CS:4980
Foundations of Embedded Systems

Safety Requirements
Part I

Requirements

Desirable properties of the executions of the system

- **Informal:** either implicit, or stated in natural language
- **Formal:** stated explicitly in a mathematically precise way

Model/design/system meets the requirements if every execution satisfies them

Clear separation between

- requirements, what needs to be implemented, and
- system, how it is implemented

Requirements

High assurance / safety-critical systems are typically provided with precise requirements

Verification Problem:

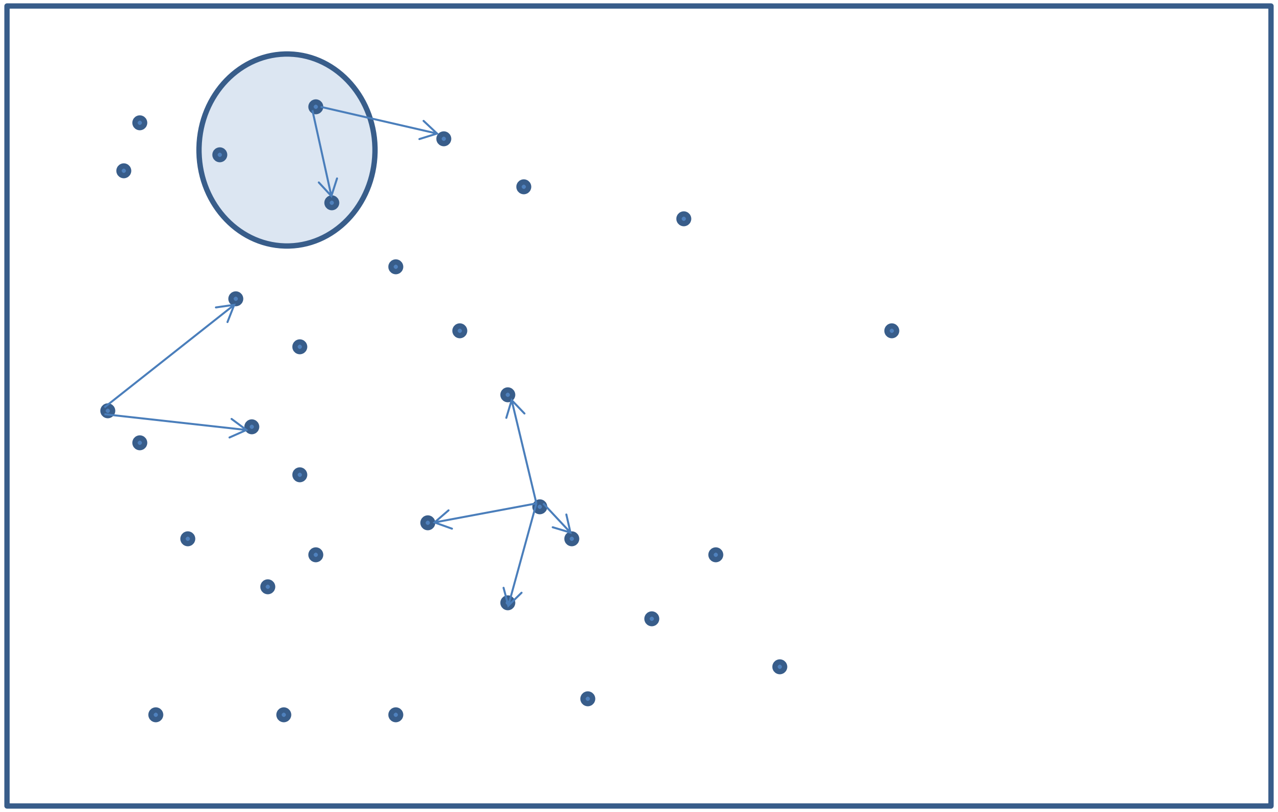
Given a formally specified requirement R and a system/model C, prove or disprove that C satisfies R

Safety and Liveness Requirements

- A *safety requirement* states that a system always stays within *good* states (i.e., nothing bad ever happens)
  - Leader election: it is never the case that two nodes consider themselves to be leaders
  - Collision avoidance: Distance between two cars is always greater than some minimum threshold
- A *liveness requirement* states that a system eventually achieves its goal (i.e., something good eventually happens)
  - Leader election: Each node eventually makes a decision
  - Cruise controller: Actual speed eventually equals desired speed
- Formalization and analysis techniques for safety and liveness differ significantly.
- We will start with safety

