Introduction

MEB: Prologue, Chapter 1

EECS3342 E: System Specification and Refinement
Fall 2025

CHEN-WEI WANG

Learning Outcomes

This module is designed to help you understand:

- What a *safety-critical* system is
- **Code of Ethics** for Professional Engineers
- What a ***Formal Method*** Is
- ***Verification*** vs. ***Validation***
- ***Model***-Based System Development

What is a Safety-Critical System (SCS)?

- A *safety-critical system (**SCS**)* is a system whose ***failure*** or ***malfunction*** has one (or more) of the following consequences:
  - death or serious injury to **people**
  - loss or severe damage to **equipment/property**
  - harm to the **environment**
- Based on the above definition, do you know of any systems that are ***safety-critical***?

Professional Engineers: Code of Ethics

- *Code of Ethics* is a basic guide for ***professional conduct*** and imposes duties on practitioners, with respect to **society**, **employers**, **clients**, **colleagues** (including employees and subordinates), the **engineering profession** and him or herself.
- It is the duty of a practitioner to act at all times with,
  1. ***fairness*** and ***loyalty*** to the practitioner's associates, employers, clients, subordinates and employees;
  2. ***fidelity*** (i.e., dedication, faithfulness) to public needs;
  3. devotion to ***high ideals*** of personal honour and professional integrity;
  4. ***knowledge*** of developments in the area of professional engineering relevant to any services that are undertaken; and
  5. ***competence*** in the performance of any professional engineering services that are undertaken.
- Consequence of misconduct?
  - ***suspension*** or ***termination*** of professional licenses
  - civil ***law suits***

Source: PEO's Code of Ethics

Developing Safety-Critical Systems

Industrial standards in various domains list *acceptance criteria* for **mission**- or **safety**-critical systems that practitioners need to comply with: e.g.,

Aviation Domain: ***RTCA DO-178C*** "*Software Considerations in Airborne Systems and Equipment Certification*"

Nuclear Domain: ***IEEE 7-4.3.2*** "*Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*"

Two important criteria are:

1. System ***requirements*** are precise and complete
2. System ***implementation*** conforms to the requirements

But how do we accomplish these criteria?

Safety-Critical vs. Mission-Critical?

- ***Critical***:
  A task whose successful completion ensures the success of a larger, more complex operation.
  e.g., Success of a pacemaker ⇒ Regulated heartbeats of a patient
- ***Safety***:
  Being free from danger/injury to or loss of human lives.
- ***Mission***:
  An operation or task assigned by a higher authority.

  Q. Formally relate being ***safety***-critical and ***mission***-critical.
  A.
  - ***safety***-critical ⇒ ***mission***-critical
  - ***mission***-critical ⇏ ***safety***-critical
- Relevant industrial standard: ***RTCA DO-178C*** (replacing RTCA DO-178B in 2012) "*Software Considerations in Airborne Systems and Equipment Certification*"

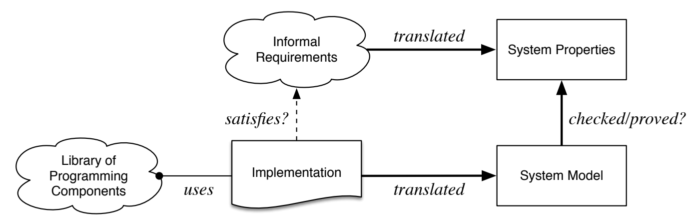
Source: Article from OpenSystems

Using Formal Methods for Certification

- A ***formal method (FM)*** is a ***mathematically rigorous*** technique for the specification, development, and verification of software and hardware systems.
- ***DO-333*** "*Formal methods supplement to DO-178C and DO-278A*" advocates the use of formal methods:

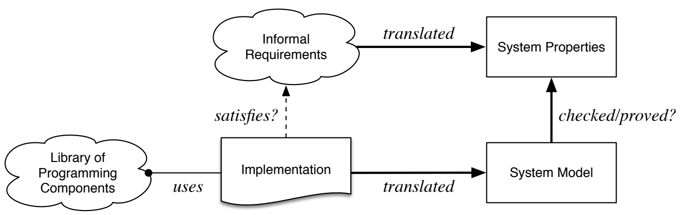
  The use of ***formal methods*** *is motivated by the expectation that, as in other engineering disciplines, performing appropriate* ***mathematical analyses*** *can contribute to establishing the* ***correctness*** *and* ***robustness*** *of a design.*
- FMs, because of their mathematical basis, are capable of:
  - *Unambiguously* describing software system requirements.
  - Enabling *precise* communication between engineers.
  - Providing ***verification (towards certification) evidence*** of:
    - A ***formal*** representation of the system being *healthy*.
    - A ***formal*** representation of the system *satisfying* *safety properties*.

Verification: Building the Product Right?



- ***Implementation*** built via ***reusable programming components***.
- *Goal* : *Implementation Satisfies Intended Requirements*
- To verify this, we ***formalize*** them as a ***system model*** and a set of (e.g., safety) ***properties***, using the specification language of a theorem prover (EECS3342) or a model checker (EECS4315).
- Two Verification Issues:
  1. Library components may ***not behave as intended***.
  2. Successful checks/proofs ensure that we *built the product right*, with respect to the informal requirements. **But**...

Validation: Building the Right Product?



- Successful checks/proofs $\not\Rightarrow$ We ***built the right product***.
- The target of our checks/proofs may not be valid:

  The requirements may be ***ambiguous***, ***incomplete***, or ***contradictory***.
- Solution: ***Precise Documentation*** [EECS4312]

Catching Defects – When?

- To minimize *development costs*, minimize *software defects*.
- Software Development Cycle:

  Requirements → *Design* → *Implementation* → Release

  Q. Design or Implementation Phase?

  Catch defects *as early as possible*.

Design and architecture	Implementation	Integration testing	Customer beta test	Postproduct release
1X*	5X	10X	15X	30X

∵ The cost of fixing defects *increases exponentially* as software progresses through the development lifecycle.

- Discovering *defects* after **release** costs up to 30 times more than catching them in the **design** phase.
- Choice of a *design language*, amendable to ***formal verification***, is therefore critical for your project.

Source: IBM Report

Model-Based System Development

- ***Modelling*** and ***formal reasoning*** should be performed **before** implementing/coding a system.
  - A system's ***model*** is its ***abstraction***, filtering irrelevant details.

    A system ***model*** means as much to a software engineer as a ***blueprint*** means to an architect.
  - A system may have a list of ***models***, "sorted" by **accuracy**:

    $$\langle m_0, m_1, \ldots, \boxed{m_i}, \boxed{m_j}, \ldots, m_n \rangle$$

    - The list starts by the most ***abstract*** model with least details.
    - A more ***abstract*** model $\boxed{m_i}$ is said to be ***refined by*** its subsequent, more ***concrete*** model $\boxed{m_j}$.
    - The list ends with the most ***concrete/refined*** model with most details.
  - It is far easier to reason about:
    - a system's ***abstract*** models (rather than its full ***implementation***)
    - ***refinement*** ***steps*** between subsequent models
- The final product is ***correct by construction***.

Learning through Case Studies

- We will study example ***models of programs/codes***, as well as ***proofs*** on them, drawn from various application domains:
  - REACTIVE Systems [sensors vs. actuators]
  - DISTRIBUTED Systems [(geographically) distributed parties]
- What you learn in this course will allow you to explore example in other application domains:
  - SEQUENTIAL Programs [single thread of control]
  - CONCURRENT Programs [interleaving processes]
- The **Rodin Platform** will be used to:
  - Construct system ***models*** using the Even-B notation.
  - Prove ***properties*** and ***refinements*** using ***classical logic*** (propositional and predicate calculus) and ***set theory***.

Index (1)

Learning Outcomes

What is a Safety-Critical System (SCS)?

Professional Engineers: Code of Ethics

Developing Safety-Critical Systems

Safety-Critical vs. Mission-Critical?

Using Formal Methods to for Certification

Verification: Building the Product Right?

Validation: Building the Right Product?

Catching Defects – When?

Model-Based System Development

Learning through Case Studies

Learning Outcomes of this Lecture

This module is designed to help you **review**:

- Propositional Logic
- Predicate Logic
- Sets, Relations, and Functions

Review of Math

MEB: Chapter 9

EECS3342 E: System Specification and Refinement
Fall 2025

CHEN-WEI WANG

YORK UNIVERSITÉ UNIVERSITY

Propositional Logic (1)

- A *proposition* is a statement of claim that must be of either *true* or *false*, but not both.
- Basic logical operands are of type Boolean: *true* and *false*.
- We use logical operators to construct compound statements.
  - Unary logical operator: negation ($\neg$)

p	$\neg p$
true	*false*
false	*true*

  - Binary logical operators: conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$), equivalence ($\equiv$), and if-and-only-if ($\iff$).

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \iff q$	$p \equiv q$
true	*true*	*true*	*true*	*true*	*true*	*true*
true	*false*	*false*	*true*	*false*	*false*	*false*
false	*true*	*false*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*true*	*true*	*true*

Propositional Logic: Implication (1)

- Written as $p \Rightarrow q$ [pronounced as "p implies q"]
  - We call p the antecedent, assumption, or premise.
  - We call q the consequence or conclusion.
- Compare the *truth* of $p \Rightarrow q$ to whether a contract is *honoured*:
  - antecedent/assumption/premise $p \approx$ promised terms [e.g., salary]
  - consequence/conclusion $q \approx$ obligations [e.g., duties]
- When the promised terms are met, then the contract is:
  - *honoured* if the obligations fulfilled. [$(true \Rightarrow true) \iff true$]
  - *breached* if the obligations violated. [$(true \Rightarrow false) \iff false$]
- When the promised terms are not met, then:
  - Fulfilling the obligation (q) or not ($\neg q$) does *not breach* the contract.

p	q	$p \Rightarrow q$
false	*true*	*true*
false	*false*	*true*

Propositional Logic: Implication (2)

There are alternative, equivalent ways to expressing $p \Rightarrow q$:

- q **if** p

  q is *true* if p is *true*
- p **only if** q

  If p is *true*, then for $p \Rightarrow q$ to be *true*, it can only be that q is also *true*.
  Otherwise, if p is *true* but q is *false*, then $(true \Rightarrow false) \equiv false$.

  Note. To prove $p \equiv q$, prove $p \iff q$ (pronounced: "p if and only if q"):
  - p **if** q [$q \Rightarrow p$]
  - p **only if** q [$p \Rightarrow q$]
- p is **sufficient** for q

  For q to be *true*, it is sufficient to have p being *true*.
- q is **necessary** for p [similar to p **only if** q]

  If p is *true*, then it is necessarily the case that q is also *true*.
  Otherwise, if p is *true* but q is *false*, then $(true \Rightarrow false) \equiv false$.
- q **unless** $\neg p$ [When is $p \Rightarrow q$ *true*?]

  If q is *true*, then $p \Rightarrow q$ *true* regardless of p.
  If q is *false*, then $p \Rightarrow q$ cannot be *true* unless p is *false*.

Propositional Logic: Implication (3)

Given an implication $p \Rightarrow q$, we may construct its:

- **Inverse**: $\neg p \Rightarrow \neg q$ [negate antecedent and consequence]
- **Converse**: $q \Rightarrow p$ [swap antecedent and consequence]
- **Contrapositive**: $\neg q \Rightarrow \neg p$ [inverse of converse]

Propositional Logic (2)

- **Axiom**: Definition of $\Rightarrow$

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

- **Theorem**: Identity of $\Rightarrow$

$$true \Rightarrow p \equiv p$$

- **Theorem**: Zero of $\Rightarrow$

$$false \Rightarrow p \equiv true$$

- **Axiom**: De Morgan

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

- **Axiom**: Double Negation

$$p \equiv \neg(\neg p)$$

- **Theorem**: Contrapositive

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

Predicate Logic (1)

- A *predicate* is a ***universal*** or ***existential*** statement about objects in some universe of disclosure.
- Unlike propositions, predicates are typically specified using ***variables***, each of which declared with some ***range*** of values.
- We use the following symbols for common numerical ranges:
  - $\mathbb{Z}$: the set of integers $[-\infty, \dots, -1, 0, 1, \dots, +\infty]$
  - $\mathbb{N}$: the set of natural numbers $[0, 1, \dots, +\infty]$
- Variable(s) in a predicate may be ***quantified***:
  - *Universal quantification* :
    All values that a variable may take satisfy certain property.
    e.g., Given that i is a natural number, i is *always* non-negative.
  - *Existential quantification* :
    Some value that a variable may take satisfies certain property.
    e.g., Given that i is an integer, i *can be* negative.

Predicate Logic (2.1): Universal Q. ($\forall$)

- A *universal quantification* has the form $(\forall X \bullet R \Rightarrow P)$
  - X is a comma-separated list of variable names
  - R is a *constraint on types/ranges* of the listed variables
  - P is a *property* to be satisfied
- ***For all*** (combinations of) values of variables listed in X that satisfies R, it is the case that P is satisfied.
  - $\forall i \bullet i \in \mathbb{N} \Rightarrow i \geq 0$ [*true*]
  - $\forall i \bullet i \in \mathbb{Z} \Rightarrow i \geq 0$ [*false*]
  - $\forall i, j \bullet i \in \mathbb{Z} \land j \in \mathbb{Z} \Rightarrow i < j \lor i > j$ [*false*]
- *Proof Strategies*
  1. How to prove $(\forall X \bullet R \Rightarrow P)$ ***true***?
     - **Hint**. When is $R \Rightarrow P$ ***true***? [*true* $\Rightarrow$ *true*, *false* $\Rightarrow$ _]
     - Show that for all instances of $x \in X$ s.t. $R(x)$, $P(x)$ holds.
     - Show that for all instances of $x \in X$ it is the case $\neg R(x)$.
  2. How to prove $(\forall X \bullet R \Rightarrow P)$ ***false***?
     - **Hint**. When is $R \Rightarrow P$ ***false***? [*true* $\Rightarrow$ *false*]
     - Give a ***witness/counterexample*** of $x \in X$ s.t. $R(x)$, $\neg P(x)$ holds.

Predicate Logic (2.2): Existential Q. ($\exists$)

- An *existential quantification* has the form $(\exists X \bullet R \land P)$
  - X is a comma-separated list of variable names
  - R is a *constraint on types/ranges* of the listed variables
  - P is a *property* to be satisfied
- ***There exist*** (a combination of) values of variables listed in X that satisfy both R and P.
  - $\exists i \bullet i \in \mathbb{N} \land i \geq 0$ [*true*]
  - $\exists i \bullet i \in \mathbb{Z} \land i \geq 0$ [*true*]
  - $\exists i, j \bullet i \in \mathbb{Z} \land j \in \mathbb{Z} \land (i < j \lor i > j)$ [*true*]
- *Proof Strategies*
  1. How to prove $(\exists X \bullet R \land P)$ ***true***?
     - **Hint**. When is $R \land P$ ***true***? [*true* $\land$ *true*]
     - Give a ***witness*** of $x \in X$ s.t. $R(x)$, $P(x)$ holds.
  2. How to prove $(\exists X \bullet R \land P)$ ***false***?
     - **Hint**. When is $R \land P$ ***false***? [*true* $\land$ *false*, *false* $\land$ _]
     - Show that for all instances of $x \in X$ s.t. $R(x)$, $\neg P(x)$ holds.
     - Show that for all instances of $x \in X$ it is the case $\neg R(x)$.

Predicate Logic (3): Exercises

- Prove or disprove: $\forall x \bullet (x \in \mathbb{Z} \land 1 \leq x \leq 10) \Rightarrow x > 0$.
  All 10 integers between 1 and 10 are greater than 0.
- Prove or disprove: $\forall x \bullet (x \in \mathbb{Z} \land 1 \leq x \leq 10) \Rightarrow x > 1$.
  Integer 1 (a witness/counterexample) in the range between 1 and 10 is *not* greater than 1.
- Prove or disprove: $\exists x \bullet (x \in \mathbb{Z} \land 1 \leq x \leq 10) \land x > 1$.
  Integer 2 (a witness) in the range between 1 and 10 is greater than 1.
- Prove or disprove that $\exists x \bullet (x \in \mathbb{Z} \land 1 \leq x \leq 10) \land x > 10$?
  All integers in the range between 1 and 10 are *not* greater than 10.

Predicate Logic (4): Switching Quantifications

Conversions between $\forall$ and $\exists$:

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

Sets: Definitions and Membership

- A *set* is a collection of objects.
  - Objects in a set are called its *elements* or *members*.
  - *Order* in which elements are arranged does not matter.
  - An element can appear *at most once* in the set.
- We may define a set using:
  - ***Set Enumeration***: Explicitly list all members in a set.
    e.g., $\{1, 3, 5, 7, 9\}$
  - ***Set Comprehension***: Implicitly specify the condition that all members satisfy.
    e.g., $\{x \mid 1 \leq x \leq 10 \wedge x \texttt{ is an odd number}\}$
- An empty set (denoted as $\{\}$ or $\varnothing$) has no members.
- We may check if an element is a *member* of a set:
  e.g., $5 \in \{1, 3, 5, 7, 9\}$ [*true*]
  e.g., $4 \notin \{x \mid x \leq 1 \leq 10, x \texttt{ is an odd number}\}$ [*true*]
- The number of elements in a set is called its *cardinality*.
  e.g., $|\varnothing| = 0$, $|\{x \mid x \leq 1 \leq 10, x \texttt{ is an odd number}\}| = 5$

Set Relations

Given two sets S_1 and S_2:

- S_1 is a ***subset*** of S_2 if every member of S_1 is a member of S_2.

$$S_1 \subseteq S_2 \iff (\forall x \bullet x \in S1 \Rightarrow x \in S2)$$

- S_1 and S_2 are ***equal*** iff they are the subset of each other.

$$S_1 = S_2 \iff S_1 \subseteq S_2 \wedge S_2 \subseteq S_1$$

- S_1 is a ***proper subset*** of S_2 if it is a strictly smaller subset.

$$S_1 \subset S_2 \iff S_1 \subseteq S_2 \wedge |S1| < |S2|$$

Set Relations: Exercises

$? \subseteq S$ always holds	[$\varnothing$ and S]
$? \subset S$ always fails	[S]
$? \subset S$ holds for some S and fails for some S	[$\varnothing$]
$S_1 = S_2 \Rightarrow S_1 \subseteq S_2$?	[Yes]
$S_1 \subseteq S_2 \Rightarrow S_1 = S_2$?	[No]

Set Operations

Given two sets S_1 and S_2:

- ***Union*** of S_1 and S_2 is a set whose members are in either.

$$S_1 \cup S_2 = \{x \mid x \in S_1 \lor x \in S_2\}$$

- ***Intersection*** of S_1 and S_2 is a set whose members are in both.

$$S_1 \cap S_2 = \{x \mid x \in S_1 \land x \in S_2\}$$

- ***Difference*** of S_1 and S_2 is a set whose members are in S_1 but not S_2.

$$S_1 \setminus S_2 = \{x \mid x \in S_1 \land x \notin S_2\}$$

Power Sets

The *power set* of a set S is a *set* of all S's *subsets*.

$$\mathbb{P}(S) = \{s \mid s \subseteq S\}$$

The power set contains subsets of *cardinalities* 0, 1, 2, ..., $|S|$.
e.g., $\mathbb{P}(\{1,2,3\})$ is a set of sets, where each member set s has cardinality 0, 1, 2, or 3:

$$\left\{ \begin{array}{l} \varnothing, \\ \{1\},\ \{2\},\ \{3\}, \\ \{1,2\},\ \{2,3\},\ \{3,1\}, \\ \{1,2,3\} \end{array} \right\}$$

Exercise: What is $\mathbb{P}(\{1,2,3,4,5\}) \setminus \mathbb{P}(\{1,2,3\})$?

