THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

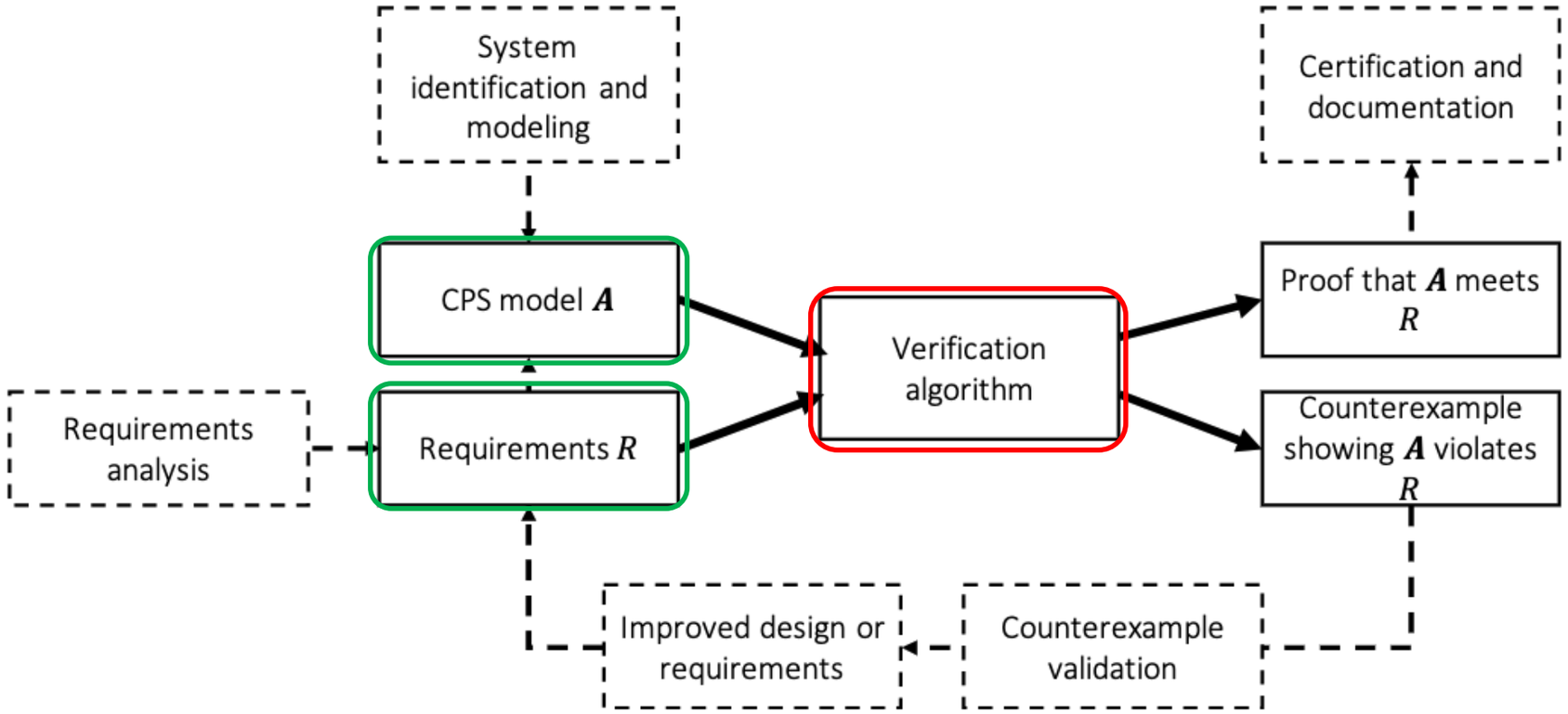
Verifying Cyber-Physical Systems

Chapter 7 – Verifying Invariants

Abenezer Taye

Oct 10, 2022

System Design Ecosystem



What is an invariant?

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

- An invariant I for a system $\mathcal{A}$ is a requirement that **always holds** (I is never violated in any execution of $\mathcal{A}$).
- **Formally**: for a hybrid automata $\mathcal{A} =< V, \Theta, A, \mathcal{D}, \mathcal{T} >$, an invariant I is a subset of the state-space V such that all the reachable states of $\mathcal{A}$ are contained in I.

Theorem 7.1. *Given an HIOA $\mathcal{A}$, if a set of states $I \subseteq val(X)$ satisfies the following:*

(a) (Start condition) For any starting state $\mathbf{x} \in \Theta$, $\mathbf{x} \in I$, and

(b) (Transition closure) for any action $a \in A$, if $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ and $\mathbf{x} \in I$ then $\mathbf{x}' \in I$, and

(c) (Trajectory closure) for any trajectory $\tau \in \mathcal{T}$, if τ.fstate $\in I$ then τ.lstate $\in I$,

then I is an inductive invariant of $\mathcal{A}$.

