Carnegie Mellon University

Pegasus: A Framework for Sound Continuous Invariant Generation

Andrew Sogokon, Stefan Mitsch, **Yong Kiam Tan**,
Katherine Kosaian, and André Platzer

Carnegie Mellon University, USA

Showcase Track, ICSE 2023, Melbourne, Australia

17 May 2023

Outline

Introduction: Formal Verification for CPS

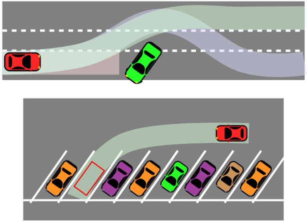
Background: Continuous Invariants and Checking

Pegasus: A Framework for Sound Continuous Invariant Generation

Conclusion

Introduction

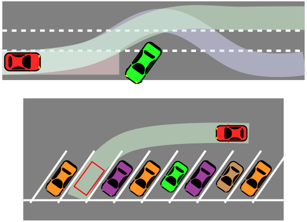
What this talk is about

Formal verification for **cyber-physical systems (CPS)**

Introduction

What this talk is about

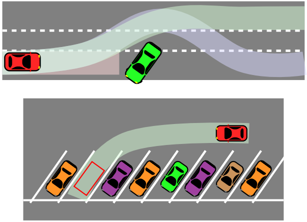


Formal verification for **cyber-physical systems (CPS)**

Discrete computational controllers interacting with **continuous** real-world physics, modeled as **hybrid systems**

Introduction

What this talk is about



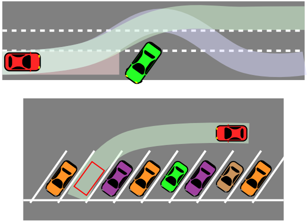
Formal verification for **cyber-physical systems (CPS)**

Mathematics-based
tools & techniques providing
rigorous guarantees
for computer systems

Discrete computational controllers
interacting with
continuous real-world physics,
modeled as **hybrid systems**

Introduction

What this talk is about



Formal verification for **cyber-physical systems (CPS)**

Mathematics-based
tools & techniques providing
rigorous guarantees
for computer systems

Discrete computational controllers
interacting with
continuous real-world physics,
modeled as **hybrid systems**

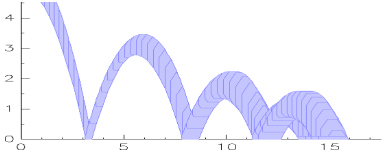
Why? Need rigorous & exhaustive proofs for these safety-critical systems

Alongside nonexhaustive simulation, field testing, and other validation methods

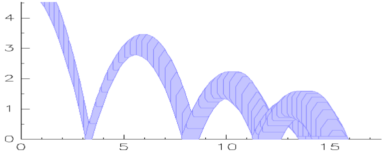
Introduction

CPS Verification Tools

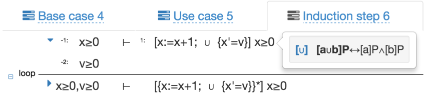
Reachability analysis:
Automatic, but overapproximate and limited to bounded space and time



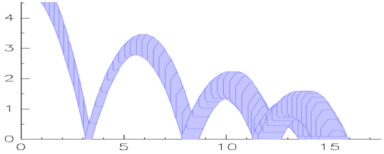
Reachability analysis:
Automatic, but overapproximate and limited to bounded space and time



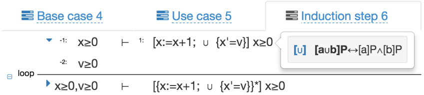
Deductive theorem proving:
More general specifications & proof techniques, but offers less automation