Transition Systems

State space + Initial states + Transitions between states



Definition of Transition System

Syntax: a transition system T has

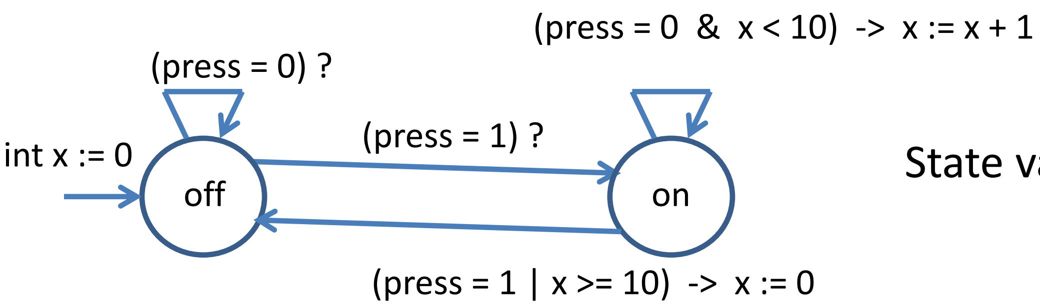
1. A set S of (typed) state variables
2. An initialization Init for state variables
3. A description Trans of how to move from one state to the next

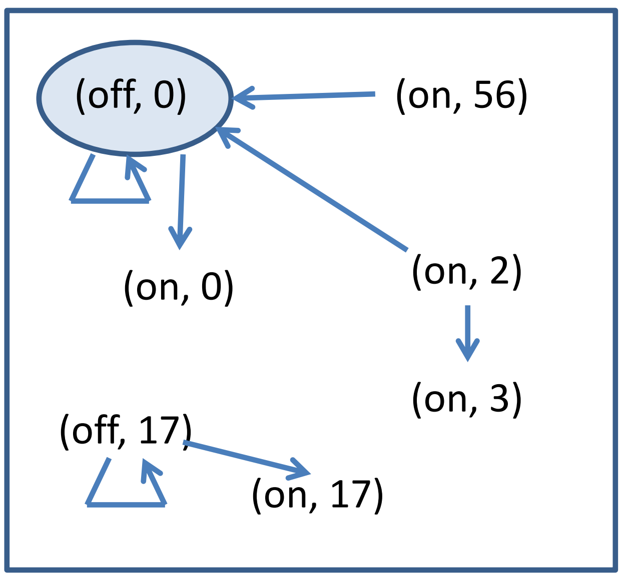
Semantics:

1. Set Q_S of states
2. Set [Init] of initial states, a subset of Q_S
3. Set [Trans] of transitions, a subset of $Q_S \times Q_S$

Synchronous reactive components, EMS, programs, and computational systems in general, all have an underlying transition system

Switch Transition System





State variables:

{off, on} mode, int x

Initialization:

mode := off ; x := 0

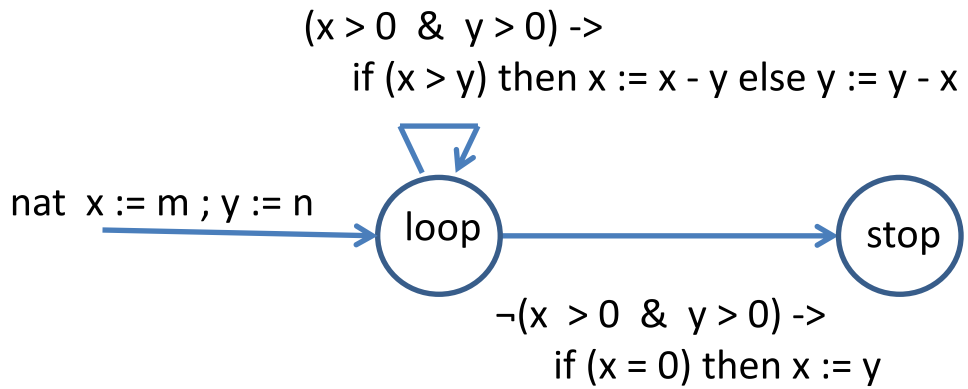
Transitions:

(off, n) -> (off, n) ;
(off, n) -> (on, n) ;
(on, n) -> (on, n+1) if n < 10 ;
(on, n) -> (off, 0)