Historical perspective on invariants

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

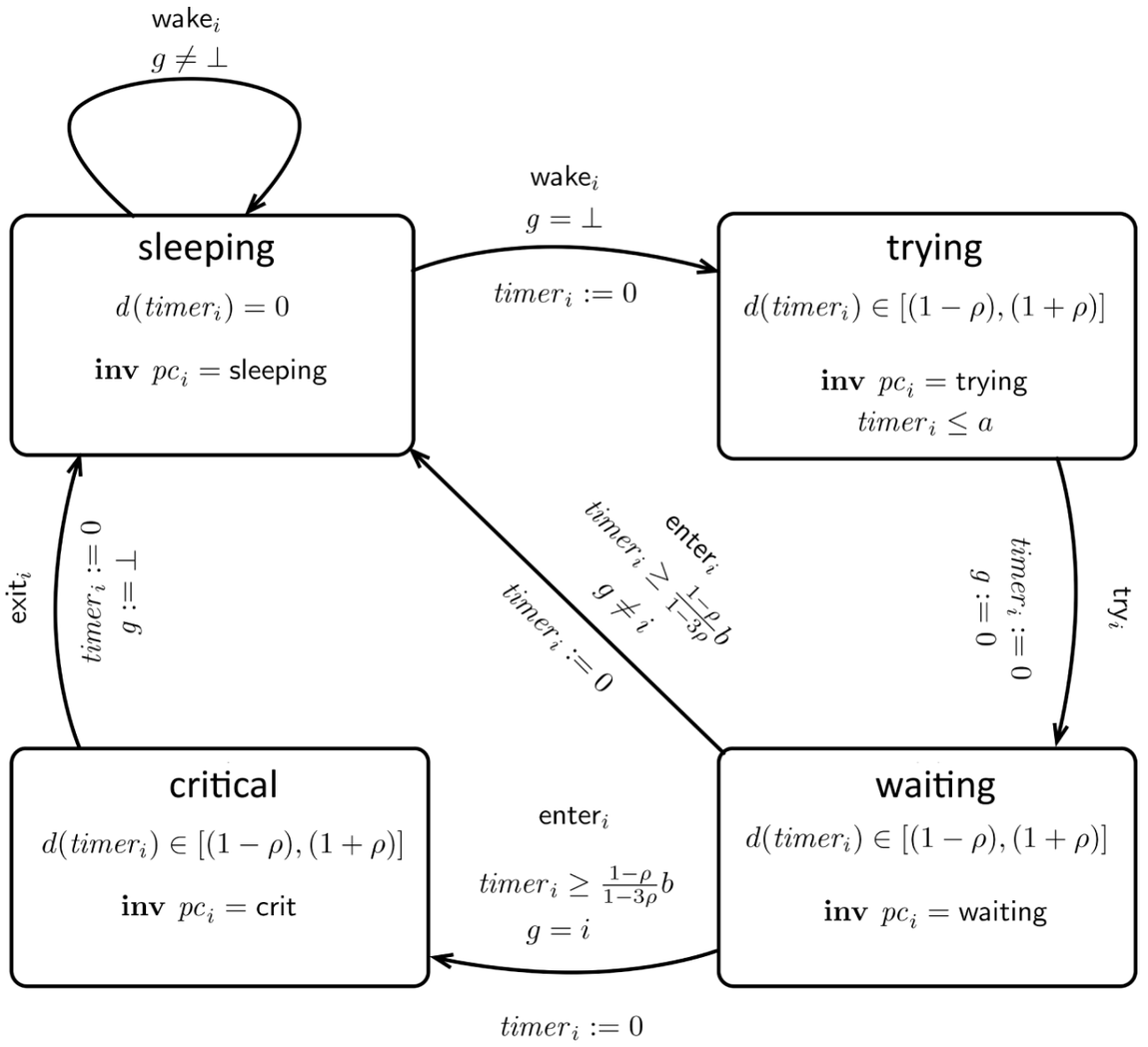
- The idea of Invariants dates to the classical program analysis technique called **Floyd-Hoare style verification** (Floyd, 1967).
- **Hoare logic/ Floyd-Hoare logic/ Hoare rules** – provides a set of logical rules for rigorously deducing the correctness of automata or programs.
- **Hoare logic** is built on a notion called **Hoare triple** – describes how the execution of a statement changes the state of the automaton.
- Hoare logic **provide axioms and inference rules** that various programming language constructs can use to infer properties from the triples.