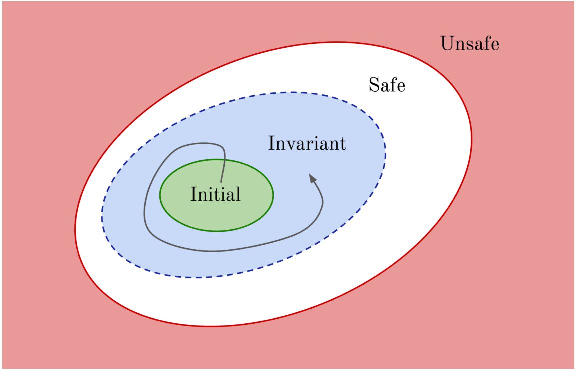
Reachability analysis:
Automatic, but overapproximate and limited to bounded space and time



Deductive theorem proving:
More general specifications & proof techniques, but offers less automation



Continuous invariants:
Set of states that can never be left when following the continuous dynamics



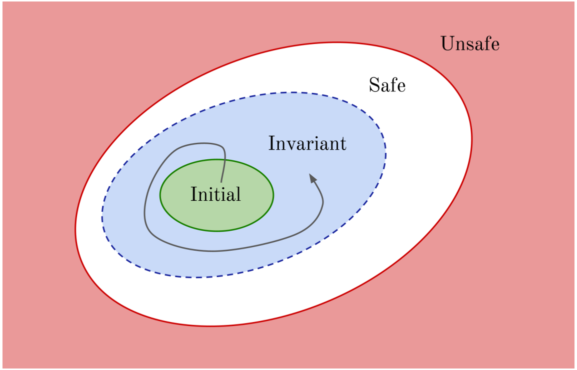
Reachability analysis:
Automatic, but overapproximate and limited to bounded space and time

Invariants help tighten overapproximations and (sometimes) extend analyses to infinite time horizons

Deductive theorem proving:
More general specifications & proof techniques, but offers less automation

Invariants are key ingredients in ODE safety proofs and part of hybrid system loop invariants

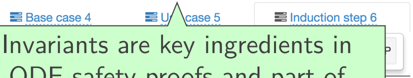
Continuous invariants:
Set of states that can never be left when following the continuous dynamics



Introduction

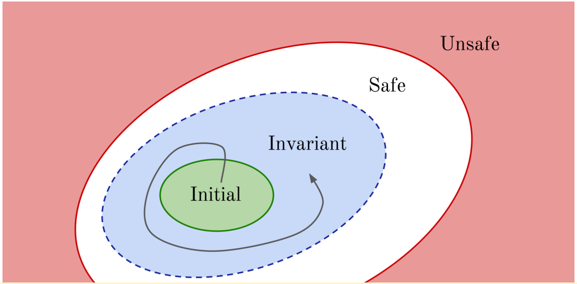
CPS Verification Tools

Deductive theorem proving:
More general specifications & proof techniques, but offers less automation



Invariants are key ingredients in ODE safety proofs and part of hybrid system loop invariants

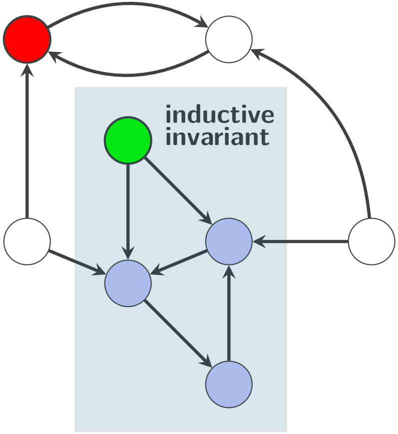
Continuous invariants:
Set of states that can never be left when following the continuous dynamics



This work: *Pegasus* continuous invariant generator & sound integration with KeYmaera X, a hybrid systems theorem prover

Outline

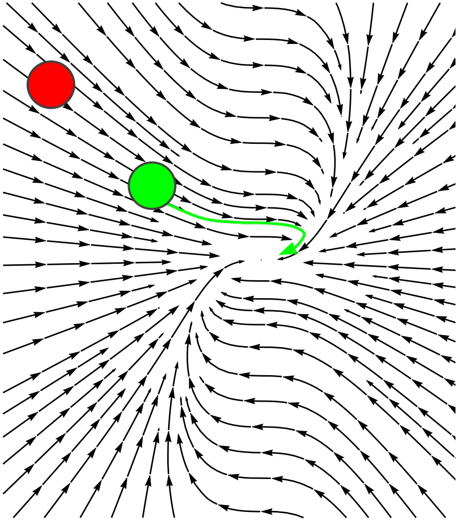
Invariants in Discrete Systems



Claim: System cannot reach bad state(s) from the initial state(s).
Justification: The invariant is closed under the system's discrete transitions.