Set of Tuples

Given n sets $S_1, S_2, \ldots, S_n$, a ***cross/Cartesian product*** of theses sets is a set of n-tuples.
Each ***n-tuple*** $(e_1, e_2, \ldots, e_n)$ contains n elements, each of which a member of the corresponding set.

$$S_1 \times S_2 \times \cdots \times S_n = \{(e_1, e_2, \ldots, e_n) \mid e_i \in S_i \land 1 \le i \le n\}$$

e.g., $\{a,b\} \times \{2,4\} \times \{\$, \&\}$ is a set of triples:

$$\begin{array}{rl} & \{a,b\} \times \{2,4\} \times \{\$, \&\} \\ = & \{\ (e_1, e_2, e_3) \mid e_1 \in \{a,b\} \land e_2 \in \{2,4\} \land e_3 \in \{\$, \&\}\ \} \\ = & \left\{ \begin{array}{l} (a,2,\$), (a,2,\&), (a,4,\$), (a,4,\&), \\ (b,2,\$), (b,2,\&), (b,4,\$), (b,4,\&) \end{array} \right\} \end{array}$$

Relations (1): Constructing a Relation

A *relation* is a set of mappings, each being an ***ordered pair*** that maps a member of set S to a member of set T.
e.g., Say $S = \{1,2,3\}$ and $T = \{a,b\}$

- $\varnothing$ is the ***minimum*** relation (i.e., an empty relation).
- $\boxed{S \times T}$ is the ***maximum*** relation (say r_1) between S and T, mapping from each member of S to each member in T:

$$\{(1,a), (1,b), (2,a), (2,b), (3,a), (3,b)\}$$

- $\{(x,y) \mid (x,y) \in S \times T \land x \neq 1\}$ is a relation (say r_2) that maps only some members in S to every member in T:

$$\{(2,a), (2,b), (3,a), (3,b)\}$$

Relations (2.1): Set of Possible Relations

- We use the ***power set*** operator to express the set of ***all possible relations*** on S and T:

$$\mathbb{P}(S \times T)$$

  Each member in $\mathbb{P}(S \times T)$ is a relation.

- To declare a relation variable r, we use the colon (`:`) symbol to mean ***set membership***:

$$r : \mathbb{P}(S \times T)$$

- Or alternatively, we write:

$$r : S \leftrightarrow T$$

  where the set $S \leftrightarrow T$ is synonymous to the set $\mathbb{P}(S \times T)$

Relations (2.2): Exercise

Enumerate $\{a, b\} \leftrightarrow \{1, 2, 3\}$.

- **Hints**:
  - You may enumerate all relations in $\mathbb{P}(\{a, b\} \times \{1, 2, 3\})$ via their *cardinalities*: 0, 1, ..., $|\{a, b\} \times \{1, 2, 3\}|$.
  - What's the *maximum* relation in $\mathbb{P}(\{a, b\} \times \{1, 2, 3\})$?

$$\{ (a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3) \}$$

- The answer is a set containing ***all*** of the following relations:
  - Relation with cardinality 0: $\varnothing$
  - How many relations with cardinality 1? $[\binom{|\{a,b\} \times \{1,2,3\}|}{1} = 6]$
  - How many relations with cardinality 2? $[\binom{|\{a,b\} \times \{1,2,3\}|}{2} = \frac{6 \times 5}{2!} = 15]$

  ...

  - Relation with cardinality $|\{a, b\} \times \{1, 2, 3\}|$:

$$\{ (a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3) \}$$

Relations (3.1): Domain, Range, Inverse

Given a relation

r = {(a, 1), (b, 2), (c, 3), (a, 4), (b, 5), (c, 6), (d, 1), (e, 2), (f, 3)}

- ***domain*** **of** r : set of first-elements from r
  - Definition: $\text{dom}(r) = \{ d \mid (d, r') \in r \}$
  - e.g., $\text{dom}(r) = \{a, b, c, d, e, f\}$
  - ASCII syntax: `dom(r)`
- ***range*** **of** r : set of second-elements from r
  - Definition: $\text{ran}(r) = \{ r' \mid (d, r') \in r \}$
  - e.g., $\text{ran}(r) = \{1, 2, 3, 4, 5, 6\}$
  - ASCII syntax: `ran(r)`
- ***inverse*** **of** r : a relation like r with elements swapped
  - Definition: $r^{-1} = \{ (r', d) \mid (d, r') \in r \}$
  - e.g., $r^{-1} = \{(1, a), (2, b), (3, c), (4, a), (5, b), (6, c), (1, d), (2, e), (3, f)\}$
  - ASCII syntax: `r~`

Relations (3.2): Image

Given a relation

r = {(a, 1), (b, 2), (c, 3), (a, 4), (b, 5), (c, 6), (d, 1), (e, 2), (f, 3)}

relational image **of** r **over set** s : sub-range of r mapped by s.

- Definition: $r[s] = \{ r' \mid (d, r') \in r \wedge d \in s \}$
- e.g., $r[\{a, b\}] = \{1, 2, 4, 5\}$
- ASCII syntax: `r[s]`

Relations (3.3): Restrictions

Given a relation

r = {(a, 1), (b, 2), (c, 3), (a, 4), (b, 5), (c, 6), (d, 1), (e, 2), (f, 3)}

- ***domain restriction*** of *r* over set *ds*: sub-relation of *r* with domain *ds*.
  - Definition: $ds \lhd r = \{ (d, r') \mid (d, r') \in r \wedge d \in ds \}$
  - e.g., $\{a, b\} \lhd r = \{(\mathbf{a}, 1), (\mathbf{b}, 2), (\mathbf{a}, 4), (\mathbf{b}, 5)\}$
  - ASCII syntax: `ds <| r`
- ***range restriction*** of *r* over set *rs*: sub-relation of *r* with range *rs*.
  - Definition: $r \rhd rs = \{ (d, r') \mid (d, r') \in r \wedge r' \in rs \}$
  - e.g., $r \rhd \{1, 2\} = \{(a, \mathbf{1}), (b, \mathbf{2}), (d, \mathbf{1}), (e, \mathbf{2})\}$
  - ASCII syntax: `r |> rs`

Relations (3.4): Subtractions

Given a relation

r = {(a, 1), (b, 2), (c, 3), (a, 4), (b, 5), (c, 6), (d, 1), (e, 2), (f, 3)}

- ***domain subtraction*** of *r* over set *ds*: sub-relation of *r* with domain not *ds*.
  - Definition: $ds \ntriangleleft r = \{ (d, r') \mid (d, r') \in r \wedge d \notin ds \}$
  - e.g., $\{a, b\} \ntriangleleft r = \{(\mathbf{c}, 3), (\mathbf{c}, 6), (\mathbf{d}, 1), (\mathbf{e}, 2), (\mathbf{f}, 3)\}$
  - ASCII syntax: `ds <<| r`
- ***range subtraction*** of *r* over set *rs*: sub-relation of *r* with range not *rs*.
  - Definition: $r \ntriangleright rs = \{ (d, r') \mid (d, r') \in r \wedge r' \notin rs \}$
  - e.g., $r \ntriangleright \{1, 2\} = \{(c, \mathbf{3}), (a, \mathbf{4}), (b, \mathbf{5}), (c, \mathbf{6}), (f, \mathbf{3})\}$
  - ASCII syntax: `r |>> rs`

Relations (3.5): Overriding

Given a relation

r = {(a, 1), (b, 2), (c, 3), (a, 4), (b, 5), (c, 6), (d, 1), (e, 2), (f, 3)}

overriding of *r* with relation *t*: a relation which agrees with *t* within $\mathrm{dom}(t)$, and agrees with *r* outside $\mathrm{dom}(t)$

- Definition: $r \mathbin{\lhd\!\!-} t = \{ (d, r') \mid (d, r') \in t \vee ((d, r') \in r \wedge d \notin \mathrm{dom}(t)) \}$
- e.g.,

$$\begin{aligned} & r \mathbin{\lhd\!\!-} \{(a,3),(c,4)\} \\ = \; & \underbrace{\{(a,3),(c,4)\}}_{\{(d,r') \mid (d,r') \in t\}} \cup \underbrace{\{(b,2),(b,5),(d,1),(e,2),(f,3)\}}_{\{(d,r') \mid (d,r') \in r \wedge d \notin \mathrm{dom}(t)\}} \\ = \; & \{(a,3),(c,4),(b,2),(b,5),(d,1),(e,2),(f,3)\} \end{aligned}$$

- ASCII syntax: `r <+ t`

Relations (4): Exercises

1. *Define $r[s]$ in terms of other relational operations.*

   Answer: $r[s] = \mathrm{ran}(s \lhd r)$

   e.g.,

$$r[\underbrace{\{a,b\}}_{s}] = \mathrm{ran}(\underbrace{\{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}}_{\{a,b\} \lhd r}) = \{1,2,4,5\}$$

2. *Define $r \mathbin{\lhd\!\!-} t$ in terms of other relational operators.*

   Answer: $r \mathbin{\lhd\!\!-} t = t \cup (\mathrm{dom}(t) \ntriangleleft r)$

   e.g.,

$$\begin{aligned} & r \mathbin{\lhd\!\!-} \underbrace{\{(a,3),(c,4)\}}_{t} \\ = \; & \underbrace{\{(a,3),(c,4)\}}_{t} \cup \underbrace{\{(b,2),(b,5),(d,1),(e,2),(f,3)\}}_{\underbrace{\mathrm{dom}(t)}_{\{a,c\}} \ntriangleleft r} \\ = \; & \{(a,3),(c,4),(b,2),(b,5),(d,1),(e,2),(f,3)\} \end{aligned}$$

Functions (1): Functional Property

- A ***relation*** r on sets S and T (i.e., $r \in S \leftrightarrow T$) is also a *function* if it satisfies the ***functional property***:

$$\mathit{isFunctional}(r) \iff \forall s, t_1, t_2 \bullet (s \in S \land t_1 \in T \land t_2 \in T) \Rightarrow ((s, t_1) \in r \land (s, t_2) \in r \Rightarrow t_1 = t_2)$$

  - That is, in a ***function***, it is forbidden for a member of S to map to more than one members of T.
  - Equivalently, in a ***function***, two distinct members of T cannot be mapped by the same member of S.
- e.g., Say $S = \{1, 2, 3\}$ and $T = \{a, b\}$, which of the following ***relations*** satisfy the above ***functional property***?
  - $S \times T$ [No]
    Witness 1: $(1, a), (1, b)$; **Witness 2**: $(2, a), (2, b)$; **Witness 3**: $(3, a), (3, b)$.
  - $(S \times T) \setminus \{(x, y) \mid (x, y) \in S \times T \land x = 1\}$ [No]
    Witness 1: $(2, a), (2, b)$; **Witness 2**: $(3, a), (3, b)$
  - $\{(1, a), (2, b), (3, a)\}$ [Yes]
  - $\{(1, a), (2, b)\}$ [Yes]

Functions (2.1): Total vs. Partial

Given a **relation** $r \in S \leftrightarrow T$

- r is a ***partial function*** if it satisfies the ***functional property***:

$$\boxed{r \in S \nrightarrow T} \iff (\mathit{isFunctional}(r) \land \mathrm{dom}(r) \subseteq S)$$

Remark. $r \in S \nrightarrow T$ means there **may (or may not) be** $s \in S$ s.t. $r(s)$ is ***undefined*** (i.e., $r[\{s\}] = \varnothing$).
  - e.g., $\{\ \{(\mathbf{2}, a), (\mathbf{1}, b)\}, \{(\mathbf{2}, a), (\mathbf{3}, a), (\mathbf{1}, b)\}\ \} \subseteq \{1, 2, 3\} \nrightarrow \{a, b\}$
  - ASCII syntax: `r : +->`
- r is a ***total function*** if there is a mapping for each $s \in S$:

$$\boxed{r \in S \rightarrow T} \iff (\mathit{isFunctional}(r) \land \mathrm{dom}(r) = S)$$

Remark. $r \in S \rightarrow T$ implies $r \in S \nrightarrow T$, but not vice versa. Why?
  - e.g., $\{(\mathbf{2}, a), (\mathbf{3}, a), (\mathbf{1}, b)\} \in \{1, 2, 3\} \rightarrow \{a, b\}$
  - e.g., $\{(\mathbf{2}, a), (\mathbf{1}, b)\} \notin \{1, 2, 3\} \rightarrow \{a, b\}$
  - ASCII syntax: `r : -->`

Functions (2.2): Relation Image vs. Function Application

- Recall: A ***function*** is a ***relation***, but a ***relation*** is not necessarily a ***function***.
- Say we have a ***partial function*** $f \in \{1, 2, 3\} \nrightarrow \{a, b\}$:

$$f = \{(\mathbf{3}, a), (\mathbf{1}, b)\}$$

  - With f wearing the ***relation*** hat, we can invoke *relational images*:

$$f[\{3\}] = \{a\}$$
$$f[\{1\}] = \{b\}$$
$$f[\{2\}] = \varnothing$$

Remark. $\Rightarrow |f[\{v\}]| \le 1$ $\because$
  - each member in $\mathrm{dom}(f)$ is mapped to at most one member in $\mathrm{ran}(f)$
  - each input set $\{v\}$ is a **singleton** set
  - With f wearing the ***function*** hat, we can invoke *functional applications*:

$$f(3) = a$$
$$f(1) = b$$
$$f(2) \text{ is } \textbf{undefined}$$

Functions (2.3): Modelling Decision

An organization has a system for keeping **track** of its employees as to where they are on the premises (e.g., ``Zone A, Floor 23''). To achieve this, each employee is issued with an active badge which, when scanned, synchronizes their current positions to a central database.

Assume the following two sets:

- *Employee* denotes the **set** of all employees working for the organization.
- *Location* denotes the **set** of all valid locations in the organization.

1. Is it appropriate to *model/formalize* such a **track** functionality as a ***relation*** (i.e., $\mathit{where\_is} \in \mathit{Employee} \leftrightarrow \mathit{Location}$)?
   Answer. No – an employee cannot be at distinct locations simultaneously.
   e.g., $\mathit{where\_is}[\mathit{Alan}] = \{$ ``Zone A, Floor 23'', ``Zone C, Floor 46'' $\}$
2. How about a ***total function*** (i.e., $\mathit{where\_is} \in \mathit{Employee} \rightarrow \mathit{Location}$)?
   Answer. No – in reality, not necessarily all employees show up.
   e.g., $\mathit{where\_is}(\mathit{Mark})$ should be ***undefined*** if Mark happens to be on vacation.
3. How about a ***partial function*** (i.e., $\mathit{where\_is} \in \mathit{Employee} \nrightarrow \mathit{Location}$)?
   Answer. Yes – this addresses the inflexibility of the total function.

Functions (3.1): Injective Functions

Given a ***function*** f (either partial or total):

- f is ***injective/one-to-one/an injection*** if f does **not** map more than one members of S to a single member of T.

  $\texttt{isInjective}(f)$
  $\iff$
  $\forall s_1, s_2, t \bullet (s_1 \in S \land s_2 \in S \land t \in T) \Rightarrow ((s_1, t) \in f \land (s_2, t) \in f \Rightarrow s_1 = s_2)$

- If f is a ***partial injection***, we write: $\boxed{f \in S \mathrel{\rightarrowtail\!\!\!\!\!\!+} T}$
  - e.g., $\{\ \varnothing, \{(1,\mathbf{a})\}, \{(2,\mathbf{a}),(3,\mathbf{b})\}\ \} \subseteq \{1,2,3\} \mathrel{\rightarrowtail\!\!\!\!\!\!+} \{a,b\}$
  - e.g., $\{(1,\mathbf{b}),(2,a),(3,\mathbf{b})\} \notin \{1,2,3\} \mathrel{\rightarrowtail\!\!\!\!\!\!+} \{a,b\}$ [total, not inj.]
  - e.g., $\{(1,\mathbf{b}),(3,\mathbf{b})\} \notin \{1,2,3\} \mathrel{\rightarrowtail\!\!\!\!\!\!+} \{a,b\}$ [partial, not inj.]
  - ASCII syntax: `f : >+>`
- If f is a ***total injection***, we write: $\boxed{f \in S \rightarrowtail T}$
  - e.g., $\{1,2,3\} \rightarrowtail \{a,b\} = \varnothing$
  - e.g., $\{(2,d),(1,a),(3,c)\} \in \{1,2,3\} \rightarrowtail \{a,b,c,d\}$
  - e.g., $\{(\mathbf{2},d),(\mathbf{1},c)\} \notin \{1,2,3\} \rightarrowtail \{a,b,c,d\}$ [not total, inj.]
  - e.g., $\{(2,\mathbf{d}),(1,c),(3,\mathbf{d})\} \notin \{1,2,3\} \rightarrowtail \{a,b,c,d\}$ [total, not inj.]
  - ASCII syntax: `f : >->`

Functions (3.2): Surjective Functions

Given a ***function*** f (either partial or total):

- f is ***surjective/onto/a surjection*** if f maps to all members of T.

  $$\texttt{isSurjective}(f) \iff \mathrm{ran}(f) = T$$

- If f is a ***partial surjection***, we write: $\boxed{f \in S \mathrel{\twoheadrightarrow\!\!\!\!\!\!\!+} T}$
  - e.g., $\{\ \{(1,\mathbf{b}),(2,\mathbf{a})\}, \{(1,\mathbf{b}),(2,\mathbf{a}),(3,\mathbf{b})\}\ \} \subseteq \{1,2,3\} \mathrel{\twoheadrightarrow\!\!\!\!\!\!\!+} \{a,b\}$
  - e.g., $\{(2,\mathbf{a}),(1,\mathbf{a}),(3,\mathbf{a})\ \} \notin \{1,2,3\} \mathrel{\twoheadrightarrow\!\!\!\!\!\!\!+} \{a,b\}$ [total, not sur.]
  - e.g., $\{(2,\mathbf{b}),(1,\mathbf{b})\} \notin \{1,2,3\} \mathrel{\twoheadrightarrow\!\!\!\!\!\!\!+} \{a,b\}$ [partial, not sur.]
  - ASCII syntax: `f : +->>`
- If f is a ***total surjection***, we write: $\boxed{f \in S \twoheadrightarrow T}$
  - e.g., $\{\ \{(2,a),(1,b),(3,a)\}, \{(2,b),(1,a),(3,b)\}\ \} \subseteq \{1,2,3\} \twoheadrightarrow \{a,b\}$
  - e.g., $\{(\mathbf{2},a),(\mathbf{3},b)\} \notin \{1,2,3\} \twoheadrightarrow \{a,b\}$ [not total, sur.]
  - e.g., $\{(2,\mathbf{a}),(3,\mathbf{a}),(1,\mathbf{a})\} \notin \{1,2,3\} \twoheadrightarrow \{a,b\}$ [total., not sur]
  - ASCII syntax: `f : -->>`

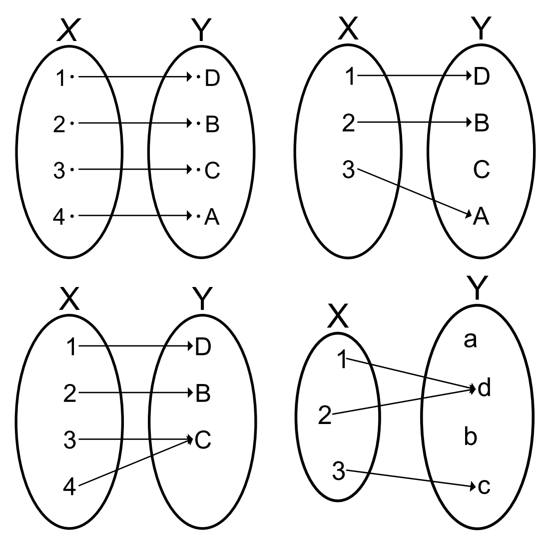
Functions (3.3): Bijective Functions

Given a function f:

f is ***bijective/a bijection****/one-to-one correspondence* if f is ***total***, ***injective***, and ***surjective***.