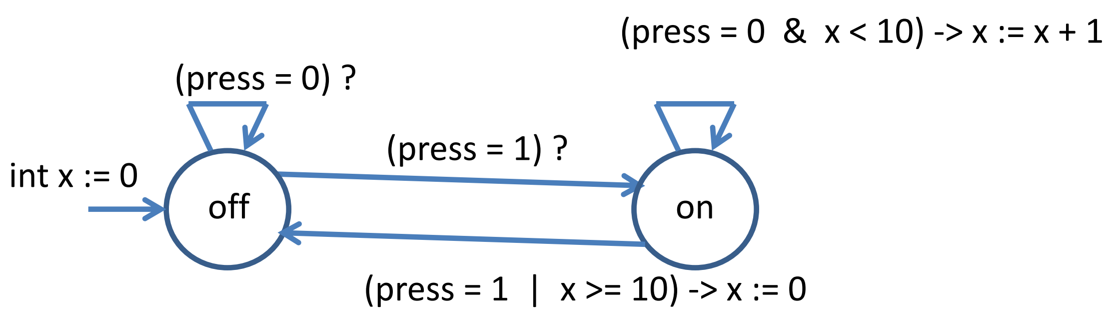
- Input/output variables become local vars
- Values for input vars are chosen non-deterministically

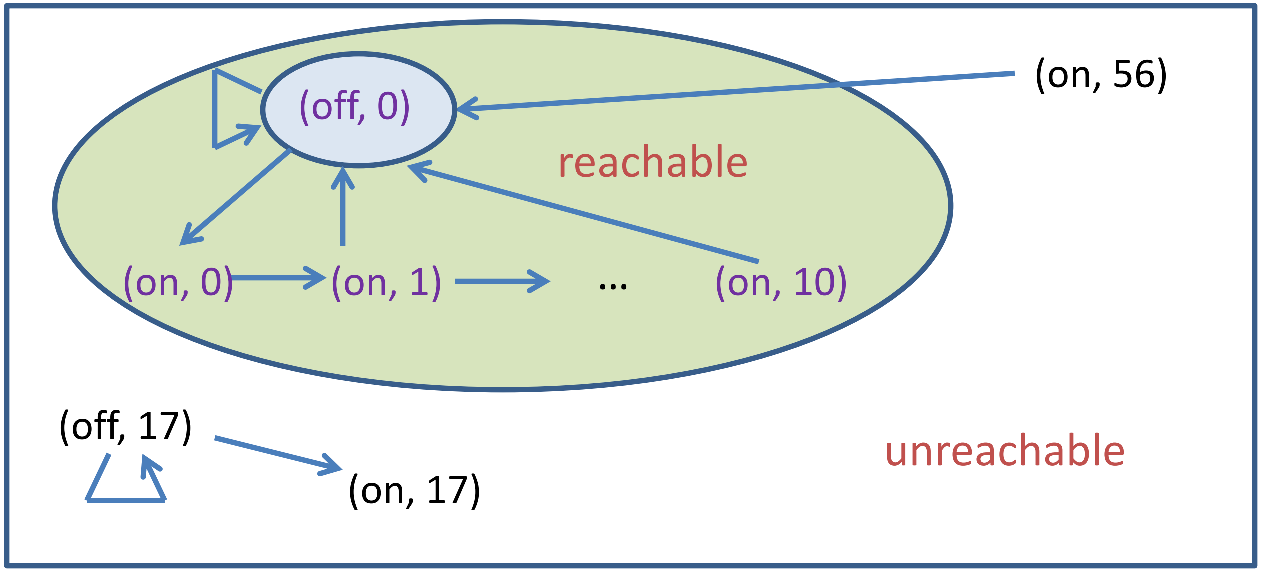
Euclid's GCD Algorithm

Classical program to compute greatest common divisor of (non-negative) input numbers m and n

