Thesis Defense

Formal Verification of Cyber-Physical Systems in the Industrial Model-Based Design Process

Nikolaos Kekatos

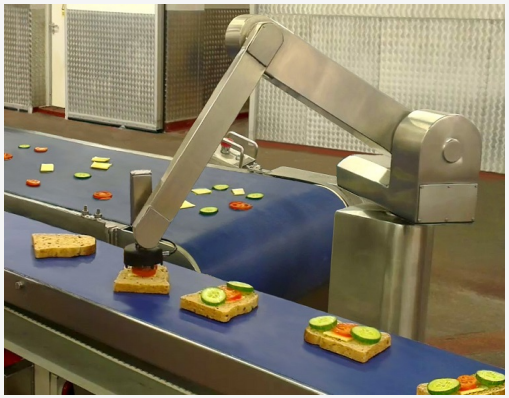
December 17, 2018
Grenoble, France

Name	Title	Institute	Role
Goran Frehse	Professor	ENSTA ParisTech	Thesis Supervisor
Thao Dang	Research Director	Univ. Grenoble Alpes	Thesis Co-Supervisor
Benoît Caillaud	Research Director	Inria Rennes - IRISA	Reviewer
Laurent Fribourg	Research Director	ENS Paris-Saclay	Reviewer
Alexandre Chapoutot	Professor	ENSTA ParisTech	Examiner

UnCoVerCPS Project

- Unifying Control and Verification of Cyber-Physical Systems - H2020
- 4-year EU project: 2015 - 2018
- 10 partners: 4 Universities and 6 Industrial Partners

