual relation
if p then q

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

when the cause is not T then it doesn't matter what consequence is (the causal rel. not violated)

q unless ¬p

conditions for ⇒ to be true

C1: q

C2: p = F

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

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

known: p T

don't care what q is.

the only way the ⇒ is T is q is also T

q is necessary to be T in order for ⇒ to be T if we know p is T

$$\boxed{p \Rightarrow q \equiv \neg p \vee q}$$

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

(1) q if p ✓

(2) q only if p ✗

$\Longleftrightarrow$

$$p \Longleftrightarrow q$$

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

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

$\neg(x < 0) \to x \geq 0$

$P \Rightarrow q$

Contrapositive (Converse of Inverse)

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

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

Converse

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

Inverse: $\neg p \Rightarrow \neg q$

equivalence proof in equational style

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

$\equiv$ {De Morgan}

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

Prove $\quad P \Leftrightarrow q$

if $\quad$ only if

Need to proof

(1) $P \Leftarrow q \quad$ P if q

$q \Rightarrow P$

(2) $P \Rightarrow q \quad$ P only if q

Identity

identities
$$\boxed{\begin{array}{l} 0 + a = a \\ 1 * a = a \end{array}}$$

$$True \Rightarrow P \equiv P$$

as if:
$$(True \Rightarrow P) \equiv P$$

$$True \wedge P \equiv P$$

$$false \vee P \equiv P$$

Precedence of operators

$\neg$

$\wedge$

$\vee$

$\Rightarrow$

$\equiv$

Zero

$$false \Rightarrow P \equiv True$$

$$false \wedge P \equiv \underline{false}$$

$$true \vee P \equiv true$$

Predicate Logic: Quantifiers

- syntax
- base cases in programming

$$\forall i \bullet R(i) \Rightarrow P(i)$$

universal

Range

① i in the range, then $P(i)$ must be true

② i not in the range, then ~~don't care~~ zero of false $\Rightarrow P \equiv$ True

empty array means $R(i)$ is false
↳ false $\Rightarrow P \equiv$ T

$$\exists i \bullet R(i) \land P(i)$$

existential

empty array means $R(i)$ is false (universe of disclosure)
↳ false $\land P \equiv$ F

$\forall i \bullet P(i)$

$\exists i \bullet P(i)$

∵ no counter-example or witness of violation can be found.

universal property

```
boolean allPos(int[] a)
{
  if (a.length == 0) {
    return true;
  }
}
```

existential property

```
boolean somePos(int[] a)
{
  if (a.length == 0) {
    return false;
  }
}
```

∵ no witness of satisfaction can be found.

Rodin

$\forall x : R(x) \Rightarrow P(x)$

$\exists x : R(x) \wedge P(x)$

TLA+

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

$R(x)$ $\Rightarrow$ $\wedge$

$\mathbb{N}$ natural #'s

↳ $0, 1, 2, 3, \ldots +\infty$

$\mathbb{Z}$ integers

↳ $-\infty, \ldots, 0, \ldots +\infty$

Logical Quantifiers: Examples

$\forall i \bullet i \in \mathbb{N} \Rightarrow i \geq 0$

$\mathbb{N} = 0, 1, 2, \ldots$

all elements in range ≥ 0

$\forall i \bullet i \in \mathbb{Z} \Rightarrow i \geq 0$.

witness/counter-example : -2

$-2 \in \mathbb{Z} \Rightarrow -2 \geq 0 \equiv T \Rightarrow F \equiv F$

$\forall i, j \bullet i \in \mathbb{Z} \land j \in \mathbb{Z} \Rightarrow i < j \lor i > j$

witness: $i = 2$, $j = 2$

$\underbrace{2 \in \mathbb{Z}}_{T} \land \underbrace{2 \in \mathbb{Z}}_{T} \Rightarrow \underbrace{2 < 2}_{F} \lor \underbrace{2 > 2}_{F}$

$T \Rightarrow F \equiv F$

$\exists i \bullet \underbrace{i \in \mathbb{N}}_{0 \in \mathbb{N}} \land \underbrace{i \geq 0}_{0 \geq 0}$ $\quad T \land T \equiv T$

witness: 0

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

witness : 0.

$\exists i, j \bullet i \in \mathbb{Z} \land j \in \mathbb{Z} \land (i < j \lor i > j)$

$\hookrightarrow$ $i = 1$, $j = 3$

Lecture 5 - January 24

Math Review

Logical Quantifications: Proof Strategies Exercises

Announcement

- Lab1 Part 2 tutorial videos released

  + ≈ 2 hours

    * $\boxed{\text{debugging}}$ using labels, error trace, state graph
    * PlusCal vs. Auto-Translated TLA+ Predicates

- Optional Textbook for Model Checking and Program Verification

  Logic in Computer Science:

  Modelling and reasoning about systems

  by M. Huth and M. Ryan

Logical Quantifiers: Examples

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

harder (1) Assume $R(i)$, prove $P(i)$

(2) Prove $\neg R(i)$ ($\approx$ empty array)

e.g. $R(i) \triangleq i < 0 \wedge i > 10$ $\Rightarrow P(i)$

there's no two disjoint ranges satisfying i at the same time. F

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

(1) Find a witness s.t. $R(i) \wedge P(i)$

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

(1) Give a witness/counter-example : $R(i) \wedge \neg P(i)$

e.g. error trace

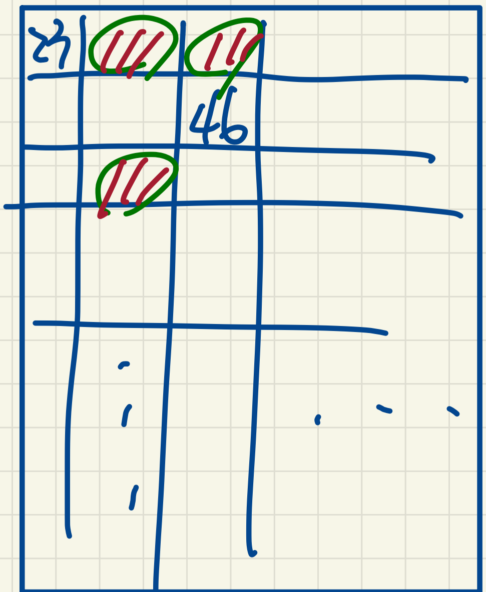
gives some context about i, might be easier to be proved s.t. $P(i)$

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

(1) Show $\neg R(i)$ ($\approx$ empty array)

harder (2) $R(i) \wedge \neg P(i) \rightsquigarrow$ for all i satisfying R, they don't satisfy P false

Sudoku



$\underbrace{M}_{\text{model}} \models \underbrace{\phi}_{\text{property}}$

$\models$ → satisfies

peg solitaire

$M \triangleq$ encoding (board)

defined as

∧

rules of game

$M \models$ no solution.

↳ failed → $\underbrace{\text{error trace}}_{\text{a way to lead to a state}}$ rep. the solution!

Prove/Disprove Logical Quantifications

- Prove or disprove: $\forall x \bullet \overbrace{\boxed{(x \in \mathbb{Z} \wedge 1 \leq x \leq 10)}}^{R} \Rightarrow \overbrace{x > 0}^{P}$.

R1 R2 R3 R4

$x \in 1..10$, each > 0

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

disprove: Counter-example: 1 $\quad \underline{1} \in \mathbb{Z} \wedge 1 \leq \underline{1} \leq 10 \Rightarrow \underline{1} > 1$

$T \Rightarrow F$

F.

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

witness : 2

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

max value is 10 but $10 > 10$ F.

Is the following statement correct:

To

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

We can just give a witness $x = 1$.

Not correct! $\because$ $\exists$ is true but

$x = 1$ not a valid witness

$(1 > 1 \equiv F)$.

$$\exists x \cdot \underbrace{x \in \mathbb{Z} \wedge 1 \le x \le 10 \Rightarrow x > 1}$$

$$\equiv \quad \exists x \cdot \text{True} \wedge (\qquad)$$

True $\wedge$ P

$\equiv$ P.

Axioms.

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

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

$\mathbb{N} = \{0, 1, \ldots, +\infty\}$

Predicate Logic: Exercise 1

Consider the following predicate:

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

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). $\boxed{5 \in \mathbb{N} \land 0 \in \mathbb{N}}$ T $\Rightarrow \boxed{5 * 0 > 0}$ F F
4. It is not a theorem, witnessed by (12, -2). correct inappropriate witness
5. It is not a theorem, witnessed by (12, 13). correct T

$12 \in \mathbb{N} \land \boxed{-2 \in \mathbb{N}} \Rightarrow 12 * -2 > 0 \equiv$ T F

$\boxed{12 \in \mathbb{N} \land 13 \in \mathbb{N} \Rightarrow 12 * 13 > 0}$

Consider the following predicate:

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

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

x : 0, 1, 2, ..., $+\infty$

y : 0, 1, 2, ..., $+\infty$

Case 1 $\quad x > 0, \; y > 0 \Rightarrow x * y > 0$

$\Rightarrow x * y \geq 0$

Case 2 $\quad x \geq 0, \; y \geq 0$ if one or both is 0

$\Rightarrow x * y = 0 \Rightarrow x * y \geq 0$

- An axiom is assumed to be true, with no need for proofs.
- A theorem is a Boolean expression that requires a proof.
  - ↳ lemma
    - ↳ sub-theorems to help.

Predicate Logic: Exercise 2

Consider the following predicate:

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

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 a theorem, provable by (–2, –3).
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).

T.

$5 \in \mathbb{N} \wedge 4 \in \mathbb{N}$

$\wedge\ 5 * 4 > 0$

not a valid witness:

$\underbrace{(-2 \in \mathbb{N} \wedge -3 \in \mathbb{N})}_{F} \wedge \underbrace{(-2 * -3 > 0)}_{T}$

F.

Logical Quantifications: Conversions

R(x): x ∈ 4315_class

P(x): x receives A+

$$\boxed{(\forall X \bullet R(X) \Rightarrow P(X))} \Leftrightarrow \neg(\exists X \bullet R(X) \land \neg P(X))$$

$\equiv$

Equational Style of Proof.

$\forall x \bullet \underbrace{\boxed{R(x) \Rightarrow P(x)}}_{Q(x)}$

$\equiv$ { Axiom: $\forall x \bullet Q(x) \equiv \neg(\exists x \bullet \neg Q(x))$ }

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

$\equiv$ { known: $p \Rightarrow q \equiv \neg p \lor q$ }

$\neg(\exists x \bullet \neg(\neg R(x) \lor P(x)))$

$\equiv$ { de Morgan: $\neg(p \lor q) \equiv \neg p \land \neg q$ }

$\neg(\exists x \bullet \neg\neg R(x) \land \neg P(x))$

$\equiv$ { double negation: $\neg\neg p \equiv p$ }

$\neg(\exists x \bullet R(x) \land \neg P(x))$

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

De Morgan

Lecture 6 - January 26

Model Checking

Introduction
Linear-time Temporal Logic (LTL): Syntax

Announcement

- Lab1 Part 2 tutorial videos released
  + Help: Scheduled Office Hours & flexible TA hours
  + ≈ 2 hours
    * debugging using labels, error trace, state graph
    * PlusCal vs. Auto-Translated TLA+ Predicates
- Optional Textbook for Model Checking and Program Verification

  Logic in Computer Science:

  Modelling and reasoning about systems

  by M. Huth and M. Ryan

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.*

checking at the machine-instruction level!

Formal Verification: Proof Based vs. Check Based

$\Gamma \vdash \phi$

property / formula

$\vdash$: derivable from left to right using the relevant inference rules (deduction)

Γ: system formula (before-after preds of actions, guards, invariants)

undecidable problem

Even if there exists a seq. of inference rule application between Γ and ϕ, a theorem prover cannot always find it.

1-9

$a^b \approx 0.5M$

$M \models \phi$

satisfies

ϕ: invariant, temporal properties (LTL, CTL)

M: system description (usually formulated as a LTS) a graph!

LTS: labeled transition system

automated

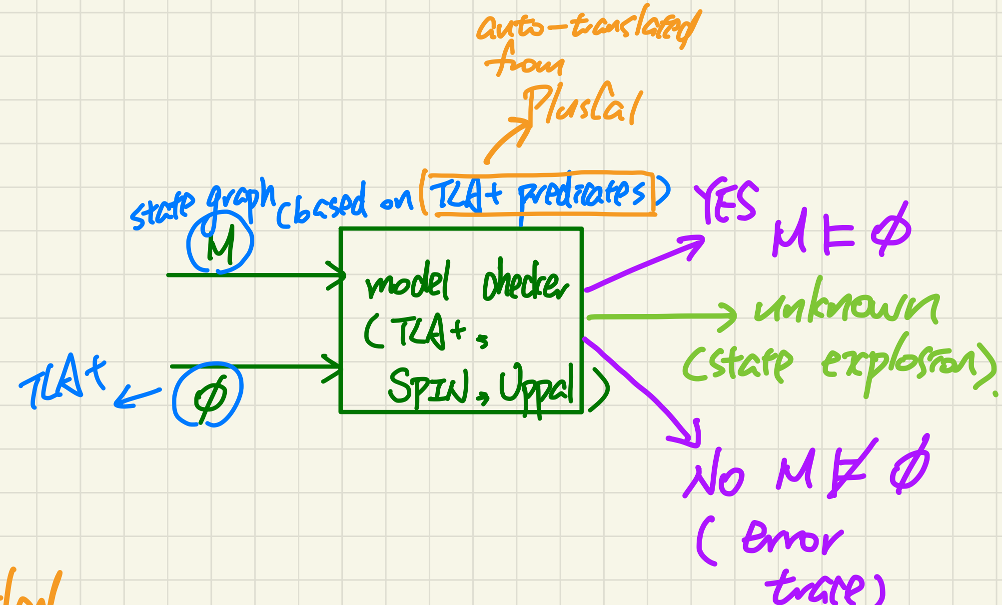
⇒ - build the graph
- traverse the graph to check.

Temporal Logic

- Syntax : structure

- Semantics : meaning

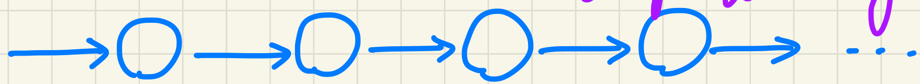
↳ (1) how to express
  (2) how to check
  (3) when the check failed, how to interpret the error trace

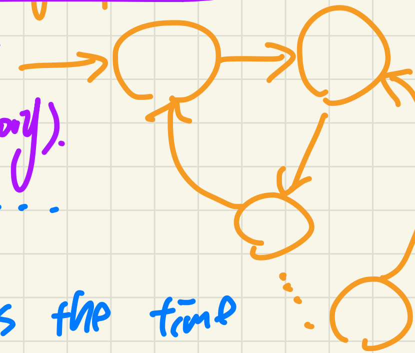


State graph (with a cycle).

(Computation) Path :

must be finite (and small enough to be fit to memory).





An infinite seq. of states that models the time

(model checking is natural for reactive system).

trace

LTL Syntax: Context-Free Grammar

$F\,\boxed{G\,\phi}$
$G\,\boxed{F\,\phi}$

ϕ	::=	$\top$	[*true*]
	|	$\bot$	[*false*]
	|	p	[propositional $\boxed{\text{atom}}$]
	|	$(\neg\phi)$	[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]

base cases.

propositional logic

atomic description (which itself does not contain any operators).

unary operators

eventually

binary operators

implicitly use $\forall$ or $\exists$.

$\phi = \underline{\underline{\phi}} \wedge \phi = p \wedge \underline{\underline{\phi}} = p \wedge X\underline{\underline{\phi}} = \boxed{p \wedge X\top}$ from the grammar.

valid LTL formula derivable

Operator Precedence

(1) $F\phi_1 \Rightarrow \phi_2$

$\hookrightarrow$ (a) $F(\phi_1 \Rightarrow \phi_2)$ (b) $(F\phi_1) \Rightarrow \phi_2$ ✓

not what (1) means

Precedence

X, F, G /* unary LTL op */

U, W, R /* binary LTL op */

$\neg$

$\wedge$

$\vee$

$\Rightarrow$

] logical op.

$\underline{X\, p \Rightarrow q}$

Letter	symbol
X	$\bigcirc$
F	$\Diamond$
G	$\Box$

$$\underline{X\,\phi} \equiv \bigcirc\,\phi$$

$$F G\,\phi \equiv \Diamond\Box\,\phi$$

Lecture 7 - January 31

Model Checking

Practical Knowledge about Parsing
Operator Precedence
Drawing Parse Trees
Left-Most Derivation (LMD)

Announcement

- Lab1 Part 2 tutorial videos released
  - + Help: Scheduled Office Hours & flexible TA hours
  - + ≈ 2 hours
    - * debugging using labels, error trace, state graph
    - * PlusCal vs. Auto-Translated TLA+ Predicates
- Optional Textbook for Model Checking and Program Verification

  Logic in Computer Science:

  Modelling and reasoning about systems

  by M. Huth and M. Ryan
- Written Test 1 approaching...

  ↳ WSC ① EECS login → lab computer

  ② PPY login → eClass

Parsing: Some Practical Knowledge

context-free grammar

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]

start variable

base cases/ terminals

non-terminals

↳ goal in derivation is to "get rid of" all non-terminals

the set of strings derivable from g

$F\,(p \Rightarrow q) \in \boxed{\mathcal{L}(g)}$

string of LTL formula

language

grammar of LTL

no complete
in-between

$p\ F\,(\neg q) \notin \mathcal{L}(g)$

$p\ U\ F\,\neg q \in \mathcal{L}(g)$

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

Interpreting a Formula: Parse Trees (1)

top down: root $\to$ leaves

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

root.

p, q, r

$c > 0$, $p^2 > 4$

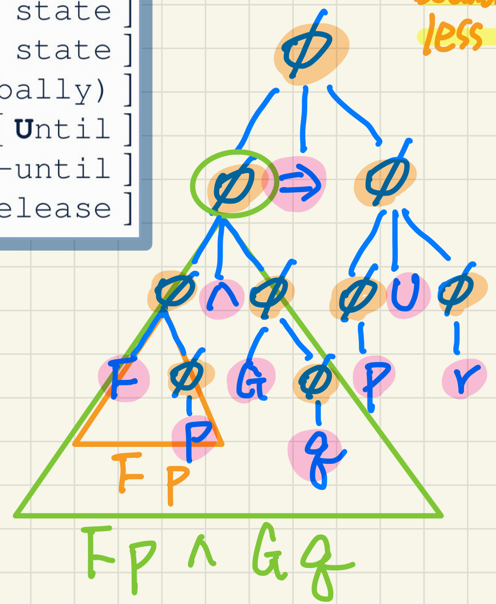
not = a keyword !