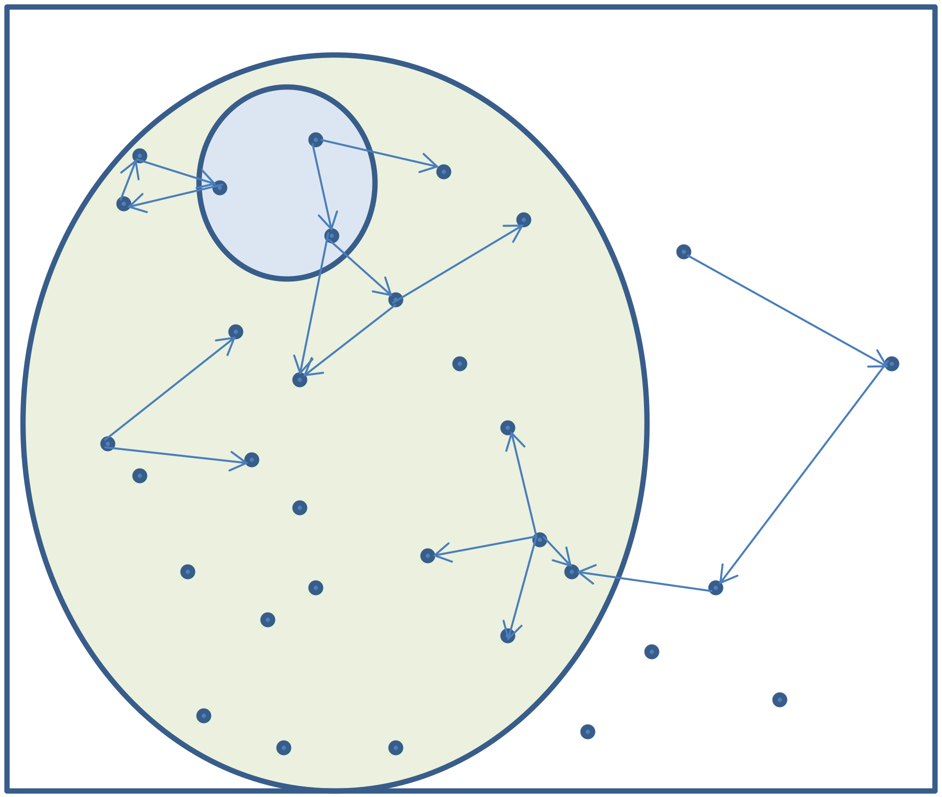


Reachable States



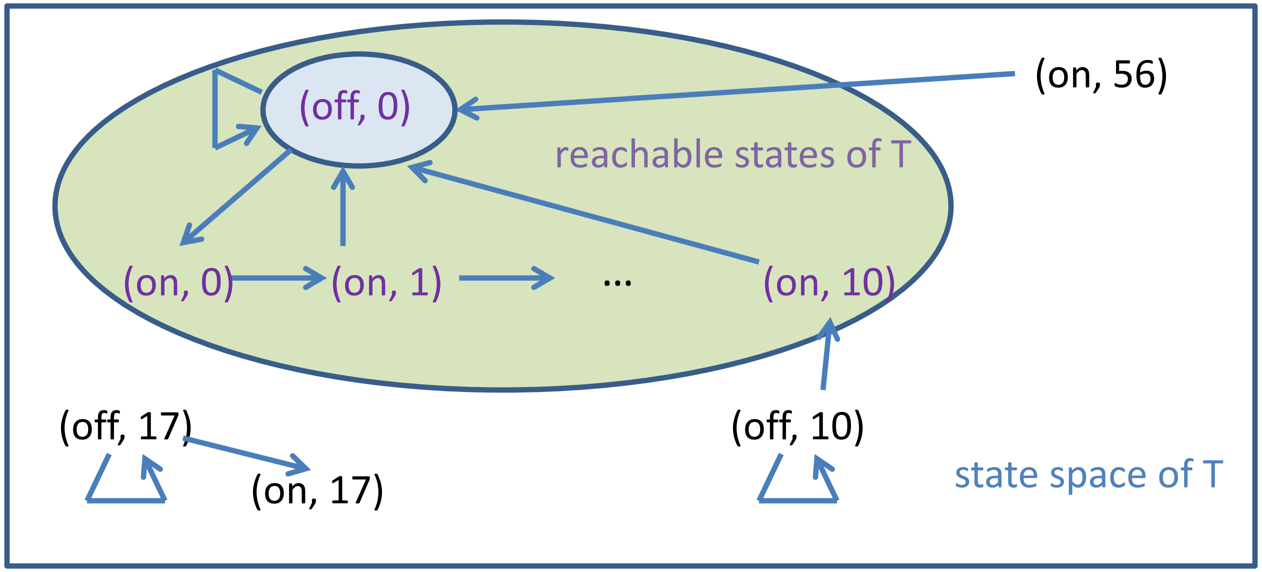


Reachable States of Transition Systems



A state s of a transition system T is *reachable* if there is an execution starting in an initial state of T and ending in s