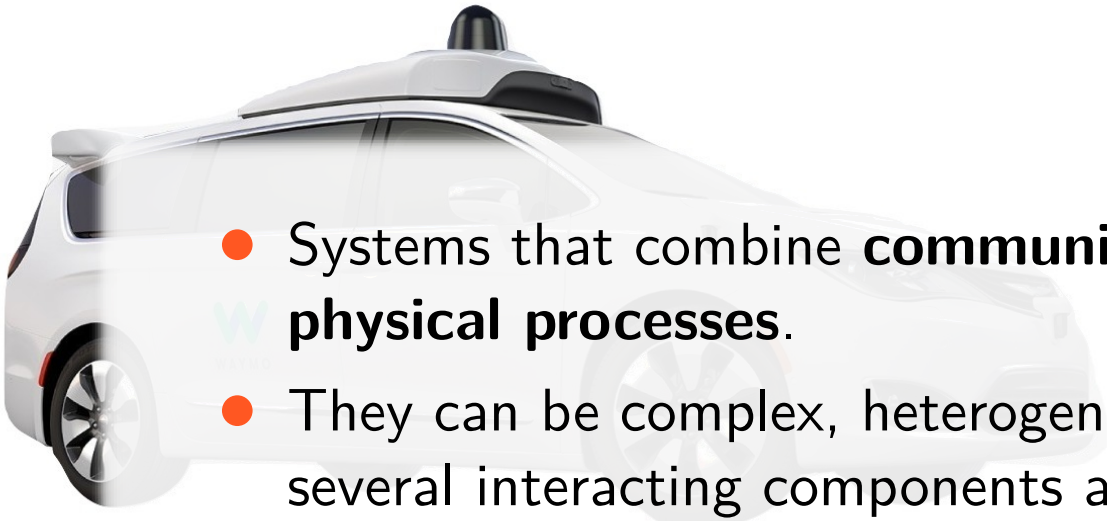
source: Technalia



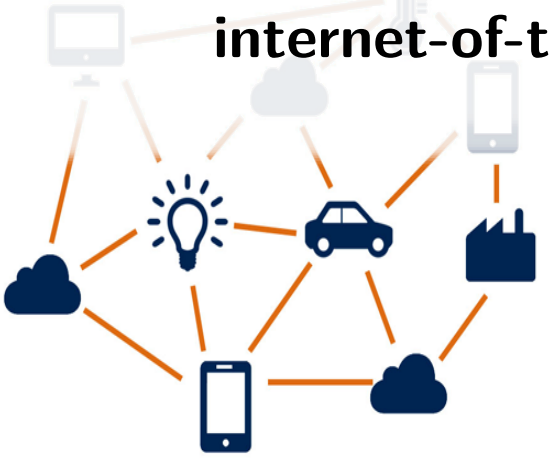
source: R.U.Robots

source: GE

Cyber-Physical Systems



- Systems that combine **communication**, **control**, and **physical processes**.
- They can be complex, heterogeneous, large-scale with several interacting components and safety critical.
- Examples include **self-driving vehicles**, **robots**, **internet-of-things**, **smart grids**, ...



Need for Safe Cyber-Physical Systems

- Cyber-physical systems are increasingly safety-critical or operation-critical!
- Formal verification can provide performance and correctness guarantees.
- Examples of insufficiently verified systems:

source: Google - TheVerge

source: Nasa

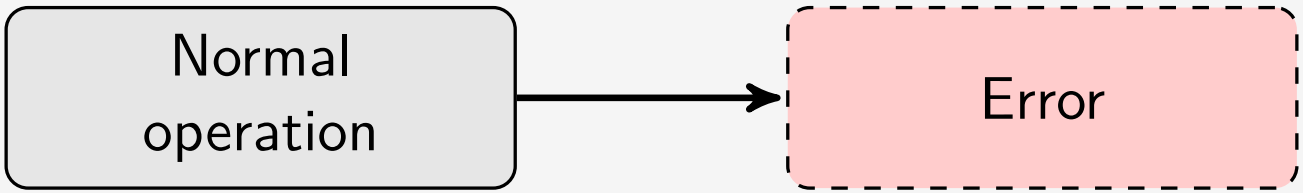
source: MediaTV.nl

What is Formal Verification?

Given a model of a system, exhaustively and automatically check whether this model meets a given specification.

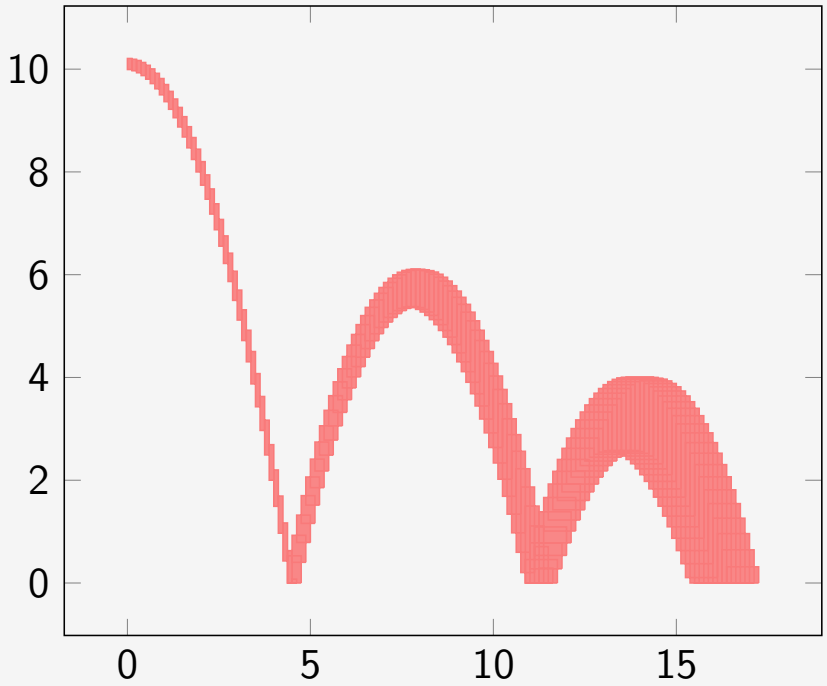
Reachability analysis is a (relatively) mature automated verification technique. It assists with

1. *Reachability Problem:* decide whether a certain state of a system is reachable from a given initial state of the system.
2. *Safety Verification:* show that an error state is not reachable.