Idea: Proving safety for system reduces to finding a suitable invariant

Invariants in Continuous Systems

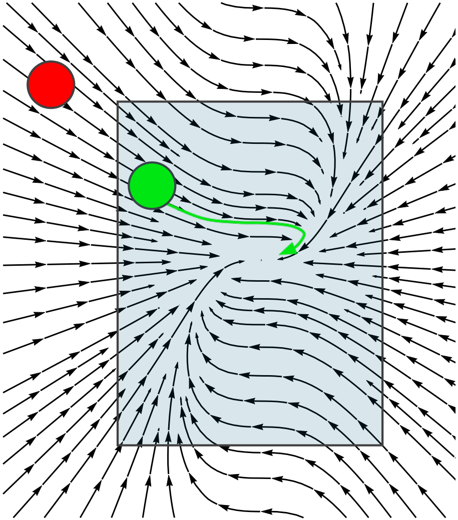


Ordinary Differential Equations (ODEs)

States: $\vec{x} \in \mathbb{R}^n$
Dynamics: $\vec{x}' = f(\vec{x})$

Trajectories of the system (in green) are solutions of the ODE from an initial state

Invariants in Continuous Systems



Ordinary Differential Equations (ODEs)

States: $\vec{x} \in \mathbb{R}^n$
Dynamics: $\vec{x}' = f(\vec{x})$

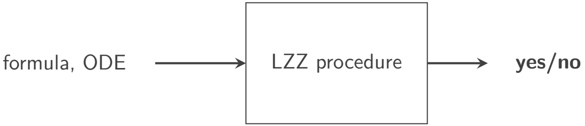
Trajectories of the system (in green) are solutions of the ODE from an initial state

Claim: System cannot reach bad state(s) from the initial state(s).
Justification: The invariant is closed under the system's **continuous** dynamics

Idea: Proving safety for **continuous** systems reduces to finding suitable **continuous** invariants

Checking Continuous Invariants

Checking whether a (polynomial arithmetic) formula defines a continuous invariant is **decidable** (Liu, Zhan & Zhao, EMSOFT 2011).



Checking Continuous Invariants

Checking whether a (polynomial arithmetic) formula defines a continuous invariant is **decidable** (Liu, Zhan & Zhao, EMSOFT 2011).

formula, ODE $\longrightarrow$ LZZ procedure $\longrightarrow$ **yes/no**

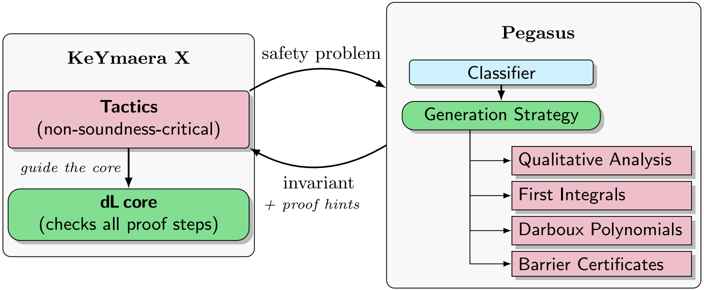
Differential dynamic logic dL is **sound and complete** for proving continuous invariance (Platzer & Tan, LICS 2018).

formula, ODE $\longrightarrow$ dL prover (**KeYmaera X**) $\longrightarrow$ **formal proof (or disproof) of invariance**