- e.g., $\{1,2,3\} \mathrel{\rightarrowtail\!\!\!\!\!\twoheadrightarrow} \{a,b\} = \varnothing$
- e.g., $\{\ \{(1,a),(2,b),(3,c)\}, \{(2,a),(3,b),(1,c)\}\ \} \subseteq \{1,2,3\} \mathrel{\rightarrowtail\!\!\!\!\!\twoheadrightarrow} \{a,b,c\}$
- e.g., $\{(\mathbf{2},b),(\mathbf{3},c),(\mathbf{4},a)\} \notin \{1,2,3,4\} \mathrel{\rightarrowtail\!\!\!\!\!\twoheadrightarrow} \{a,b,c\}$ [not total, inj., sur.]
- e.g., $\{(1,\mathbf{a}),(2,b),(3,c),(4,\mathbf{a})\} \notin \{1,2,3,4\} \mathrel{\rightarrowtail\!\!\!\!\!\twoheadrightarrow} \{a,b,c\}$ [total, not inj., sur.]
- e.g., $\{(1,\mathbf{a}),(2,\mathbf{c})\} \notin \{1,2\} \mathrel{\rightarrowtail\!\!\!\!\!\twoheadrightarrow} \{a,b,c\}$ [total, inj., not sur.]
- ASCII syntax: `f : >->>`

Functions (4.1): Exercises



Functions (4.2): Modelling Decisions

1. Should an array a declared as "`String[] a`" be *modelled/formalized* as a *partial* function (i.e., $a \in \mathbb{Z} \nrightarrow String$) or a *total* function (i.e., $a \in \mathbb{Z} \rightarrow String$)?
   Answer. $a \in \mathbb{Z} \rightarrow String$ is not appropriate as:
   - Indices are non-negative (i.e., $a(i)$, where $i < 0$, is ***undefined***).
   - Each array size is finite: not all positive integers are valid indices.
2. What does it mean if an **array** is *modelled/formalized* as a partial ***injection*** (i.e., $a \in \mathbb{Z} \rightarrowtail String$)?
   Answer. It means that the array does **not** contain any duplicates.
3. Can an integer array "`int[] a`" be *modelled/formalized* as a partial ***surjection*** (i.e., $a \in \mathbb{Z} \twoheadrightarrow \mathbb{Z}$)?
   Answer. Yes, if a stores all 2^{32} integers (i.e., $[-2^{31}, 2^{31} - 1]$).
4. Can a string array "`String[] a`" be *modelled/formalized* as a partial ***surjection*** (i.e., $a \in \mathbb{Z} \twoheadrightarrow String$)?
   Answer. No $\because$ # possible strings is ∞.
5. Can an integer array "`int[]`" storing all 2^{32} values be *modelled/formalized* as a ***bijection*** (i.e., $a \in \mathbb{Z} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbb{Z}$)?
   Answer. No, because it cannot be ***total*** (as discussed earlier).

Beyond this lecture . . .

- For the *where_is* $\in$ *Employee* $\nrightarrow$ *Location* model, what does it mean when it is:
  - ***Injective*** [*where_is* $\in$ *Employee* $\rightarrowtail$ *Location*]
  - ***Surjective*** [*where_is* $\in$ *Employee* $\twoheadrightarrow$ *Location*]
  - ***Bijective*** [*where_is* $\in$ *Employee* $\rightarrowtail\!\!\!\!\twoheadrightarrow$ *Location*]
- Review examples discussed in your earlier math courses on ***logic*** and ***set theory***.

Index (1)

Learning Outcomes of this Lecture
Propositional Logic (1)
Propositional Logic: Implication (1)
Propositional Logic: Implication (2)
Propositional Logic: Implication (3)
Propositional Logic (2)
Predicate Logic (1)
Predicate Logic (2.1): Universal Q. (∀)
Predicate Logic (2.2): Existential Q. (∃)
Predicate Logic (3): Exercises
Predicate Logic (4): Switching Quantifications

Index (2)

Sets: Definitions and Membership
Set Relations
Set Relations: Exercises
Set Operations
Power Sets
Set of Tuples
Relations (1): Constructing a Relation
Relations (2.1): Set of Possible Relations
Relations (2.2): Exercise
Relations (3.1): Domain, Range, Inverse
Relations (3.2): Image

Index (3)

Relations (3.3): Restrictions

Relations (3.4): Subtractions

Relations (3.5): Overriding

Relations (4): Exercises

Functions (1): Functional Property

Functions (2.1): Total vs. Partial

Functions (2.2):
Relation Image vs. Function Application

Functions (2.3): Modelling Decision

Functions (3.1): Injective Functions

Functions (3.2): Surjective Functions

Specifying & Refining a Bridge Controller

MEB: Chapter 2

EECS3342 E: System Specification and Refinement
Fall 2025

CHEN-WEI WANG

Index (4)

Functions (3.3): Bijective Functions

Functions (4.1): Exercises

Functions (4.2): Modelling Decisions

Beyond this lecture . . .

Learning Outcomes

This module is designed to help you understand:

- What a ***Requirement Document*** (***RD***) is
- What a ***refinement*** is
- Writing ***formal specifications***
  - (Static) contexts: constants, axioms, theorems
  - (Dynamic) machines: variables, invariants, events, guards, actions
- ***Proof Obligations*** (***POs***) associated with proving:
  - ***refinements***
  - system ***properties***
- Applying ***inference rules*** of the ***sequent calculus***

Recall: Correct by Construction

- Directly reasoning about **source code** (written in a programming language) is too complicated to be feasible.
- Instead, given a ***requirements document***, prior to **implementation**, we develop ***models*** through a series of ***refinement*** steps:
  - Each model formalizes an ***external observer***'s perception of the system.
  - Models are "sorted" with ***increasing levels of accuracy*** w.r.t. the system.
  - The ***first model***, though the most ***abstract***, can already be proved satisfying some ***requirements***.
  - Starting from the ***second model***, each model is analyzed and proved ***correct*** relative to two criteria:
    1. Some ***requirements*** (i.e., R-descriptions)
    2. ***Proof Obligations*** (***POs***) related to the ***preceding model*** being ***refined by*** the ***current model*** (via "extra" ***state*** variables and ***events***).
  - The ***last model*** (which is *correct by construction*) should be **sufficiently close** to be transformed into a **working program** (e.g., in C).

State Space of a Model

- A model's *state space* is the set of **all** configurations:
  - Each ***configuration*** assigns values to **constants** & **variables**, subject to:
    - ***axiom*** (e.g., typing constraints, assumptions)
    - ***invariant*** properties/theorems
  - Say an initial model of a bank system with two constants and a variable:

    $c \in \mathbb{N}1 \wedge L \in \mathbb{N}1 \wedge accounts \in String \mathrel{+\!\!\!\!\to} \mathbb{Z}$ /* typing constraint */

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

    Q. What is the ***state space*** of this initial model?

    A. All valid combinations of c, L, and $accounts$.
    - Configuration 1: $(c = 1,000, L = 500,000, b = \varnothing)$
    - Configuration 2: $(c = 2,375, L = 700,000, b = \{(\text{"}id1\text{"}, 500), (\text{"}id2\text{"}, 1,250)\})$

    ... [Challenge: ***Combinatorial Explosion***]
  - Model Concreteness ↑ ⇒ (State Space ↑ ∧ Verification Difficulty ↑)
- A model's ***complexity*** should be guided by those properties intended to be verified against that model.

  ⇒ ***Infeasible*** to prove **all** desired properties on a model.

  ⇒ ***Feasible*** to distribute desired properties over a list of ***refinements***.

Roadmap of this Module

- We will walk through the *development process* of constructing ***models*** of a control system regulating cars on a bridge.

  Such controllers exemplify a ***reactive system***.

  (with **sensors** and **actuators**)
- Always stay on top of the following roadmap:
  1. A ***Requirements Document*** (***RD***) of the bridge controller
  2. A brief overview of the ***refinement strategy***
  3. An initial, the most ***abstract*** model
  4. A subsequent ***model*** representing the ***1st refinement***
  5. A subsequent ***model*** representing the ***2nd refinement***
  6. A subsequent ***model*** representing the ***3rd refinement***

Requirements Document: Mainland, Island

Imagine you are asked to build a bridge (as an alternative to ferry) connecting the downtown and Toronto Island.

Page Source: `https://soldbyshane.com/area/toronto-islands/`

Requirements Document: E-Descriptions

Each ***E-Description*** is an **atomic** ***specification*** of a ***constraint*** or an ***assumption*** of the system's working environment.

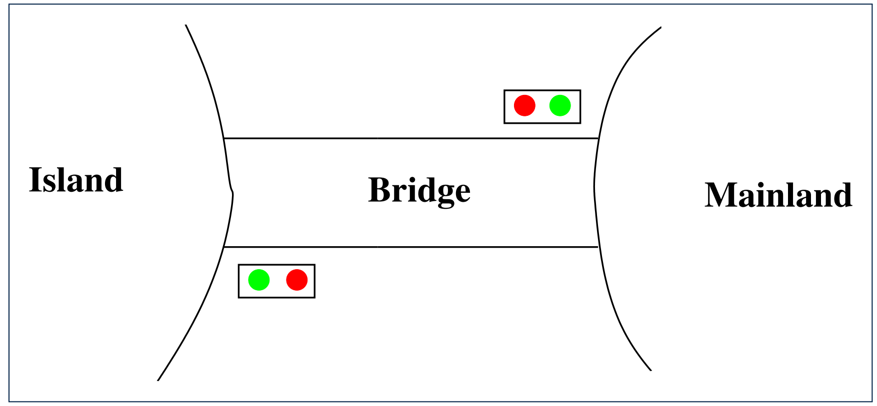
ENV1	The system is equipped with two traffic lights with two colors: green and red.
ENV2	The traffic lights control the entrance to the bridge at both ends of it.
ENV3	Cars are not supposed to pass on a red traffic light, only on a green one.
ENV4	The system is equipped with four sensors with two states: on or off.
ENV5	The sensors are used to detect the presence of a car entering or leaving the bridge: "on" means that a car is willing to enter the bridge or to leave it.

Requirements Document: R-Descriptions

Each ***R-Description*** is an **atomic** ***specification*** of an intended ***functionality*** or a desired ***property*** of the working system.

REQ1	The system is controlling cars on a bridge connecting the mainland to an island.
REQ2	The number of cars on bridge and island is limited.
REQ3	The bridge is one-way or the other, not both at the same time.

Requirements Document: Visual Summary of Equipment Pieces



Refinement Strategy

- Before diving into details of the ***models***, we first clarify the adopted ***design strategy of progressive refinements***.
  0. The ***initial model*** (m_0) will address the intended functionality of a limited number of cars on the island and bridge.
  [**REQ2**]
  1. A ***1st refinement*** (m_1 which ***refines*** m_0) will address the intended functionality of the *bridge being one-way*.
  [**REQ1**, **REQ3**]
  2. A ***2nd refinement*** (m_2 which ***refines*** m_1) will address the environment constraints imposed by *traffic lights*.
  [**ENV1, ENV2, ENV3**]
  3. A ***final, 3rd refinement*** (m_3 which ***refines*** m_2) will address the environment constraints imposed by *sensors* and the *architecture*: controller, environment, communication channels.
  [**ENV4, ENV5**]
- Recall *Correct by Construction* :

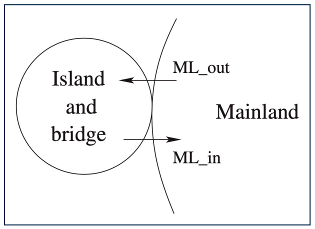
  From each ***model*** to its ***refinement***, only a manageable amount of details are added, making it ***feasible*** to conduct **analysis** and **proofs**.

Model m_0: Abstraction

- In this <u>most</u> ***abstract*** perception of the bridge controller, we do **<u>not</u>** even consider the bridge, traffic lights, and sensors!
- Instead, we focus on this single ***requirement***:

REQ2	The number of cars on bridge and island is limited.

- **<u>Analogies</u>**:
  - Observe the system from the sky: island and bridge appear only as a <u>compound</u>.



  - "***Zoom in***" on the system as ***refinements*** are introduced.

Model m_0: State Space

1. The ***static*** part is fixed and may be seen/imported.
   A ***constant*** d denotes the <u>maximum</u> number of cars allowed to be on the ***island-bridge compound*** at any time.
   (whereas cars on the mainland is <u>unbounded</u>)

   constants: d

   axioms:
   axm0_1 : $d \in \mathbb{N}$

   <u>Remark</u>. ***Axioms*** are <u>assumed true</u> and may be used to prove theorems.
2. The ***dynamic*** part changes as the system ***evolves***.
   A ***variable*** n denotes the actual number of cars, at a given moment, in the ***island-bridge compound***.

   variables: n

   invariants:
   inv0_1 : $n \in \mathbb{N}$
   inv0_2 : $n \leq d$

   <u>Remark</u>. ***Invariants*** should be (subject to **<u>proofs</u>**):
   - ***Established*** when the system is first <u>initialized</u>
   - ***Preserved***/***Maintained*** <u>after</u> any ***<u>enabled</u> event***'s actions take effect

Model m_0: State Transitions via Events

- The system acts as an **ABSTRACT STATE MACHINE (ASM)** : it ***<u>evolves</u>*** as ***actions of <u>enabled</u> events*** change values of variables, subject to ***invariants***.
- At any given ***state*** (a <u>valid</u> ***configuration*** of constants/variables):
  - An event is said to be ***<u>enabled</u>*** if its guard evaluates to ***true***.
  - An event is said to be ***<u>disabled</u>*** if its guard evaluates to ***false***.
  - An ***<u>enabled</u>*** event makes a ***state transition*** if it occurs and its ***actions*** take effect.
- *<u>1st</u>* ***event***: A car <u>exits</u> mainland (and <u>enters</u> the island-bridge <u>compound</u>).

```
ML_out
  begin
    n := n + 1
  end
```

Correct Specification? Say $d = 2$.
<u>Witness</u>: ***Event Trace*** $\langle init, ML\_out, ML\_out, ML\_out \rangle$

- *<u>2nd</u>* ***event***: A car <u>enters</u> mainland (and <u>exits</u> the island-bridge <u>compound</u>).

```
ML_in
  begin
    n := n - 1
  end
```

Correct Specification? Say $d = 2$.
<u>Witness</u>: ***Event Trace*** $\langle init, ML\_in \rangle$

Model m_0: Actions vs. Before-After Predicates

- When an <u>enabled</u> event e occurs there are two notions of ***state***:
  - ***Before-/Pre-State***: Configuration just ***<u>before</u>*** e's actions take effect
  - ***After-/Post-State***: Configuration just ***<u>after</u>*** e's actions take effect

  <u>Remark</u>. When an <u>enabled</u> event occurs, its ***action(s)*** cause a **<u>transition</u>** from the ***pre-state*** to the ***post-state***.
- As examples, consider ***actions*** of m_0's two events:

Events	ML_out $n := n + 1$	ML_in $n := n - 1$
before–after predicates	$n' = n + 1$	$n' = n - 1$

  - An event ***action*** "$n := n + 1$" is <u>not</u> a variable assignment; instead, it is a ***specification***: "*n becomes n + 1 (when the state transition completes)*".
  - The ***before-after predicate*** (***BAP***) "$n' = n + 1$" expresses that n' (the ***post-state*** value of n) is one more than n (the ***pre-state*** value of n).
- When we express ***proof obligations*** (***POs***) associated with ***events***, we use ***BAP***.

Design of Events: Invariant Preservation

- Our design of the two events

```
ML_out
  begin
    n := n + 1
  end
```

```
ML_in
  begin
    n := n - 1
  end
```

only specifies how the ***variable*** n should be updated.

- Remember, ***invariants*** are conditions that should never be ***violated***!

invariants:
  inv0_1 : $n \in \mathbb{N}$
  inv0_2 : $n \leq d$

- By simulating the system as an ***ASM***, we discover ***witnesses*** (i.e., event traces) of the ***invariants*** not being preserved all the time.

$$\exists s \bullet s \in \text{STATE SPACE} \Rightarrow \neg \textit{invariants}(s)$$

- We formulate such a commitment to preserving ***invariants*** as a ***proof obligation (PO)*** rule (a.k.a. a ***verification condition (VC)*** rule).

Sequents: Syntax and Semantics

- We formulate each *PO/VC* rule as a (horizontal or vertical) ***sequent***:

$$H \vdash G \qquad \begin{array}{c} H \\ \vdash \\ G \end{array}$$

  - The symbol $\vdash$ is called the ***turnstile***.
  - H is a set of predicates forming the ***hypotheses/assumptions***. [assumed as ***true***]
  - G is a set of predicates forming the ***goal/conclusion***. [claimed to be ***provable*** from H]
- Informally:
  - $H \vdash G$ is ***true*** if G can be proved by assuming H. [i.e., We say "H ***entails*** G" or "H ***yields*** G"]
  - $H \vdash G$ is ***false*** if G cannot be proved by assuming H.
- Formally: $H \vdash G \iff (H \Rightarrow G)$

  Q. What does it mean when H is empty (i.e., no hypotheses)?

  A. $\vdash G \equiv$ ***true*** $\vdash G$ [Why not $\vdash G \equiv$ ***false*** $\vdash G$?]

PO of Invariant Preservation: Sketch

- Here is a sketch of the PO/VC rule for *invariant preservation*:

Axioms
Invariants Satisfied at *Pre-State*
Guards of the Event
$\vdash$
Invariants Satisfied at *Post-State*

INV

- Informally, this is what the above PO/VC *requires to prove*:

Assuming **all** axioms, invariants, and the event's guards hold at the ***pre-state***,
after the ***state transition*** is made by the event,
all invariants hold at the ***post-state***.

PO of Invariant Preservation: Components

constants: d

variables: n

```
ML_out
  begin
    n := n + 1
  end
```

axioms:
  axm0_1 : $d \in \mathbb{N}$

invariants:
  inv0_1 : $n \in \mathbb{N}$
  inv0_2 : $n \leq d$

```
ML_in
  begin
    n := n - 1
  end
```

- c: list of ***constants*** $\langle d \rangle$
- $A(c)$: list of ***axioms*** $\langle$**axm0_1**$\rangle$
- ***v*** and ***v'***: list of ***variables*** in ***pre***- and ***post***-states $v \cong \langle n \rangle$, $v' \cong \langle n' \rangle$
- $I(c, v)$: list of ***invariants*** $\langle$**inv0_1**, **inv0_2**$\rangle$
- $G(c, v)$: the ***event***'s list of guards

  $G(\langle d \rangle, \langle n \rangle)$ of *ML_out* $\cong \langle$***true***$\rangle$, $G(\langle d \rangle, \langle n \rangle)$ of *ML_in* $\cong \langle$***true***$\rangle$
- $E(c, v)$: effect of the ***event***'s actions i.t.o. what variable values ***become***

  $E(\langle d \rangle, \langle n \rangle)$ of *ML_out* $\cong \langle \mathbf{n+1} \rangle$, $E(\langle d \rangle, \langle n \rangle)$ of *ML_out* $\cong \langle \mathbf{n-1} \rangle$
- $v' = E(c, v)$: ***before-after predicate*** formalizing E's actions

  BAP of *ML_out*: $\langle \mathbf{n'} \rangle = \langle \mathbf{n+1} \rangle$, BAP of *ML_in*: $\langle \mathbf{n'} \rangle = \langle \mathbf{n-1} \rangle$

Rule of Invariant Preservation: Sequents

- Based on the components $(c, A(c), v, I(c,v), E(c,v))$, we are able to formally state the ***PO/VC Rule of Invariant Preservation***:

$$\frac{\begin{array}{l} A(c) \\ I(c, v) \\ G(c, v) \\ \vdash \\ I_i(c, E(c, v)) \end{array}}{} \quad \text{INV}$$

where I_i denotes a single invariant condition

  - Accordingly, how many ***sequents*** to be proved? [# events × # invariants]
  - We have two ***sequents*** generated for ***event*** *ML_out* of model m_0:

$$d \in \mathbb{N},\ n \in \mathbb{N},\ n \le d \vdash n + 1 \in \mathbb{N} \quad \text{ML\_out/inv0\_1/INV}$$

$$d \in \mathbb{N},\ n \in \mathbb{N},\ n \le d \vdash n + 1 \le d \quad \text{ML\_out/inv0\_2/INV}$$

Exercise. Write the ***POs of invariant preservation*** for event *ML_in*.

- Before claiming that a ***model*** is *correct*, outstanding ***sequents*** associated with all ***POs*** must be proved/discharged.

Inference Rules: Syntax and Semantics

- An *inference rule (IR)* has the following form:

$$\frac{A}{C}\ \text{L}$$

Formally: $A \Rightarrow C$ is an axiom.

Informally: To prove C, it is sufficient to prove A instead.

Informally: C is the case, assuming that A is the case.

  - L is a name label for referencing the ***inference rule*** in proofs.
  - A is a **set** of sequents known as ***antecedents*** of rule L.
  - C is a **single** sequent known as ***consequent*** of rule L.
- Let's consider ***inference rules (IRs)*** with two different flavours:

$$\frac{H1 \vdash G}{H1, H2 \vdash G}\ \text{MON} \qquad \frac{}{n \in \mathbb{N} \vdash n + 1 \in \mathbb{N}}\ \text{P2}$$

  - IR **MON**: To prove $H1, H2 \vdash G$, it suffices to prove $H1 \vdash G$ instead.
  - IR **P2**: $n \in \mathbb{N} \vdash n + 1 \in \mathbb{N}$ is an ***axiom***.