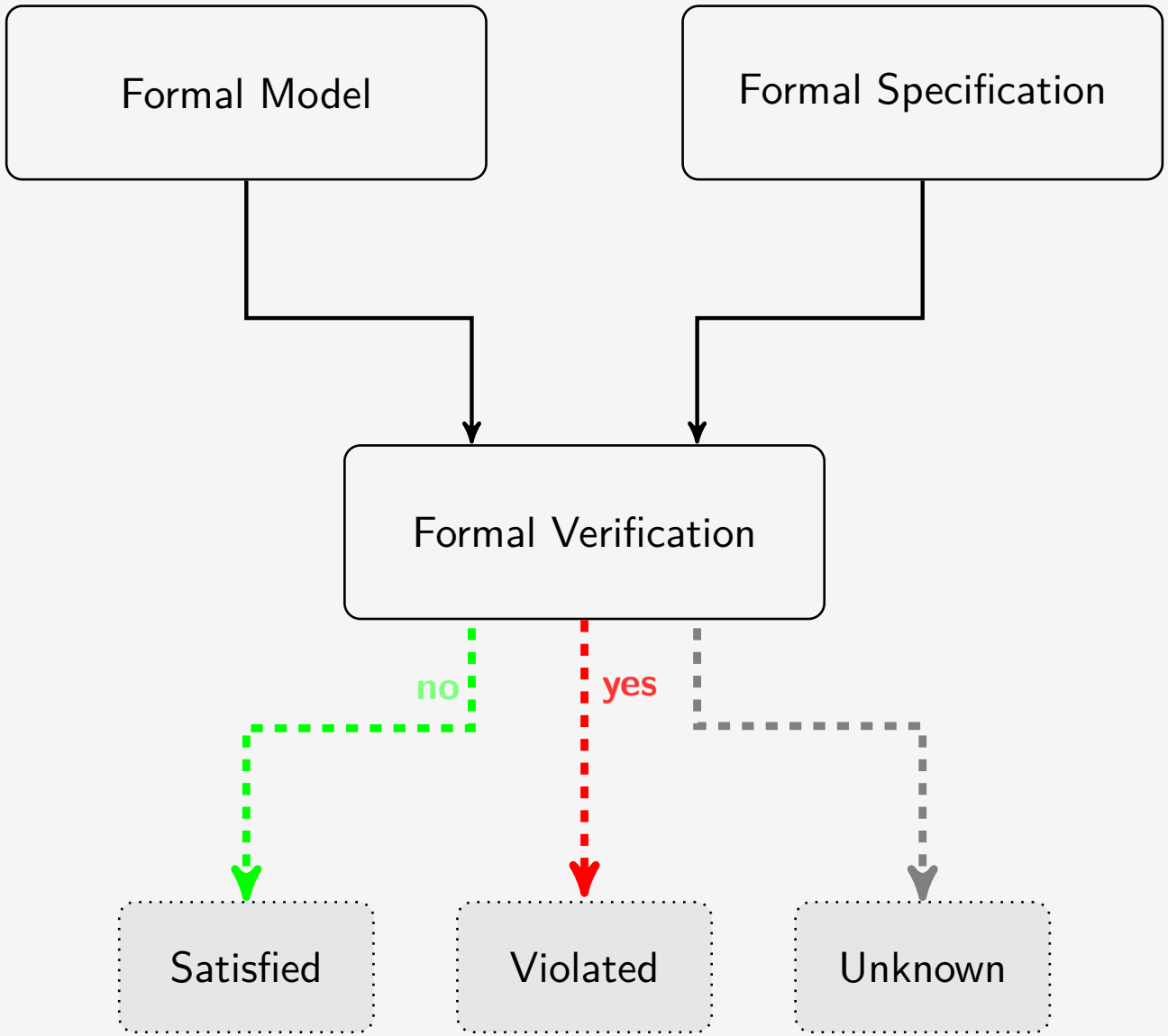
Set-based Reachability Analysis

- Instead of single points we employ sets and model nondeterministic behaviors.
- Tool support: SpaceEx, CORA, HyPRO, Flow*, JuliaReach ...