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

logical — bounds less tightly

temporal op

atom



Interpreting a Formula: Parse Trees (1)

top down:
root → leaves

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]
	|	$(\mathbf{X}\,\phi)$	[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]

root.

p, q, r
$c > 0,\ p^2 > 4$

not = a keyword!

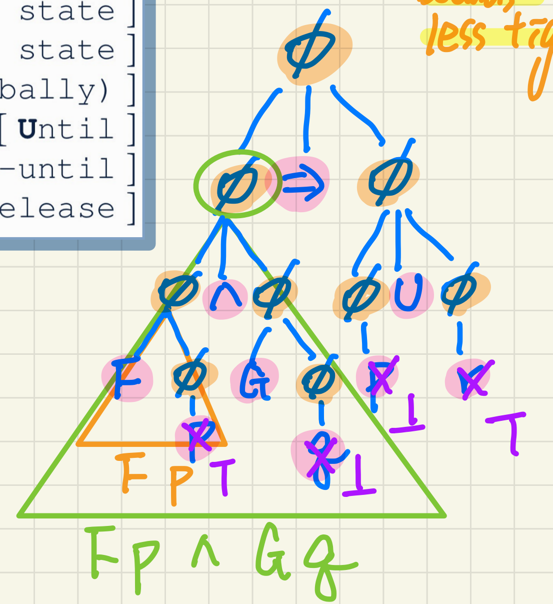
$$\boxed{\boxed{\mathbf{F}\ \cancel{p}} \wedge \boxed{\mathbf{G}\ \cancel{q}}} \Rightarrow \boxed{\boxed{\cancel{p}\ \mathbf{U}\ \cancel{r}}}$$

$\top$ $\bot$ $\bot$ $\top$

logical ⇒ binds less tightly

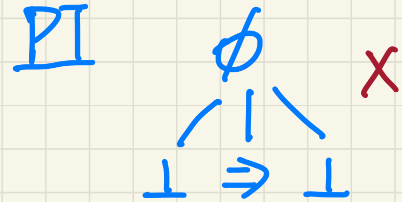
temporal op

atom



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

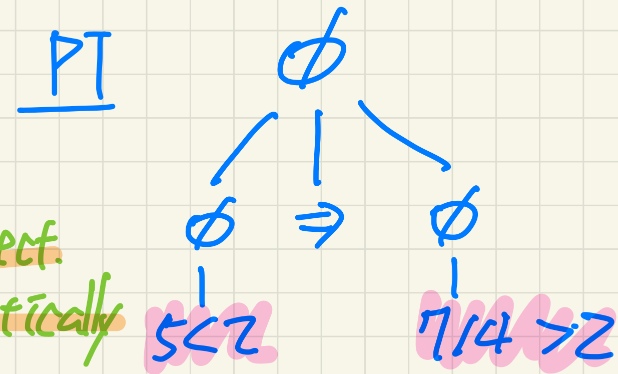
$5 < 2 \Rightarrow 7/4 > 2$



(no evaluation should be done)

- syntax
- $\boxed{\text{semantics}}$

↳ only makes sense if the formula is correct syntactically

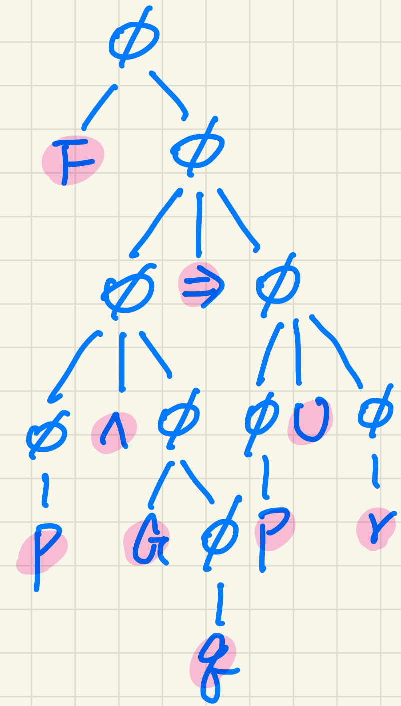


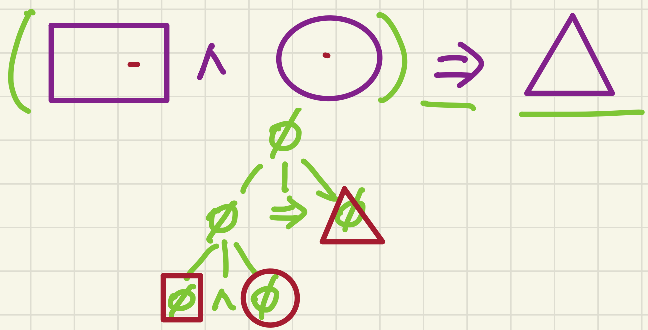
Interpreting a Formula: Parse Trees (2)

Q. $p \Rightarrow q \Rightarrow r$?

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]

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





$p \wedge q \equiv q \wedge p$ but different PTs.

Given two formula strings f1 and f2

different spellings.

(1) If $\boxed{f1 \neq f2}$, but f1 and f2 have the same parse tree, f1 and f2 are considered as semantically equivalent.

$F p \wedge G q \Rightarrow p \, U \, r$

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

① part of the input string to force some order of interpretation.

② parentheses are omitted in PTs.

(optional) (2) If f1 = f2, but f1 and f2 have different PTs, this means the grammar is ambiguous.

Interpreting a Formula: Parse Trees (3)

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

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

Interpreting a Formula: Parse Trees (4)

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

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

is derived to $\Rightarrow$

ϕ → left-most non-terminal

$\phi \Rightarrow \phi$

implication

left-most non-terminal

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

$\Rightarrow F\,\phi \wedge \phi \Rightarrow \phi$

$\Rightarrow F\,p \wedge \phi \Rightarrow \phi$

(to be continued ...).

Lecture 8 - February 2

Model Checking

Comparison: Parse Trees, LMDs, RMDs
Deriving Subformulas
Labelled Transition System (LTS)

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]

each step of derivation is based on a rule in the grammar

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

is derived to

$\Rightarrow$

ϕ → left-most non-terminal

$\phi \Rightarrow \phi$

left-most non-terminal

implication

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

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

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

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

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

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

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

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

when there's no non-terminal ⇒ done!

Interpreting a Formula: LMD (2)

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

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

Interpreting a Formula: LMD (3)

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

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

Interpreting a Formula: LMD (4)

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

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

Interpreting a Formula: RMD (1)

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

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

Interpreting a Formula: RMD (2)

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

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

Interpreting a Formula: RMD (3)

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

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

Interpreting a Formula: RMD (4)

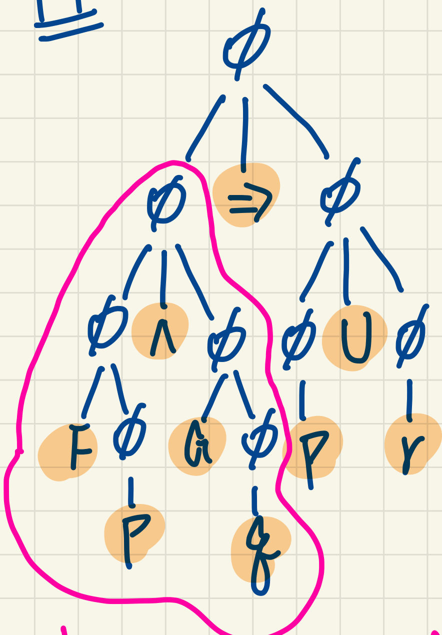
ϕ	$::=$	$\top$	[*true*]
	$\mid$	$\bot$	[*false*]
	$\mid$	p	[propositional atom]
	$\mid$	$(\neg\phi)$	[logical negation]
	$\mid$	$(\phi \wedge \phi)$	[logical conjunction]
	$\mid$	$(\phi \vee \phi)$	[logical disjunction]
	$\mid$	$(\phi \Rightarrow \phi)$	[logical implication]
	$\mid$	$(\mathbf{X}\,\phi)$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

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

Interpreting a Formula: PT vs. LMD vs. RMD

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

PT



subtree: $F\,p \wedge G\,q$

LMD

ϕ

① $\Rightarrow \phi \Rightarrow \phi$

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

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

④ $\Rightarrow F\,p \wedge \phi \Rightarrow \phi$

⑤ $\Rightarrow F\,p \wedge G\,\phi \Rightarrow \phi$

⑥ $\Rightarrow F\,p \wedge G\,q \Rightarrow \phi$

⑦ $\Rightarrow F\,p \wedge G\,q \Rightarrow \phi\ U\ \phi$

⑧ $\Rightarrow F\,p \wedge G\,q \Rightarrow p\ U\ \phi$

⑨ $\Rightarrow F\,p \wedge G\,q \Rightarrow p\ U\ r$

RMD

ϕ

$\Rightarrow \phi \Rightarrow \phi$

$\Rightarrow \phi \Rightarrow \phi\ U\ \phi$

$\Rightarrow \phi \Rightarrow \phi\ U\ r$

$\Rightarrow \phi \Rightarrow p\ U\ r$

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

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

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

$\Rightarrow F\,\phi \wedge G\,q \Rightarrow p\ U\ r$

$\Rightarrow F\,p \wedge G\,q \Rightarrow p\ U\ r$

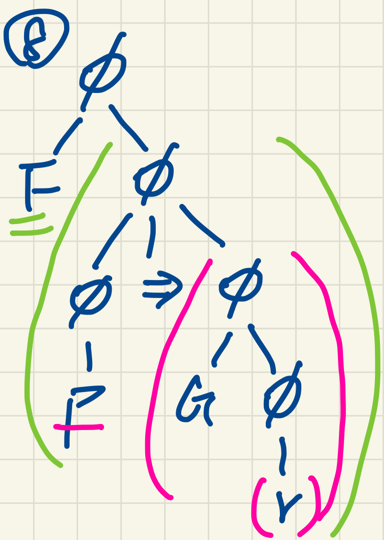
Deriving Subformulas from a Parse Tree

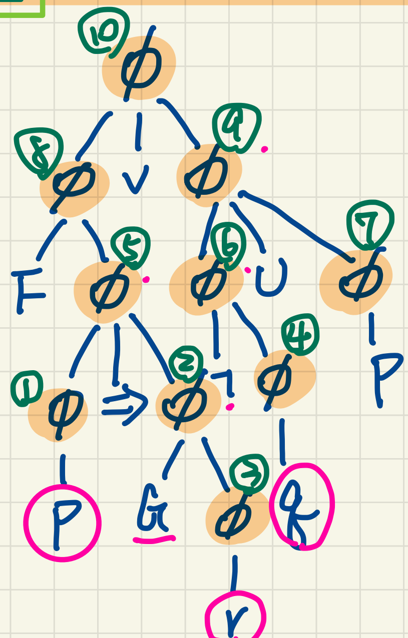
Instead, bracket strings obtained from subtrees.

* and ** are not the same
∵ in ** F is applied last
in * F is applied first.

Enumerate all subformulas of:

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





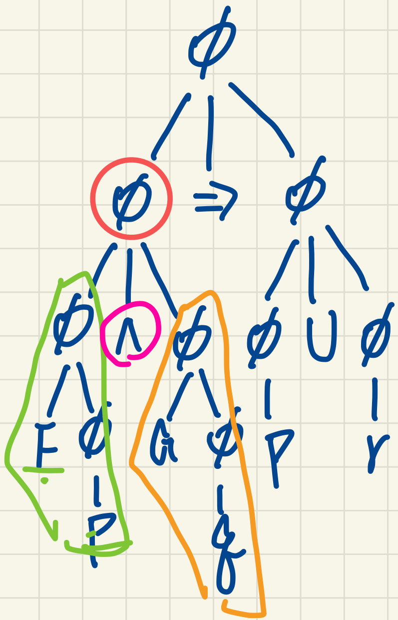
Q1: How many subformulas?
↳ Count how many ϕ's.

10

Q2. Enumerate all subformulas.

① p
② $G\,r$
③ r
④ q
⑤ $p \Rightarrow G\,r$
⑥ $\neg q$
⑦ p *
⑧ $\boxed{F\,p \Rightarrow G\,r}$ $F(p \Rightarrow (G\,r))$
⑨ $(\neg q)\ U\ p$
⑩ $F(p \Rightarrow (G\,r)) \vee ((\neg q)\ U\ p)$

Given a PT:



Enumerate all subformulas:

$(F\ (p)) \wedge (G\ (q))$

Context-Free Grammar (CFG): Exercise

(optional)

dangling else

Is the following CFG ambiguous?

$$
\begin{array}{lcl}
\textit{Statement} & \rightarrow & \texttt{if}\ \textit{Expr}\ \texttt{then}\ \textit{Statement} \\
& | & \texttt{if}\ \textit{Expr}\ \texttt{then}\ \textit{Statement}\ \texttt{else}\ \textit{Statement} \\
& | & \textit{Assignment} \\
& \ldots &
\end{array}
$$

Example:

if *Expr1* **then if** *Expr2* **then** *Assignment1* **else** *Assignment2*

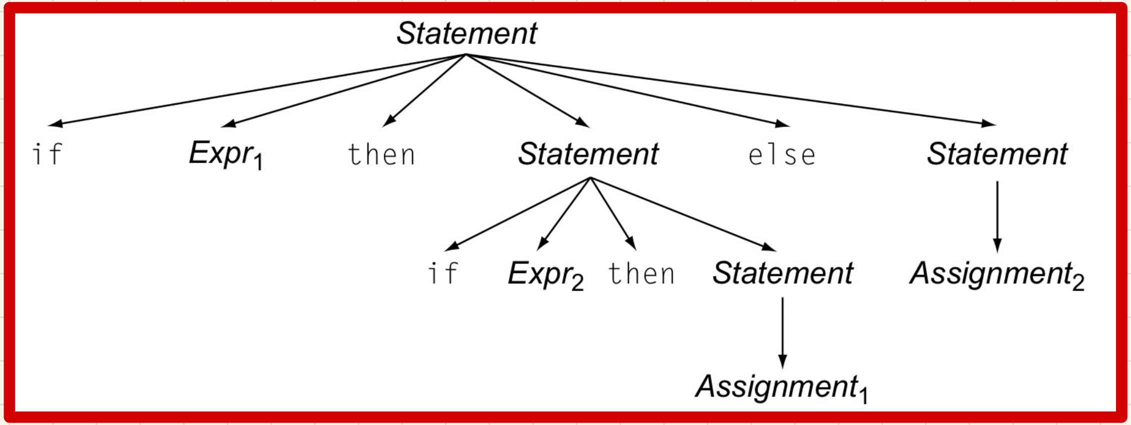
Context-Free Grammar (CFG): Exercise

Is the following CFG ambiguous?

Statement → `if` *Expr* `then` *Statement*
  | `if` *Expr* `then` *Statement* `else` *Statement*
  | *Assignment*
  . . .

Example: A Possible **Semantic Interpretation**?

if *Expr1* **then** **if** *Expr2* **then** *Assignment1* **else** *Assignment2*



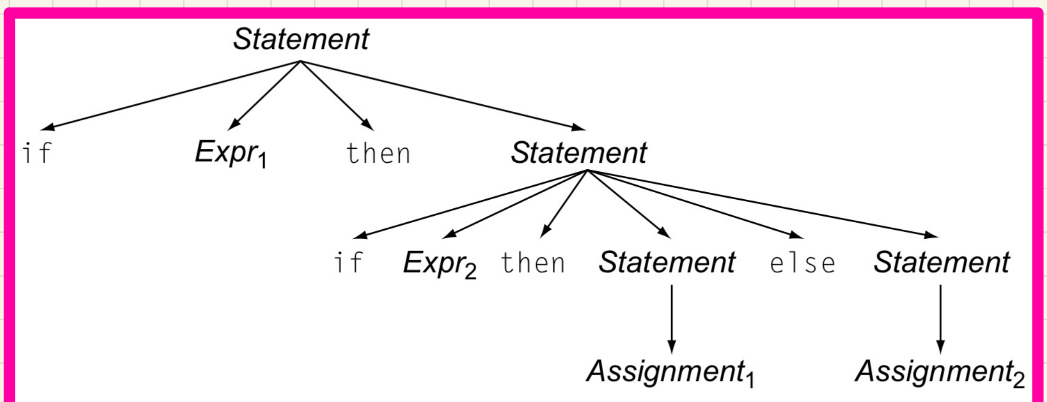
Context-Free Grammar (CFG): Exercise

Is the following CFG ambiguous?

```
Statement  →  if Expr then Statement
           |  if Expr then Statement else Statement
           |  Assignment
              ...
```

Example: A Possible **Semantic Interpretation**?

if *Expr1* **then** **if** *Expr2* **then** *Assignment1* **else** *Assignment2*



Labelled Transition System (LTS)

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

labelling function $L \in S \to \mathbb{P}(P)$ — a set of atoms that are satisfied by the state

~~$L \in S \to P$~~ — given a state, return a member in P

S: a finite set of states ↳ values of variables

$\longrightarrow$: transition relation ↳ set of pairs.

$\longrightarrow \in \boxed{S \leftrightarrow S}$ the set of all relations on states.

P: a set of atomic propositions (which evaluate to T or F)

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

S_0: $x=2$ $\quad S_1$: $x=-1$ $\quad S_2$: $x=6$

$L(S_0) = \{x>0\}$
$L(S_1) = \{\ \}$
$L(S_2) = \{x>0, x>4\}$

Q. Formulate **deadlock freedom**:

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

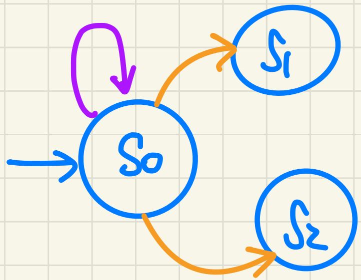
$s \to s_1$

$$\forall \underline{s} \cdot s \in S \Rightarrow (\exists s' \cdot s' \in S \wedge \boxed{(s, s') \in \longrightarrow})$$

~~$L(S) \neq \emptyset$~~

$\to \in S \leftrightarrow S$

$\to \in S \to S$



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

(s_0, s_0)

<u>not</u> a function, a relation!