[proved automatically without further justifications]

Proof of Sequent: Steps and Structure

- To prove the following sequent (related to ***invariant preservation***):

$$d \in \mathbb{N},\ n \in \mathbb{N},\ n \le d \vdash n + 1 \in \mathbb{N} \quad \text{ML\_out/inv0\_1/INV}$$

1. Apply a ***inference rule***, which ***transforms*** some "outstanding" **sequent** to one or more other **sequents** to be proved instead.
2. Keep applying ***inference rules*** until **all** ***transformed*** **sequents** are ***axioms*** that do **not** require any further justifications.

- Here is a *formal proof* of ML_out/**inv0_1**/INV, by applying IRs **MON** and **P2**:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \le d \\ \vdash \\ n + 1 \in \mathbb{N} \end{array} \quad \text{MON} \quad \begin{array}{l} n \in \mathbb{N} \\ \vdash \\ n + 1 \in \mathbb{N} \end{array} \quad \text{P2}$$

Example Inference Rules (1)

$$\frac{}{\vdash 0 \in \mathbb{N}}\ \text{P1}$$

1st Peano axiom: 0 is a natural number.

$$\frac{}{n \in \mathbb{N} \vdash n + 1 \in \mathbb{N}}\ \text{P2}$$

2nd Peano axiom: $n + 1$ is a natural number, assuming that n is a natural number.

$$\frac{}{0 < n \vdash n - 1 \in \mathbb{N}}\ \text{P2'}$$

$n - 1$ is a natural number, assuming that n is positive.

$$\frac{}{n \in \mathbb{N} \vdash 0 \le n}\ \text{P3}$$

3rd Peano axiom: n is non-negative, assuming that n is a natural number.

Example Inference Rules (2)

$$\frac{}{n < m \vdash n+1 \leq m}\ \textbf{INC}$$

$n+1$ is less than or equal to m,
assuming that n is strictly less than m.

$$\frac{}{n \leq m \vdash n-1 < m}\ \textbf{DEC}$$

$n-1$ is strictly less than m,
assuming that n is less than or equal to m.

Example Inference Rules (3)

$$\frac{H1 \vdash G}{H1, H2 \vdash G}\ \textbf{MON}$$

To prove a goal under certain hypotheses,
it suffices to prove it under less hypotheses.

$$\frac{H, P \vdash R \qquad H, Q \vdash R}{H, P \vee Q \vdash R}\ \textbf{OR\_L}$$

Proof by Cases:
To prove a goal under a disjunctive assumption,
it suffices to prove **independently**
the same goal, twice, under each disjunct.

$$\frac{H \vdash P}{H \vdash P \vee Q}\ \textbf{OR\_R1}$$

To prove a disjunction,
it suffices to prove the left disjunct.

$$\frac{H \vdash Q}{H \vdash P \vee Q}\ \textbf{OR\_R2}$$

To prove a disjunction,
it suffices to prove the right disjunct.

Revisiting Design of Events: *ML_out*

- Recall that we already proved ***PO*** ML_out/**inv0_1**/INV:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n+1 \in \mathbb{N} \end{array} \quad \textbf{MON} \quad \begin{array}{l} n \in \mathbb{N} \\ \vdash \\ n+1 \in \mathbb{N} \end{array} \quad \textbf{P2}$$

∴ ***ML_out/inv0_1/INV*** succeeds in being discharged.

- How about the other ***PO*** ML_out/**inv0_2**/INV for the same event?

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n+1 \leq d \end{array} \quad \textbf{MON} \quad \begin{array}{l} n \leq d \\ \vdash \\ n+1 \leq d \end{array} \quad \textbf{?}$$

∴ ***ML_out/inv0_2/INV*** fails to be discharged.

Revisiting Design of Events: *ML_in*

- How about the ***PO*** ML_in/**inv0_1**/INV for *ML_in*:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n-1 \in \mathbb{N} \end{array} \quad \textbf{MON} \quad \begin{array}{l} n \in \mathbb{N} \\ \vdash \\ n-1 \in \mathbb{N} \end{array} \quad \textbf{?}$$

∴ ***ML_in/inv0_1/INV*** fails to be discharged.

- How about the other ***PO*** ML_in/**inv0_2**/INV for the same event?

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n-1 \leq d \end{array} \quad \textbf{MON} \quad \begin{array}{l} n \leq d \\ \vdash \\ n-1 < d \vee n-1 = d \end{array} \quad \textbf{OR\_1} \quad \begin{array}{l} n \leq d \\ \vdash \\ n-1 < d \end{array} \quad \textbf{DEC}$$

∴ ***ML_in/inv0_2/INV*** succeeds in being discharged.

Fixing the Design of Events

- Proofs of ***ML_out/inv0_2/INV*** and ***ML_in/inv0_1/INV*** fail due to the two events being *enabled when they should not*.
- Having this feedback, we add proper ***guards*** to *ML_out* and *ML_in*:

```
ML_out
  when
    n < d
  then
    n := n + 1
  end
```

```
ML_in
  when
    n > 0
  then
    n := n - 1
  end
```

- Having changed both events, updated ***sequents*** will be generated for the PO/VC rule of ***invariant preservation***.
- All ***sequents*** ($\{ML\_out, ML\_in\} \times \{\textbf{inv0\_1}, \textbf{inv0\_2}\}$) now ***provable***?

Revisiting Fixed Design of Events: *ML_out*

- How about the ***PO*** ML_out/**inv0_1**/INV for *ML_out*:

$$\begin{array}{|l|}\hline d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \le d \\ n < d \\ \vdash \\ n+1 \in \mathbb{N} \\ \hline\end{array} \quad \textbf{MON} \quad \begin{array}{|l|}\hline n \in \mathbb{N} \\ \vdash \\ n+1 \in \mathbb{N} \\ \hline\end{array} \quad \textbf{P2}$$

$\therefore$ ***ML_out/inv0_1/INV*** still succeeds in being discharged!

- How about the other ***PO*** ML_out/**inv0_2**/INV for the same event?

$$\begin{array}{|l|}\hline d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \le d \\ n < d \\ \vdash \\ n+1 \le d \\ \hline\end{array} \quad \textbf{MON} \quad \begin{array}{|l|}\hline n < d \\ \vdash \\ n+1 \le d \\ \hline\end{array} \quad \textbf{INC}$$

$\therefore$ ***ML_out/inv0_2/INV*** now succeeds in being discharged!

Revisiting Fixed Design of Events: *ML_in*

- How about the ***PO*** ML_in/**inv0_1**/INV for *ML_in*:

$$\begin{array}{|l|}\hline d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \le d \\ n > 0 \\ \vdash \\ n-1 \in \mathbb{N} \\ \hline\end{array} \quad \textbf{MON} \quad \begin{array}{|l|}\hline n > 0 \\ \vdash \\ n-1 \in \mathbb{N} \\ \hline\end{array} \quad \textbf{P2'}$$

$\therefore$ ***ML_in/inv0_1/INV*** now succeeds in being discharged!

- How about the other ***PO*** ML_in/**inv0_2**/INV for the same event?

$$\begin{array}{|l|}\hline d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \le d \\ n > 0 \\ \vdash \\ n-1 \le d \\ \hline\end{array} \quad \textbf{MON} \quad \begin{array}{|l|}\hline n \le d \\ \vdash \\ n-1 < d \lor n-1 = d \\ \hline\end{array} \quad \textbf{OR\_1} \quad \begin{array}{|l|}\hline n \le d \\ \vdash \\ n-1 < d \\ \hline\end{array} \quad \textbf{DEC}$$

$\therefore$ ***ML_in/inv0_2/INV*** still succeeds in being discharged!

Initializing the Abstract System m_0

- Discharging the four ***sequents*** proved that both ***invariant*** conditions are ***preserved*** between occurrences/interleavings of ***events*** `ML_out` and `ML_in`.
- But how are the ***invariants established*** in the first place?

  Analogy. Proving P via ***mathematical induction***, two cases to prove:
  - $P(1), P(2), \ldots$ [***base*** cases ≈ ***establishing*** inv.]
  - $P(n) \Rightarrow P(n+1)$ [***inductive*** cases ≈ ***preserving*** inv.]
- Therefore, we specify how the ***ASM***'s ***initial state*** looks like:

```
init
  begin
    n := 0
  end
```

✓ The IB compound, once ***initialized***, has no cars.

✓ Initialization always possible: guard is ***true***.

✓ There is no ***pre-state*** for *init*.

$\therefore$ The RHS of := must not involve variables.

$\therefore$ The RHS of := may only involve constants.

✓ There is only the ***post-state*** for *init*.

$\therefore$ Before-***After Predicate***: $n' = 0$

PO of Invariant Establishment

init
begin
$n := 0$
end

- ✓ An ***reactive system***, once ***initialized***, should never terminate.
- ✓ Event *init* cannot "preserve" the ***invariants***.
  ∵ State before its occurrence (***pre-state***) does not exist.
- ✓ Event *init* only required to ***establish*** invariants for the first time

- A new formal component is needed:
  - $K(c)$: effect of ***init***'s actions i.t.o. what variable values ***become***
    e.g., $K(\langle d \rangle)$ of *init* $\widehat{=} \langle 0 \rangle$
  - $v' = K(c)$: ***before-after predicate*** formalizing *init*'s actions
    e.g., BAP of *init*: $\langle \mathbf{n'} \rangle = \langle 0 \rangle$
- Accordingly, PO of ***invariant establishment*** is formulated as a ***sequent***:

Axioms ⊢ *Invariants* Satisfied at *Post-State*	INV	$A(c) \vdash I_i(c, \mathbf{K(c)})$	INV

Discharging PO of Invariant Establishment

- How many ***sequents*** to be proved? [# invariants]
- We have two ***sequents*** generated for ***event*** *init* of model m_0:

$d \in \mathbb{N} \vdash 0 \in \mathbb{N}$ init/inv0_1/INV

$d \in \mathbb{N} \vdash 0 \le d$ init/inv0_2/INV

- Can we discharge the ***PO*** init/**inv0_1**/INV ?

$d \in \mathbb{N} \vdash 0 \in \mathbb{N}$ **MON** $\vdash 0 \in \mathbb{N}$ **P1**

∴ ***init/inv0_1/INV*** succeeds in being discharged.

- Can we discharge the ***PO*** init/**inv0_2**/INV ?

$d \in \mathbb{N} \vdash 0 \le d$ **P3**

∴ ***init/inv0_2/INV*** succeeds in being discharged.

System Property: Deadlock Freedom

- So far we have proved that our initial model m_0 is s.t. all ***invariant conditions*** are:
  - Established when system is first initialized via *init*
  - Preserved whenevner there is a ***state transition***
    (via an enabled event: *ML_out* or *ML_in*)
- However, whenever ***event occurrences*** are conditional (i.e., ***guards*** stronger than ***true***), there is a possibility of ***deadlock***:
  - A state where ***guards*** of all events evaluate to ***false***
  - When a ***deadlock*** happens, none of the ***events*** is ***enabled***.
    ⇒ The system is blocked and not reactive anymore!
- We express this ***non-blocking*** property as a new requirement:

REQ4	Once started, the system should work for ever.

PO of Deadlock Freedom (1)

- Recall some of the formal components we discussed:
  - c: list of ***constants*** $\langle d \rangle$
  - $A(c)$: list of ***axioms*** $\langle \mathbf{axm0\_1} \rangle$
  - $\mathbf{v}$ and $\mathbf{v'}$: list of ***variables*** in ***pre***- and ***post***-states $\mathbf{v} \mathrel{\widehat{=}} \langle n \rangle$, $\mathbf{v'} \mathrel{\widehat{=}} \langle n' \rangle$
  - $I(c, v)$: list of ***invariants*** $\langle \mathbf{inv0\_1}, \mathbf{inv0\_2} \rangle$
  - $G(c, v)$: the event's list of ***guards***
    $G(\langle d \rangle, \langle n \rangle)$ of *ML_out* $\widehat{=} \langle n < d \rangle$, $G(\langle d \rangle, \langle n \rangle)$ of *ML_in* $\widehat{=} \langle n > 0 \rangle$
- A system is ***deadlock-free*** if at least one of its ***events*** is ***enabled***:

Axioms, *Invariants* Satisfied at *Pre-State* ⊢ Disjunction of the guards satisfied at *Pre-State*	DLF	$A(c), I(c, \mathbf{v}) \vdash G_1(c, \mathbf{v}) \vee \cdots \vee G_m(c, \mathbf{v})$	DLF

To prove about deadlock freedom
- An event's effect of state transition is **not** relevant.
- Instead, the evaluation of all events' ***guards*** at the ***pre-state*** is relevant.

PO of Deadlock Freedom (2)

- ***Deadlock freedom*** is not necessarily a desired property.
  ⇒ When it is (like m_0), then the generated ***sequents*** must be discharged.
- Applying the PO of ***deadlock freedom*** to the initial model m_0:

$$\begin{array}{|l|}\hline A(c)\\ I(c,v)\\ \vdash\\ G_1(c,v)\vee\cdots\vee G_m(c,v)\\ \hline\end{array} \quad \underline{\text{DLF}} \quad \begin{array}{|l|}\hline d\in\mathbb{N}\\ n\in\mathbb{N}\\ n\le d\\ \vdash\\ n<d\vee n>0\\ \hline\end{array} \quad \underline{\text{DLF}}$$

  Our bridge controller being ***deadlock-free*** means that cars can ***always*** enter (via *ML_out*) or leave (via *ML_in*) the island-bridge compound.

- Can we formally discharge this ***PO*** for our ***initial model*** m_0?

Example Inference Rules (4)

$$\frac{}{H, P \vdash P}\ \textbf{HYP}$$

A goal is proved if it can be assumed.

$$\frac{}{\bot \vdash P}\ \textbf{FALSE\_L}$$

Assuming ***false*** ($\bot$),
anything can be proved.

$$\frac{}{P \vdash \top}\ \textbf{TRUE\_R}$$

true ($\top$) is proved,
regardless of the assumption.

$$\frac{}{P \vdash E = E}\ \textbf{EQ}$$

An expression being equal to itself is proved,
regardless of the assumption.

Example Inference Rules (5)

$$\frac{H(F),\ E = F \ \vdash\ P(F)}{H(E),\ E = F \ \vdash\ P(E)}\ \textbf{EQ\_LR}$$

To prove a goal $P(E)$ assuming $H(E)$,
where both P and H depend on expression E,
it suffices to prove $P(F)$ assuming $H(F)$,
where both P and H depend on expresion F,
given that E is equal to F.

$$\frac{H(E),\ E = F \ \vdash\ P(E)}{H(F),\ E = F \ \vdash\ P(F)}\ \textbf{EQ\_RL}$$

To prove a goal $P(F)$ assuming $H(F)$,
where both P and H depend on expression F,
it suffices to prove $P(E)$ assuming $H(E)$,
where both P and H depend on expresion E,
given that E is equal to F.

Discharging PO of DLF: Exercise

$$\begin{array}{|l|}\hline A(c)\\ I(c,v)\\ \vdash\\ G_1(c,v)\vee\cdots\vee G_m(c,v)\\ \hline\end{array} \quad \underline{\text{DLF}} \quad \begin{array}{|l|}\hline d\in\mathbb{N}\\ n\in\mathbb{N}\\ n\le d\\ \vdash\\ n<d\vee n>0\\ \hline\end{array} \quad ??$$

Discharging PO of DLF: First Attempt

$$d \in \mathbb{N},\; n \in \mathbb{N},\; n \leq d \vdash n < d \lor n > 0$$

$\equiv$

- $d \in \mathbb{N},\; n \in \mathbb{N},\; n < d \lor n = d \vdash n < d \lor n > 0$ **MON** $n < d \lor n = d \vdash n < d \lor n > 0$ **OR_L**
  - $n < d \vdash n < d \lor n > 0$ **OR_R1** $n < d \vdash n < d$ **HYP**
  - $n = d \vdash n < d \lor n > 0$ **EQ_LR, MON** $\vdash d < d \lor d > 0$ **OR_R2** $\vdash d > 0$ **?**

Why Did the DLF PO Fail to Discharge?

- In our first attempt, proof of the 2nd case failed: $\vdash d > 0$
- This ***unprovable*** sequent gave us a good hint:
  - For the model under consideration (m_0) to be ***deadlock-free***, it is required that $d > 0$. [≥ 1 car allowed in the IB compound]
  - But current ***specification*** of m_0 ***not*** strong enough to entail this:
    - $\neg(d > 0) \equiv d \leq 0$ is possible for the current model
    - Given **axm0_1** : $d \in \mathbb{N}$

    $\Rightarrow d = 0$ is allowed by m_0 which causes a ***deadlock***.
- Recall the *init* event and the two ***guarded*** events:

```
init
  begin
    n := 0
  end

ML_out
  when
    n < d
  then
    n := n + 1
  end

ML_in
  when
    n > 0
  then
    n := n − 1
  end
```

When $d = 0$, the disjunction of guards evaluates to ***false***: $0 < 0 \lor 0 > 0$
$\Rightarrow$ As soon as the system is initialized, it ***deadlocks immediately***
as no car can either enter or leave the IR compound!!

Fixing the Context of Initial Model

- Having understood the <u>failed</u> proof, we add a proper ***axiom*** to m_0:

  axioms:
  axm0_2 : $d > 0$

- We have effectively elaborated on **REQ2**:

REQ2	The number of cars on bridge and island is limited but positive.

- Having changed the context, an <u>updated</u> ***sequent*** will be generated for the PO/VC rule of ***deadlock freedom***.
- Is this new sequent now ***provable***?

Discharging PO of DLF: Second Attempt

$$d \in \mathbb{N},\; d > 0,\; n \in \mathbb{N},\; n \leq d \vdash n < d \lor n > 0$$

$\equiv$

- $d \in \mathbb{N},\; d > 0,\; n \in \mathbb{N},\; n < d \lor n = d \vdash n < d \lor n > 0$ **MON** $d > 0,\; n < d \lor n = d \vdash n < d \lor n > 0$ **OR_L**
  - $d > 0,\; n < d \vdash n < d \lor n > 0$ **OR_R1** $d > 0,\; n < d \vdash n < d$ **HYP**
  - $d > 0,\; n = d \vdash n < d \lor n > 0$ **EQ_LR, MON** $d > 0 \vdash d < d \lor d > 0$ **OR_R2** $d > 0 \vdash d > 0$ **HYP**

Initial Model: Summary

- The final version of our ***initial model*** m_0 is ***provably correct*** w.r.t.:
  - Establishment of ***Invariants***
  - Preservation of ***Invariants***
  - ***Deadlock*** Freedom
- Here is the final ***specification*** of m_0:

constants: d

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

variables: n

invariants:
inv0_1 : $n \in \mathbb{N}$
inv0_2 : $n \leq d$

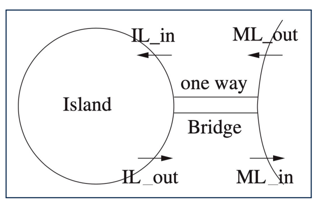
init
begin
$n := 0$
end

ML_out
when
$n < d$
then
$n := n + 1$
end

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

Model m_1: "More Concrete" Abstraction

- First ***refinement*** has a more ***concrete*** perception of the bridge controller:
  - We "**zoom in**" by observing the system from **closer to the ground**, so that the island-bridge compound is split into:
    - the island
    - the (one-way) bridge



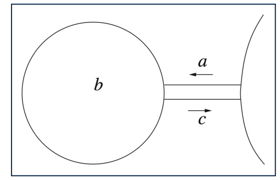
  - Nonetheless, traffic lights and sensors remain ***abstracted*** away!
- That is, we focus on these two ***requirement***:

REQ1	The system is controlling cars on a bridge connecting the mainland to an island.
REQ3	The bridge is one-way or the other, not both at the same time.

- We are ***obliged to prove*** this ***added concreteness*** is ***consistent*** with m_0.

Model m_1: Refined State Space

1. The **static** part is the same as m_0's: **constants:** d **axioms:** **axm0_1** : $d \in \mathbb{N}$ **axm0_2** : $d > 0$
2. The **dynamic** part of the ***concrete state*** consists of three ***variables***:



- ***a***: number of cars on the bridge, heading to the island
- ***b***: number of cars on the island
- ***c***: number of cars on the bridge, heading to the mainland

variables: a, b, c

invariants:
inv1_1 : $a \in \mathbb{N}$
inv1_2 : $b \in \mathbb{N}$
inv1_3 : $c \in \mathbb{N}$
inv1_4 : ??
inv1_5 : ??