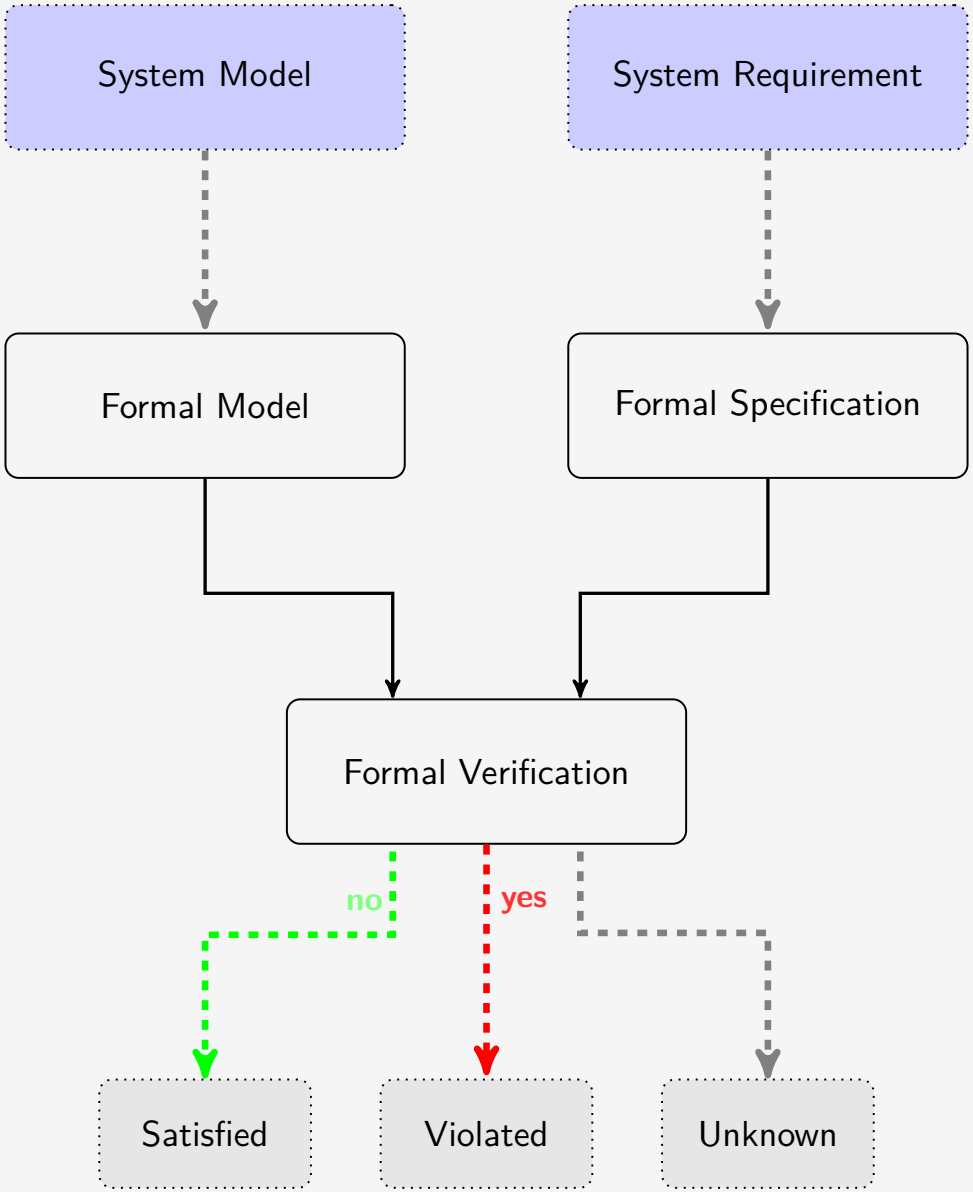


Bouncing Ball - SpaceEx