Lecture 9 - February 7

Model Checking

Examples: LTS Formulation
Paths, Unwinding All Possible Paths
Path Satisfaction: X, G, F

Announcements

- Lab2 released
- WrittenTest1 coming

  ↳ cover until and including today

  + some left-over examples

  (to be finished within first 20 min on Thursday).

Labelled Transition System (LTS)

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

labelling function $L \in S \to \mathbb{P}(P)$

a set of atoms that are satisfied by the state

~~$L \in S \to P$~~

given a state, return a member in P

S: a finite set of states

↳ values of variables

$\longrightarrow$: transition relation

set of pairs.

$\longrightarrow \in \boxed{S \leftrightarrow S}$

the set of all relations on states.

P: a set of atomic propositions (which evaluate to T or F)

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

s_0: $x=2$ s_1: $x=-1$ s_2: $x=6$

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

$L(s_1) = \{\ \}$

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

Q. Formulate **deadlock freedom**:

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

$s. \to s_1.$

$$\forall \underline{s} \cdot s \in S \Rightarrow (\exists s' \cdot s' \in S \land \boxed{(s, s') \in \longrightarrow})$$

~~$L(s) \neq \emptyset$~~

Labelled Transition System (LTS)

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

$$0 \le c \le 3$$

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: $\{\boxed{c \ge 1}, \boxed{c \le 1}\}$

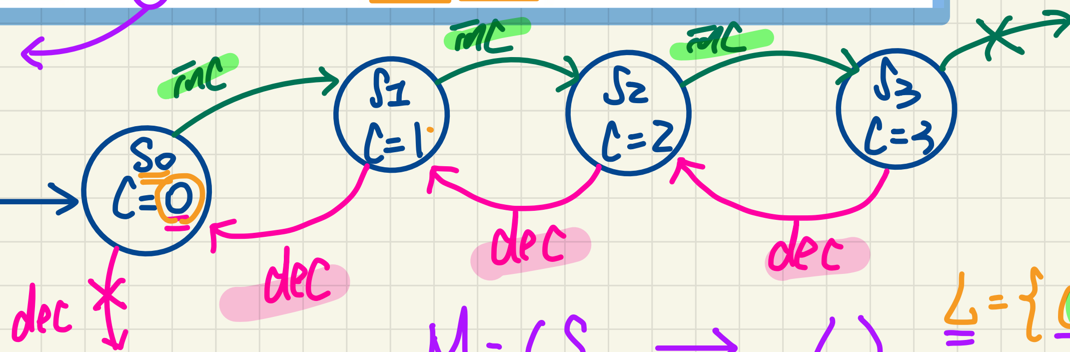
Exercise

$0 < c_1 \le 2$ $\quad inc_1$, dec_1

$3 \le c_2 \le 5$ $\quad inc_2$, dec_2

init: $c_1 = 1$, $c_2 = 3$

properties that we're interested in verifying.



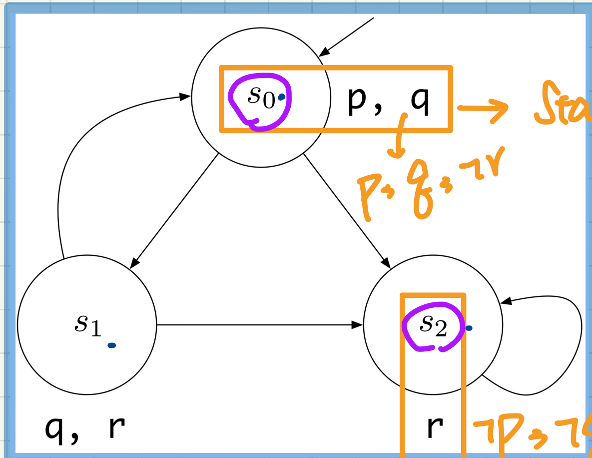
$S = \{s_0, s_1, s_2, s_3\}$

$\longrightarrow = \{(s_0, s_1), (s_1, s_2), (s_2, s_3), (s_3, s_2), (s_2, s_1), (s_1, s_0)\}$

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

$L = \{(s_0, \{c \le 1\}), (s_1, \{c \ge 1, c \le 1\}), (s_2, \{c \ge 1\}), (s_3, \{c \ge 1\})\}$

Labelled Transition System (LTS): Formulation & Paths



$p, q, \neg r$

$\neg p, \neg q$

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

State s_0 satisfies p and q

(implicitly, r is not satisfied)

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

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

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

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

state names

Path: $\pi = s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$

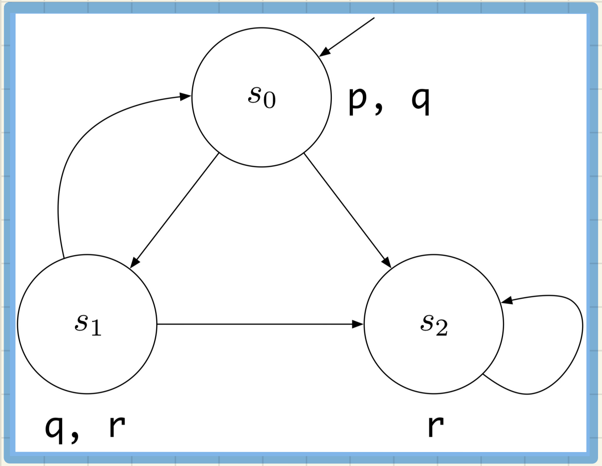
$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow \cdots$

indices

1st state in path

2nd state in path

3rd state in path



$$\pi^3 = s_0 \to s_1 \to s_0 \to s_1 \to \cdots$$

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

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

$$= \pi^4$$

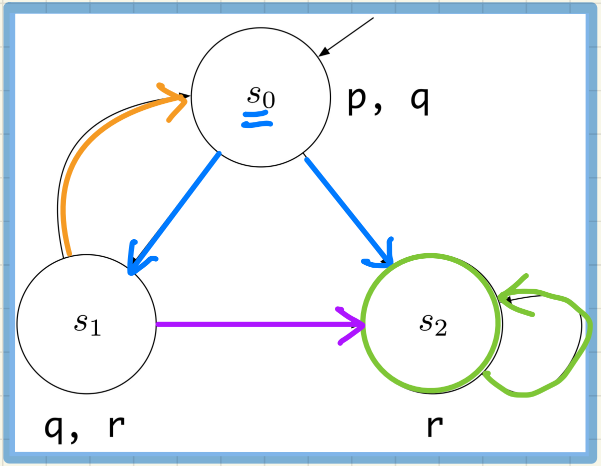
$$\pi = s_0 \to s_1 \to s_0 \to s_1 \to s_0 \to s_1 \to \cdots$$

Pattern: s_1 s_2 s_3 s_4 s_5 s_6

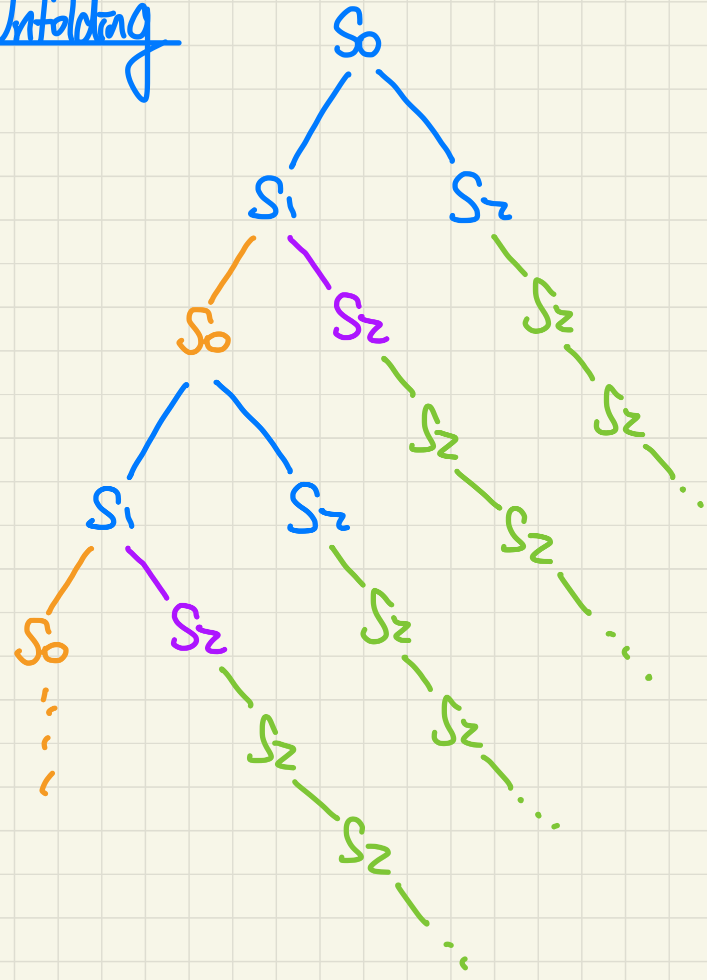
π^0

$$\pi^1 = \pi$$

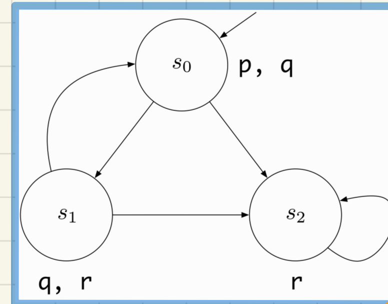
$$\pi^2 = s_1 \to s_0 \to s_1 \to s_0 \to s_1 \to \cdots$$



Unfolding



Path Satisfaction: Logical Operations



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

$s_1 \to s_2 \to \cdots \to s_{i-1} \to s_i \to s_{i+1} \to \cdots$

first state in π

e.g. $\pi = s_0 \to s_2 \to s_2 \to \cdots$

1st state

$\pi \models p$

$\pi \not\models r$

$\pi \models p \iff p \in L(s_1)$ 1st state in path

$\pi \models \top$ ✓ 1st state satisfies $\top$. labelling function

$\pi \not\models \bot \iff \neg(\pi \models \bot)$

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

$\pi \models \phi 1 \wedge \phi 2 \iff \pi \models \phi_1 \wedge \pi \models \phi_2$

$\pi \models \phi 1 \vee \phi 2$

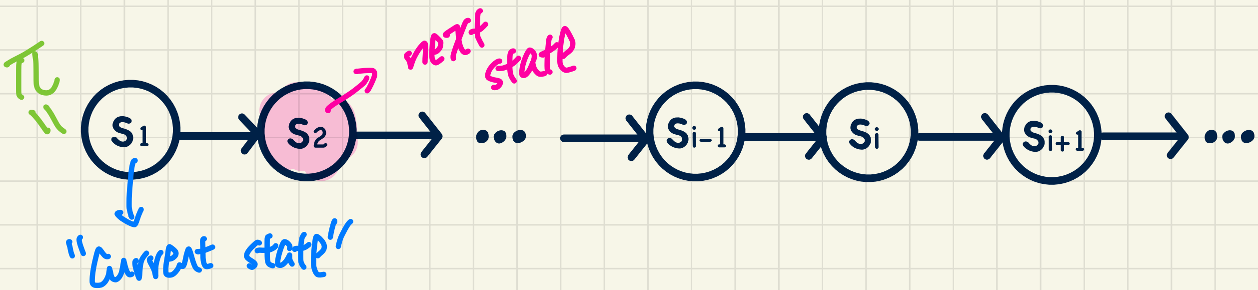
$\pi \models \phi 1 \Rightarrow \phi 2$

[redacted]

Path Satisfaction: Temporal Operations (1)

A **path** satisfies $X\phi$

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



Formulation (over a path)

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

*

π^3

Q. What is $\pi 3 \models X\ p$ checking?

Path Satisfaction: Temporal Operations (2)

Global

A **path** satisfies $G\phi$
if the every state satisfies it.

Formulation (over a path)

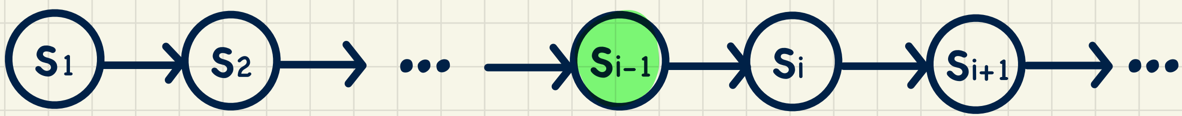
$$\pi \models G\phi \iff \forall i \cdot i \geq 1 \Rightarrow \boxed{\pi^{i} \models \phi}$$

Path Satisfaction: Temporal Operations (3)

Future

A path satisfies $F\phi$

if some future state satisfies it.



Formulation (over a path)

$$\pi \models F\phi \iff \exists i.\ i \geq 1 \wedge \boxed{\pi^i \models \phi}$$

Lecture 10 - February 9

Model Checking

Path Satisfaction vs. Model Satisfaction
Unary Temporal Operators: X, G, F

Announcements

- Lab1 solution coming soon!
- Lab2 released
- WrittenTest1 guide & example questions released
  + Verify EECS account on a WSC machine
  + Verify PPY account and Duo Mobile on eClass
- Review session on Monday? 1pm or 2pm?

  Zoom!

Satisfaction relations

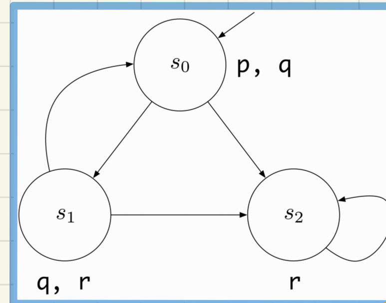
(1) $\underbrace{\pi}_{\text{path}} \models \phi$

(2) $\underbrace{s}_{\text{state}} , \underbrace{M}_{\text{model}} \models \phi$

need to consider **all** π starting from state s.

Path Satisfaction: Logical Operations

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



first state in π

$s_1 \to s_2 \to \cdots \to s_{i-1} \to s_i \to s_{i+1} \to \cdots$

e.g. $\pi = s_0 \to s_2 \to s_2 \to \cdots$ (1st state)

$\pi \models p$

$\pi \not\models r$

$\pi \models p \iff p \in L(s_1)$ 1st state in path

$\pi \models \top$ ✓ 1st state satisfies $\top$ labelling function

$\pi \not\models \bot \iff \neg(\pi \models \bot)$

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

$\pi \models \phi 1 \wedge \phi 2 \iff \pi \models \phi_1 \wedge \pi \models \phi_2$

$\pi \models \phi 1 \vee \phi 2$

$\pi \models \phi 1 \Rightarrow \phi 2$

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

$\boxed{S_2 \text{ satisfies } p}$ $\pi^2 \models p$

$\boxed{S_4 \text{ satisfies } p}$ $\pi^4 \models p$

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

Q: Express that <u>all</u> the even-numbered states satisfies a proposition p.

Path Satisfaction: Temporal Operations (1)

A **path** satisfies $X\phi$
if the **next state** (of the "current state") satisfies it.

$\pi =$ $s_1 \to s_2 \to \cdots \to s_{i-1} \to s_i \to s_{i+1} \to \cdots$

next state

"current state"

Formulation (over a path)

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

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

π^3

$(\pi^3)^2$

$$\boxed{\pi^3 \models X p}$$

$$\iff (\pi^3)^2 \models p$$

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

*

π^3

Q. What is π3 ⊨ X p checking?

Model Satisfaction

Given:

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

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

$s \vDash \phi$

Formulation (over all paths)

model satisfaction

$\pi = s \rightarrow \cdots$ (a valid path w.r.t. M)

$$s \vDash \phi \iff \forall \pi \cdot \boxed{\pi \text{ starts with } s} \Rightarrow \boxed{\pi \vDash \phi}$$

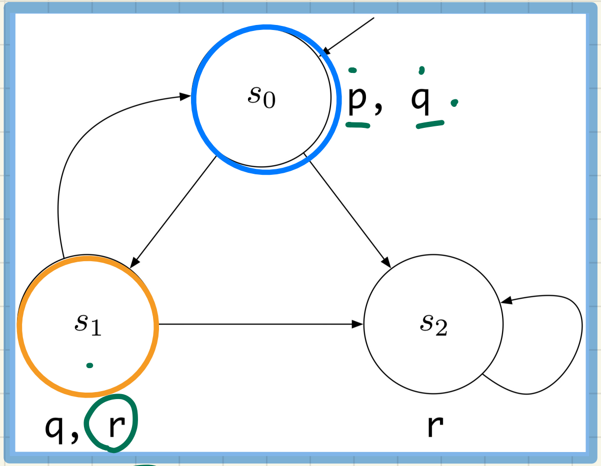
↳ path satisfaction

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

(1) To prove $s \vDash \phi$, need to show for every possible path π, $\pi \vDash \phi$

(2) To disprove $s \vDash \phi$, provide a witness $\pi = s \rightarrow \cdots$, $\pi \nvDash \phi$.