✓ **inv1_1**, **inv1_2**, **inv1_3** are ***typing*** constraints.

✓ **inv1_4** ***links***/***glues*** the ***abstract*** and ***concrete*** states.

✓ **inv1_5** specifies that the bridge is one-way.

Model m_1: State Transitions via Events

- The system acts as an **ABSTRACT STATE MACHINE (ASM)**: it ***evolves*** as ***actions of enabled events*** change values of variables, subject to ***invariants***.
- We first consider the "old" ***events*** already existing in m_0.
- ***Concrete***/***Refined*** version of ***event*** *ML_out*:

ML_out
when
??
then
$a := a + 1$
end

  - Meaning of *ML_out* is ***refined***: a car exits mainland (getting on the bridge).
  - *ML_out* ***enabled*** only when:
    - the bridge's current traffic flows to the island
    - number of cars on both the bridge and the island is limited
- ***Concrete***/***Refined*** version of ***event*** *ML_in*:

ML_in
when
??
then
$c := c - 1$
end

  - Meaning of *ML_in* is ***refined***: a car enters mainland (getting off the bridge).
  - *ML_in* ***enabled*** only when:

    there is some car on the bridge heading to the mainland.

Model m_1: Actions vs. Before-After Predicates

- Consider the ***concrete***/***refined*** version of ***actions*** of m_0's two events:

Events	ML_in **when** $0 < c$ **then** $c := c - 1$ **end**	ML_out **when** $a + b < d$ $c = 0$ **then** $a := a + 1$ **end**
Before–after predicates	$a' = a \land b' = b \land c' = c - 1$	$a' = a + 1 \land b' = b \land c' = c$

  - An event's ***actions*** are a **specification**: "c becomes c - 1 after the transition".
  - The ***before-after predicate*** (***BAP***) "$c' = c - 1$" expresses that c' (the ***post-state*** value of c) is one less than c (the ***pre-state*** value of c).
  - Given that the ***concrete state*** consists of three variables:
    - An event's ***actions*** only specify those changing from ***pre***-state to ***post***-state. [e.g., $c' = c - 1$]
    - Other unmentioned variables have their ***post***-state values remain unchanged. [e.g., $a' = a \land b' = b$]
- When we express ***proof obligations*** (***POs***) associated with ***events***, we use ***BAP***.

States & Invariants: Abstract vs. Concrete

- m_1 refines m_0 by introducing more ***variables***:
  - ***Abstract*** State (of m_0 being refined): **variables:** n
  - ***Concrete*** State (of the refinement model m_1): **variables:** a, b, c
- Accordingly, ***invariants*** may involve different ***states***:
  - ***Abstract*** Invariants (involving the ***abstract*** state only):

```
invariants:
  inv0_1 : n ∈ ℕ
  inv0_2 : n ≤ d
```

  - ***Concrete*** Invariants (involving at least the ***concrete*** state):

```
invariants:
  inv1_1 : a ∈ ℕ
  inv1_2 : b ∈ ℕ
  inv1_3 : c ∈ ℕ
  inv1_4 : a + b + c = n
  inv1_5 : a = 0 ∨ c = 0
```


Events: Abstract vs. Concrete

- When an ***event*** exists in both models m_0 and m_1, there are two versions of it:
  - The ***abstract*** version modifies the ***abstract*** state.

(abstract_)ML_out **when** $n < d$ **then** $n := n + 1$ **end**	(abstract_)ML_in **when** $n > 0$ **then** $n := n - 1$ **end**

  - The ***concrete*** version modifies the ***concrete*** state.

(concrete_)ML_out **when** $a + b < d$ $c = 0$ **then** $a := a + 1$ **end**	(concrete_)ML_in **when** $c > 0$ **then** $c := c - 1$ **end**

- A ***new event*** may **only** exist in m_1 (the ***concrete*** model): we will deal with this kind of events later, separately from "redefined/overridden" events.

PO of Refinement: Components (1)

```
constants: d

axioms:
  axm0_1 : d ∈ ℕ
  axm0_2 : d > 0

variables: a, b, c

invariants:
  inv1_1 : a ∈ ℕ
  inv1_2 : b ∈ ℕ
  inv1_3 : c ∈ ℕ
  inv1_4 : a + b + c = n
  inv1_5 : a = 0 ∨ c = 0

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

ML_in
  when
    c > 0
  then
    c := c − 1
  end
```

- c: list of ***constants*** $\langle d \rangle$
- $A(c)$: list of ***axioms*** $\langle$**axm0_1**$\rangle$
- v and v': ***abstract variables*** in pre- & post-states $v \mathrel{\hat=} \langle n \rangle, v' \mathrel{\hat=} \langle n \rangle$
- w and w': ***concrete variables*** in pre- & post-states $w \mathrel{\hat=} \langle a, b, c \rangle, w' \mathrel{\hat=} \langle a', b', c' \rangle$
- $I(c, v)$: list of ***abstract invariants*** $\langle$**inv0_1**, **inv0_2**$\rangle$
- $J(c, v, w)$: list of ***concrete invariants*** $\langle$**inv1_1**, **inv1_2**, **inv1_3**, **inv1_4**, **inv1_5**$\rangle$

PO of Refinement: Components (2)

constants: d

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

variables: a, b, c

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

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

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

- $G(c, v)$: list of guards of the ***abstract event***

$$G(\langle d \rangle, \langle n \rangle) \text{ of } ML\_out \mathrel{\widehat{=}} \langle n < d \rangle,\ G(c, v) \text{ of } ML\_in \mathrel{\widehat{=}} \langle n > 0 \rangle$$

- $H(c, w)$: list of guards of the ***concrete event***

$$H(\langle d \rangle, \langle a, b, c \rangle) \text{ of } ML\_out \mathrel{\widehat{=}} \langle a + b < d, c = 0 \rangle,\ H(c, w) \text{ of } ML\_in \mathrel{\widehat{=}} \langle c > 0 \rangle$$

PO of Refinement: Components (3)

constants: d

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

variables: a, b, c

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

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

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

- $E(c, v)$: effect of the ***abstract event***'s actions i.t.o. what variable values **become**

$$E(\langle d \rangle, \langle n \rangle) \text{ of } ML\_out \mathrel{\widehat{=}} \langle \mathbf{n} + 1 \rangle,\ E(\langle d \rangle, \langle n \rangle) \text{ of } ML\_in \mathrel{\widehat{=}} \langle \mathbf{n} - 1 \rangle$$

- $F(c, w)$: effect of the ***concrete event***'s actions i.t.o. what variable values **become**

$$F(c, w) \text{ of } ML\_out \mathrel{\widehat{=}} \langle \mathbf{a} + 1, b, c \rangle,\ F(c, w) \text{ of } ML\_in \mathrel{\widehat{=}} \langle a, b, \mathbf{c} - 1 \rangle$$

Sketching PO of Refinement

The PO/VC rule for a *proper refinement* consists of two parts:

1. ***Guard Strengthening***

   Axioms
   Abstract Invariants Satisfied at Pre-State
   Concrete Invariants Satisfied at Pre-State
   Guards of the *Concrete Event*
   $\vdash$
   Guards of the *Abstract Event*
   — GRD

   - A ***concrete*** transition always has an ***abstract*** counterpart.
   - A ***concrete*** event is enabled *only if* ***abstract*** counterpart is enabled.

2. ***Invariant Preservation***

   Axioms
   Abstract Invariants Satisfied at Pre-State
   Concrete Invariants Satisfied at Pre-State
   Guards of the *Concrete Event*
   $\vdash$
   Concrete Invariants Satisfied at **Post**-State
   — INV

   - A ***concrete*** event performs a ***transition*** on ***concrete*** states.
   - This ***concrete*** state ***transition*** must be consistent with how its ***abstract*** counterpart performs a corresponding ***abstract transition***.

Note. ***Guard strengthening*** and ***invariant preservation*** are only applicable to events that might be ***enabled*** after the system is launched.
The special, non-guarded `init` event will be discussed separately later.

Refinement Rule: Guard Strengthening

- Based on the components, we are able to formally state the ***PO/VC Rule of Guard Strengthening for Refinement***:

$$\frac{A(c) \quad I(c, \mathbf{v}) \quad J(c, \mathbf{v}, \mathbf{w}) \quad H(c, \mathbf{w}) \vdash G_i(c, \mathbf{v})}{} \quad \text{GRD}$$

where G_i denotes a single ***guard*** condition of the ***abstract*** event

  - How many ***sequents*** to be proved? [# *abstract* guards]
  - For *ML_out*, only one ***abstract*** guard, so one ***sequent*** is generated :

$$\begin{array}{l} d \in \mathbb{N} \quad d > 0 \\ n \in \mathbb{N} \quad n \le d \\ a \in \mathbb{N} \quad b \in \mathbb{N} \quad c \in \mathbb{N} \quad a + b + c = n \quad a = 0 \lor c = 0 \\ a + b < d \quad c = 0 \\ \vdash \\ n < d \end{array} \quad \text{ML\_out/GRD}$$

- **Exercise**. Write *ML_in*'s ***PO of Guard Strengthening for Refinement***.

PO Rule: Guard Strengthening of *ML_out*

axm0_1	$\{\ d \in \mathbb{N}$
axm0_2	$\{\ d > 0$
inv0_1	$\{\ n \in \mathbb{N}$
inv0_2	$\{\ n \leq d$
inv1_1	$\{\ a \in \mathbb{N}$
inv1_2	$\{\ b \in \mathbb{N}$
inv1_3	$\{\ c \in \mathbb{N}$
inv1_4	$\{\ a + b + c = n$
inv1_5	$\{\ a = 0 \lor c = 0$
Concrete guards of *ML_out*	$\{\ a + b < d$, $c = 0$
	$\vdash$
Abstract guards of *ML_out*	$\{\ n < d$

ML_out/GRD

PO Rule: Guard Strengthening of *ML_in*

axm0_1	$\{\ d \in \mathbb{N}$
axm0_2	$\{\ d > 0$
inv0_1	$\{\ n \in \mathbb{N}$
inv0_2	$\{\ n \leq d$
inv1_1	$\{\ a \in \mathbb{N}$
inv1_2	$\{\ b \in \mathbb{N}$
inv1_3	$\{\ c \in \mathbb{N}$
inv1_4	$\{\ a + b + c = n$
inv1_5	$\{\ a = 0 \lor c = 0$
Concrete guards of *ML_in*	$\{\ c > 0$
	$\vdash$
Abstract guards of *ML_in*	$\{\ n > 0$

ML_in/GRD

Proving Refinement: ML_out/GRD

$$\begin{array}{l} d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \lor c = 0 \\ a + b < d \\ c = 0 \\ \vdash \\ n < d \end{array} \ \textbf{MON} \ \begin{array}{l} a + b + c = n \\ a + b < d \\ c = 0 \\ \vdash \\ n < d \end{array} \ \textbf{EQ\_LR, MON} \ \begin{array}{l} a + b + 0 = n \\ a + b < d \\ \vdash \\ n < d \end{array} \ \textbf{ARI} \ \begin{array}{l} a + b = n \\ a + b < d \\ \vdash \\ n < d \end{array} \ \textbf{EQ\_LR, MON} \ \begin{array}{l} n < d \\ \vdash \\ n < d \end{array} \ \textbf{HYP}$$

Proving Refinement: ML_in/GRD

$$\begin{array}{l} d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \lor c = 0 \\ c > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{MON} \ \begin{array}{l} b \in \mathbb{N} \\ a + b + c = n \\ a = 0 \lor c = 0 \\ c > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{OR\_L}$$

$$\begin{cases} \begin{array}{l} b \in \mathbb{N} \\ a + b + c = n \\ a = 0 \\ c > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{EQ\_LR, MON} \ \begin{array}{l} b \in \mathbb{N} \\ 0 + b + c = n \\ c > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{ARI} \ \begin{array}{l} b \in \mathbb{N} \\ b + c = n \\ c > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{ARI} \ \begin{array}{l} c \leq n \\ c > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{ARI} \ \begin{array}{l} n > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{HYP} \\ \begin{array}{l} b \in \mathbb{N} \\ a + b + c = n \\ c = 0 \\ c > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{EQ\_LR} \ \begin{array}{l} b \in \mathbb{N} \\ a + b + 0 = n \\ c = 0 \\ 0 > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{MON} \ \begin{array}{l} 0 > 0 \\ \vdash \\ n > 0 \end{array} \ \textbf{ARI} \ \begin{array}{l} \bot \\ \vdash \\ n > 0 \end{array} \ \textbf{FALSE\_L} \end{cases}$$

Refinement Rule: Invariant Preservation

- Based on the components, we are able to formally state the ***PO/VC Rule of Invariant Preservation for Refinement***:

$$\begin{array}{l} A(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ \vdash \\ J_i(c, E(c, v), F(c, w)) \end{array} \quad \text{INV} \qquad \text{where } J_i \text{ denotes a single } \textbf{\textit{concrete invariant}}$$

  - \# ***sequents*** to be proved? [# *concrete*, **old** evts × # *concrete* invariants]
  - Here are two (of the ten) ***sequents*** generated:

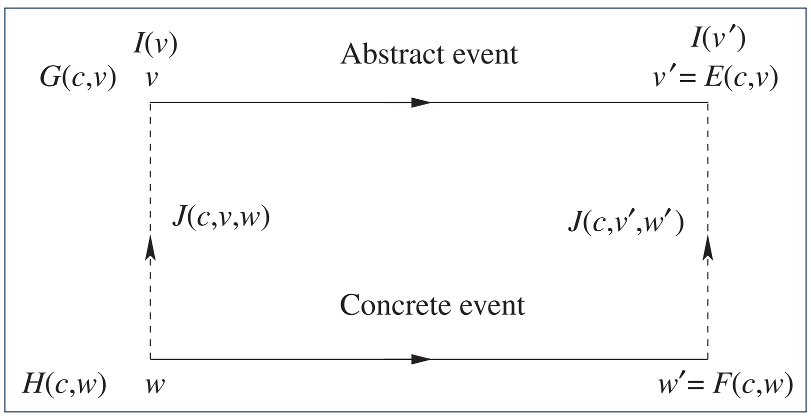
$$\begin{array}{l} d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \le d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \lor c = 0 \\ a + b < d \\ c = 0 \\ \vdash \\ (a+1) + b + c = (n+1) \end{array} \quad \text{ML\_out/inv1\_4/INV} \qquad \begin{array}{l} d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \le d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \lor c = 0 \\ c > 0 \\ \vdash \\ a = 0 \lor (c-1) = 0 \end{array} \quad \text{ML\_in/inv1\_5/INV}$$

- **Exercises**. Specify and prove other **eight** ***POs of Invariant Preservation***.

Visualizing Inv. Preservation in Refinement

Each ***concrete*** event (w to w') is ***simulated by*** an ***abstract*** event (v to v'):

- ***abstract*** & ***concrete*** pre-states related by ***concrete*** invariants $J(c, v, w)$
- ***abstract*** & ***concrete*** post-states related by ***concrete*** invariants $J(c, v', w')$



INV PO of m_1: ML_out/inv1_4/INV

axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \le d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \lor c = 0$
Concrete guards of *ML_out*	$\{ a + b < d$, $c = 0$
	$\vdash$
Concrete invariant **inv1_4** with *ML_out*'s effect in the post-state	$\{ (a+1) + b + c = (n+1)$

ML_out/inv1_4/INV

INV PO of m_1: ML_in/inv1_5/INV

axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \le d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \lor c = 0$
Concrete guards of *ML_in*	$\{ c > 0$
	$\vdash$
Concrete invariant **inv1_5** with *ML_in*'s effect in the post-state	$\{ a = 0 \lor (c-1) = 0$

ML_in/inv1_5/INV

Proving Refinement: ML_out/inv1_4/INV

$$\begin{array}{l} d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \lor c = 0 \\ a + b < d \\ c = 0 \\ \vdash \\ (a + 1) + b + c = (n + 1) \end{array} \quad \textbf{MON} \quad \begin{array}{l} a + b + c = n \\ \vdash \\ (a + 1) + b + c = (n + 1) \end{array} \quad \textbf{ARI} \quad \begin{array}{l} a + b + c = n \\ \vdash \\ a + b + c + 1 = n + 1 \end{array} \quad \textbf{EQ\_LR, MON} \quad \begin{array}{l} \vdash \\ n + 1 = n + 1 \end{array} \quad \textbf{EQ}$$

Proving Refinement: ML_in/inv1_5/INV

$$\begin{array}{l} d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \lor c = 0 \\ c > 0 \\ \vdash \\ a = 0 \lor (c - 1) = 0 \end{array} \quad \textbf{MON} \quad \begin{array}{l} a = 0 \lor c = 0 \\ c > 0 \\ \vdash \\ a = 0 \lor (c - 1) = 0 \end{array} \quad \textbf{OR\_L}$$

$$\begin{cases} \begin{array}{l} a = 0 \\ c > 0 \\ \vdash \\ a = 0 \lor (c - 1) = 0 \end{array} \quad \textbf{OR\_R1} \quad \begin{array}{l} a = 0 \\ c > 0 \\ \vdash \\ a = 0 \end{array} \quad \textbf{HYP} \\ \begin{array}{l} c = 0 \\ c > 0 \\ \vdash \\ a = 0 \lor (c - 1) = 0 \end{array} \quad \textbf{EQ\_LR, MON} \quad \begin{array}{l} 0 > 0 \\ \vdash \\ a = 0 \lor (0 - 1) = 0 \end{array} \quad \textbf{ARI} \quad \begin{array}{l} \bot \\ \vdash \\ a = 0 \lor -1 = 0 \end{array} \quad \textbf{FALSE\_L} \end{cases}$$

Initializing the Refined System m_1

- Discharging the **twelve sequents** proved that:
  - ***concrete invariants*** preserved by `ML_out` & `ML_in`
  - ***concrete guards*** of *ML_out* & *ML_in* entail their ***abstract*** counterparts
- What's left is the specification of how the ***ASM***'s ***initial state*** looks like:

```
init
  begin
    a := 0
    b := 0
    c := 0
  end
```

✓ No cars on bridge (heading either way) and island

✓ Initialization always possible: guard is ***true***.

✓ There is no ***pre-state*** for *init*.

∴ The RHS of := must not involve variables.

∴ The RHS of := may only involve constants.

✓ There is only the ***post-state*** for *init*.

∴ Before-***After Predicate***: $a' = 0 \land b' = 0 \land c' = 0$

PO of m_1 Concrete Invariant Establishment

- Some (new) formal components are needed:
  - $K(c)$: effect of ***abstract init***'s actions:
    e.g., $K(\langle d \rangle)$ of *init* $\widehat{=}$ $\langle 0 \rangle$
  - $v' = K(c)$: ***before-after predicate*** formalizing ***abstract*** *init*'s actions
    e.g., BAP of *init*: $\langle n' \rangle = \langle 0 \rangle$
  - $L(c)$: effect of ***concrete init***'s actions:
    e.g., $K(\langle d \rangle)$ of *init* $\widehat{=}$ $\langle 0, 0, 0 \rangle$
  - $w' = L(c)$: ***before-after predicate*** formalizing ***concrete*** *init*'s actions
    e.g., BAP of *init*: $\langle a', b', c' \rangle = \langle 0, 0, 0 \rangle$
- Accordingly, PO of ***invariant establishment*** is formulated as a **sequent**:

$$\begin{array}{l} \text{Axioms} \\ \vdash \\ \textit{Concrete Invariants} \text{ Satisfied at Post-State} \end{array} \quad \text{INV} \qquad \begin{array}{l} A(c) \\ \vdash \\ J_i(c, K(c), L(c)) \end{array} \quad \text{INV}$$

Discharging PO of m_1 Concrete Invariant Establishment

- How many ***sequents*** to be proved? [# ***concrete*** invariants]
- Two (of the five) sequents generated for ***concrete*** *init* of m_1:

$$d \in \mathbb{N},\ d > 0 \vdash 0 + 0 + 0 = 0 \quad \text{init/inv1\_4/INV}$$

$$d \in \mathbb{N},\ d > 0 \vdash 0 = 0 \lor 0 = 0 \quad \text{init/inv1\_5/INV}$$

- Can we discharge the ***PO*** init/**inv1_4**/INV ?

$$d \in \mathbb{N},\ d > 0 \vdash 0 + 0 + 0 = 0 \quad \textbf{ARI, MON} \quad \vdash \top \quad \textbf{TRUE\_R}$$

∴ ***init/inv1_4/INV*** succeeds in being discharged.