Proving invariants via inductive reasoning

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

- Example: Fisher's mutual exclusion algorithm
- The algorithm allows n processes to coordinate and access a critical resource exclusively.
  - No two processes have simultaneous access, and all requesting processes eventually gain access
- **Our aim**: establish the correctness of a distributed mutual exclusion algorithm
- $g - single\ shared\ memory$
- $phases - sleeping, trying, waiting, and\ critical$
- $internal\ variables$: $pc_i -$ $program\ counter, and\ timer_i - stopwatch$

$internal\ actions$: $wake, try, enter, exit$



Analysis of Fisher's mutual exclusion

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

- **Ultimate goal**: to establish the safety requirement for $\mathcal{A}$ that no two processes enter the critical section simultaneously.

$$\forall\, i, j \in [n], i \neq j \wedge pc_i = critical\ implies\ pc_j \neq critical$$

- **Approach**: establish a sequence of simpler invariants
- **Invariant 1**: $\forall\, i \in [n], if\ pc_i = sleeping,\ then\ timer_i = 0; otherwise\ timer_i \geq 0$
  - This simply states that for any process i, the timer is always nonnegative and stays at 0 when the process is sleeping
- **Invariant 2**: $\forall\, i, j \in [n],\ pc_i = waiting \wedge pc_j = trying \wedge g = i\ implies$

$$timer_i - timer_j < \frac{2\rho}{1-\rho} timer_i + b - a$$

  - This asserts the gap between the timer values of two processes that are attempting to access the critical section
- **Invariant 3**: $\forall\, i, j \in [n]\ pc_i = critical\ implies\ pc_j \neq trying \wedge g = i$
  - If process i is already in the critical section, then $g = i$, and no other process could be trying.

Barriers to automated invariant proving

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

- Two barriers to automating the process of proving invariants
- **Barrier 1**: automating the strengthening step → requires an understanding of how the system works
- **Barrier 2**: need for explicit trajectories for checking the trajectory condition of Theorem 7.1
- Addressing barrier 2: proving invariants without solving ODEs
- **Main idea**: look at the boundary of the invariant set I, and prove that the trajectories cannot escape the boundary
- **Lemma**: I is an inductive invariant of an ODE $\dot{x} = f(x)$ if at every state x on the boundary of I, the vector $f(x)$ is pointing inward from the boundary $\frac{\partial F(x)}{\partial x} . f(x) \geq 0$.
- The above condition is called **sub-tangential condition**, and the boundary is called **barrier**.

Barrier certificates

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

- They can be used to construct safety proofs relative to an unsafe set.
- Key idea: find a barrier function that separates the unsafe set from the reachset

Theorem 7.5. *Let $\mathcal{A} = (V, \Theta, A, \mathcal{D}, \mathcal{T})$ be a simple hybrid automaton with $V = X \cup loc$, set of locations $L = type(loc)$, and unsafe set $U \subseteq val(V)$. Suppose that there is a collection $\{B_\ell(\boldsymbol{x})\}$ of functions B_ℓ: $val(X) \to \mathbb{R}$ such that for all $\ell \in L$, B_ℓ is differentiable with respect to its argument, and it satisfies:*

(a) (Start condition) *For any starting state $\boldsymbol{v} \in \Theta_\ell$, $B_\ell(\boldsymbol{x}) \leq 0$.*

(b) (Unsafe condition) *For any unsafe state $\boldsymbol{v} \in U_\ell$, $B_\ell(\boldsymbol{x}) > 0$.*

(c) (Transition condition) *For any transition $(\mathbf{x}, \ell) \xrightarrow{a} (\mathbf{x}', \ell')$, $B_\ell(\mathbf{x}) \leq 0 \Rightarrow B'_\ell(\mathbf{x}') \leq 0$.*

(d) (Flow condition) *For any state $\boldsymbol{x} \in I_\ell$, on the boundary with $B_\ell(\boldsymbol{x}) = 0$, $\frac{\partial_\ell}{\partial \mathbf{x}} f_\ell(\mathbf{x}) \leq 0$.*

Then $\mathcal{A}$ is safe with respect to U (i.e., $\mathsf{Reach}_{\mathcal{A}} \cap U = \emptyset$).