Model vs. Path Satisfaction: Exercises (1.1)



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

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

$\pi \models \top$ T

$\pi \not\models \bot$ T

$\pi \models p \land q$ T

$\pi \models p \lor q$ T

$\pi \models p \Rightarrow q$ T

$\pi \models r$ F

$\pi \models r \Rightarrow p \land q \land r$ T

1st state s_0 TS

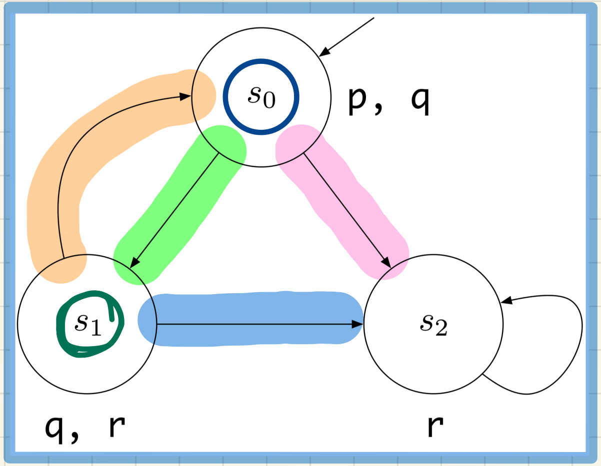
$\pi^2 \models p \Rightarrow q$ T

$\pi^2 \models r$ T

$\pi^2 \models r \Rightarrow p \land q \land r$ F

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

Model vs. Path Satisfaction: Exercises (1.2)



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

$s_0 \models \top$ T

$s_0 \not\models \bot$ T

$s_0 \models p \land q$

$s_0 \models p \lor q$

$s_0 \models p \Rightarrow q$

$s_0 \models r$

$s_0 \models r \Rightarrow p \land q \land r$

(1) all possible paths starting from s_0 has s_0 as the first state

(2) $\pi \models p \Leftrightarrow$

$p \in L(\underline{\underline{s_1}})$ s_0

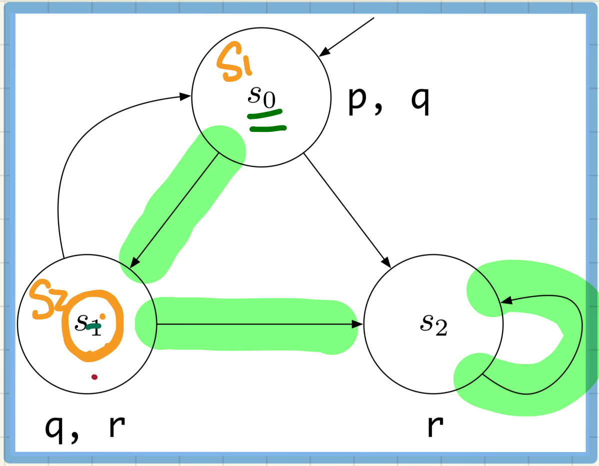
$s_1 \models \overset{F}{p} \Rightarrow q$ T

$s_1 \models r$ T

$s_1 \models \underset{T}{r} \Rightarrow \overset{F}{p} \land q \land r$ F

Exercise: What if we change the LHS to s_1?

Model vs. Path Satisfaction: Exercises (2.1)



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

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

2nd state.

$\pi \models X\,\top \Leftrightarrow \pi^2 \models \top$ T

$\pi \not\models X\,\bot$ T

$\pi \models X\,(q \land r) \Leftrightarrow \pi^2 \models q \land r$ T

$\pi \models X\,q \land r$ F

$\pi \models X q \land \pi \models r$ F

T $\pi^2 \models q$

$\therefore$ s_0 doesn't satisfy r

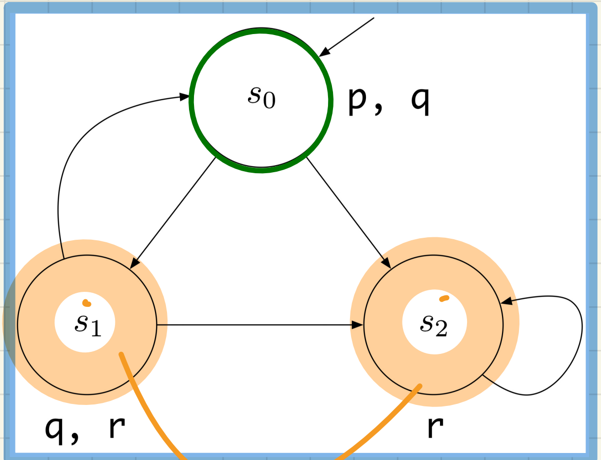
$\pi \models X\,(q \Rightarrow r) \Leftrightarrow \pi^2 \models q \Rightarrow r$ T.

$\pi \models X\,q \Rightarrow r \Leftrightarrow \underbrace{\pi^2 \models q}_{T} \Rightarrow \underbrace{\pi \models r}_{F}$

F

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

Model vs. Path Satisfaction: Exercises (2.2)



possible next states for paths starting from s_0

$s \vDash \phi \Leftrightarrow$ **all** π starting at s, $\pi \vDash \phi$

need to consider all paths starting from s_0

possible next states from s_0?

$s_0 \vDash \mathbf{X}\ \top$ (T)

$s_0 \nvDash \mathbf{X}\ \bot$ (T)

$s_0 \vDash \mathbf{X}\ (q \wedge r)$ (F) Witness: $s_0 \to s_2 \to \ldots$

$s_0 \vDash \mathbf{X}\ q \wedge r$ (F) Witness: $s_0 \to s_1 \to \ldots$ not satisfying r

$s_0 \vDash \mathbf{X}\ (q \Rightarrow r)$ (T)

$s_0 \vDash \mathbf{X}\ q \Rightarrow r$ (F) Witness: $s_0 \to s_1 \to \ldots$

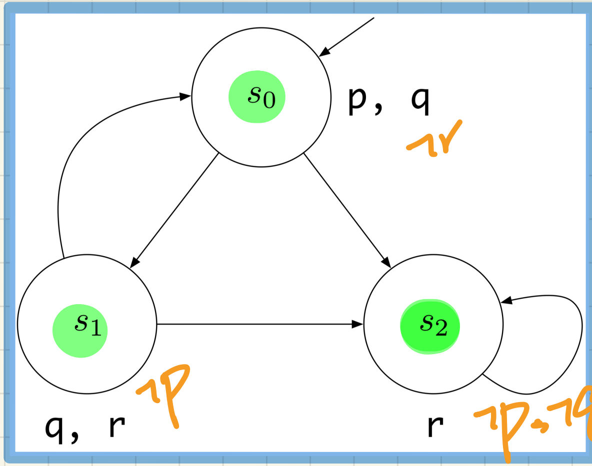
$T \Rightarrow F \equiv F$

r is (F) q is (T)

try!

Exercise: What if we change the LHS to s_1?

Model vs. Path Satisfaction: Exercises (3.1)



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

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

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

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

$\pi \models \mathbf{G}\,\neg(p \wedge r) \rightarrow \neg p \vee \neg r$ T

$\pi \models \mathbf{G}\,r$ F $s_0 \rightarrow s1 \rightarrow \ldots$ ¬r

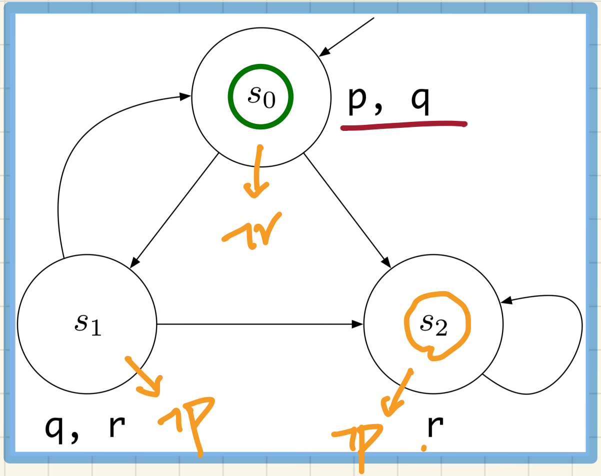
$\boxed{\pi^2 \models \mathbf{G}\,r} \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots \models \mathbf{G}\,r$

T

To disprove path satisfaction, give a witness state.

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, $\pi \models \phi$

✓

$s_0 \models G\ \top$ T

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

$s_0 \models G\ \neg(p \land r)$ $\equiv \neg p \lor \neg r$ T → all paths starting from s_0 over all states

$s_0 \models G\ r$ F

$s_2 \models G\ r$ T

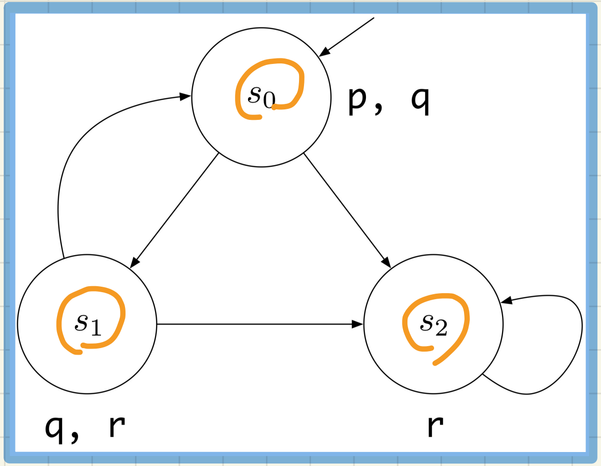
witness:

$s_0 \rightarrow \cdots$

¬r

Exercise: What if we change the LHS to s_1?

Model vs. Path Satisfaction: Exercises (4.1)



$\pi \models \mathbf{F}\ \phi \iff \exists i \bullet i \geq 1 \land \pi^i \models \phi$

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

$\pi \models \mathbf{F}\ \top$: T   T

$\pi \not\models \mathbf{F}\ \bot$: T   T

$\pi \models \mathbf{F}\ \neg(p \land r)$ T

all states in π actually satisfies $\neg p \lor \neg r$

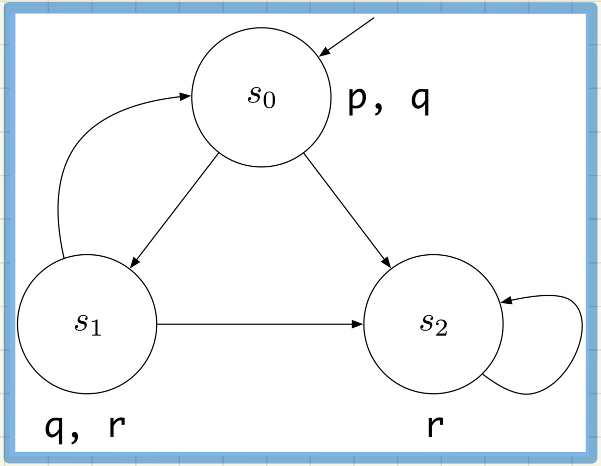
$\pi \models \mathbf{F}\ r$

$\pi \models \mathbf{F}\ (q \land r)$ T

witness: s_1 witness: s_1

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

Model vs. Path Satisfaction: Exercises (4.2)



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

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

$s_0 \not\models \mathbf{F}\ \perp$ T

$s_0 \models \mathbf{F}\ \boxed{\neg(p \wedge r)}$ T ∵ every state satisfies $\neg p \vee \neg r$.

$s_0 \models \mathbf{F}\ r$ T

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

Witness: $s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$ ($q \wedge r$ never satisfied)

$S \models F\ \phi$

↳ for each path starting from s, ($\pi = s \rightarrow \ldots$) there's one state satisfying ϕ.

Exercise: What if we change the LHS to s_1?

Sunday, February 12

Written Test 1 Review

$\models$

- Consider the following check (where `s` is a state of some model `M`, and `phi` is a syntactically-correct LTL formula):

state s, M |= F phi

In order to show that the above model satisfaction relation does **not** hold, we need to show that for every path pi (i.e., a witness) of M, phi does not hold at any state in pi.

disproving

E

$\pi = s \rightarrow \cdots$

$\forall$ path • path starts with state s $\Rightarrow$

$\pi \models F\,\phi$

path

to disprove this, need a witness path

P

$\exists i \cdot i \in \mathbb{N} \wedge i \geq 1 \wedge \pi^i \models \phi$

for the witness path chosen, all states do not satisfy ϕ

Prove vs. Disprove model satisfaction of G.

$s, M \models G\,\phi$

$\forall \pi \cdot \pi = s \to \cdots \Rightarrow$

$\boxed{\pi \models G\,\phi}$

↳ $\forall \underline{i} \cdot i \in \mathbb{N} \wedge i \geq 1 \Rightarrow$

$\pi^i \models \phi$

$\pi : s \to \cdots \to \boxed{s_i \to \cdots}$

$\pi^i \not\models \phi$ e.g. p

e.g. $X\phi$

In general,

ϕ may be just a prop. atom, or

ϕ can be complicated, including temporal operators.

to disprove:

find a witness i s.t.

$i \in \mathbb{N} \wedge i \geq 1 \wedge \boxed{\pi^i \not\models \phi}$

WTI.

Prove vs. Disprove model, path sat.

you'll only be given options to choose from.

e.g. $s_2 \models X\,p$

↳ to disprove, give a path

$\pi = s_2 \to s$

$p \notin L(s)$

$\mathbf{F}\ \boxed{p} \land (\boxed{\mathbf{G}\ \boxed{q}\ =>}\ \mathbf{U}\ \boxed{r})$

Consequence is missing!

Can this be the LHS of U op?

NO!

WTI: stick to letters of temporal operators.

$\boxed{\mathbf{F}\ \boxed{p}} \land (\boxed{\mathbf{G}\ \boxed{q}}\ => \boxed{\boxed{r}\ \mathbf{U}\ \boxed{s}})$

an operator with lower precedence than U

alternatively:

$(\underline{G\,q \Rightarrow r}\ U\ s)$

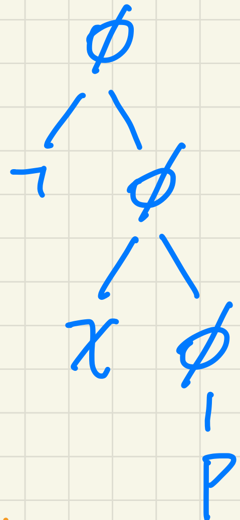
not right: $\Rightarrow$ has lower precedence than U.

Operator Precedence

Unary temporal	$\underline{X}$, F, G
Binary temporal	U, W, R
Unary Prop.	$\neg$
	$\land$
	$\lor$
	$\Rightarrow$

Assume: questions will not req. a decision on the associativity of $\Rightarrow$, X.

$\neg \boxed{\chi \boxed{p}}$



$\dot{\vee}$ $\dot{U}$

Q. Consider:

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

$$\begin{aligned} \phi &\Rightarrow \phi \; U \; \phi \\ &\Rightarrow p \; U \; (\phi \wedge \phi) \\ &\Rightarrow p \; U \; (\phi \wedge r) \\ &\Rightarrow p \; U \; (q \wedge r) \end{aligned}$$

not LM

Is this a valid LMD?

parse trees

LMD

RMD

- drag and drop lines of derivation steps in correct order

-

make solution available.

Lecture 11 - February 28

Model Checking

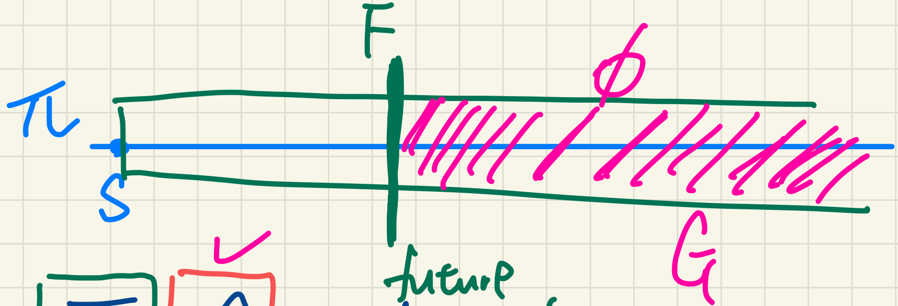
Path Satisfaction: Nested LTL Operators
FG vs. F => FG

Announcements

- Released: WrittenTest1, Lab2 solution
- To be released:
  + ProgTest1 Guide (by the end of Wednesday)
  + ProgTest1 practice questions (by Thursday class)

progtest1

- 1 ~ 2 algorithms
- conditionals, loops, tuples
- assertions (postcondition)



future point

$s \models FG\phi$

some state

model satisfaction
(consider all
paths starting with s).

arbitrary
LTL formula

$\forall x \cdot P(x)$

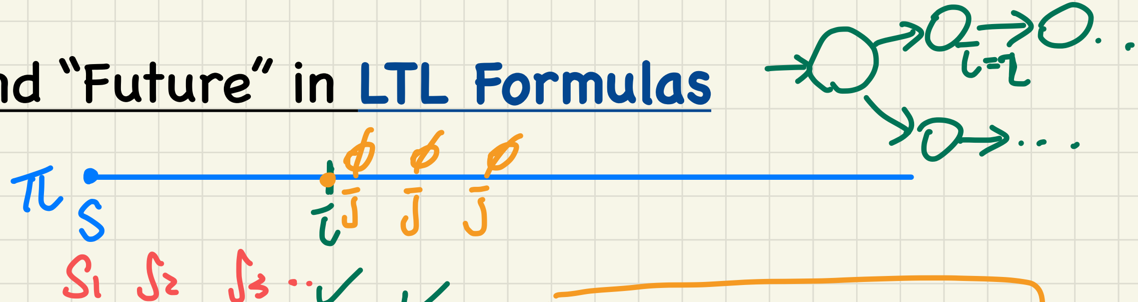
disprove:
- find an x s.t. $\neg P(x)$

$\exists x \cdot P(x)$

disprove:
- find all possible x s.t. $\neg P(x)$

Nesting "Global" and "Future" in LTL Formulas

$$s \models \mathbf{FG}\,\phi$$



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

Q. Formulate the above nested pattern of LTL operator.

specific to the π being considered

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

$(**\exists i \bullet i \geq 1 \wedge (\forall j \bullet j \geq i \Rightarrow (\pi^j \models \phi)))$

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

*① consider all path patterns starting from s

**② find such i ***③ each state subsequent to ith state (including ith state) satisfies ϕ

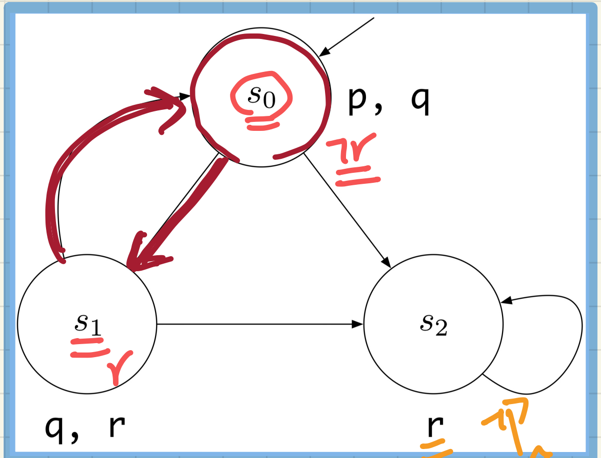
Q. How to disprove the above nested pattern of LTL operators?