- Can we discharge the ***PO*** init/**inv1_5**/INV ?

$$d \in \mathbb{N},\ d > 0 \vdash 0 = 0 \lor 0 = 0 \quad \textbf{ARI, MON} \quad \vdash \top \quad \textbf{TRUE\_R}$$

∴ ***init/inv1_5/INV*** succeeds in being discharged.

Model m_1: New, Concrete Events

- The system acts as an **ABSTRACT STATE MACHINE (ASM)**: it ***evolves*** as ***actions of enabled events*** change values of variables, subject to ***invariants***.
- Considered ***concrete/refined events*** already existing in m_0: *ML_out* & *ML_in*
- ***New event*** *IL_in*:

```
IL_in
  when
    ??
  then
    ??
  end
```

  - *IL_in* denotes a car entering the island (getting off the bridge).
  - *IL_in* ***enabled*** only when:
    - The bridge's current traffic flows to the island.
      Q. Limited number of cars on the bridge and the island?
      A. Ensured when the earlier *ML_out* (of same car) occurred
- ***New event*** *IL_out*:

```
IL_out
  when
    ??
  then
    ??
  end
```

  - *IL_out* denotes a car exiting the island (getting on the bridge).
  - *IL_out* ***enabled*** only when:
    - There is some car on the island.
    - The bridge's current traffic flows to the mainland.

Model m_1: BA Predicates of Multiple Actions

Consider ***actions*** of m_1's two ***new*** events:

```
IL_in
  when
    a > 0
  then
    a := a − 1
    b := b + 1
  end
```

```
IL_out
  when
    b > 0
    a = 0
  then
    b := b − 1
    c := c + 1
  end
```

- What is the ***BAP*** of *ML_in*'s ***actions***?

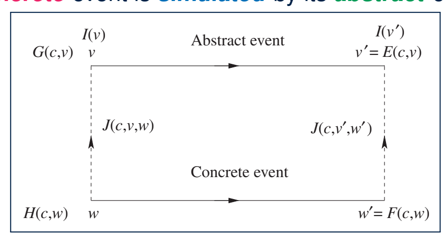
$$a' = a - 1 \land b' = b + 1 \land c' = c$$

- What is the ***BAP*** of *ML_in*'s ***actions***?

$$a' = a \land b' = b - 1 \land c' = c + 1$$

Visualizing Inv. Preservation in Refinement

- Recall how a ***concrete*** event is ***simulated*** by its ***abstract*** counterpart:



- For each ***new*** event:
  - Strictly speaking, it does **not** have an ***abstract*** counterpart.
  - It is ***simulated by*** a special ***abstract*** event (transforming v to v'):

```
skip
  begin
  end
```

- *skip* is a "dummy" event: non-guarded and does nothing
- **Q**. ***BAP*** of the skip event?
  A. $n' = n$

Refinement Rule: Invariant Preservation

- The new events *IL_in* and *IL_out* do not exist in $\mathbf{m_0}$, but:
  - They **exist** in $\mathbf{m_1}$ and may impact upon the ***concrete*** state space.
  - They ***preserve*** the ***concrete invariants***, just as *ML_out* & *ML_in* do.
- Recall the ***PO/VC Rule of Invariant Preservation for Refinement***:

$$\begin{array}{|l|}\hline A(c) \\ I(c, v) \\ J(c, v, w) \\ H(c, w) \\ \vdash \\ J_i(c, E(c, v), F(c, w)) \\ \hline \end{array} \quad \underline{\text{INV}}$$

where J_i denotes a single ***concrete invariant***

  - How many ***sequents*** to be proved? [# *new* evts × # *concrete* invariants]
  - Here are two (of the ten) ***sequents*** generated:

$$\begin{array}{|l|}\hline d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \vee c = 0 \\ a > 0 \\ \vdash \\ (a-1) + (b+1) + c = n \\ \hline \end{array} \quad \underline{\text{IL\_in/inv1\_4/INV}} \qquad \begin{array}{|l|}\hline d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \vee c = 0 \\ a > 0 \\ \vdash \\ (a-1) = 0 \vee c = 0 \\ \hline \end{array} \quad \underline{\text{IL\_in/inv1\_5/INV}}$$

- **Exercises**. Specify and prove other **eight** ***POs of Invariant Preservation***.

INV PO of m_1: IL_in/inv1_4/INV

axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \leq d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \vee c = 0$
Guards of *IL_in*	$\{ a > 0$
	$\vdash$
Concrete invariant **inv1_4** with *IL_in*'s effect in the post-state	$\{ (a-1) + (b+1) + c = n$

IL_in/inv1_4/INV

INV PO of m_1: IL_in/inv1_5/INV

axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \leq d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \vee c = 0$
Guards of *IL_in*	$\{ a > 0$
	$\vdash$
Concrete invariant **inv1_5** with *IL_in*'s effect in the post-state	$\{ (a-1) = 0 \vee c = 0$

IL_in/inv1_5/INV

Proving Refinement: IL_in/inv1_4/INV

$$\begin{array}{|l|}\hline d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a + b + c = n \\ a = 0 \vee c = 0 \\ a > 0 \\ \vdash \\ (a-1) + (b+1) + c = n \\ \hline \end{array} \; \textbf{MON} \; \begin{array}{|l|}\hline a + b + c = n \\ \vdash \\ (a-1) + (b+1) + c = n \\ \hline \end{array} \; \textbf{ARI} \; \begin{array}{|l|}\hline a + b + c = n \\ \vdash \\ a + b + c = n \\ \hline \end{array} \; \textbf{HYP}$$

Proving Refinement: IL_in/inv1_5/INV

$$\begin{array}{l} d \in \mathbb{N} \\ d > 0 \\ n \in \mathbb{N} \\ n \leq d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \\ a+b+c=n \\ a = 0 \lor c = 0 \\ a > 0 \\ \vdash \\ (a-1) = 0 \lor c = 0 \end{array} \quad \textbf{MON} \quad \begin{array}{l} a = 0 \lor c = 0 \\ a > 0 \\ \vdash \\ (a-1) = 0 \lor c = 0 \end{array} \quad \textbf{OR\_L}$$

$$\left\{ \begin{array}{l} \begin{array}{l} a = 0 \\ a > 0 \\ \vdash \\ (a-1) = 0 \lor c = 0 \end{array} \quad \textbf{EQ\_LR, MON} \quad \begin{array}{l} 0 > 0 \\ \vdash \\ (0-1) = 0 \lor c = 0 \end{array} \quad \textbf{ARI} \quad \begin{array}{l} \bot \\ \vdash \\ -1 = 0 \lor c = 0 \end{array} \quad \textbf{FALSE\_L} \\ \begin{array}{l} c = 0 \\ a > 0 \\ \vdash \\ (a-1) = 0 \lor c = 0 \end{array} \quad \textbf{OR\_R2} \quad \begin{array}{l} c = 0 \\ a > 0 \\ \vdash \\ c = 0 \end{array} \quad \textbf{HYP} \end{array} \right.$$

Livelock Caused by New Events Diverging

- An alternative m_1 (with **inv1_4**, **inv1_5**, and **guards** of new events removed):

constants: d

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

variables: a, b, c

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

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

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

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

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

Concrete invariants are **under-specified**: only typing constraints.

Exercises: Show that **Invariant Preservation** is provable, but **Guard Strengthening** is not.

- Say this alternative m_1 is implemented as is:
  IL_in and *IL_out* **always** ***enabled*** and may occur ***indefinitely***, preventing other "old" events (*ML_out* and *ML_in*) from ever happening:
  $\langle init, ML\_out, IL\_in, IL\_out, IL\_in, IL\_out, \dots \rangle$
  Q: What are the corresponding ***abstract*** transitions?
  A: $\langle init, ML\_out, skip, skip, skip, skip, \dots \rangle$ [≈ executing `while(true);`]
- We say that these two ***new*** events *diverge*, creating a *livelock*:
  - Different from a ***deadlock*** ∵ **always** an event occurring (*IL_in* or *IL_out*).
  - But their ***indefinite*** occurrences contribute **nothing** useful.

PO of Convergence of New Events

The PO/VC rule for *non-divergence/livelock freedom* consists of two parts:
- Interleaving of ***new*** events characterized as an integer expr.: ***variant***.
- A variant $V(c, w)$ may refer to constants and/or ***concrete*** variables.
- In the original m_1, let's try **variants** : $2 \cdot a + b$

1. ***Variant Stays Non-Negative***

$$\frac{\begin{array}{l} A(c) \\ I(c,v) \\ J(c,v,w) \\ H(c,w) \\ \vdash \\ V(c,w) \in \mathbb{N} \end{array}}{} \quad \text{NAT}$$

- Variant $V(c, w)$ measures how many more times the ***new*** events can occur.
- If a ***new*** event is ***enabled***, then $V(c, w) > 0$.
- When $V(c, w)$ reaches 0, some "old" events must happen s.t. $V(c, w)$ goes back above 0.

2. ***A New Event Occurrence Decreases Variant***

$$\frac{\begin{array}{l} A(c) \\ I(c,v) \\ J(c,v,w) \\ H(c,w) \\ \vdash \\ V(c,F(c,w)) < V(c,w) \end{array}}{} \quad \text{VAR}$$

- If a ***new*** event is ***enabled*** and occurs, the value of $V(c, w) \downarrow$.

PO of Convergence of New Events: NAT

- Recall: PO related to ***Variant Stays Non-Negative***:

$$\frac{\begin{array}{l} A(c) \\ I(c,v) \\ J(c,v,w) \\ H(c,w) \\ \vdash \\ V(c,w) \in \mathbb{N} \end{array}}{} \quad \text{NAT}$$

How many ***sequents*** to be proved? [# *new* events]

- For the ***new*** event *IL_in*:

$$\frac{\begin{array}{lll} d \in \mathbb{N} & d > 0 & \\ n \in \mathbb{N} & n \leq d & \\ a \in \mathbb{N} & b \in \mathbb{N} & c \in \mathbb{N} \\ a+b+c=n & a = 0 \lor c = 0 & \\ a > 0 & & \\ \vdash & & \\ 2 \cdot a + b \in \mathbb{N} & & \end{array}}{} \quad \text{IL\_in/NAT}$$

Exercises: Prove **IL_in/NAT** and Formulate/Prove **IL_out/NAT**.

PO of Convergence of New Events: VAR

- Recall: PO related to ***A New Event Occurrence Decreases Variant***

$$\frac{A(c) \quad I(c,v) \quad J(c,v,w) \quad H(c,w) \vdash V(c,F(c,w)) < V(c,w)}{}\ \text{VAR}$$

How many ***sequents*** to be proved? [# *new* events]

- For the ***new*** event *IL_in*:

$$\begin{array}{lll} d \in \mathbb{N} & d > 0 & \\ n \in \mathbb{N} & n \le d & \\ a \in \mathbb{N} & b \in \mathbb{N} & c \in \mathbb{N} \\ a+b+c=n & a=0 \lor c=0 & \\ a>0 & & \\ \vdash & & \\ 2\cdot(a-1)+(b+1) < 2\cdot a+b & & \end{array} \quad \text{IL\_in/VAR}$$

Exercises: Prove **IL_in/VAR** and Formulate/Prove **IL_out/VAR**.

Convergence of New Events: Exercise

Given the original $\mathbf{m_1}$, what if the following ***variant*** expression is used:

variants : $a + b$

Are the formulated sequents still ***provable***?

PO of Refinement: Deadlock Freedom

- Recall:
  - We proved that the initial model m_0 is deadlock free (see **DLF**).
  - We proved, according to ***guard strengthening***, that if a ***concrete*** event is enabled, then its ***abstract*** counterpart is enabled.
- PO of ***relative deadlock freedom*** for a ***refinement*** model:

$$\frac{A(c) \quad I(c,v) \quad J(c,v,w) \quad G_1(c,v) \lor \cdots \lor G_m(c,v) \vdash H_1(c,w) \lor \cdots \lor H_n(c,w)}{}\ \text{DLF}$$

If an ***abstract*** state does not *deadlock* (i.e., $G_1(c,v) \lor \cdots \lor G_m(c,v)$), then its ***concrete*** counterpart does not *deadlock* (i.e., $H_1(c,w) \lor \cdots \lor H_n(c,w)$).

- Another way to think of the above PO:

  The ***refinement*** does **not** introduce, in the ***concrete***, any "new" *deadlock* scenarios **not** existing in the ***abstract*** state.

PO Rule: Relative Deadlock Freedom m_1

axm0_1	$d \in \mathbb{N}$	
axm0_2	$d > 0$	
inv0_1	$n \in \mathbb{N}$	
inv0_2	$n \le d$	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	
inv1_3	$c \in \mathbb{N}$	
inv1_4	$a + b + c = n$	
inv1_5	$a = 0 \lor c = 0$	
Disjunction of *abstract* guards	$n < d$	**guards of** *ML_out* **in** m_0
	$\lor\ n > 0$	**guards of** *ML_in* **in** m_0
$\vdash$		
Disjunction of *concrete* guards	$a + b < d \land c = 0$	**guards of** *ML_out* **in** m_1
	$\lor\ c > 0$	**guards of** *ML_in* **in** m_1
	$\lor\ a > 0$	**guards of** *IL_in* **in** m_1
	$\lor\ b > 0 \land a = 0$	**guards of** *IL_out* **in** m_1

DLF

Example Inference Rules (6)

$$\frac{H, \neg P \vdash Q}{H \vdash P \vee Q} \quad \textbf{OR\_R}$$

To prove a ***disjunctive goal***,
it suffices to prove one of the disjuncts,
with the the negation of the the other disjunct
serving as an additional hypothesis.

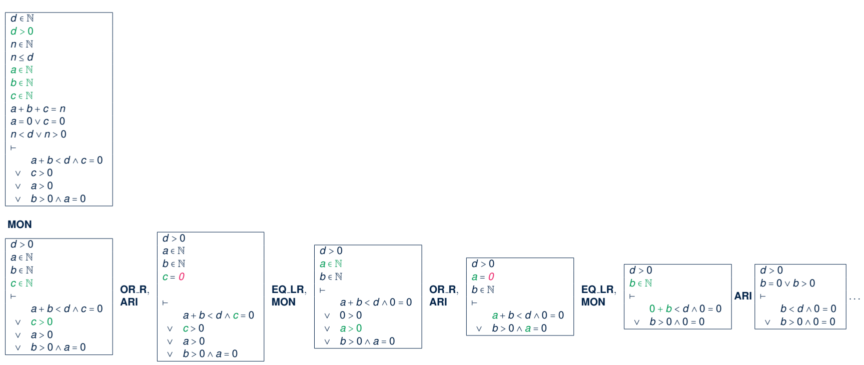
$$\frac{H, P, Q \vdash R}{H, P \wedge Q \vdash R} \quad \textbf{AND\_L}$$

To prove a goal with a ***conjunctive hypothesis***,
it suffices to prove the same goal,
with the two conjuncts
serving as two separate hypotheses.

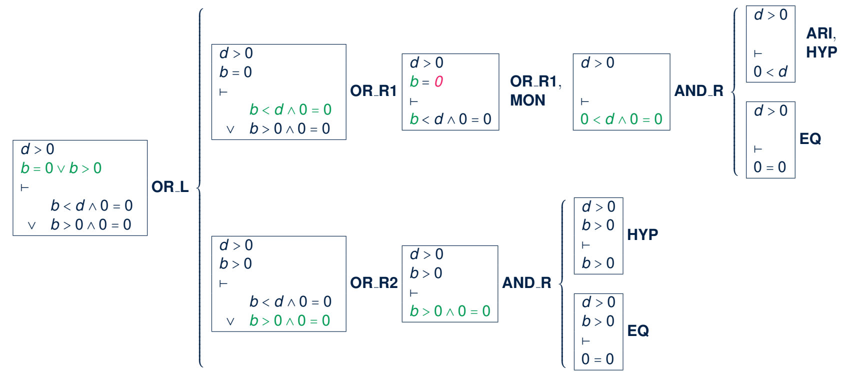
$$\frac{H \vdash P \qquad H \vdash Q}{H \vdash P \wedge Q} \quad \textbf{AND\_R}$$

To prove a goal with a ***conjunctive goal***,
it suffices to prove each conjunct
as a separate goal.

Proving Refinement: DLF of m_1



Proving Refinement: DLF of m_1 (continued)



First Refinement: Summary

- The final version of our ***first refinement*** m_1 is ***provably correct*** w.r.t.:
  - Establishment of ***Concrete Invariants*** [*init*]
  - Preservation of ***Concrete Invariants*** [old & new events]
  - Strengthening of ***guards*** [old events]
  - ***Convergence*** (a.k.a. livelock freedom, non-divergence) [new events]
  - Relative ***Deadlock*** Freedom
- Here is the final specification of m_1:

```
constants: d

axioms:
  axm0_1 : d ∈ ℕ
  axm0_2 : d > 0

variables: a, b, c

invariants:
  inv1_1 : a ∈ ℕ
  inv1_2 : b ∈ ℕ
  inv1_3 : c ∈ ℕ
  inv1_4 : a + b + c = n
  inv1_5 : a = 0 ∨ c = 0

variants:
  2 · a + b

init
  begin
    a := 0
    b := 0
    c := 0
  end

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

ML_in
  when
    c > 0
  then
    c := c − 1
  end

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

IL_out
  when
    b > 0
    a = 0
  then
    b := b − 1
    c := c + 1
  end
```

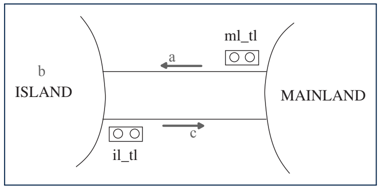
Model m_2: "More Concrete" Abstraction

- 2nd ***refinement*** has even more ***concrete*** perception of the bridge controller:
  - We "**zoom in**" by observing the system from **even closer to the ground**, so that the one-way traffic of the bridge is controlled via:

ml_tl: a traffic light for exiting the ML

il_tl: a traffic light for exiting the IL

abstract variables ***a***, ***b***, ***c*** from m_1 still used (instead of being replaced)



  - Nonetheless, sensors remain ***abstracted*** away!
- That is, we focus on these three ***environment constraints***:

ENV1	The system is equipped with two traffic lights with two colors: green and red.
ENV2	The traffic lights control the entrance to the bridge at both ends of it.
ENV3	Cars are not supposed to pass on a red traffic light, only on a green one.

- We are ***obliged to prove*** this ***added concreteness*** is ***consistent*** with m_1.

Model m_2: Refined, Concrete State Space

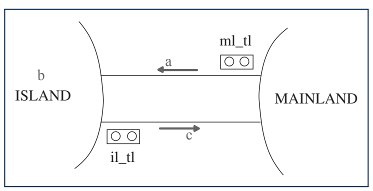
1. The **static** part introduces the notion of traffic light colours:

sets: *COLOR*

constants: *red*, *green*

axioms:
axm2_1 : $COLOR = \{green, red\}$
axm2_2 : $green \neq red$

2. The **dynamic** part shows the ***superposition refinement*** scheme:



- ***Abstract*** variables ***a***, ***b***, ***c*** from m_1 are still in use in ***m_2***.
- Two new, ***concrete*** variables are introduced: ***ml_tl*** and ***il_tl***
- **Constrast**: In m_1, ***abstract*** variable *n* is replaced by ***concrete*** variables *a*, *b*, *c*.

variables:
a, b, c
ml_tl
il_tl

invariants:
inv2_1 : $ml\_tl \in COLOUR$
inv2_2 : $il\_tl \in COLOUR$
inv2_3 : ??
inv2_4 : ??