Invariants



- A *property* of a transition system T is a Boolean-valued expression P over state variables
- Property P is an *invariant* of T if every reachable state satisfies P
- Some invariants for T above: x <= 10, x <= 50, mode = off => x = 0
- Some non-invariants for T above: x < 10, mode = off

Invariants

- We express safety requirements for a transition system T as properties P of T's state variables
  - If P is invariant then T is *safe* (wrt P)
  - If P is not invariant, then some *bad* state, satisfying ¬P, is reachable

    (the execution leading to such a state is a *counterexample*)

- Leader election:

  $(r_n = N) \Rightarrow (id_n = \max I)$ I : set of identifiers of all nodes

- Euclid's GCD Program:

  (mode = stop) => (x = gcd(m, n))

Formal Verification

Grand challenge: automate verification as much as possible!

Analysis Techniques

- Dynamic Analysis (runtime)
  - Execute the system, possibly multiple times with different inputs
  - Check if every execution meets the desired requirement
- Static Analysis (design time)
  - Analyze the source code or the model for possible bugs
- Trade-offs
  - Dynamic analysis is incomplete but accurate (checks real system, and discovered bugs are real bugs)
  - Static analysis can be complete and can catch design bugs early but many static analysis techniques are not scalable (solution: analyze approximate models, can lead to false warnings)

Invariant Verification

Simulation

- Simulate the model, possibly multiple times with different inputs
- Easy to implement, scalable, but no correctness guarantees

Deductive verification

- Construct a proof that system satisfies the invariant
- Usually requires manual effort (but partial automation often possible)

Model checking

- Automatically explores all reachable states to check invariants
- Not scalable, but current tools can analyze many real-world designs (relies on many interesting theoretical advances)

Note: Newer techniques are blurring the differences between deductive verification and model checking

Proving Invariants

- Given a transition system T = (S, Init, Trans), and a property P, prove that all reachable states of T satisfy P

- Inductive definition of *reachable state*:
  - All initial states are reachable in 0 transitions
  - If a state s is reachable in k transitions and s -> t is a transition, then the state t is reachable in k+1 transitions
  - Reachable = Reachable in n transitions, for some n

- Prove: for all n, states reachable in n transitions satisfy P
  - *Base case*: Show that all initial states satisfy P
  - *Inductive case*:
    1. Assume that a state s satisfies P
    2. Show that if s -> t is a transition then t must satisfy P

Recall: Inductive Proofs in Arithmetic

- To show that a statement P holds for all natural numbers n,
  - Base case: Prove that P holds for $n = 0$
  - Assume that P holds for an arbitrary natural k
  - Using the assumption, prove that P holds for $k+1$

- Example statement: For all n,

$$(0 + 1 + 2 + \ldots + n) = n(n+1)/2$$

Inductive Invariant

A property P is an *inductive invariant* of transition system T if

1. Every initial state of T satisfies P
2. If a state satisfies P and s -> t is a transition of T, then t satisfies P

Note:

1. If P is an inductive invariant of T, then all reachable states of T must satisfy P, and thus, it is an invariant of T

2. There are invariants which are not inductive

Proving Inductive Invariant Example (1)

Consider transition system T given by

- State variable int x, initialized to 0
- Transition description given by if (x < m) then x := x+1 for some m >= 0

Is the property P : 0 <= x <= m an inductive invariant of T?

- ❑ Base case: Consider initial state x := 0. Check that it satisfies P
- ❑ Inductive case:
  - Consider an arbitrary state s, suppose s(x) = a
  - Assume that s satisfies P, that is, assume 0 <= a <= m
  - Consider the state t obtained by executing a transition from s
  - If a < m then t(x) = a+1, else t(x) = a
  - In either case, 0 <= t(x) <= m
  - So t satisfies the property P, and the proof is complete

Proving Inductive Invariant Example (2)

Consider transition system T given by

- State variables int x, y; initially: x := 0 ; y := m for some m > 0
- Transition description given by if (x < m) then { x := x+1 ; y := y-1 }

Is the property P : 0 <= y <= m an inductive invariant of T?

- ❑ Base case: Consider initial state (x := 0, y := m). Check that it satisfies P
- ❑ Inductive case:
  - Consider an arbitrary state s with x = a and y = b
  - Assume that s satisfies P, that is, assume 0 <= b <= m
  - Consider the state t obtained by executing a transition from s
  - If a < m then t(y) = b-1, else t(y) = b
  - Can we conclude that 0 <= t(y) <= m?
  - No! When b = 0, t(y) is negative.
  - The proof fails. In fact, P is not an inductive invariant of T!