Outline

Pegasus Overview

This work: Sound and automated continuous invariant generation with *Pegasus*



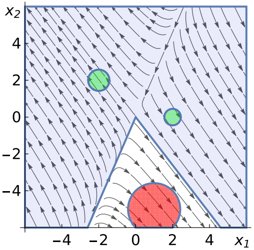
Pegasus (implemented in Wolfram Language) has:

- a simple continuous safety verification problem **classifier**
- invariant generation **primitive** methods and **strategies** for combining them
- **proof hints** for sound invariant checking in KeYmaera X

Primitives

Qualitative analysis

Idea: Perform discrete abstraction using heuristics & other sources in the input problem (using LZZ to test the transition relation)



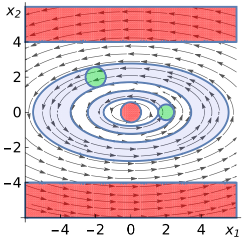
Some sources of predicates:

- right-hand sides of ODEs, their factors, etc.
- functions defining the pre/postcondition and domains
- physically meaningful quantities (e.g. divergence of the vector field)

Primitives

First integrals and Darboux polynomials

Idea: Find **conserved quantities** of the continuous system
Functions p such that $p' = 0$ (i.e. the rate of change of p w.r.t. the ODEs is 0)



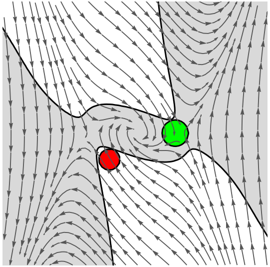
Techniques:

- **Polynomial** first integrals (of bounded degree) can be generated using linear algebra
- **Darboux polynomials** $p' = \alpha p$, for polynomial α (gen. with computer algebra techniques)
- **Rational functions** $p = \frac{q}{r}$, for polynomials q, r (combine Darboux poly. and lin. algebra)

Primitives

Barrier certificates

Idea: find a continuous invariant $p \leq 0$ numerically (Prajna and Jadbabaie, HSCC 2004)
Generalizes to **vector barrier certificates** (our work, FM 2018)



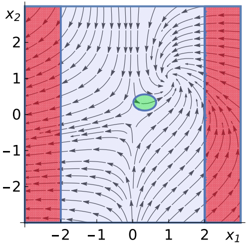
Techniques:

- **differential inequalities**, e.g. $p' \leq 0$, $p' \leq \lambda p$ ($\lambda \in \mathbb{R}$), and
- **sum-of-squares decomposition** (via semidefinite programming)
- **linear programming relaxations**

Strategies

Differential saturation

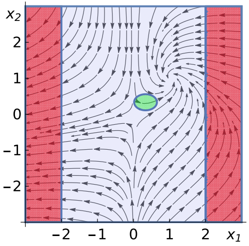
Idea: Iteratively refine the candidate invariant by cycling through primitive methods until saturation or a suitable invariant is found

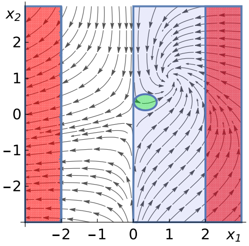


Strategies

Differential saturation

Idea: Iteratively refine the candidate invariant by cycling through primitive methods until saturation or a suitable invariant is found



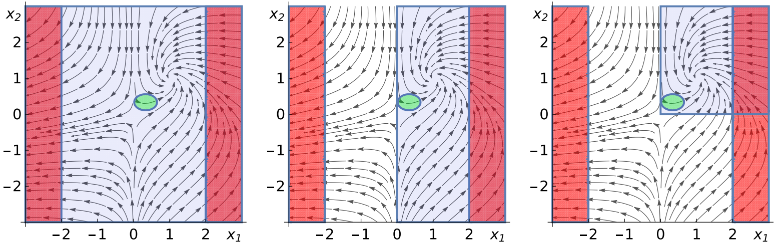


- Refinement 1 (using a Darboux polynomial: $x_1 > 0$)