- **inv2_1** & **inv2_2**: typing constraints
- **inv2_3**: being allowed to exit ML **means** cars within limit and no opposite traffic
- **inv2_4**: being allowed to exit IL **means** some car in IL and no opposite traffic

Model m_2: Refining Old, Abstract Events

- The system acts as an **ABSTRACT STATE MACHINE (ASM)**: it ***evolves*** as ***actions of enabled events*** change values of variables, subject to ***invariants***.
- ***Concrete/Refined*** version of ***event*** *ML_out*:

```
ML_out
  when
    ??
  then
    a := a + 1
  end
```

  - Recall the ***abstract*** guard of *ML_out* in m_1: $(c = 0) \land (a + b < d)$
    ⇒ Unrealistic as drivers should **not** know about *a*, *b*, *c*!
  - *ML_out* is ***refined***: a car exits the ML (to the bridge) only when:
    - the traffic light *ml_tl* allows
- ***Concrete/Refined*** version of ***event*** *IL_out*:

```
IL_out
  when
    ??
  then
    b := b − 1
    c := c + 1
  end
```

  - Recall the ***abstract*** guard of *IL_out* in m_1: $(a = 0) \land (b > 0)$
    ⇒ Unrealistic as drivers should **not** know about *a*, *b*, *c*!
  - *IL_out* is ***refined***: a car exits the IL (to the bridge) only when:
    - the traffic light *il_tl* allows

Q1. How about the other two "old" ***events*** *IL_in* and *ML_in*?
A1. No need to ***refine*** as already ***guarded*** by *ML_out* and *IL_out*.
Q2. What if the driver disobeys *ml_tl* or *il_tl*? [**A2**. **ENV3**]

Model m_2: New, Concrete Events

- The system acts as an **ABSTRACT STATE MACHINE (ASM)**: it ***evolves*** as ***actions of enabled events*** change values of variables, subject to ***invariants***.
- Considered ***events*** already existing in m_1:
  - *ML_out* & *IL_out* [REFINED]
  - *IL_in* & *ML_in* [UNCHANGED]
- ***New event*** *ML_tl_green*:

```
ML_tl_green
  when
    ??
  then
    ml_tl := green
  end
```

  - *ML_tl_green* denotes the traffic light *ml_tl* turning green.
  - *ML_tl_green* ***enabled*** only when:
    - the traffic light not already green
    - limited number of cars on the bridge and the island
    - No opposite traffic

  [⇒ *ML_out*'s ***abstract*** guard in m_1]
- ***New event*** *IL_tl_green*:

```
IL_tl_green
  when
    ??
  then
    il_tl := green
  end
```

  - *IL_tl_green* denotes the traffic light *il_tl* turning green.
  - *IL_tl_green* ***enabled*** only when:
    - the traffic light not already green
    - some cars on the island (i.e., island not empty)
    - No opposite traffic

  [⇒ *IL_out*'s ***abstract*** guard in m_1]

Invariant Preservation in Refinement m_2

constants: d

sets: $COLOR$

axioms:
- **axm0_1** : $d \in \mathbb{N}$
- **axm0_2** : $d > 0$
- **axm2_1** : $COLOR = \{green, red\}$
- **axm2_2** : $green \neq red$

variables: a, b, c, ml_tl, il_tl

invariants:
- **inv2_1** : $ml\_tl \in COLOUR$
- **inv2_2** : $il\_tl \in COLOUR$
- **inv2_3** : $ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
- **inv2_4** : $il\_tl = green \Rightarrow b > 0 \wedge a = 0$

ML_tl_green
when $ml\_tl = red$, $a + b < d$, $c = 0$
then $ml\_tl := green$
end

ML_out
when $ml\_tl = green$
then $a := a + 1$
end

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

IL_tl_green
when $il\_tl = red$, $b > 0$, $a = 0$
then $il\_tl := green$
end

IL_out
when $il\_tl = green$
then $b := b - 1$, $c := c + 1$
end

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

Recall the ***PO/VC Rule of Invariant Preservation for Refinement***:

$$\frac{A(c),\ I(c, v),\ J(c, v, w),\ H(c, w) \vdash J_i(c, E(c, v), F(c, w))}{} \quad \text{INV}$$

where J_i denotes a single ***concrete invariant***

- How many ***sequents*** to be proved? [# *concrete* evts × # *concrete* invariants = 6 × 4]
- We discuss two sequents: *ML_out*/**inv2_4**/*INV* and *IL_out*/**inv2_3**/*INV*

Exercises. Specify and prove (some of) other twenty-two ***POs of Invariant Preservation***.

INV PO of m_2: ML_out/inv2_4/INV

ML_out/inv2_4/INV

axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
axm2_1	$\{ COLOUR = \{green, red\}$
axm2_2	$\{ green \neq red$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \leq d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \vee c = 0$
inv2_1	$\{ ml\_tl \in COLOUR$
inv2_2	$\{ il\_tl \in COLOUR$
inv2_3	$\{ ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
inv2_4	$\{ il\_tl = green \Rightarrow b > 0 \wedge a = 0$
Concrete guards of *ML_out*	$\{ ml\_tl = green$
	$\vdash$
Concrete invariant **inv2_4** with *ML_out*'s effect in the post-state	$\{ il\_tl = green \Rightarrow b > 0 \wedge (a + 1) = 0$

INV PO of m_2: IL_out/inv2_3/INV

IL_out/inv2_3/INV

axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
axm2_1	$\{ COLOUR = \{green, red\}$
axm2_2	$\{ green \neq red$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \leq d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \vee c = 0$
inv2_1	$\{ ml\_tl \in COLOUR$
inv2_2	$\{ il\_tl \in COLOUR$
inv2_3	$\{ ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
inv2_4	$\{ il\_tl = green \Rightarrow b > 0 \wedge a = 0$
Concrete guards of *IL_out*	$\{ il\_tl = green$
	$\vdash$
Concrete invariant **inv2_3** with *ML_out*'s effect in the post-state	$\{ ml\_tl = green \Rightarrow a + (b - 1) < d \wedge (c + 1) = 0$

Example Inference Rules (7)

$$\frac{H, P, Q \vdash R}{H, P, P \Rightarrow Q \vdash R} \quad \textbf{IMP\_L}$$

If a hypothesis ***P*** matches the assumption of another ***implicative hypothesis P ⇒ Q***, then the conclusion ***Q*** of the ***implicative hypothesis*** can be used as a new hypothesis for the sequent.

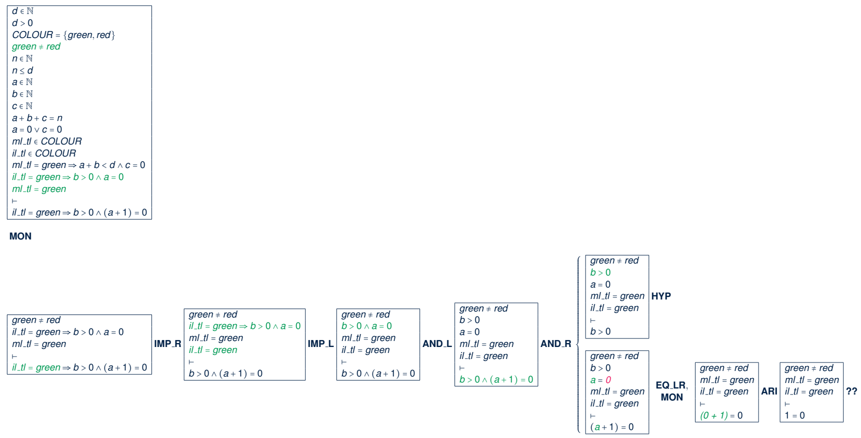
$$\frac{H, P \vdash Q}{H \vdash P \Rightarrow Q} \quad \textbf{IMP\_R}$$

To prove an ***implicative goal P ⇒ Q***, it suffices to prove its conclusion ***Q***, with its assumption ***P*** serving as a new hypotheses.

$$\frac{H, \neg Q \vdash P}{H, \neg P \vdash Q} \quad \textbf{NOT\_L}$$

To prove a goal ***Q*** with a ***negative hypothesis*** $\neg$ ***P***, it suffices to prove the negated hypothesis $\neg(\neg P) \equiv$ ***P*** with the negated original goal $\neg$ ***Q*** serving as a new hypothesis.

Proving ML_out/inv2_4/INV: First Attempt



Failed: ML_out/inv2_4/INV, IL_out/inv2_3/INV

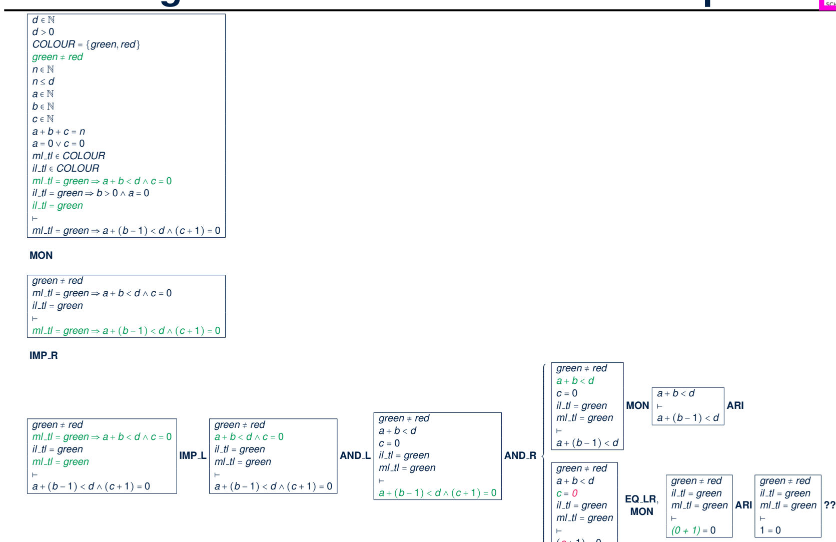
- Our first attempts of proving ***ML_out/inv2_4/INV*** and ***IL_out/inv2_3/INV*** both failed the 2nd case (resulted from applying IR **AND_R**):

$$green \neq red \land il\_tl = green \land ml\_tl = green \vdash 1 = 0$$

- This ***unprovable*** sequent gave us a good hint:
  - Goal $1 = 0 \equiv$ **false** suggests that the ***safety requirements*** $a = 0$ (for **inv2_4**) and $c = 0$ (for **inv2_3**) ***contradict*** with the current m_2.
  - Hyp. $il\_tl = green = ml\_tl$ suggests a ***possible, dangerous state*** of m_2, where two cars heading different directions are on the one-way bridge:

init	ML_tl_green	ML_out	IL_in	IL_tl_green	IL_out	ML_out
$d = 2$	$d = 2$	$d = 2$	$d = 2$	$d = 2$	$d = 2$	$d = 2$
$a' = 0$	$a' = 0$	**$a' = 1$**	**$a' = 0$**	$a' = 0$	$a' = 0$	**$a' = 1$**
$b' = 0$	$b' = 0$	$b' = 0$	**$b' = 1$**	$b' = 1$	**$b' = 0$**	$b' = 0$
$c' = 0$	$c' = 0$	$c' = 0$	$c' = 0$	$c' = 0$	**$c' = 1$**	$c' = 1$
$ml\_tl' = red$	**$ml\_tl' = green$**	$ml\_tl' = green$	$ml\_tl' = green$	$ml\_tl' = green$	$ml\_tl' = green$	$ml\_tl' = green$
$il\_tl' = red$	$il\_tl' = red$	$il\_tl' = red$	$il\_tl' = red$	**$il\_tl' = green$**	$il\_tl' = green$	$il\_tl' = green$

Proving IL_out/inv2_3/INV: First Attempt



Fixing m_2: Adding an Invariant

- Having understood the failed proofs, we add a proper ***invariant*** to m_2:

> **invariants:**
> ...
> **inv2_5** : $ml\_tl = red \lor il\_tl = red$

- We have effectively resulted in an improved m_2 more faithful w.r.t. **REQ3**:

REQ3	The bridge is one-way or the other, not both at the same time.

- Having added this new invariant ***inv2_5***:
  - Original 6×4 generated sequents to be updated: **inv2_5** a new hypothesis e.g., Are ***ML_out/inv2_4/INV*** and ***IL_out/inv2_3/INV*** now ***provable***?
  - Additional 6×1 sequents to be generated due to this new invariant e.g., Are ***ML_tl_green/inv2_5/INV*** and ***IL_tl_green/inv2_5/INV*** ***provable***?

INV PO of m_2: ML_out/inv2_4/INV – Updated

axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
axm2_1	$\{ COLOUR = \{green, red\}$
axm2_2	$\{ green \neq red$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \leq d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \vee c = 0$
inv2_1	$\{ ml\_tl \in COLOUR$
inv2_2	$\{ il\_tl \in COLOUR$
inv2_3	$\{ ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
inv2_4	$\{ il\_tl = green \Rightarrow b > 0 \wedge a = 0$
inv2_5	$\{ ml\_tl = red \vee il\_tl = red$
Concrete guards of *ML_out*	$\{ ml\_tl = green$
	$\vdash$
Concrete invariant **inv2_4** with *ML_out*'s effect in the post-state	$\{ il\_tl = green \Rightarrow b > 0 \wedge (a + 1) = 0$

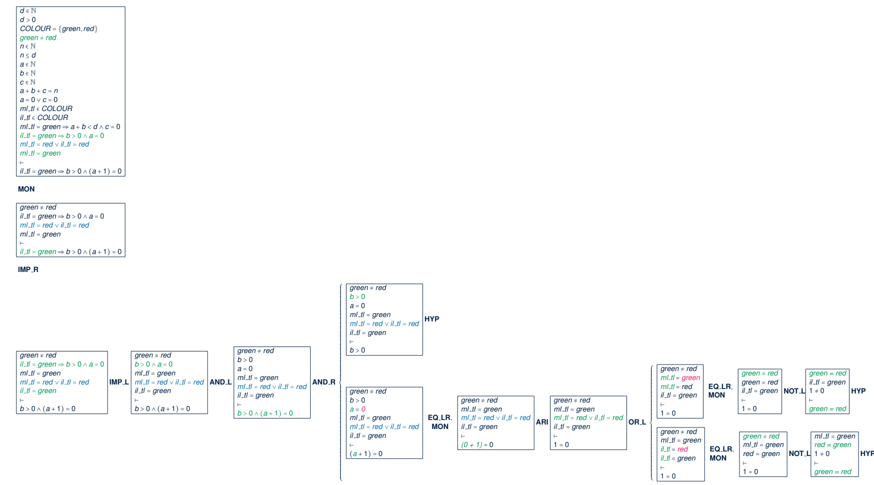
ML_out/inv2_4/INV

INV PO of m_2: IL_out/inv2_3/INV – Updated

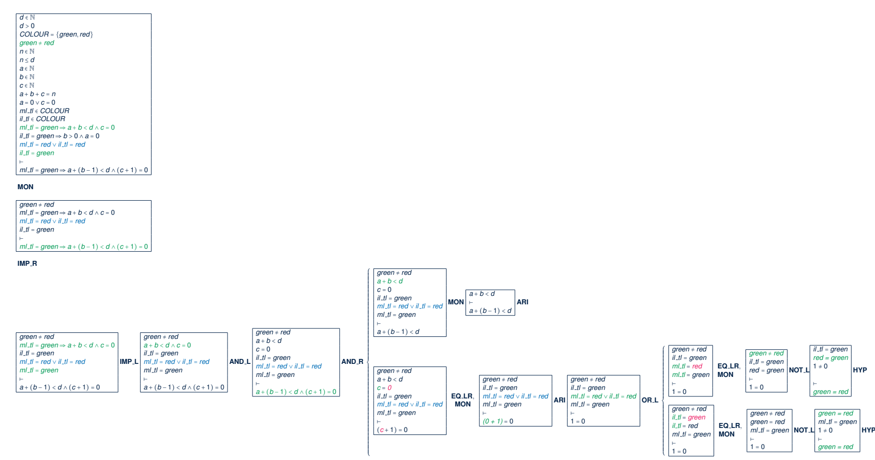
axm0_1	$\{ d \in \mathbb{N}$
axm0_2	$\{ d > 0$
axm2_1	$\{ COLOUR = \{green, red\}$
axm2_2	$\{ green \neq red$
inv0_1	$\{ n \in \mathbb{N}$
inv0_2	$\{ n \leq d$
inv1_1	$\{ a \in \mathbb{N}$
inv1_2	$\{ b \in \mathbb{N}$
inv1_3	$\{ c \in \mathbb{N}$
inv1_4	$\{ a + b + c = n$
inv1_5	$\{ a = 0 \vee c = 0$
inv2_1	$\{ ml\_tl \in COLOUR$
inv2_2	$\{ il\_tl \in COLOUR$
inv2_3	$\{ ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
inv2_4	$\{ il\_tl = green \Rightarrow b > 0 \wedge a = 0$
inv2_5	$\{ ml\_tl = red \vee il\_tl = red$
Concrete guards of *IL_out*	$\{ il\_tl = green$
	$\vdash$
Concrete invariant **inv2_3** with *ML_out*'s effect in the post-state	$\{ ml\_tl = green \Rightarrow a + (b - 1) < d \wedge (c + 1) = 0$

IL_out/inv2_3/INV

Proving ML_out/inv2_4/INV: Second Attempt



Proving IL_out/inv2_3/INV: Second Attempt



Fixing m_2: Adding Actions

- Recall that an ***invariant*** was added to m_2:

  invariants:
  inv2_5 : $ml\_tl = red \lor il\_tl = red$

- Additional 6×1 sequents to be generated due to this new invariant:
  - e.g., ***ML_tl_green*/inv2_5/INV** [for ***ML_tl_green*** to preserve **inv2_5**]
  - e.g., ***IL_tl_green*/inv2_5/INV** [for ***IL_tl_green*** to preserve **inv2_5**]
- For the above ***sequents*** to be ***provable***, we need to revise the two events:

```
ML_tl_green
  when
    ml_tl = red
    a + b < d
    c = 0
  then
    ml_tl := green
    il_tl := red
  end
```

```
IL_tl_green
  when
    il_tl = red
    b > 0
    a = 0
  then
    il_tl := green
    ml_tl := red
  end
```

Exercise: Specify and prove ***ML_tl_green*/inv2_5/INV** & ***IL_tl_green*/inv2_5/INV**.