Why did the proof fail?

- Consider the state s with x = 0 and y = 0
  - State s satisfies P: $0 <= y <= m$
  - Executing a transition from s leads to state t with x = 1 and y = -1
  - State t does not satisfy P
- However, the state s in above argument is not reachable!
- **Cause of failure:** The property P did not capture correlation between the state components x and y
- **Solution:** *Inductive Strengthening*
  - Consider property Q : $(0 <= y <= m)$ & $(x + y = m)$
  - Property Q implies property P
  - While P is not an inductive invariant, Q is!
  - It follows that all reachable states must satisfy P

Proving Inductive Invariant Example (3)

Consider transition system T given by

- State variables int x, y; initially: x := 0 ; y := m for some m > 0
- Transition description given by if (x < m) then { x := x+1 ; y := y-1 }

Property Q : (0 <= y <= m) & (x + y = m)

- ❑ Base case: Consider initial state (x := 0, y := m). Check that it satisfies Q
- ❑ Inductive case:
  - Consider an arbitrary state s with x = a and y = b
  - Assume that s satisfies Q, that is, assume 0 <= b <= m and a+b = m
  - Consider the state t obtained by executing a transition from s
  - If a < m then t(x) = a+1 and t(y) = b-1, else t(x) = a and t(y) = b
  - But if a < m, since b = m-a, then b > 0, and thus b-1 >= 0
  - In either case, the condition (0 <= t(y) <= m) & (t(x)+t(y) = m) holds
- ❑ Conclusion: Property Q is an inductive invariant