Strategies

Differential saturation

Idea: Iteratively refine the candidate invariant by cycling through primitive methods until saturation or a suitable invariant is found

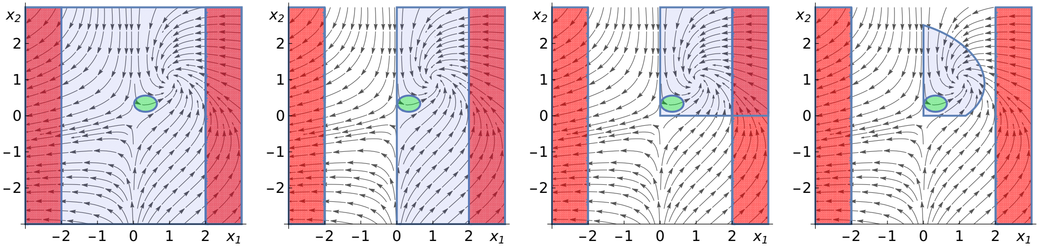


- Refinement 1 (using a Darboux polynomial: $x_1 > 0$)
- Refinement 2 (using qualitative analysis $x_1 > 0 \wedge x_2 > 0$)

Strategies

Differential saturation

Idea: Iteratively refine the candidate invariant by cycling through primitive methods until saturation or a suitable invariant is found

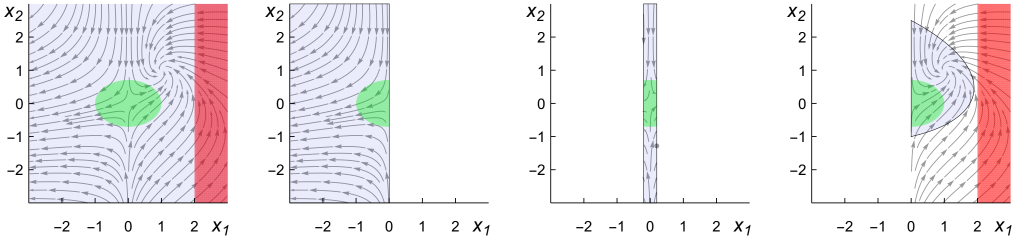


- Refinement 1 (using a Darboux polynomial: $x_1 > 0$)
- Refinement 2 (using qualitative analysis $x_1 > 0 \wedge x_2 > 0$)
- Refinement 3 (using a barrier certificate $x_1 > 0 \wedge x_2 > 0 \wedge p \leq 0$)
  $$p = \frac{3}{8}x_1 + \frac{23}{56}x_1^2 - \frac{123}{56}x_2 + \frac{3}{14}x_1x_2 + \frac{29}{28}x_2^2 - 1$$

Strategies

Differential divide-and-conquer

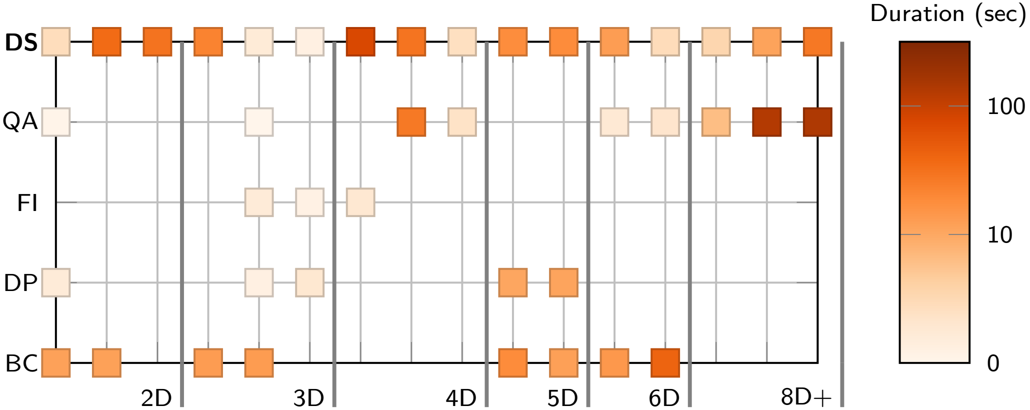
Idea: Divide state space using an equational invariant $p = 0$, and find suitable invariants for each partition recursively