*① Find a witness $\pi = s \rightarrow \cdots$

**② Show that for each state in π.

***③ there's one subsequent state that violates ϕ.

Path Satisfaction: Exercises (5.1)



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

$s_0 \models \mathbf{FG}\ r$ false

Witness: $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$

violates r, cannot be ignored in path

$s_0 \models \mathbf{F}\boxed{\mathbf{G}\ (p \vee q)}$ false

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

get stuck here, and both p and q are violated

$s_0 \models \mathbf{FG}\ (p \vee r)$ True

states involved in all possible paths satisfy $p \vee q$.

① $s_0 \to s_2 \to s_2 \to \cdots$

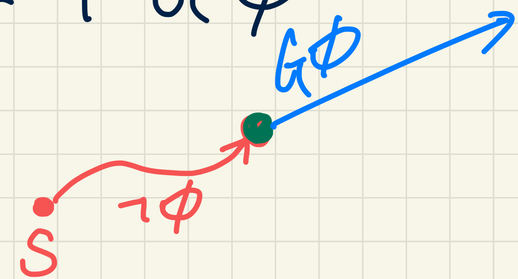
② $s_0 \to s_1 \to s_2 \to s_2 \to \cdots$

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

Exercise: What if we change the LHS to s_2?

$S \models G\phi$

$S \models FG\phi$

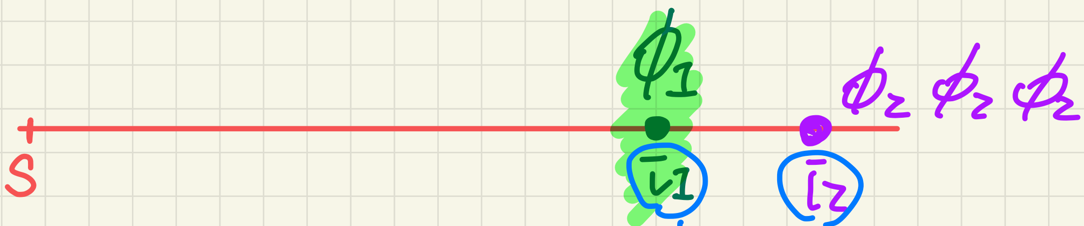


for each π.

$$S \models \boxed{F\phi_1} \Rightarrow \boxed{\underline{\underline{E}}G\phi_2}$$

Consider $\underline{\underline{\text{all}}}$ $\underline{\underline{\text{paths}}}$

$\pi = S \rightarrow \cdots$



may or may $\underline{\underline{\text{not}}}$ be the same

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

$s \models \boxed{F\phi 1} \Rightarrow \boxed{FG\phi_2}$

Each path π starting with s is s.t. if eventually ϕ1 holds on π, then ϕ2 eventually holds on π continuously.

Q. Formulate the above nested pattern of LTL operators.

dummies at different scopes

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

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

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

Lecture 12 - March 2

Model Checking

Path Satisfaction: Nested LTL Operators

F $\phi 1 \Rightarrow$ ***FG*** $\phi 2$

Nesting "Global" and "Future" in LTL Formulas

model sat.

$$s \models \boxed{\mathbf{F}\phi 1} \Rightarrow \boxed{\mathbf{F}\mathbf{G}\phi_2}$$

Each path π starting with s is s.t. if eventually ϕ1 holds on π, then ϕ2 eventually holds on π continuously.

Q. Formulate the above nested pattern of LTL operators.

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

$\big((\exists i_1 \bullet i_1 \geq 1 \land \pi^{i_1} \models \phi_1)$

$** \Rightarrow$

$(\exists i_2 \bullet i_2 \geq 1 \land (\forall j \bullet j \geq i_2 \Rightarrow \pi^{j} \models \phi_2)) \big)$

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

① Consider all path patterns ② a. T ⇒ T b. F ⇒ _ c. _ ⇒ T

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

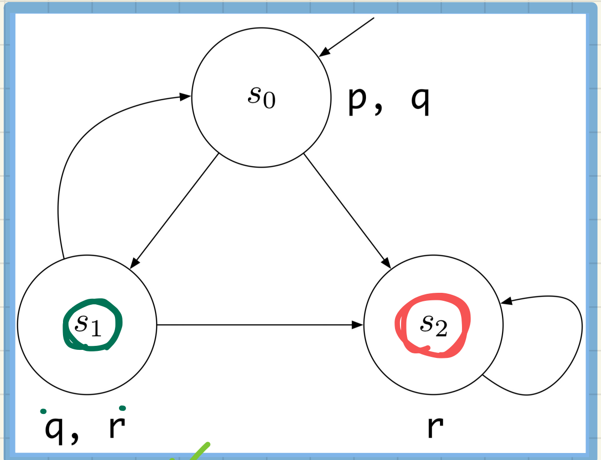
① Find a witness path ② T ⇒ F.

alt to *: s_2 satisfies $\neg q \wedge r$, then from s_2 r is cont. T

Path Satisfaction: Exercises (5.2)

④ $s_0 \to s_1 \to s_0 \to s_1 \to \cdots \to s_2 \to \cdots$

⑤ $s_0 \to s_1 \to s_0 \to s_1 \to \cdots \to s_0 \to s_2 \to \cdots$



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

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

* $s_0 \models \mathbf{F}\,(\neg q \wedge r) \Rightarrow \mathbf{FG}\ r$

① $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$

↳ no state in this path satisfies $\neg q \wedge r$

↳ $F(\neg q \wedge r)$ is false

false $\Rightarrow P \equiv$ $\boxed{\text{True}}$

② $s_0 \to \boxed{s_2} \to s_2 \to \cdots$

↳ satisfies $\neg q \wedge r$

↳ $F(\neg q \wedge r)$ is T

starting from s_2, Gr is satisfied. T $\Rightarrow$ T $\equiv$ T

$s_0 \models \mathbf{F}\,(\neg q \vee r) \Rightarrow \mathbf{FG}\ r$ F

T

Witness: $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$

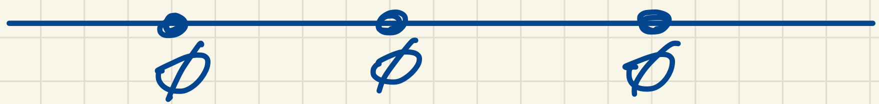
↳ satisfies $F(\neg q \vee r)$: s_1

↳ violates: $FG\ r$: s_0 does not satisfy r.

Exercise: What if we change the LHS to s_2?

FG $\emptyset$

GE $\emptyset$



Lab2 Solution: getAllSuffixes (V2: Tuple of Tuples)

```
---------------------------- MODULE getAllSuffixes_v2 -----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v2 {
    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 vars will be constants ↳ need to be instantiated for model checking

outputs will be variables

algo

\E

input: [23, 46, 69]

result: Len(input)

[[23, 46, 69], 3

[46, 69], 2

[69]]

I	Len(I)	J	Len(r[J])
	3	1	3
	3	2	2
	3	3	1

```
---------------------------- MODULE getAllSuffixes_v2 ----------------------------
EXTENDS Integers, Sequences, TLC
CONSTANT input
ASSUME Len(input) > 0
(*
--algorithm getAllSuffixes_v2 {
    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]);
    }
}
*)
```

an item in some result tuple

offset

input: [23, 46, 69]

result:

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

use k to refer to an item in tuple

use j to refer to a suffix

j

1 k = 1, 2, 3

2 k = 1, 2

3 k = 1

(Len(result[1]))

Lab2 Solution: getRightShifts

B ⇒ P

∧

≡ ¬B ⇒ Q

```
------------------------------ MODULE getRightShifts ------------------------------
EXTENDS Integers, Naturals, Sequences, TLC
CONSTANT input, n
ASSUME /\ Len(input) > 0
       /\ n >= 0

(*
--algorithm getRightShifts {
    variable result = input, nos = n % Len(input) (* number of shifts *), i = 1;
    {
        while (i <= Len(input)) {
            if (((i + Len(input)) - nos) % Len(input) = 0) {
                result[i] := input[Len(input)];
            } else {
                result[i] := input[((i + Len(input)) - nos) % Len(input)];
            };
            i := i + 1;
        };
        \* version 1 of postcondition: for each index in the result, what is the corresponding index in the input?
        assert \A j \in 1..Len(result): IF ((j + Len(result)) - (n % Len(result))) % Len(result) = 0
                                        THEN result[j] = input[Len(input)]
                                        ELSE result[j] = input[((j + Len(input)) - (n % Len(input))) % Len(input)];
        \* version 2 of postcondition: for each index in the input, what is the corresponding index in the result?
        assert \A j \in 1..Len(input): IF (j + (n % Len(input))) % Len(input) = 0
                                       THEN result[Len(input)] = input[j]
                                       ELSE result[((j + (n % Len(input))) % Len(input))] = input[j];
    }
}
*)
```

IF B
THEN P
ELSE Q

shift to R by one pos.

1 2 3

input: [23, 46, 69]

1 2 3

result: [69, 23, 46]

+1

input		output
1	2	2
2	3	3
3	4	1

4 % 3

Assertion: explicit about the variables that can be used.