INV PO of m_2: ML_out/inv2_3/INV

axm0_1	$d \in \mathbb{N}$
axm0_2	$d > 0$
axm2_1	$COLOUR = \{green, red\}$
axm2_2	$green \neq red$
inv0_1	$n \in \mathbb{N}$
inv0_2	$n \leq d$
inv1_1	$a \in \mathbb{N}$
inv1_2	$b \in \mathbb{N}$
inv1_3	$c \in \mathbb{N}$
inv1_4	$a + b + c = n$
inv1_5	$a = 0 \lor c = 0$
inv2_1	$ml\_tl \in COLOUR$
inv2_2	$il\_tl \in COLOUR$
inv2_3	$ml\_tl = green \Rightarrow a + b < d \land c = 0$
inv2_4	$il\_tl = green \Rightarrow b > 0 \land a = 0$
inv2_5	$ml\_tl = red \lor il\_tl = red$
Concrete guards of *ML_out*	$ml\_tl = green$
	$\vdash$
Concrete invariant **inv2_3** with *ML_out*'s effect in the post-state	$ml\_tl = green \Rightarrow (a+1) + b < d \land c = 0$

ML_out/inv2_3/INV

Proving ML_out/inv2_3/INV: First Attempt

$$d \in \mathbb{N},\ d > 0,\ COLOUR = \{green, red\},\ green \neq red,\ n \in \mathbb{N},\ n \leq d,\ a \in \mathbb{N},\ b \in \mathbb{N},\ c \in \mathbb{N},\ a + b + c = n,\ a = 0 \lor c = 0,\ ml\_tl \in COLOUR,\ il\_tl \in COLOUR,\ ml\_tl = green \Rightarrow a + b < d \land c = 0,\ il\_tl = green \Rightarrow b > 0 \land a = 0,\ ml\_tl = red \lor il\_tl = red,\ ml\_tl = green \vdash ml\_tl = green \Rightarrow (a+1) + b < d \land c = 0$$

MON

$$ml\_tl = green \Rightarrow a + b < d \land c = 0 \vdash ml\_tl = green \Rightarrow (a+1) + b < d \land c = 0$$

IMP_R

$$ml\_tl = green \Rightarrow a + b < d \land c = 0,\ ml\_tl = green \vdash (a+1) + b < d \land c = 0$$

IMP_R

$$a + b < d \land c = 0,\ ml\_tl = green \vdash (a+1) + b < d \land c = 0$$

AND_L

$$a + b < d,\ c = 0,\ ml\_tl = green \vdash (a+1) + b < d \land c = 0$$

AND_R

$$a + b < d,\ c = 0,\ ml\_tl = green \vdash (a+1) + b < d \quad \textbf{??}$$

$$a + b < d,\ c = 0,\ ml\_tl = green \vdash c = 0 \quad \textbf{HYP}$$

Failed: ML_out/inv2_3/INV

- Our first attempt of proving ***ML_out/inv2_3/INV*** failed the 1st case (resulted from applying IR **AND_R**):

$$a + b < d \land c = 0 \land ml\_tl = green \vdash (a+1) + b < d$$

- This ***unprovable*** sequent gave us a good hint:
  - Goal $\underbrace{(a+1)}_{a'} + \underbrace{b}_{b'} < d$ specifies the ***capacity requirement***.
  - Hypothesis $\boxed{c = 0 \land ml\_tl = green}$ assumes that it's safe to exit the ML.
  - Hypothesis $\boxed{a + b < d}$ is **not** strong enough to entail $(a+1) + b < d$.

    e.g., $d = 3$, $b = 0$, $a = 0$ [$(a+1) + b < d$ evaluates to ***true***]
    e.g., $d = 3$, $b = 1$, $a = 0$ [$(a+1) + b < d$ evaluates to ***true***]
    e.g., $d = 3$, $b = 0$, $a = 1$ [$(a+1) + b < d$ evaluates to ***true***]
    e.g., $d = 3$, $b = 0$, $a = 2$ [$(a+1) + b < d$ evaluates to ***false***]
    e.g., $d = 3$, $b = 1$, $a = 1$ [$(a+1) + b < d$ evaluates to ***false***]
    e.g., $d = 3$, $b = 2$, $a = 0$ [$(a+1) + b < d$ evaluates to ***false***]
  - Therefore, $a + b < d$ (allowing one more car to exit ML) should be split:

    $a + b + 1 \neq d$ [more later cars may exit ML, ***ml_tl*** remains ***green***]
    $a + b + 1 = d$ [no more later cars may exit ML, ***ml_tl*** turns ***red***]

Fixing m_2: Splitting *ML_out* and *IL_out*

- Recall that ***ML_out/inv2_3/INV*** failed ∵ two cases not handled separately:

  $a + b + 1 \neq d$ [more later cars may exit ML, ***ml_tl*** remains **green**]

  $a + b + 1 = d$ [no more later cars may exit ML, ***ml_tl*** turns **red**]

- Similarly, ***IL_out/inv2_4/INV*** would fail ∵ two cases not handled separately:

  $b - 1 \neq 0$ [more later cars may exit IL, ***il_tl*** remains **green**]

  $b - 1 = 0$ [no more later cars may exit IL, ***il_tl*** turns **red**]

- Accordingly, we split ***ML_out*** and ***IL_out*** into two with corresponding guards.

```
ML_out_1
  when
    ml_tl = green
    a + b + 1 ≠ d
  then
    a := a + 1
  end
```

```
ML_out_2
  when
    ml_tl = green
    a + b + 1 = d
  then
    a := a + 1
    ml_tl := red
  end
```

```
IL_out_1
  when
    il_tl = green
    b ≠ 1
  then
    b := b − 1
    c := c + 1
  end
```

```
IL_out_2
  when
    il_tl = green
    b = 1
  then
    b := b − 1
    c := c + 1
    il_tl := red
  end
```

Exercise: Given the latest m_2, how many sequents to prove for ***invariant preservation***?
Exercise: Specify and prove ***ML_out_i*/inv2_3/INV** & ***IL_out_i*/inv2_4/INV** (where $i \in 1 .. 2$).
Exercise: Each split event (e.g., *ML_out_1*) refines its ***abstract*** counterpart (e.g., *ML_out*)?

m_2 Livelocks: New Events Diverging

- Recall that a system may ***livelock*** if the new events diverge.
- Current m_2's two new events ***ML_tl_green*** and ***IL_tl_green*** may *diverge* :

```
ML_tl_green
  when
    ml_tl = red
    a + b < d
    c = 0
  then
    ml_tl := green
    il_tl := red
  end
```

```
IL_tl_green
  when
    il_tl = red
    b > 0
    a = 0
  then
    il_tl := green
    ml_tl := red
  end
```

- *ML_tl_green* and *IL_tl_green* both ***enabled*** and may occur ***indefinitely***, preventing other "old" events (e.g., *ML_out*) from ever happening:

⟨ *init*, *ML_tl_green*, *ML_out_1*, *IL_in*, *IL_tl_green*, *ML_tl_green*, *IL_tl_green*, ... ⟩

init	*ML_tl_green*	*ML_out_1*	*IL_in*	*IL_tl_green*	*ML_tl_green*	*IL_tl_green*
$d = 2$	$d = 2$	$d = 2$	$d = 2$	$d = 2$	$d = 2$	$d = 2$
$a' = 0$	$a' = 0$	$a' = 1$	$a' = 0$	$a' = 0$	$a' = 0$	$a' = 0$
$b' = 0$	$b' = 0$	$b' = 0$	$b' = 1$	$b' = 1$	$b' = 1$	$b' = 1$
$c' = 0$	$c' = 0$	$c' = 0$	$c' = 0$	$c' = 0$	$c' = 0$	$c' = 0$
$ml\_tl = red$	$ml\_tl' = green$	$ml\_tl' = green$	$ml\_tl' = green$	$ml\_tl' = red$	$ml\_tl' = green$	$ml\_tl' = red$
$il\_tl = red$	$il\_tl' = red$	$il\_tl' = red$	$il\_tl' = red$	$il\_tl' = green$	$il\_tl' = red$	$il\_tl' = green$

⇒ Two traffic lights keep changing colors so rapidly that **no** drivers can ever pass!

- **Solution**: Allow color changes between traffic lights in a disciplined way.

Fixing m_2: Regulating Traffic Light Changes

We introduce two variables/flags for regulating traffic light changes:

- ***ml_pass*** is ***1*** **if**, since ***ml_tl*** was last turned **green**, at least one car exited the ML onto the bridge. Otherwise, ***ml_pass*** is ***0***.
- ***il_pass*** is ***1*** **if**, since ***il_tl*** was last turned **green**, at least one car exited the IL onto the bridge. Otherwise, ***il_pass*** is ***0***.

variables: *ml_pass, il_pass*

invariants:
inv2_6 : $ml\_pass \in \{0,1\}$
inv2_7 : $il\_pass \in \{0,1\}$
inv2_8 : $ml\_tl = red \Rightarrow ml\_pass = 1$
inv2_9 : $il\_tl = red \Rightarrow il\_pass = 1$

```
ML_out_1
  when
    ml_tl = green
    a + b + 1 ≠ d
  then
    a := a + 1
    ml_pass := 1
  end
```

```
ML_out_2
  when
    ml_tl = green
    a + b + 1 = d
  then
    a := a + 1
    ml_tl := red
    ml_pass := 1
  end
```

```
IL_out_1
  when
    il_tl = green
    b ≠ 1
  then
    b := b − 1
    c := c + 1
    il_pass := 1
  end
```

```
IL_out_2
  when
    il_tl = green
    b = 1
  then
    b := b − 1
    c := c + 1
    il_tl := red
    il_pass := 1
  end
```

```
ML_tl_green
  when
    ml_tl = red
    a + b < d
    c = 0
    il_pass = 1
  then
    ml_tl := green
    il_tl := red
    ml_pass := 0
  end
```

```
IL_tl_green
  when
    il_tl = red
    b > 0
    a = 0
    ml_pass = 1
  then
    il_tl := green
    ml_tl := red
    il_pass := 0
  end
```


Fixing m_2: Measuring Traffic Light Changes

- Recall:
  - Interleaving of ***new*** events charactered as an integer expression: ***variant***.
  - A variant $V(c, w)$ may refer to constants and/or ***concrete*** variables.
  - In the latest m_2, let's try **variants** : $ml\_pass + il\_pass$
- Accordingly, for the ***new*** event *ML_tl_green*:

$d \in \mathbb{N}$	$d > 0$	
$COLOUR = \{green, red\}$	$green \neq red$	
$n \in \mathbb{N}$	$n \leq d$	
$a \in \mathbb{N}$	$b \in \mathbb{N}$	$c \in \mathbb{N}$
$a + b + c = n$	$a = 0 \lor c = 0$	
$ml\_tl \in COLOUR$	$il\_tl \in COLOUR$	
$ml\_tl = green \Rightarrow a + b < d \land c = 0$	$il\_tl = green \Rightarrow b > 0 \land a = 0$	
$ml\_tl = red \lor il\_tl = red$		
$ml\_pass \in \{0,1\}$	$il\_pass \in \{0,1\}$	
$ml\_tl = red \Rightarrow ml\_pass = 1$	$il\_tl = red \Rightarrow il\_pass = 1$	
$ml\_tl = red$	$a + b < d$	$c = 0$
$il\_pass = 1$		
$\vdash$		
$0 + il\_pass < ml\_pass + il\_pass$		

ML_tl_green/VAR

Exercises: Prove **ML_tl_green/VAR** and Formulate/Prove **IL_tl_green/NAT**.

PO Rule: Relative Deadlock Freedom of m_2

axm0_1 $d \in \mathbb{N}$
axm0_2 $d > 0$
axm2_1 $COLOUR = \{green, red\}$
axm2_2 $green \neq red$
inv0_1 $n \in \mathbb{N}$
inv0_2 $n \leq d$
inv1_1 $a \in \mathbb{N}$
inv1_2 $b \in \mathbb{N}$
inv1_3 $c \in \mathbb{N}$
inv1_4 $a + b + c = n$
inv1_5 $a = 0 \lor c = 0$
inv2_1 $ml\_tl \in COLOUR$
inv2_2 $il\_tl \in COLOUR$
inv2_3 $ml\_tl = green \Rightarrow a + b < d \land c = 0$
inv2_4 $il\_tl = green \Rightarrow b > 0 \land a = 0$
inv2_5 $ml\_tl = red \lor il\_tl = red$
inv2_6 $ml\_pass \in \{0, 1\}$
inv2_7 $il\_pass \in \{0, 1\}$
inv2_8 $ml\_tl = red \Rightarrow ml\_pass = 1$
inv2_9 $il\_tl = red \Rightarrow il\_pass = 1$

Disjunction of *abstract* guards:
- $a + b < d \land c = 0$ — guards of *ML_out* in m_1
- $\lor$ $c > 0$ — guards of *ML_in* in m_1
- $\lor$ $a > 0$ — guards of *IL_in* in m_1
- $\lor$ $b > 0 \land a = 0$ — guards of *IL_out* in m_1

$\vdash$

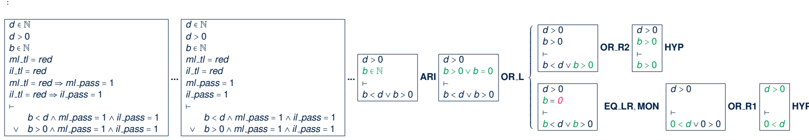
Disjunction of *concrete* guards:
- $ml\_tl = red \land a + b < d \land c = 0 \land il\_pass = 1$ — guards of *ML_tl_green* in m_2
- $\lor$ $il\_tl = red \land b > 0 \land a = 0 \land ml\_pass = 1$ — guards of *IL_tl_green* in m_2
- $\lor$ $ml\_tl = green \land a + b + 1 \neq d$ — guards of *ML_out_1* in m_2
- $\lor$ $ml\_tl = green \land a + b + 1 = d$ — guards of *ML_out_2* in m_2
- $\lor$ $il\_tl = green \land b \neq 1$ — guards of *IL_out_1* in m_2
- $\lor$ $il\_tl = green \land b = 1$ — guards of *IL_out_2* in m_2
- $\lor$ $c > 0$ — guards of *ML_in* in m_2
- $\lor$ $a > 0$ — guards of *IL_in* in m_2

DLF

Proving Refinement: DLF of m_2

$d \in \mathbb{N}$
$d > 0$
$COLOUR = \{green, red\}$
$green \neq red$
$n \in \mathbb{N}$
$n \leq d$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$a + b + c = n$
$a = 0 \lor c = 0$
$ml\_tl \in COLOUR$
$il\_tl \in COLOUR$
$ml\_tl = green \Rightarrow a + b < d \land c = 0$
$il\_tl = green \Rightarrow b > 0 \land a = 0$
$ml\_tl = red \lor il\_tl = red$
$ml\_pass \in \{0, 1\}$
$il\_pass \in \{0, 1\}$
$ml\_tl = red \Rightarrow ml\_pass = 1$
$il\_tl = red \Rightarrow il\_pass = 1$
$a + b < d \land c = 0$
$\lor$ $c > 0$
$\lor$ $a > 0$
$\lor$ $b > 0 \land a = 0$
$\vdash$
$ml\_tl = red \land a + b < d \land c = 0 \land il\_pass = 1$
$\lor$ $il\_tl = red \land b > 0 \land a = 0 \land ml\_pass = 1$
$\lor$ $ml\_tl = green$
$\lor$ $il\_tl = green$
$\lor$ $a > 0$
$\lor$ $c > 0$

⋮



Second Refinement: Summary

- The final version of our ***second refinement*** m_2 is ***provably correct*** w.r.t.:
  - Establishment of ***Concrete Invariants*** [*init*]
  - Preservation of ***Concrete Invariants*** [old & new events]
  - Strengthening of ***guards*** [old events]
  - ***Convergence*** (a.k.a. livelock freedom, non-divergence) [new events]
  - Relative ***Deadlock*** Freedom
- Here is the final specification of m_2:

constants: d

sets: $COLOR$

axioms:
- **axm0_1** : $d \in \mathbb{N}$
- **axm0_2** : $d > 0$
- **axm2_1** : $COLOR = \{green, red\}$
- **axm2_2** : $green \neq red$

variables: a, b, c, ml_tl, il_tl, ml_pass, il_pass

invariants:
- **inv2_1** : $ml\_tl \in COLOUR$
- **inv2_2** : $il\_tl \in COLOUR$
- **inv2_3** : $ml\_tl = green \Rightarrow a + b < d \land c = 0$
- **inv2_4** : $il\_tl = green \Rightarrow b > 0 \land a = 0$
- **inv2_5** : $ml\_tl = red \lor il\_tl = red$
- **inv2_6** : $ml\_pass \in \{0, 1\}$
- **inv2_7** : $il\_pass \in \{0, 1\}$
- **inv2_8** : $ml\_tl = red \Rightarrow ml\_pass = 1$
- **inv2_9** : $il\_tl = red \Rightarrow il\_pass = 1$

variants: $ml\_pass + il\_pass$

```
ML_tl_green
  when
    ml_tl = red
    a + b < d
    c = 0
    il_pass = 1
  then
    ml_tl := green
    il_tl := red
    ml_pass := 0
  end

IL_tl_green
  when
    il_tl = red
    b > 0
    a = 0
    ml_pass = 1
  then
    il_tl := green
    ml_tl := red
    il_pass := 0
  end

ML_out_1
  when
    ml_tl = green
    a + b + 1 ≠ d
  then
    a := a + 1
    ml_pass := 1
  end

ML_out_2
  when
    ml_tl = green
    a + b + 1 = d
  then
    a := a + 1
    ml_tl := red
    ml_pass := 1
  end

IL_out_1
  when
    il_tl = green
    b ≠ 1
  then
    b := b − 1
    c := c + 1
    il_pass := 1
  end

IL_out_2
  when
    il_tl = green
    b = 1
  then
    b := b − 1
    c := c + 1
    il_tl := red
    il_pass := 1
  end

ML_in
  when
    c > 0
  then
    c := c − 1
  end

IL_in
  when
    a > 0
  then
    a := a − 1
    b := b + 1
  end
```


Index (1)

Learning Outcomes
Recall: Correct by Construction
State Space of a Model
Roadmap of this Module
Requirements Document: Mainland, Island
Requirements Document: E-Descriptions
Requirements Document: R-Descriptions
Requirements Document: Visual Summary of Equipment Pieces
Refinement Strategy
Model m_0: Abstraction

Index (2)

Model m_0: State Space

Model m_0: State Transitions via Events

Model m_0: Actions vs. Before-After Predicates

Design of Events: Invariant Preservation

Sequents: Syntax and Semantics

PO of Invariant Preservation: Sketch

PO of Invariant Preservation: Components

Rule of Invariant Preservation: Sequents

Inference Rules: Syntax and Semantics

Proof of Sequent: Steps and Structure

Example Inference Rules (1)

Index (3)

Example Inference Rules (2)

Example Inference Rules (3)

Revisiting Design of Events: *ML_out*

Revisiting Design of Events: *ML_in*

Fixing the Design of Events

Revisiting Fixed Design of Events: *ML_out*

Revisiting Fixed Design of Events: *ML_in*

Initializing the Abstract System m_0

PO of Invariant Establishment

Discharging PO of Invariant Establishment

System Property: Deadlock Freedom

Index (4)

PO of Deadlock Freedom (1)

PO of Deadlock Freedom (2)

Example Inference Rules (4)

Example Inference Rules (5)

Discharging PO of DLF: Exercise

Discharging PO of DLF: First Attempt

Why Did the DLF PO Fail to Discharge?

Fixing the Context of Initial Model

Discharging PO of DLF: Second Attempt

Initial Model: Summary

Model m_1: "More Concrete" Abstraction

Index (5)

Model m_1: Refined State Space

Model m_1: State Transitions via Events

Model m_1: Actions vs. Before-After Predicates

States & Invariants: Abstract vs. Concrete

Events: Abstract vs. Concrete

PO of Refinement: Components (1)

PO of Refinement: Components (2)

PO of Refinement: Components (3)

Sketching PO of Refinement

Refinement Rule: Guard Strengthening

PO Rule: Guard Strengthening of *ML_out*

Index (6)

PO Rule: Guard Strengthening of *ML_in*

Proving Refinement: ML_out/GRD

Proving Refinement: ML_in/GRD

Refinement Rule: Invariant Preservation

Visualizing Inv. Preservation in Refinement

INV PO of m_1: ML_out/inv1_4/INV

INV PO of m_1: ML_in/inv1_5/INV

Proving Refinement: ML_out/inv1_4/INV

Proving Refinement: ML_in/inv1_5/INV

Initializing the Refined System m_1

PO of m_1 Concrete Invariant Establishment

Index (7)

Discharging PO of m_1
Concrete Invariant Establishment

Model m_1: New, Concrete Events

Model m_1: BA Predicates of Multiple Actions

Visualizing Inv. Preservation in Refinement

Refinement Rule: Invariant Preservation

INV PO of m_1: IL_in/inv1_4/INV

INV PO of m_1: IL_in/inv1_5/INV

Proving Refinement: IL_in/inv1_4/INV

Proving Refinement: IL_in/inv1_5/INV

Livelock Caused by New Events Diverging

Index (8)

PO of Convergence of New Events

PO of Convergence of New Events: NAT

PO of Convergence of New Events: VAR

Convergence of New Events: Exercise

PO of Refinement: Deadlock Freedom

PO Rule: Relative Deadlock Freedom of m_1

Example Inference Rules (6)

Proving Refinement: DLF of m_1

Proving Refinement: DLF of m_1 (continued)

First Refinement: Summary

Model m_2: "More Concrete" Abstraction

Index (9)

Model m_2: Refined, Concrete State Space

Model m_2: Refining Old, Abstract Events

Model m_2: New, Concrete Events

Invariant Preservation in Refinement m_2

INV PO of m_2: ML_out/inv2_4/INV

INV PO of m_2: IL_out/inv2_3/INV

Example Inference Rules (7)

Proving ML_out/inv2_4/INV: First Attempt

Proving IL_out/inv2_3/INV: First Attempt

Failed: ML_out/inv2_4/INV, IL_out/inv2_3/INV

Fixing m_2: Adding an Invariant

Index (10)

INV PO of m_2: ML_out/inv2_4/INV – Updated

INV PO of m_2: IL_out/inv2_3/INV – Updated

Proving ML_out/inv2_4/INV: Second Attempt

Proving IL_out/inv2_3/INV: Second Attempt

Fixing m_2: Adding Actions

INV PO of m_2: ML_out/inv2_3/INV

Proving ML_out/inv2_3/INV: First Attempt

Failed: ML_out/inv2_3/INV

Fixing m_2: Splitting *ML_out* and *IL_out*

m_2 Livelocks: New Events Diverging

Fixing m_2: Regulating Traffic Light Changes

Index (11)

Fixing m_2: Measuring Traffic Light Changes

PO Rule: Relative Deadlock Freedom of m_2

Proving Refinement: DLF of m_2

Second Refinement: Summary