- State space is partitioned into regions $x_1 < 0$, $x_1 = 0$, $x_1 > 0$, which have no transitions between them
- For $x_1 \leq 0$, no unsafe states, so the trivial invariant suffices
- For $x_1 > 0$, a barrier certificate separates initial from unsafe states

Benchmarks

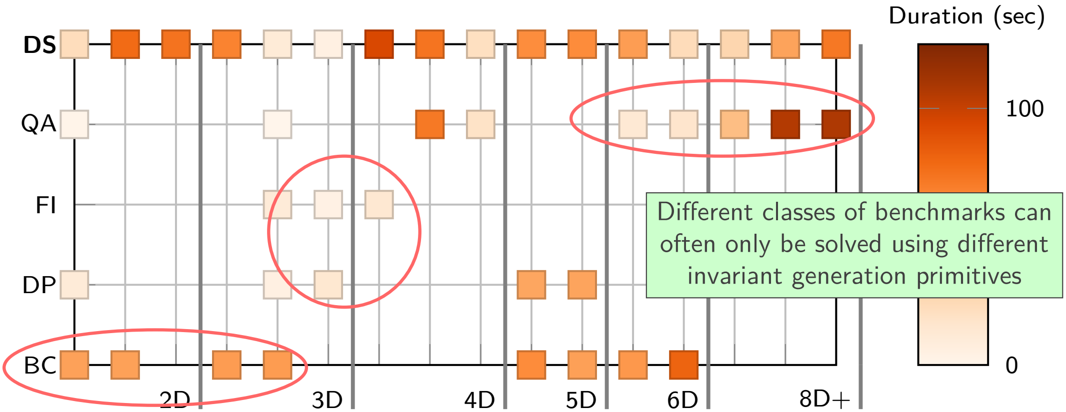
- Benchmark suite of **150** continuous safety verification problems drawn from the literature
- Variety of benchmarks: 2–16 dim, (non-)linear ODEs, syntactic complexity, topology, etc.



Selected benchmark problems (dimension: 2D-16D)

Benchmarks

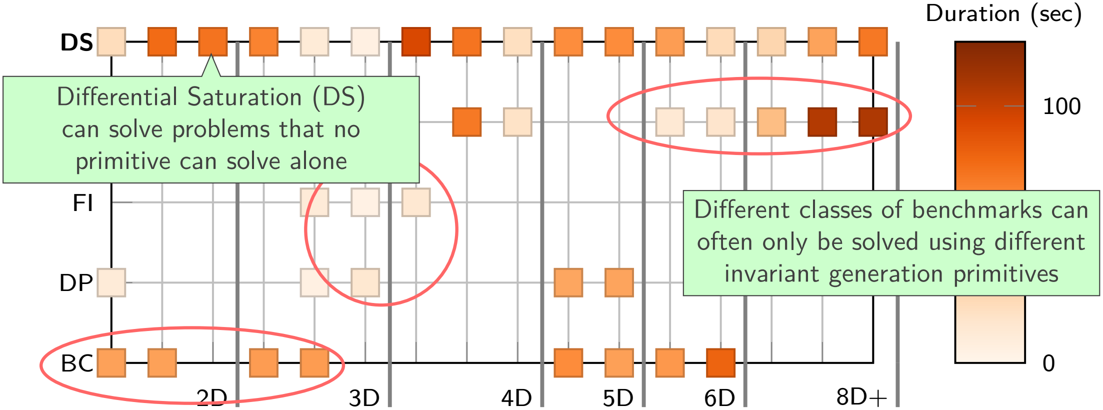
Benchmark suite of **150** continuous safety verification problems drawn from the literature
Variety of benchmarks: 2–16 dim, (non-)linear ODEs, syntactic complexity, topology, etc.