```
<<1, 2, 3>> \o <<4, 5, 6>>

output1 \o <<i>>
```

Lecture 13 - March 14

Model Checking

Model Satisfaction: Nested LTL Operators
$GF\phi$, $GF\phi \Rightarrow GF\phi$
LTL Operators: Until, Weak Until, Release

Announcements

- ProgTest1 result to be released by Friday
- Lab3 to be released by the end of Thursday

mutual
exclusion

external

G
T
F.

Nesting "Global" and "Future" in LTL Formulas

$s \models GF\,\phi$

infinitely often

π : ○ → | ○ → ○ | → . . . ○ → . . .

(1) ϕ should "almost" always true

(2) relative to any state, can't wait indefinitely for ϕ to be true again

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 path patterns (2) argue for each state on the path p.

(3) for each state, where's the future state

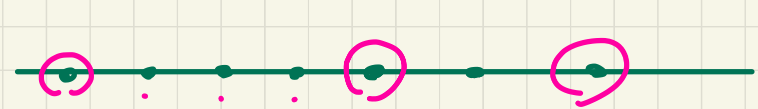
Q. How to disprove the above nested pattern of LTL operators?

* Give a witness path π $G\neg\phi$ that satisfies ϕ.

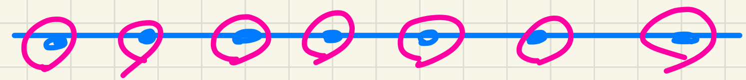
** Give a witness state on π, say s'

$\boxed{\text{*** relative to } s' \text{ on } \pi,\ \phi \text{ is never true.}}$

(1) GF $\underline{\underline{\emptyset}}$

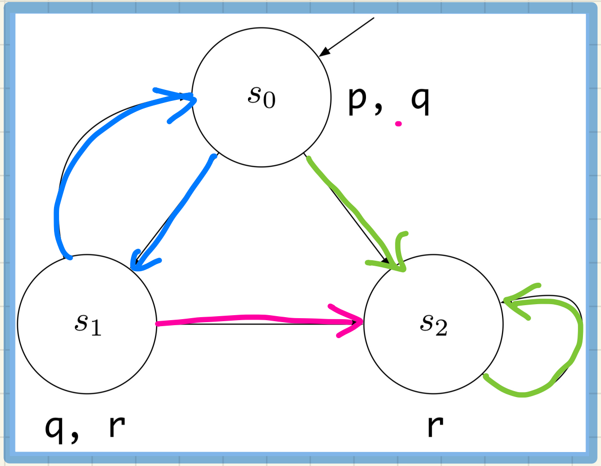


(2) $\underline{\underline{G}}$ $\underline{\underline{\emptyset}}$



$$\boxed{\begin{array}{l} (2) \Rightarrow (1) \\ (1) \not\Rightarrow (2) \end{array}}$$

Model Satisfaction: Exercises (6.1)



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

$s_0 \models GF\ p$ $\boxed{\text{false}}$

$\boxed{s_0} \rightarrow \boxed{s_2} \rightarrow \boxed{s_2} \rightarrow \cdots$

p $\qquad \neg p$

$s_0 \models GF\ (p \lor q)$ $\boxed{\text{false}}$

$\boxed{s_0} \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$

$p \lor q$ $\qquad \neg(p \lor q)$

$\because s_0 \models G(p \lor r)$

$\Rightarrow s_0 \models GF(p \lor r)$

$s_0 \models GF(p \lor r)$

$\boxed{\text{True}}$

↳ ∵ all states satisfy $p \lor r$

Path Patterns

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

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

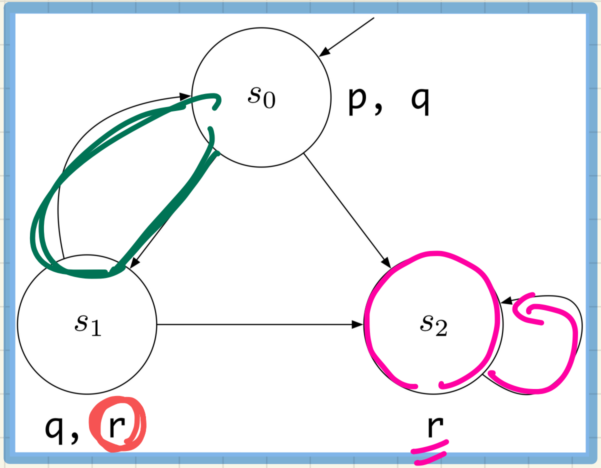
(3) $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots$

(4) $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \cdots \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$

(5) $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_2 \rightarrow \cdots$

Exercise: What if we change the LHS to s_2?

Model Satisfaction: Exercises (6.2)



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

$s_0 \models GF\ p \Rightarrow G\ r$ false

Just. For $s_0 \to s_1 \to \cdots$, r is not continuously true.

$s_0 \models GF\ p \Rightarrow GF\ r$ true

$\forall \pi \cdot \pi = s \to \cdots \Rightarrow ($

$(\forall i \cdots \Rightarrow \exists j \cdots \wedge \pi^j \models p)$

$\Rightarrow (\forall i \cdot \cdots \Rightarrow (\exists j \cdot \cdots \wedge \pi^j \models r)))$

(1) assume GF p: only path to consider is $s_0 \to s_1 \to s_0 \to \cdots$

(2) In that path: GF r true.

$s0 \models GF\ r \Rightarrow GF\ p$ False.

Witness

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

GF r is True

but GF p is false

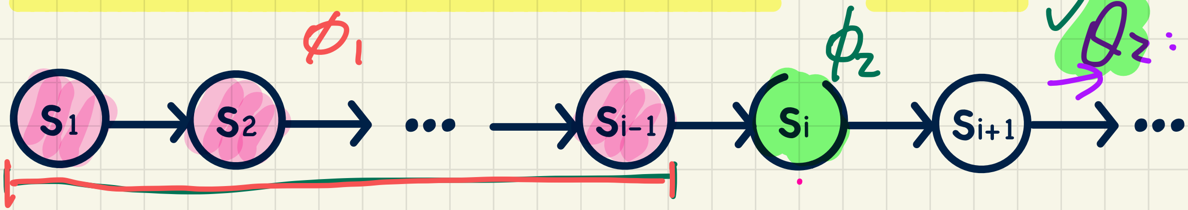
$T \Rightarrow F = F$.

Exercise: What if we change the LHS to s_2?

Path Satisfaction: Temporal Operations (4)

$\pi \models \phi_1 \, \mathrm{U} \, \phi_2$ until

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



Θ_1: Is it ok that $G\,\phi_1$ but $G\,\neg\phi_2$? ×

✓ Θ_2: Is it ok that $G\,\phi_1$ and $F\,\phi_2$?

Formulation (over a path)

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

Path Satisfaction: Temporal Operations (5)

weak until

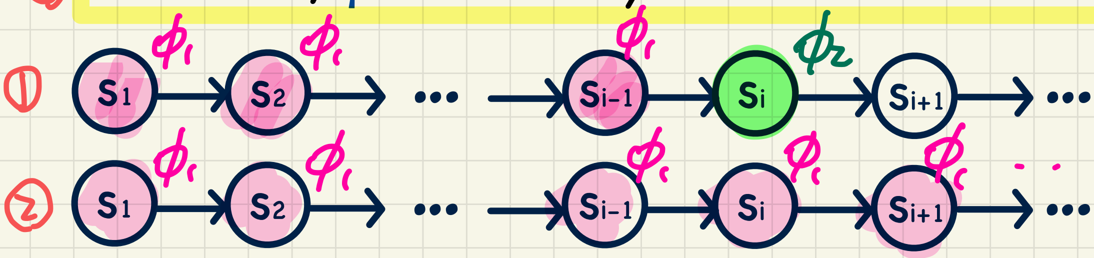
$\pi \models \phi_1 \, W \, \phi_2$

① If there is ever a future state that satisfies ϕ_2, then until then, all states satisfy ϕ_1.

② ~~Otherwise~~ Or, ϕ_1 must always be the case.

Q1. Is it ok that $G\,\phi_1$ but $\boxed{G\,\neg\phi_2}$. ok

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



Formulation (over a path)

$$\phi_1 \, \mathcal{W} \, \phi_2 \iff \phi_1 \, \mathcal{U} \, \phi_2 \lor (\forall k \cdot k \geq 1 \Rightarrow \pi^k \models \phi_1)$$

$$\phi_1 \, \mathcal{W} \, \phi_2 \iff \phi_1 \, \mathcal{U} \, \phi_2 \lor G\,\phi_1$$

P

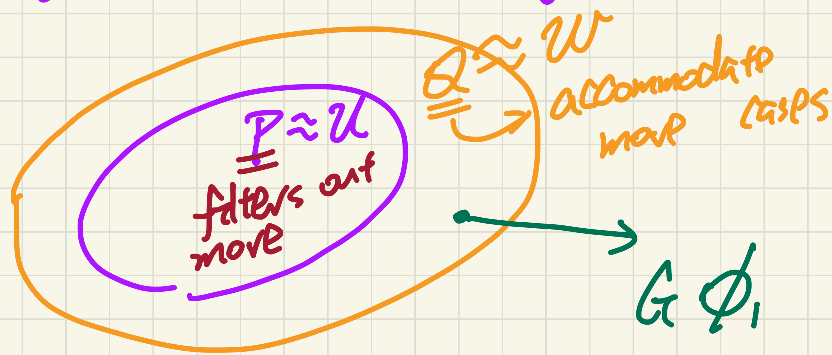
q

$P \Rightarrow q$

stronger   weaker

Satisfying values

$$\{x \mid P(x)\} \subseteq \{x \mid q(x)\}$$



Path Satisfaction: Temporal Operations (6)

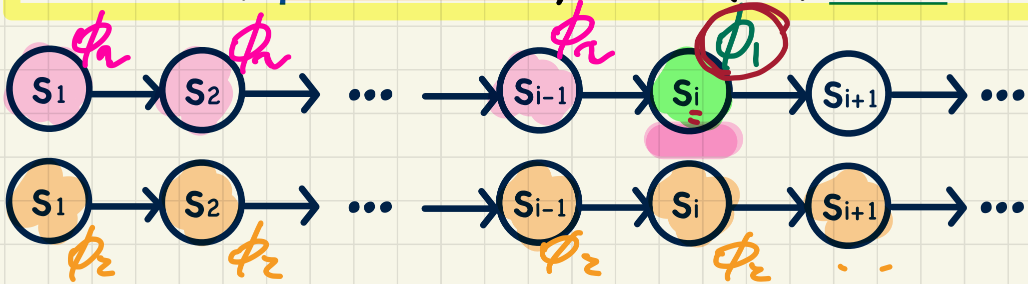
release

$\pi \models \phi 1 \, \mathbf{R} \, \phi 2$ → "ϕ_2 has been holding, ϕ_1 releases it"

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

OV

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



Formulation (over a path)

$$\pi \models \phi_1 \mathcal{R} \phi_2 \iff \left(\begin{array}{l} \left(\exists i \cdot i \geq 1 \wedge \left(\pi^i \models \phi_1 \wedge \left(\forall j \cdot j \leq i \cdot \pi^j \models \phi_2\right)\right)\right) \\ \vee \\ \underbrace{\left(\forall k \cdot k \geq 1 \Rightarrow \pi^k \models \phi_2\right)}_{G\,\phi_2} \end{array} \right)$$

this disjunct put constraints on ϕ_1 being satisfied

Lecture 14 - March 16

Model Checking

LTL Examples: Until, Weak Until, Release
Formulating Natural Language in LTL

Announcements

✓ 6:45 7
lecture video

- Mar 23 class?
- ProgTest1 result to be released by the end of Friday
- Lab3 released

PPY & D.M.
eecs login.

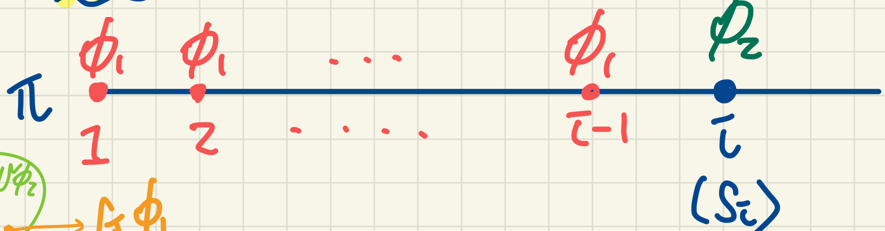
- WrittenTest2 example questions to be released
- Review Q&A session: 7pm on Sunday, March 19?

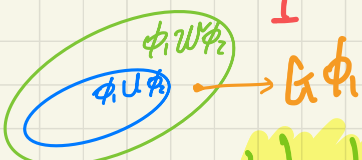
cont.

U, W, R

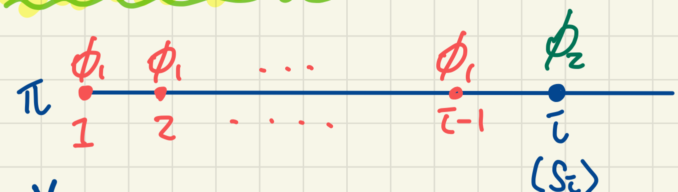
Until $\phi_1 \, U \, \phi_2$

starting from S_i, ϕ_1 being true or not would not impact the evaluation of the formula

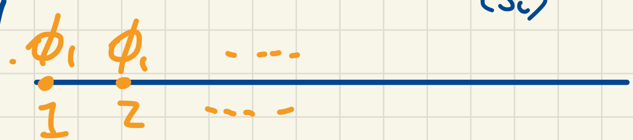




Weak Until $\phi_1 \, W \, \phi_2$

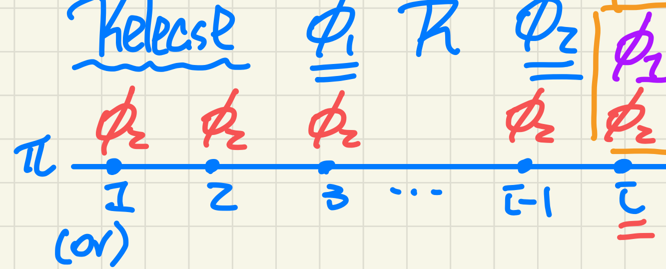


(or) $\vee$

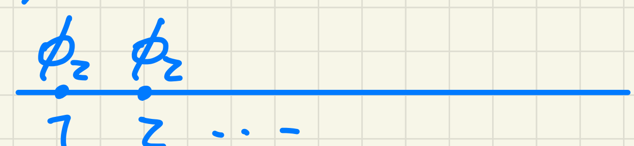


Release $\underline{\underline{\phi_1}} \, R \, \underline{\underline{\phi_2}}$

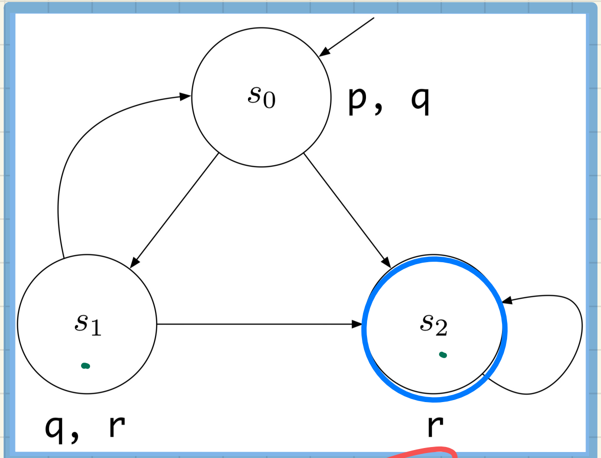
if there's no state satisfying $\phi_1 \wedge \phi_2$, then this scenario cannot satisfy.



(or)



Model Satisfaction: Exercises (7.1)



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

$s_0 \models p \ \mathcal{U} \ r$ True

P: $s_0 \to s_1 \to \cdots$ (r at s_1)

$s_0 \to s_2 \to \cdots$

(i-1) (i)

$s_0 \models p \ \mathcal{W} \ r$ True

$\because \phi_1 \ \mathcal{U} \ \phi_2 \Rightarrow \phi_1 \ \mathcal{W} \ \phi_2$

$s_2 \models r \ \mathcal{U} \ p$ false

$\because$ no state satisfying p

$\to s_2 \models r \ \mathcal{W} \ p$

always true

True

* $s_2 \models p \ \mathcal{U} \ r$ True

$s_2 \to s_2 \to \cdots$

i=1

r already satisfied and $(\forall j \cdot 1 \le j \le i-1 \Rightarrow \pi^j \models p)$ true

$s_0 \models r \ \mathcal{R} \ p$

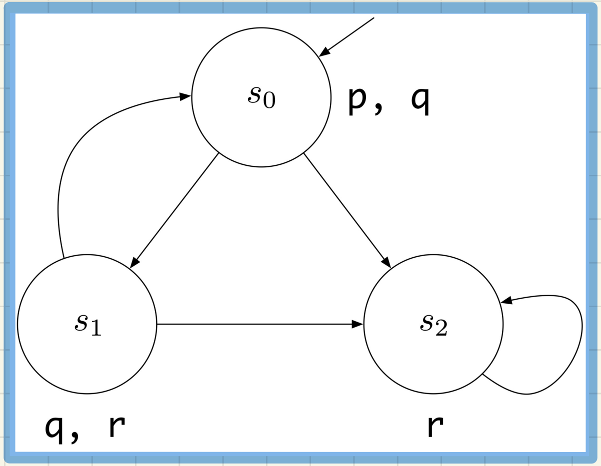
Witness $s_0 \to s_1 \to s_0 \to \cdots$

(1) no state that satisfies $p \wedge r$

(2) not the case that p is always true

Exercise: What if we change the LHS to s_2?

Model Satisfaction: Exercises (7.2)



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

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

$\because$ $p \wedge r$ is never satisfie

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

↳ We know: $(p \vee r)\ \mathcal{U}\ (p \wedge r)$ false

But: $(p \vee r)$ is always satisfied.

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

↓

always true

$s_0 \models G\,(p \vee r)$

$\boxed{\text{2nd case of } R \text{ satisfied.}}$

Exercise: What if we change the LHS to s_2?

Formulating Natural Language in LTL (1)

Ex: I smoked $\mathcal{U}$ G(I was 22 $\wedge$ ...).

Natural Language:

I had smoked until I was 22.

I was 22

s s s ... s ¬s ¬s ¬s

Atom t: I was 22

Atom s: I smoke

Q. Is s U t an appropriate formulation?

Ex

I smoked $\mathcal{U}$

(I was 22 $\wedge$

¬(I smoked))

s s s s

s s s

i I was 22

$\forall j \cdot 1 \le j \le i-1 \Rightarrow \pi^j \models s$

$\pi^i \models t$

Solution $\underbrace{\text{(I smoked)}}_{\phi_1}$ $\mathcal{U}$ ϕ_2 (I was 22 $\wedge$ G(¬ I smoked))

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

Formulating Natural Language in LTL (2.1)

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

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

Natural Language:

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

$\neg F(\text{started} \wedge \neg \text{ready})$

Assumed atoms:

- started
- ready

LTL Formulation

$G(\neg(\text{started} \wedge \neg \text{ready}))$

$G(\neg \text{started} \vee \text{ready}) \rightarrow G(\text{started} \Rightarrow \text{ready})$

Formulating Natural Language in LTL (2.2)

Natural Language:

Whenever a request is made,
it will be acknowledged eventually.

no starvation

Assumed atoms:

- requested
- acknowledged

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

LTL Formulation

Formulating Natural Language in LTL (2.3)

Natural Language:

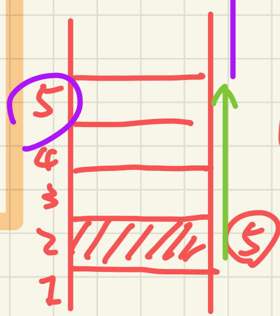
An elevator traveling upwards at the 2nd floor
does not change its direction
when it has passengers wishing to to to the 5th floor.

Assumed atoms:

- floor2, floor5
- directionUp
- buttonPressed5

elevator state

LTL Formulation



$$G\big(floor2 \wedge buttonPressed5 \Rightarrow (directionUp\ \mathcal{U}\ floor5)\big)$$

$\mathcal{W}$ ✗

Sunday, March 19

Written Test 2 Review

algorithm

input ... output ...

e.g. a tuple of integers

a tuple of integers sorted in non-descending order

postcondition: assertions relating the input & output.

output

last-set value

* assert \A i . i \in 1..len(output)-1 : output[i] <= output[i+1]

the only postcondition

Q. Is this postcondition correct & complete?

- Given a correct pair of input/output, assertion should pass.

input: <<5, 3, 1, 2>>
output: <<1, 2, 3, 5>>

- Given an incorrect pair of input/output, assertion should fail.

input: <<5, 3, 1, 2>>
output: <<23, 46, 89, 92>>

assertion should fail on this, but it actually passes.

- $\boxed{\text{syntax}}$
  - ↳ derivable from grammar
    - ↳ formal meaning
- Correct or not?

$$\underline{s\ \mathcal{U}}\ t \quad \times \quad \text{vs} \quad \underline{s\ \mathcal{U}\ (t \wedge G(\neg s))} \quad \checkmark$$

$$G(\underline{\phi_1\ \mathcal{U}\ \phi_2})$$

compare to $\underline{GF\phi}$.

p : I eat

q : I go to school

$$G((X\ q) \Rightarrow p)$$

$$s_0 \models \boxed{G}\, \boxed{F\,(\, \boxed{G\phi_1 \Rightarrow F\phi_2}\,)}$$

$$\forall \pi \cdot \pi = s_0 \to \cdots \Rightarrow$$

$$\forall i \cdot i \geq 1 \Rightarrow$$

$$\left(\exists j \cdot j \geq i \land \left(\overbrace{\boxed{\forall k \cdot k \geq j \Rightarrow \pi^k \models \phi_1}}^{G} \Rightarrow \overbrace{\boxed{\exists l \cdot l \geq j \land \pi^l \models \phi_2}}^{F} \right) \right)$$

$$s_0 \models \underline{\underline{F}}\left(\boxed{\phi_1 \, \mathcal{U} \, \phi_2}\right)$$

$$\forall \pi \cdot \pi = s_0 \to \cdots \Rightarrow$$

$$\left(\exists i \cdot i \geq \underline{\underline{1}} \;\wedge \left(\exists j \cdot j \geq \underline{\underline{i}} \right) \right)$$

Lecture 15 - March 28

Program Verification

Stronger vs. Weaker Assertions
Total vs. Partial Correctness

Announcements

Lab3
↳ grace period until 12noon

- Bonus Opportunity - Course Evaluation
- ProgTest1: Echo (eMail, Zoom); Jackie (Office Hour)
- Lab3 due tomorrow
- ProgTest2 :: -
- Final Exam: Review Q&A Sessions

↳ data sheet
↳ one side only: put anything you like
↳ computer-typed ≥ 10pt

Lecture

Program Verification

Correctness - Motivating Examples

no overlap $\Rightarrow$ can't say weaker or stronger

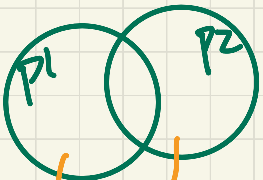
Assertions: precondition, postcondition, invariant

$x>4 \Rightarrow x>3$

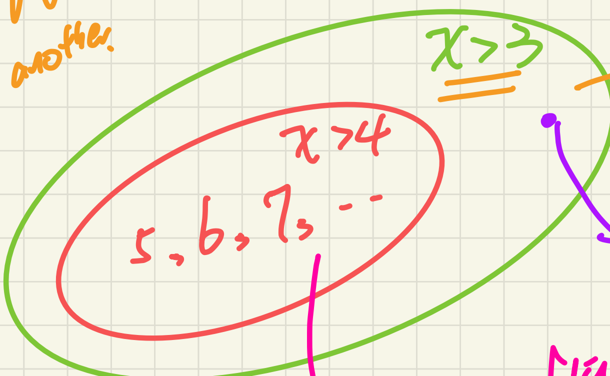
$$\boxed{\begin{array}{c} x>3 \\ x>4 \end{array}}$$

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

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



In this case the sets of satisfying values are not subset of one another.



more satisfying values $\Rightarrow$ weaker.

fewer satisfying values $\Rightarrow$ stronger

$P1 \Rightarrow P2$

P1 stronger

P2 weaker

Weakest predicate: True

Strongest predicate: False

-- algorithm foo

assert $\boxed{}$

$x > 3$

weaker

vs. $x > 4$

stronger

more input values can be used for computation.

false precond.

↳ no input value accepted.

Imp.

Postcond. $\boxed{}$

P_1 vs. $\boxed{P_2}$

weaker ↳ more computation outcomes are acceptable

true post.

↳ no way to check computation

known: $P1 \Rightarrow \boxed{P2}$

Preconditions

known: $p1 \Rightarrow p2$

P_1 vs. P_2

stronger: require more (on input values)

weaker: require less (on input values)

search(int[] a, int tar)

↳ precond. for linear search:

a != null

↳ precond for binary search:

a != null

a is sorted

Postconditions

known: $p1 \Rightarrow p2$

P_1 vs. P_2

P_1 ↳ stronger: ensure more on the output result

P_2 ↳ weaker: ensure less on the output result.

Program Correctness: Example (1)

traceability of specification

```
--algorithm increment_by_9 {
  variable i;
  {
    (* precondition *)
    assert i > 3
    (* implementation *)
    i := i + 9;
    (* postcondition *)
    assert i > 13
  }
}
```

fix: i > 4

4

4

13

too weak: value 4 is accepted, and it's going to cause the terminating state to violate the postcond.

specification

for wp calculation, assume imp. and postcond. are fixed

not correct!

Correctness

1. Relative concept
2. For input values satisfying the precondition, executing the implementation will:
   (1) terminate
   (2) output/input satisfies the postcond upon termination.

Is this program correct?

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

Program Correctness: Example (2)

```
--algorithm increment_by_9 {
  variable i;
  {
    (* precondition *)
    assert  i > 5
    (* implementation *)
    i := i + 9;

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

6, 7, 8, 9, ...

15, 16, 17, ...

Is this precond. too strong?

∵ 5 is disallowed by the precond.

need to check the REQ.

fixed.

Is this program correct?

$\{i > 5\}\ i := i + 9\ \{i > 13\}$ → provable as a theorem

Lecture

Program Verification

Hoare Triple and Weakest Precondition

e.g. conjunction of a list of assertions

$$\{\underbrace{\quad\quad}_{\text{precondition}}\}\ \underbrace{\quad\quad S \quad\quad}_{\text{programming statement.}}\ \{\underbrace{\quad\quad}_{\text{postcondition}}\}$$

can be transformed into a Boolean predicate

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

└→ Starting in a state satisfying Q, executing S will $\boxed{\text{terminate}}$ in a state satisfying R.

PO1 ∧ ¬PO2 ⇒ incorrect.