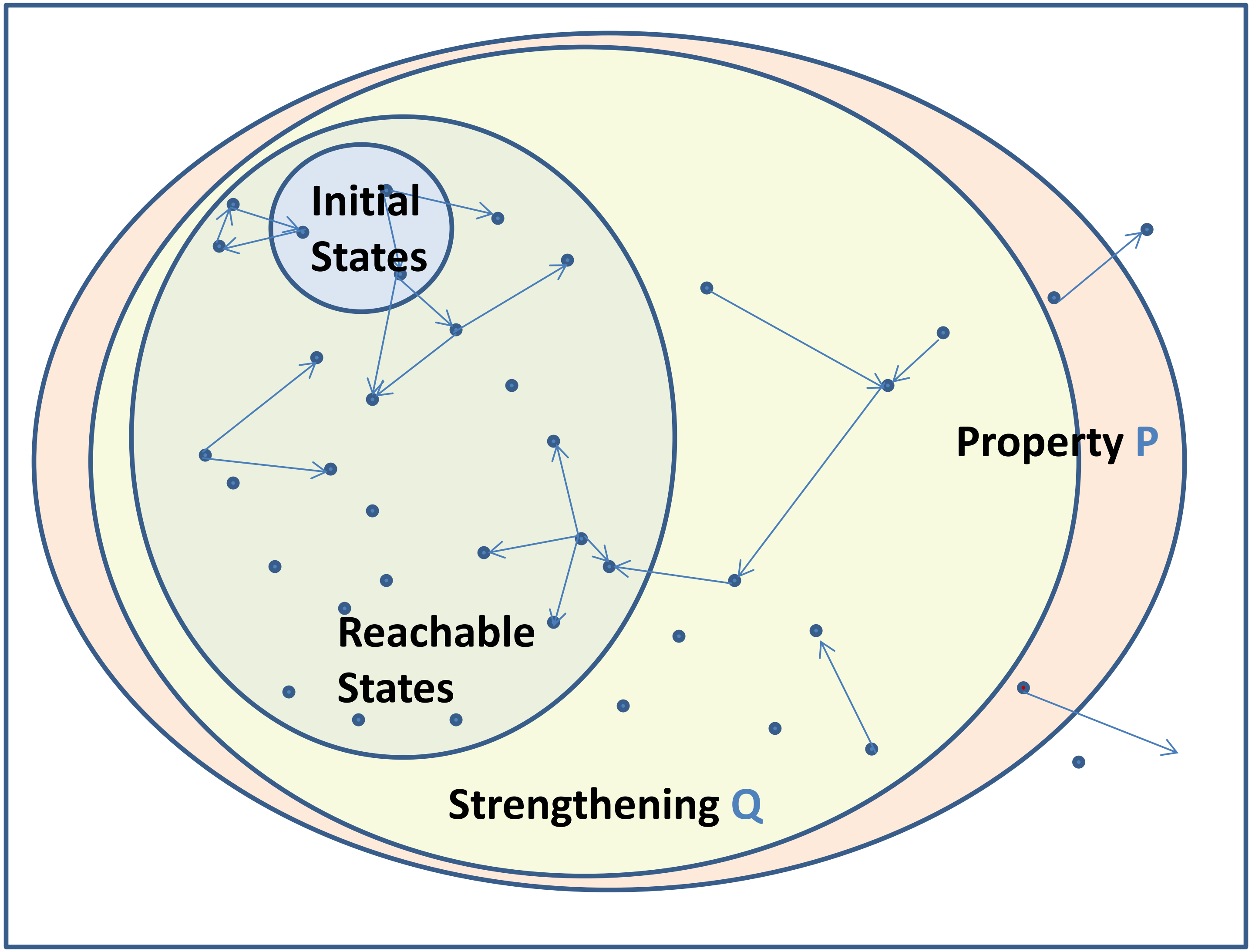
Proof Rule for Proving Invariants

- To establish that a property P is an invariant of transition system T
- Find an *inductive strengthening* of P: a property Q such that
  1. Q implies P (i.e., every state satisfying Q also satisfies P)
  2. Q is an inductive invariant:
     - all initial states satisfies Q
     - For any states s, t such as s satisfies Q and s -> t is a transition, t satisfies Q
- This is a sound and complete strategy for establishing invariants

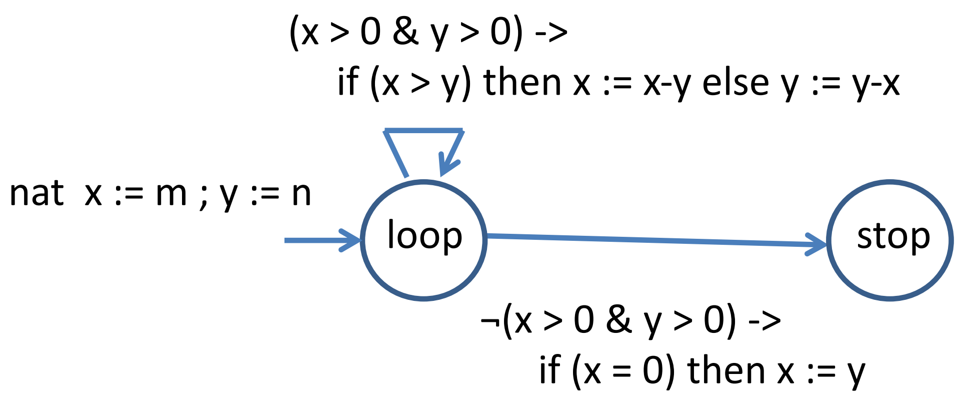
  Sound: If P has an inductive strengthening Q then P is indeed invariant

  Complete: If P is an invariant, then it has an inductive strengthening Q (however, it may not be representable in the chosen property language)

Inductive Strengthening



Correctness of GCD



- **Property** P **:** gcd(x, y) = gcd(m, n)
- Verify that P is an inductive invariant (Exercise)
- Captures the core logic of the program: even though x and y are updated at every step, their gcd stays unchanged
- When switching to stop, if x is 0, then gcd(x, y) is y; if y = 0, then gcd(x, y) = x, and thus x = gcd(m, n) upon switching to stop
- **Note:** (mode = stop) => (x = gcd(m, n)) is invariant, but not inductive

Transition System for Leader Election

For each node n

Initial state:

$\text{int } id_n := n$; $\text{int } r_n := 1$

State transition update:

- Round counters r_n :

  if $r_n < N$ then $r_n := r_n + 1$

- Identifiers r_n :

  $id_n := \max \{id_n, \max \{id_m \mid m \text{ is connected to } n\}\}$

Invariants for Leader Election

Initial state: for all n, int id_n := n; int r_n := 1

State transition update: for all n,

- if $r_n < N$ then $r_n := r_n + 1$
- $id_n := \max \{id_n, \max \{id_m \mid m \text{ is connected to } n\}\}$

Property: for all n, $id_n >= n$

- Obviously an invariant; is it an inductive invariant?