Satisfiability and satisfiability modulo theories

- Satisfiability:
  - Given a propositional logic formula ∅ over a set of variables X, decide if there is a valuation that satisfies ∅.
- For example: given the propositional formula

$$\phi(x_1, x_2, x_3) := (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_1),$$

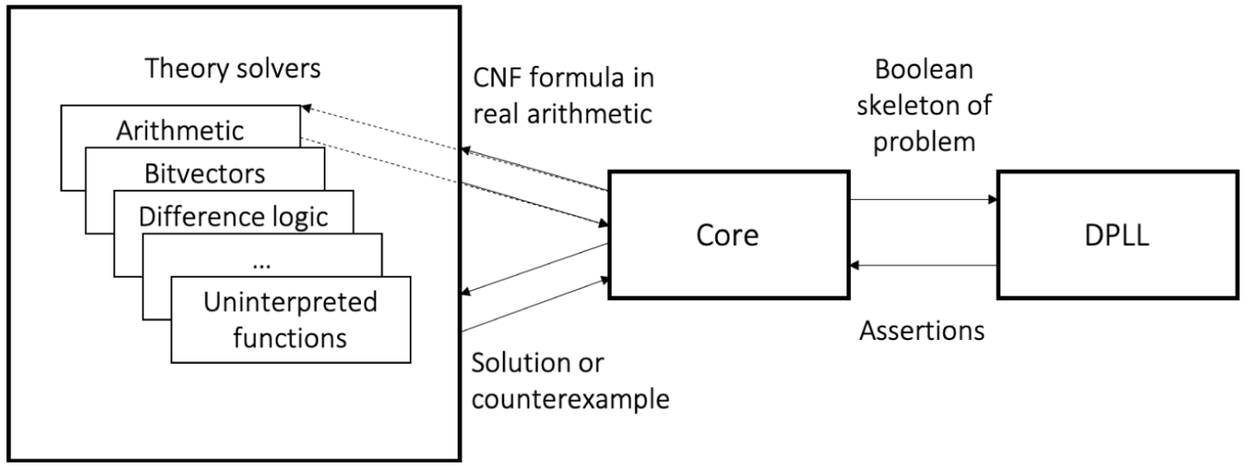
- The SAT problem decides whether $\exists\ x_1, x_2, x_3 : \phi(x_1, x_2, x_3).$
- Approaches to solving the SAT problem
  - Brute-force → check ∅ for each possible valuation $x \in val(X)$ of X ➔ requires $O(2^n)$ operations for a set of n variables
  - Davis Putnam Logeman and Loveland (DPLL) → backtracking algorithm that most solvers build on

Satisfiability modulo theory (SMT)

- SMT is a generalization of satisfiability in which some of the binary variables are replaced by predicates over nonbinary variables
- Example: $\phi_1(w, x, y, z) := (x - y = 5) \wedge (z - y \geq 2) \wedge (z - x > 2) \wedge (w - x = 2)$

  $\phi_2(x, y, z) := (3x^2 - 4y + 5z \leq 5) \wedge (-2x + 5z^3 \leq 7).$
- $\emptyset_1$ is an example of a predicate in **difference logic (DL),** and $\emptyset_2$ is a predicate in **real arithmetic**
- The satisfaction of predicates is defined in terms of the appropriate theories
- For example:
  - Linear inequalities over real variables are evaluated using the rules of the theory of linear real arithmetic
  - Predicates involving uninterpreted function symbols are evaluated using the rules of the theory of uninterpreted functions
  - These theories are called background theories for SMT
- An SMT solver uses theory solvers together with DPLL

Combine theory solvers with DPLL



Lazy approach to combine theory solvers with DPLL

1. Construct a Boolean abstraction $abs(\phi)$ of the given formula ϕ.
2. Use a SAT solver to find a satisfying solution x for $abs(\phi)$; if the SAT solver returns "unsatisfiable," then conclude that ϕ is unsatisfiable; otherwise, go to step 3.
3. Use a theory solver to check the satisfiability of $abs^{-1}(x)$ obtained from step 2.
4. If the theory solver returns "unsatisfiable," then a new constraint is added to $abs(\phi)$, and steps 1–4 are repeated.

Finding and learning invariants

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

- The problem of finding adequately strong invariants for hybrid automata is a challenging one
- **Approach 1: Invariant generation with templates:**
  - Basic idea is to choose a template form of the invariants $I(p)$ and then solve constraints imposed by Theorem 7.1 to find the unknown coefficients p
  - One example can be using the template shape of boxes or hyperrectangles defined by parameters $l, u \in R^n$

$$Box(l,u) = \{v \in val(V) || l_i \leq v_i \leq u_i\}$$

  - To find an instance of the template Box, we have to solve a constraint satisfaction problem of the following form

$$\exists l, u, \forall v: v \in Box(l,u) \Longrightarrow Post_A(v) \in Box(l,u)$$

- **Approach 2: Learning invariants by using execution data**
  - (Si et al, 2018) used RL approach to find invariant from execution data
  - (Kozarev et al, 2016) show how SVM and RL can be used to classify whether states satisfy an invariant based on execution data

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Thank You!!!

Q&A