PO1

PO2

PO2 (without termination) ⇒ partial correctness

PO2 ∧ PO1 ⇒ total correctness

Lecture 16 - March 30

Program Verification

Weakest Precondition (WP)
WP Rules

Announcements

- Lab3 due tomorrow Friday, noon
- $\boxed{\text{ProgTest2}}$

  level of difficulty ≈ EECS1022/1022

Algorithm
- precond Q
- postcond R
- imp S

formulate

$\{Q\}\ S\ \{R\}$

how to transform a Hoare Triple into a predicate?

prove or disprove.

POs for partial correctness & termination

$\begin{cases} \boxed{\text{provable}} \Rightarrow \text{algo. correct} \\ \text{otherwise} \Rightarrow \text{incorrect.} \end{cases}$

Hoare Triple as a Predicate

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

Hoare Triple

Correct Program

precond.

imp.

wp(S, R)

imp.

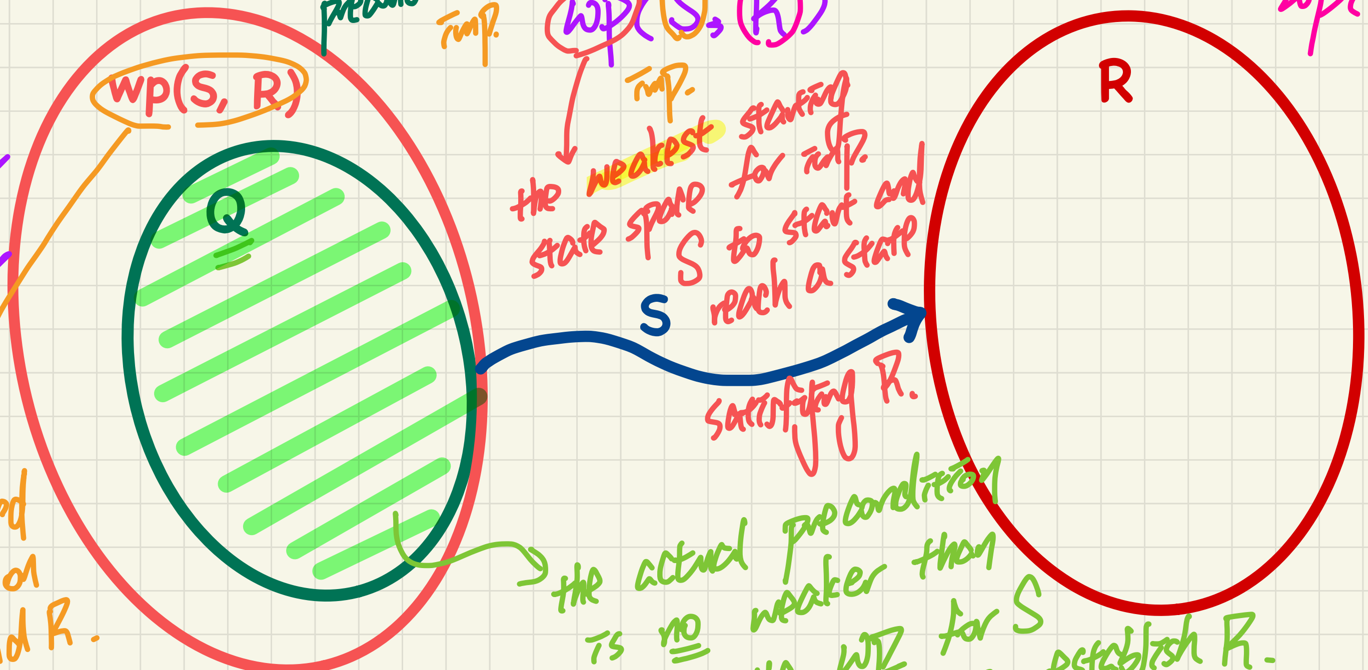
postcond.

If Q is false, program is trivially correct.

express: Q is stronger than wp(S, R)

⇒ fails to hold if Q is weaker than wp(S, R)

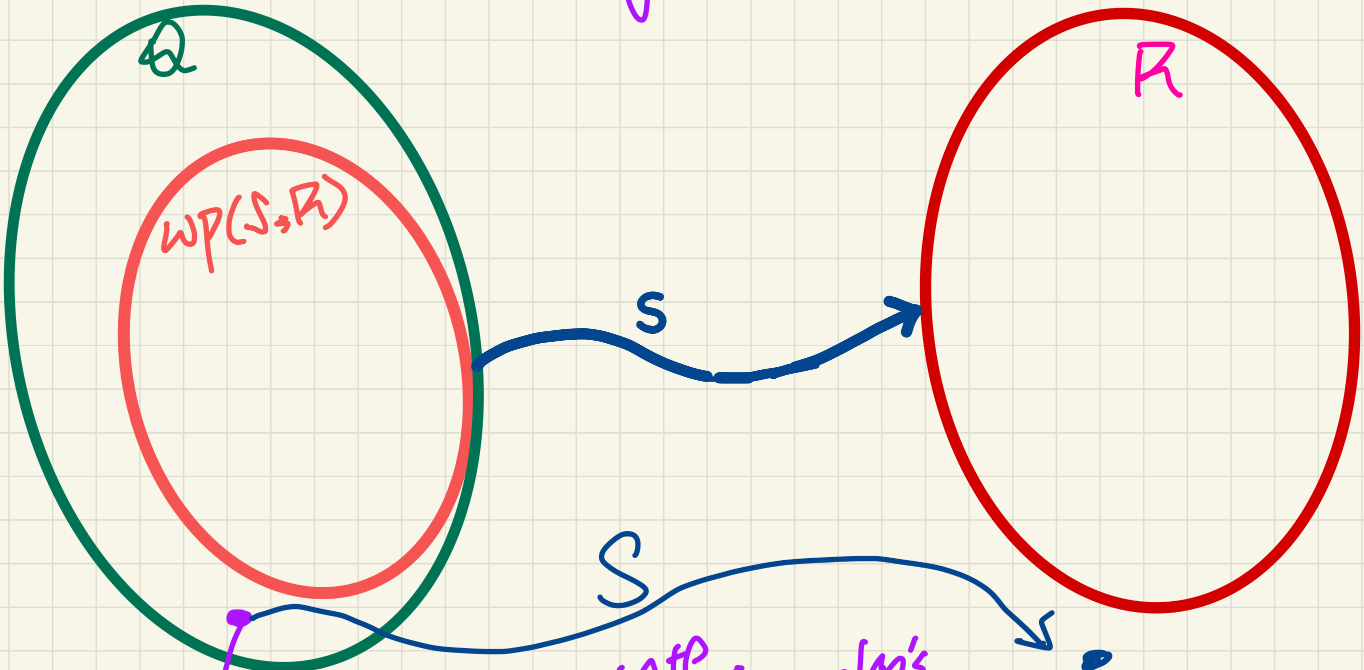
the weakest starting state space for imp. S to start and reach a state satisfying R.



calculated based on S and R.

the actual precondition is no weaker than the wp for S to establish R.

Incorrect Program



starting from this state which satisfies the algo's precondition will <u>not</u> result in a state satisfying R pre-state value of b (value of b at the beginning of algorithm)

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

tmp?

usually, subscript 0 is omitted

the post-state value of b equals the pre-state value of b minus a

3342:

b → pre-state

b' → post-state

4315

b_0 → pre-state

b → post-state

Lecture

Program Verification

Rules of wp Calculus

Rules of Weakest Precondition: Assignment

base case for wp calculation

$\{Q\}\ x := e\ \{R\}$

$\hookrightarrow Q \Rightarrow \boxed{wp(x := e, R)}$

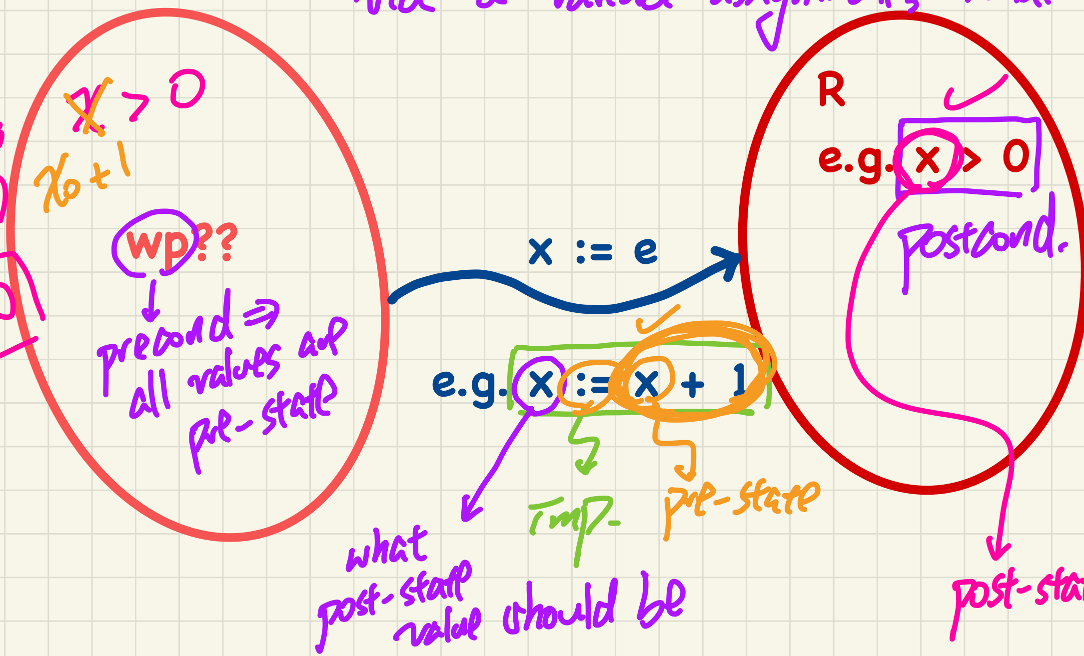
$R[x := e]$

$$wp(\underbrace{x := e}_{S}, R) = R[x := e]$$

to achieve the postcond. R, via a variable assignment, what's the wp to start?

$wp(x := x+1, x > 0)$

$= \boxed{x_0 + 1 > 0}$



$\boxed{wp}(x := 23 \; , \; \underline{\underline{x = 46}})$

$= \{ \text{wp value for } := \}$

$\quad \underline{\underline{x}} = 46 \, [x := 23]$

$= \; 23 = 46$

$= \boxed{False}$

acceptable no input values can guarantee that "x := 23" will establish 46.

In case, better off you're just fixing S or R.

the only way to have a correct program is:

Q

$\{ \underline{False} \}$

$x := 23$

$\{ x = 46 \}$

$\because False \Rightarrow False$

$\equiv T$

Correctness of Programs: Assignment (1)

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

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

$$wp(x := \boxed{x+1}^{e},\ \underline{x > x_0})$$

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

$$\underline{x > x_0}\,[x := x_0 + 1]$$

$$= x_0 + 1 > x_0$$

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

any precondition will be correct

$\text{“}\_\text{”} \Rightarrow$ True

Correctness of Programs: Assignment (2)

Exercise

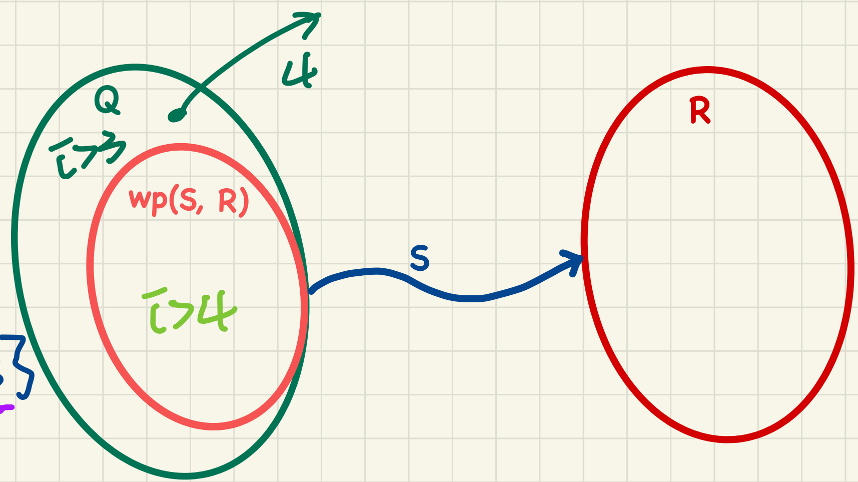
What is the weakest precondition for a program `x := x + 1` to establish the postcondition $x > x_0$?

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

Program Correctness: Revisiting Example (1)

```
--algorithm increment_by_9 {
  variable i;
  {
    (* precondition *)
    assert i > 3                Q
    (* implementation *)
    i := i + 9;                 S
    (* postcondition *)
    assert i > 13               R
  }
}
```

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



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

$\Leftrightarrow$

$i > 4$

$i > 3 \Rightarrow \boxed{wp(i := i + 9,\ i > 13)}$

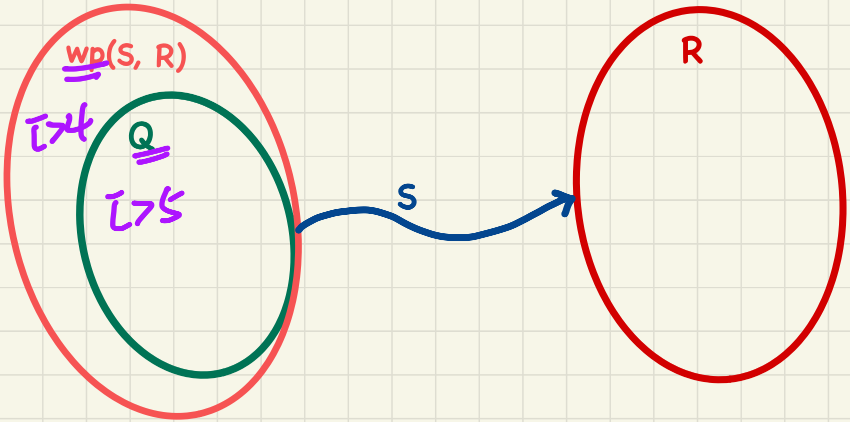
Program Correctness: Revisiting Example (2)

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

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

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

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



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

$= i > 4$

Argue: $i > 5 \Rightarrow i > 4$

Rules of Weakest Precondition: Conditionals

branch 1 Imp. branch 2 Imp.

$$wp(\text{if } B \text{ then } S1 \text{ else } S2 \text{ end}, R)$$

$$\begin{array}{l} B \Rightarrow wp(S_1, R) \\ \land \\ \neg B \Rightarrow wp(S_2, R) \end{array}$$

both branches should be able to establish the R. by the corresponding statement.

$$\{Q\}\ \overbrace{\text{if } B \text{ then } S_1 \text{ else } S_2}^{S}\ \{R\}$$

$$Q \Rightarrow \underbrace{\left(\begin{array}{l} B \Rightarrow wp(S_1, R) \\ \land \\ \neg B \Rightarrow wp(S_2, R) \end{array} \right)}_{wp(S,\ R)}$$

Correctness of Programs: Conditionals

Is this program correct?

$\{x > 0 \wedge y > 0\}$
```
if x > y then
  bigger := x ; smaller := y
else
  bigger := y ; smaller := x
end
```
$\{bigger \geq smaller\}$

B

S_1

S_2

(Step 3)

Argue: $x > 0 \wedge y > 0 \overset{?}{\Rightarrow} wp$

(Step 1) Formulate Hoare Triple

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

(Step 2) Calculate wp(if B then S_1 else S_2, $bigger \geq smaller$).