Property: $id_1 \in ID$ with ID set of identifiers of all nodes

- Not an inductive invariant!
- During a transition s -> t, value of id_1 in state t may equal value of id_m in state s, but property says nothing about $s(id_m)$
- What about: forall n, $id_n \in ID$? (Exercise)

Correctness of Leader Election

We expect id_n to be maximum of all identifiers after N rounds

Property: for all n, $(r_n = N) \Rightarrow (id_n = \max ID)$

- Not inductive

Goal: Find inductive strengthening capturing co-relation among all state variables at intermediate steps

Observe: for all nodes n, after k rounds r_n is k and id_n is max of ids of nodes that are $< k$ hops away from n

Property:

P_1 : for all m, n, $r_m = r_n$

& P_2 : for all n, $id_n = \max \{ c \mid distance(c, n) < r_n \}$

Let's prove that P_1 & P_2 is an inductive invariant!

Proof: Base Case

Initial state s: for each node n, $s(id_n) = n$ and $s(r_n) = 1$

Goal: Show that the following both hold in this initial state s

P_1 : for all m, n, $r_m = r_n$

P_2 : for all n, $id_n = \max \{ c \mid distance(c, n) < r_n \}$

P_1) $s(r_m) = s(r_n) = 1$; so P_1 holds

P_2) Consider a node n, we want to show

$$s(id_n) = \max \{ c \mid distance(c, n) < 1 \}$$

The only node c with $distance(c, n) < 1$ is n itself, and $s(id_n) = n$, so P_2 holds

Proof: Inductive Case

- Consider an arbitrary state s, and assume both P_1 and P_2 hold
- Let $s(r_n) = k$, for each node n
- For $k < N$, consider a successor state t of s
- Goal: Show that both P_1 and P_2 hold in state t
- To show P_1 , consider two nodes m and n
  - $t(r_m) = s(r_m) + 1 = k+1$, and similarly, $t(r_n) = k+1$, so P_1 holds in t
- To show P_2, consider a node n, we want to show

  $$t(id_n) = \max \{ c \mid distance(c, n) < k+1 \}$$

  - Assumption 1 (from inductive hypothesis): for all m,

    $s(id_m) = \max \{ c \mid distance(c, m) < k\}$
  - Assumption 2 (from the transition relation):

    $t(id_n) = \max \{ s(id_n), \max \{s(id_c) \mid c \text{ is linked to } n \} \}$

Proof: Inductive Case (Continued)

- Let $h = \max \{ c \mid distance(c, n) < k+1 \}$ and $d = distance(h, n)$
- Goal: show that $t(id_n) = h$
- Since $d < k+1$, either $d < k$ or $d = k$

  Case ($d < k$)

  - By Assumption 1, $s(id_n)$ cannot be $< h$, so must be h
  - By Assumption 2, $t(id_n)$ cannot be $< s(id_n)$, so must be h

  Case ($d = k$)

  - By basic properties of graphs, there must be a node m such that $distance(h, m) = k - 1$ and m is linked to n
  - By Assumption 1, $s(id_m)$ cannot be $< h$, so must be h
  - By Assumption 2, $t(id_n)$ cannot be $< s(id_m)$, so must be h
- The proof is complete!

Summary of Invariants

- General way to formulate and prove safety properties of programs/models/systems
- Inductive invariant:
  - Holds in initial states
  - Is preserved by every transition
- To be inductive, property needs to capture relevant relationships among all state variables
- Benefit of finding inductive invariants:
  - Correctness reasoning becomes local (one needs to think about what happens in one step)
  - Tools available to check if a property is an inductive invariant
- Area of active research: can a tool discover them automatically?

Automated Invariant Verification

Can such a verifier exist?

If so, what is the computational complexity of the verification problem?

Solving Invariant Verification

Establishing system safety is important, but there is no generally efficient procedure to solve the verification problem

Solution 1: Simulation-based analysis

- Simulate the model multiple times, and check that each state encountered on each execution satisfies desired safety property
- Useful, practical in real-world, but gives only partial guarantee (and is known to miss hard-to-find bugs)

Solution 2: Semi-automated formal proofs using inductive invariants

- Only partial tool support possible, so requires considerable effort
- Recent successes: verified microprocessor, web browser, JVM

Solution 3: Exhaustive search through state-space

- Fully automated, but with scalability limitations (may not work!)
- Complementary to simulation, increasingly used in industry
- Two approaches: On-the-fly enumerative search, symbolic search

Credits

Notes based on Chapter 3 of

Principles of Cyber-Physical Systems
by Rajeev Alur
MIT Press, 2015