Selected benchmark problems (dimension: 2D-16D)

Benchmarks

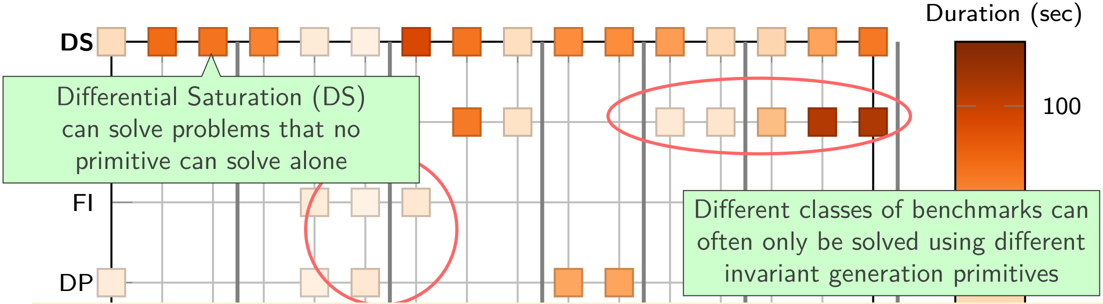
Benchmark suite of **150** continuous safety verification problems drawn from the literature
Variety of benchmarks: 2–16 dim, (non-)linear ODEs, syntactic complexity, topology, etc.



Selected benchmark problems (dimension: 2D-16D)

Benchmarks

Benchmark suite of **150** continuous safety verification problems drawn from the literature
Variety of benchmarks: 2–16 dim, (non-)linear ODEs, syntactic complexity, topology, etc.

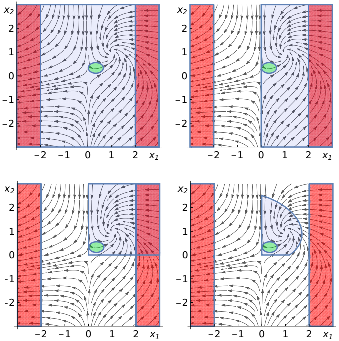


More extensive experiments & results in paper:

- Summary: 107/150 solved automatically, DS strategy is highly effective at combining invariant generation primitives
- Various configuration parameters for differential saturation
- Effect of proof hints on sound invariant checking for KeYmaera X

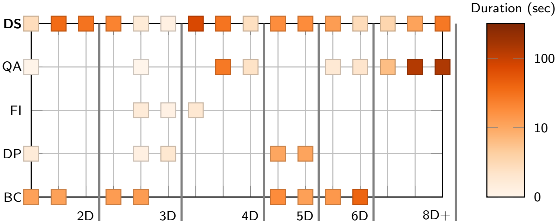
Outline

Thank you for your attention!



Pegasus is an **effective** and **sound** continuous invariant generator for hybrid systems verification

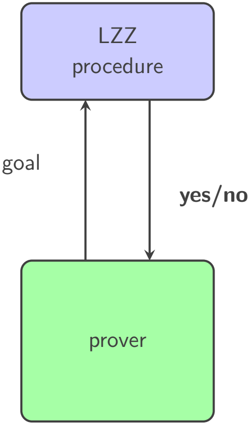
http://pegasus.keymaeraX.org



Selected benchmark problems (dimension: 2D-16D)

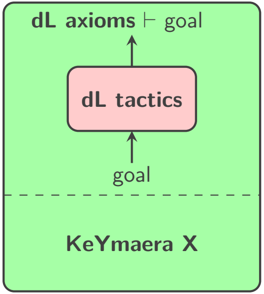
Handling Invariants

Design choices in proof assistants



Using external oracles

Less soundness-critical code



(Untrusted, but checked) proof using tactics

Primitives

Discrete abstraction

Idea: Partition $\mathbb{R}^n$ into discrete states $S_1, \ldots, S_k$ defined by some predicates & compute the discrete transition relation in the resulting abstraction