Exercise.

$$wp(\underbrace{S_1}_{\text{phase 1}} ;\ \underbrace{S_2}_{\text{remaining phases}},\ R)$$

$$\mid\ S_1\ ;\ \mid\ S_2\ \mid\ R$$

$$wp(S_1 ;\ wp(S_2 ; R))\qquad \boxed{wp(S_2 ; R)}$$

Correctness of Programs: Sequential Composition

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

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

$= \{ wp \text{ rule for } ; \}$

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

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

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

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

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

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

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

$\boxed{y > x}$

wp-

(Step 2) $True \Rightarrow y > x$

not a theorem.

counter ex: $x = 2$, $y = 1$

Lecture 17 - April 6

Program Verification

Contracts of Loops: Invariant vs. Variant
Correctness of Loops

Announcements

- Lab4 released
- Exam guide released

Program Verification

- Predicates : Stronger $\Rightarrow$ Weaker
- Hoare Triple $\{Q\}\ S\ \{R\}$

$\Leftrightarrow\ Q \Rightarrow wp(S, R)$

① :=
② if then else
③ ;

① Show wp calculation ② Prove

one example done in the math review lecture (equational style)

$= \{$ justification $\}$

$\vdots$

$\equiv \{ \ldots \}$

$\equiv \{ \ldots \}$

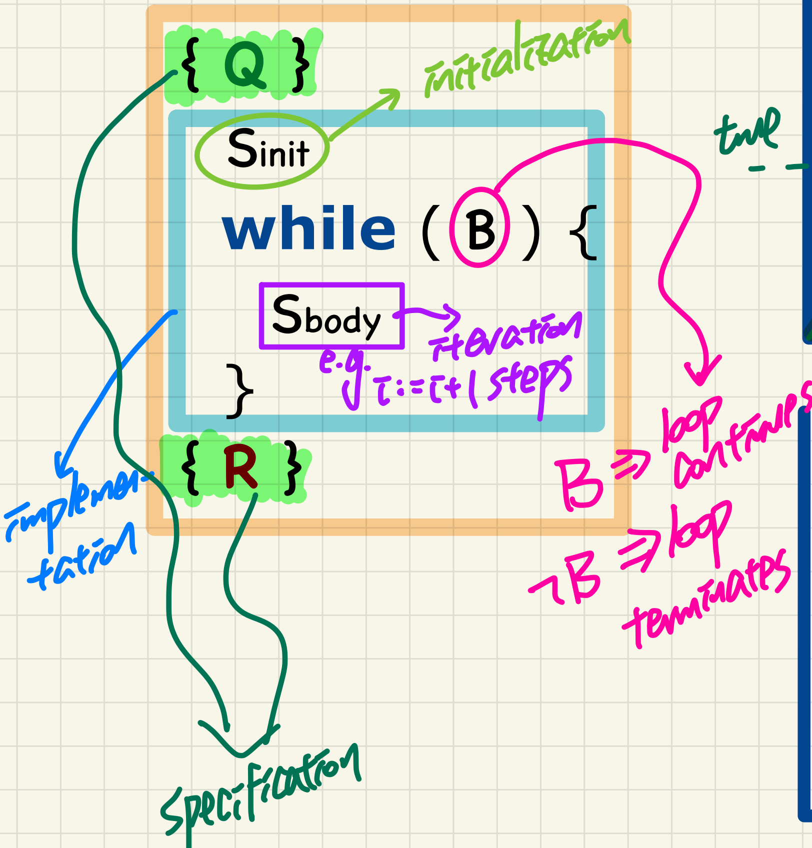
$\vdots$

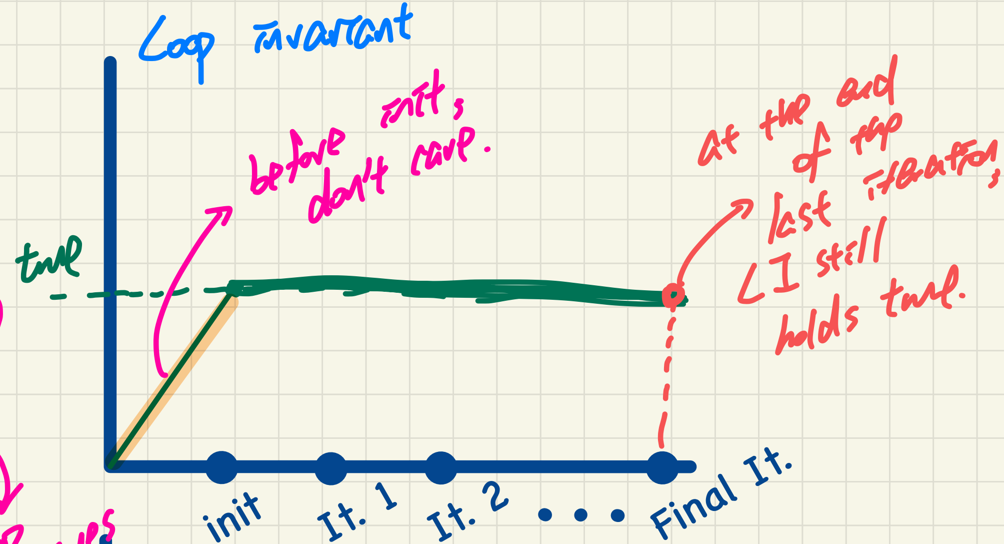
Lecture

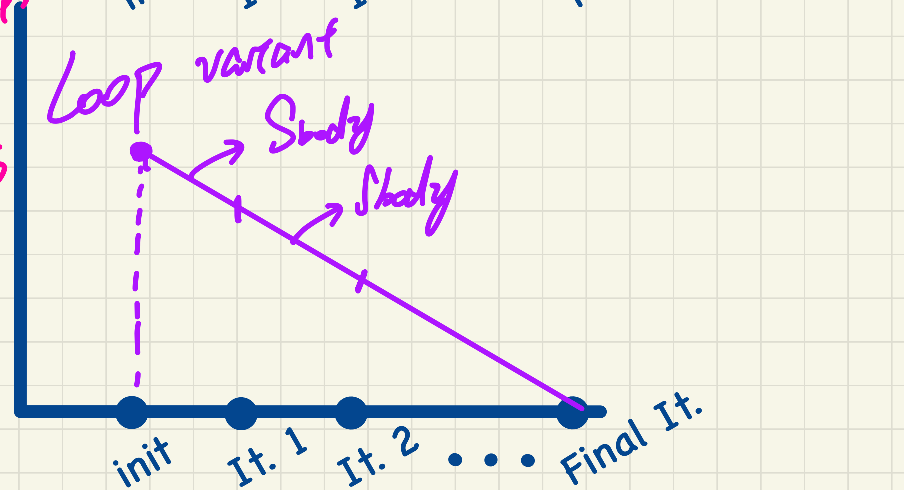
Program Verification

Contracts of Loops

Correctness of Loops

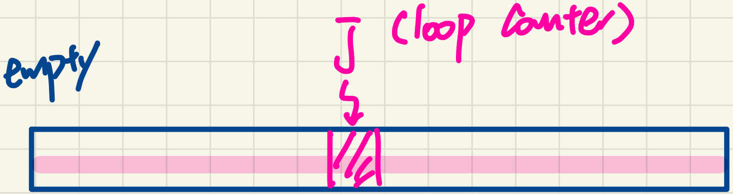






non-empty

Input:



Output: index i s.t. input[i] is max.

- Exercise. Write an assertion for the postcondition.

- Exercise 2: loop invariant.

  ↳ Hint: loop counter

  Hint: inclusive of j or not?

Contracts of Loops

Syntax

input

output

```
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 *)
  }
}
```

precond

LI established and maintained

body of loop

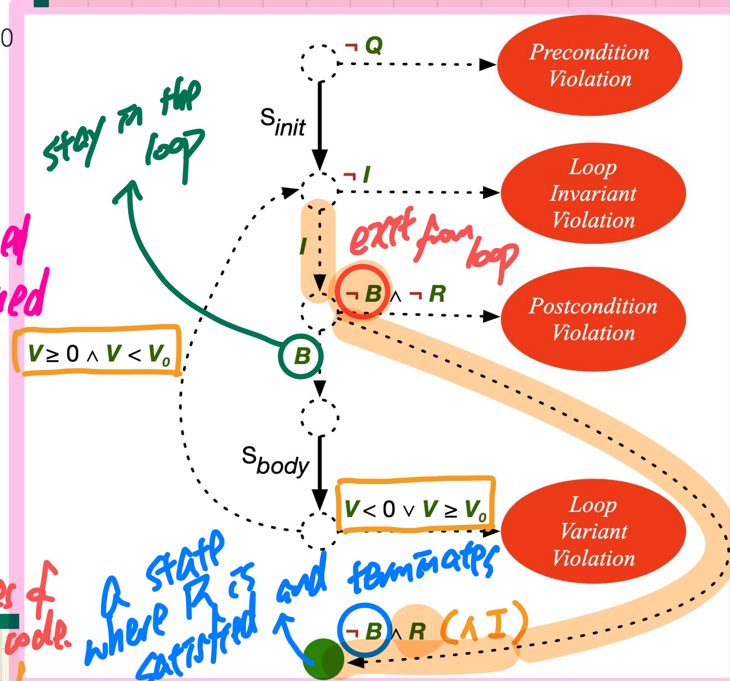
post cond

* R only needs to be established after these lines of code.

∴ In case there's code between end of loop and "assert R";

* LV: ① ∈ ℕ ② LV < LV₀

Runtime Checks



Contracts of Loops: Example

Assume: Q and R are true

Specification

```
I(i) == (1 <= i) /\ (i <= 6)
V(i) == 6 - i
--algorithm loop_invariant_test
  variables i = 1, variant_pre = 0, variant_post = 0;
  {
    assert I(i);
    while (i <= 5) {
      variant_pre := V(i);
      i := i + 1;
      variant_post := V(i);
      assert variant_post >= 0;
      assert variant_post < variant_pre;
      assert I(i);
    } ;
  }
```

① precond: true

② Start is "i=1"

B

end of iteration	i	I	V	B
0 → Start	1	T	(5)	T
1	2	T	4	T
2	3	T	3	T
3	4	T	2	T
4	5	T	1	T
5	6	T	0	F

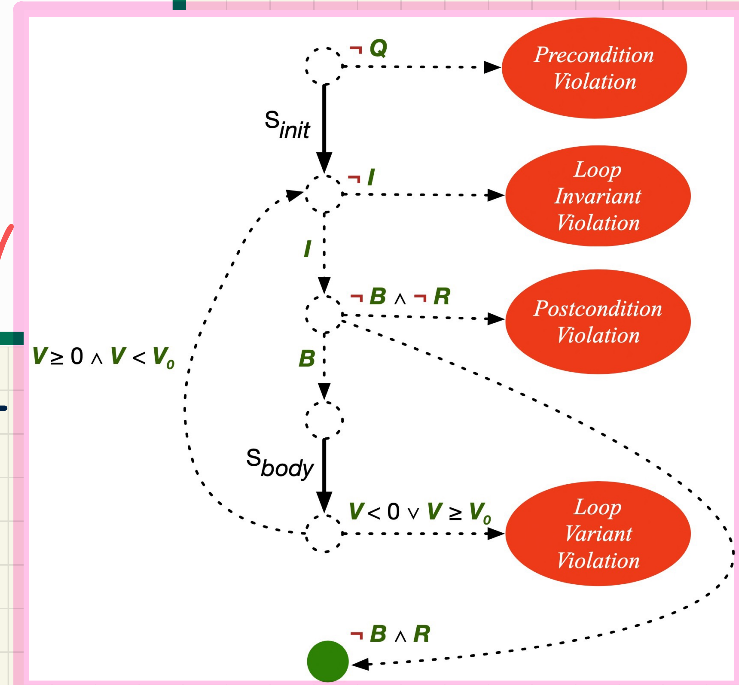
established

maintained

① ≥0

② dec

Runtime Checks



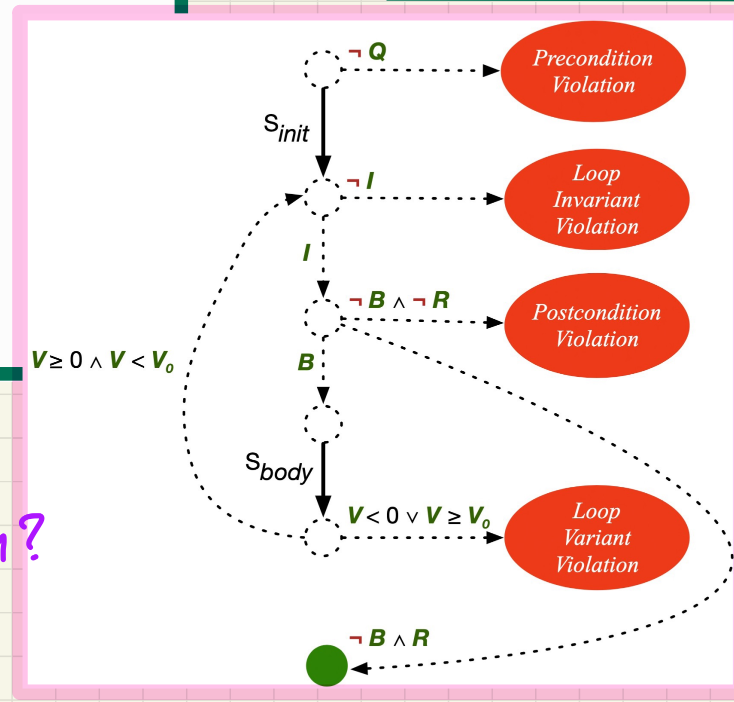
Contracts of Loops: Violations

Assume: Q and R are true

Specification

```
I(i) == (1 <= i) /\ (i <= 6)
V(i) == 6 - i
--algorithm loop_invariant_test
 variables i = 1, variant_pre = 0, variant_post = 0;
 {
   assert I(i);
   while (i <= 5) {
     variant_pre := V(i);
     i := i + 1;
     variant_post := V(i);
     assert variant_post >= 0;
     assert variant_post < variant_pre;
     assert I(i);
   } ;
 }
```

Runtime Checks



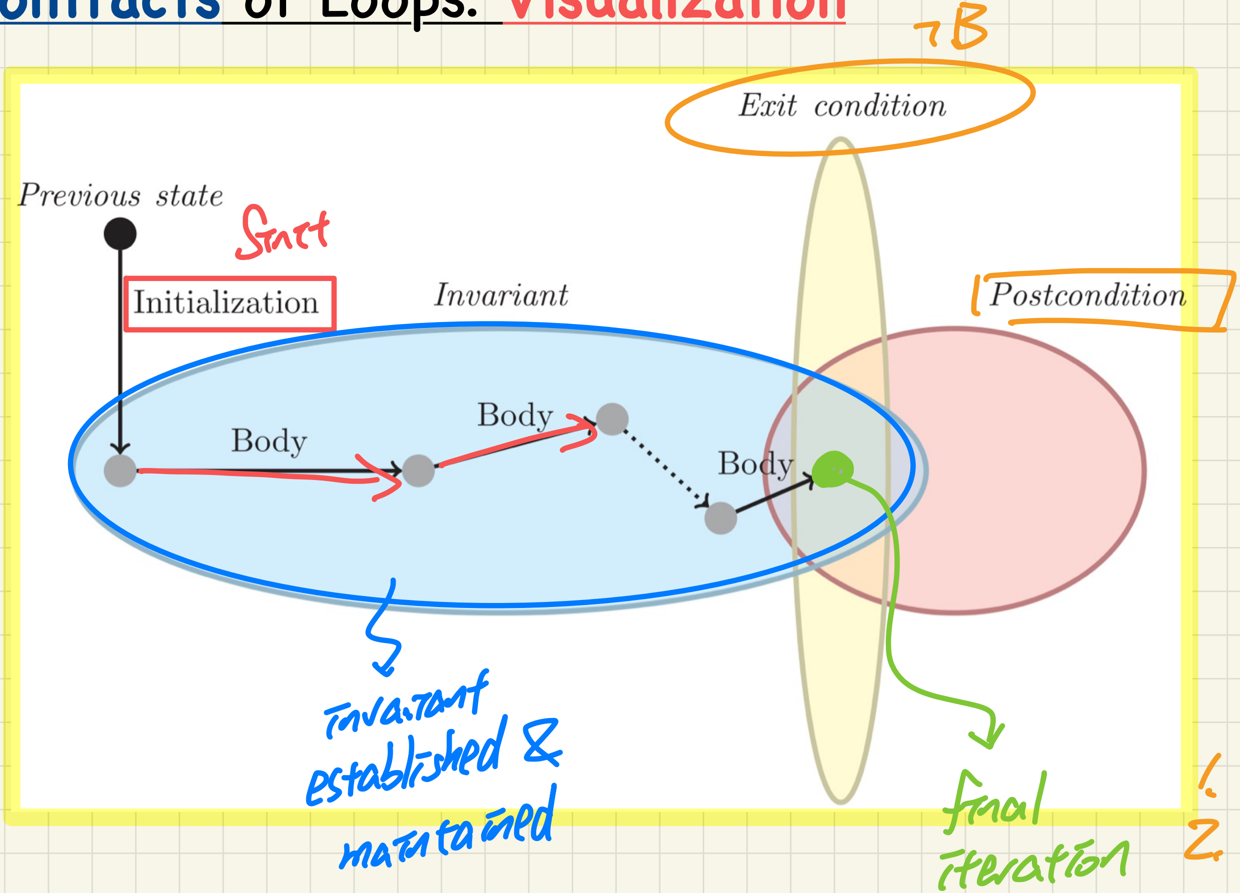
violation?

which iteration?

invariant: 1 <= i <= 5

variant: 5 - i

Contracts of Loops: Visualization



1. LI
2. ¬B
3. R

Lecture

Program Verification

Correctness Proofs of Loops

Correct Loops: Proof Obligations

- A loop is ***partially correct*** if:
  - Given precondition Q, the initialization step S_{init} establishes *LI* I.

    $\{Q\}\ \underline{S_{init}}\ \{I\}$

    $$\{Q\}\ S_{init}\ \{I\}$$

  - At the end of S_{body}, if not yet to exit, *LI* I is maintained.

    $\{I \wedge B\}\ S_{body}\ \{I\}$

    $$\{I \wedge B\}\ S_{body}\ \{I\}$$

  - If ready to exit and *LI* I maintained, postcondition R is established.

    $\neg B \wedge I \Rightarrow R$

    $$I \wedge \neg B \Rightarrow R$$

- A loop ***terminates*** if:
  - Given *LI* I, and not yet to exit, S_{body} maintains *LV* V as non-negative.

    $\{I \wedge B\}\ S_{body}\ \{V \geq 0\}$

    $$\{I \wedge B\}\ S_{body}\ \{V \geq 0\}$$

  - Given *LI* I, and not yet to exit, S_{body} decrements *LV* V.

    $\{I \wedge B\}\ S_{body}\ \{V < V_0\}$

    $$\{I \wedge B\}\ S_{body}\ \{V < V_0\}$$

```
{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}
```

means variant_post at the end of iteration

Correct Loops: Proof Obligations

Example

```
I(i) == (1 <= i) /\ (i <= 6)
V(i) == 6 - i
--algorithm loop_invariant_test
  variables i = 1, variant_pre = 0, variant_post = 0;
  {
    assert I(i);
    while (i <= 5) {
      variant_pre := V(i);
      i := i + 1;
      variant_post := V(i);
      assert variant_post >= 0;
      assert variant_post < variant_pre;
      assert I(i);
    } ;
  }
```

Specification

not to be included in

Start

① $\{ True \}$ $i := 1$ $\{ i \leq 6 \}$

$1 \leq i \wedge$

② $\{ 1 \leq i \wedge i \leq 6 \wedge i \leq 5 \}$

$i := i + 1$

$\{ 1 \leq i \wedge i \leq 6 \}$

- A loop is ***partially** correct* if:
  - Given precondition $\boldsymbol{Q}$, the initialization step S_{init} establishes ***LI*** I.

    ①

    $$\{Q\}\ S_{init}\ \{I\}$$

  - At the end of S_{body}, if not yet to exit, ***LI*** I is maintained.

    ②

    $$\{I \wedge B\}\ S_{body}\ \{I\}$$

  - If ready to exit and ***LI*** I maintained, postcondition $\boldsymbol{R}$ is established.

    [redacted]

- A loop ***terminates*** if:
  - Given ***LI*** I, and not yet to exit, S_{body} maintains ***LV*** V as non-negative.

    [redacted]

  - Given ***LI*** I, and not yet to exit, S_{body} decrements ***LV*** V.

    [redacted]

$\{Q\}$

loop

$\boxed{S_{extra}}$ assume: no loop

$\{R\}$

$wp(S_{extra}, R)$ -

I hope you enjoyed learning with me

All the best to you !