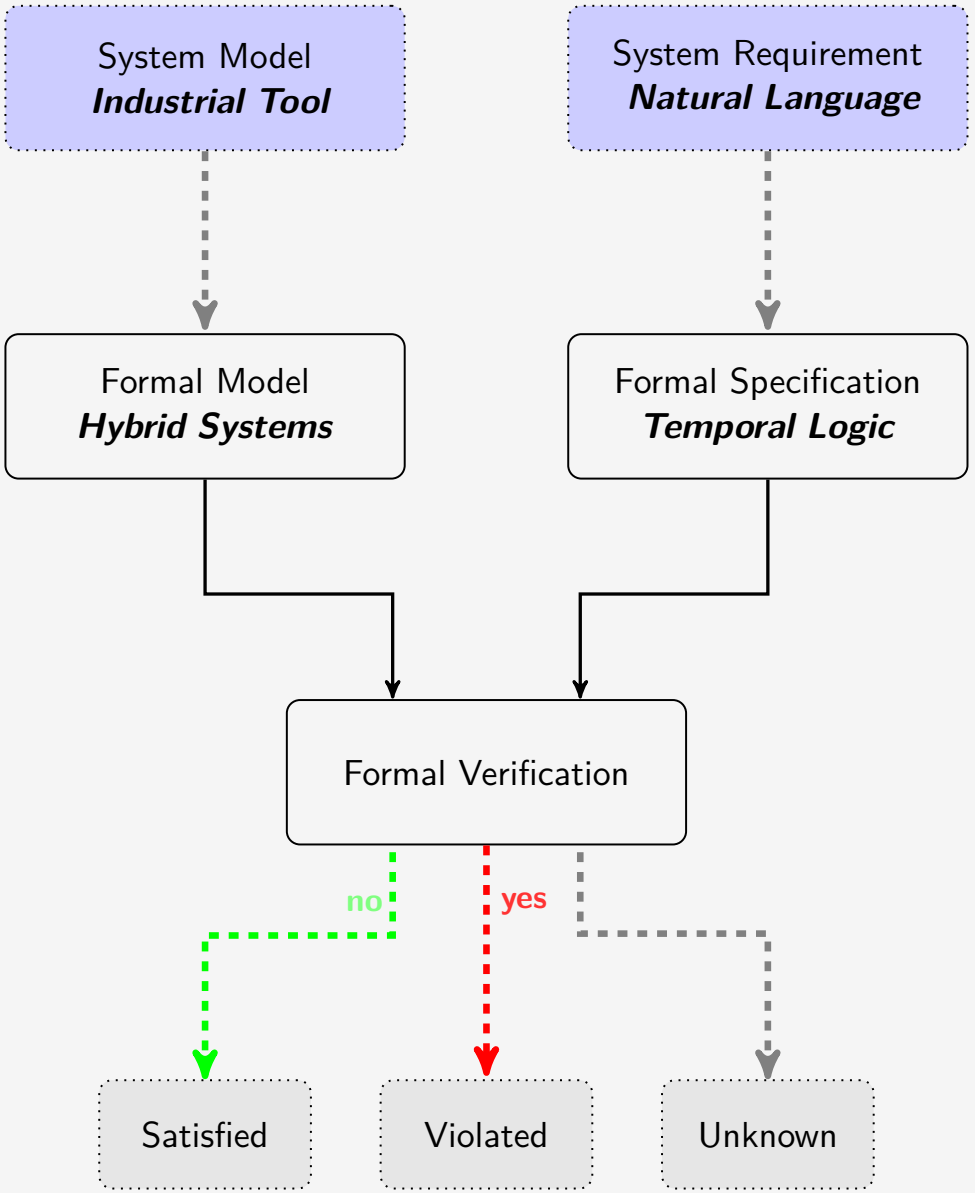
Verification Workflow



Industrial Verification Workflow



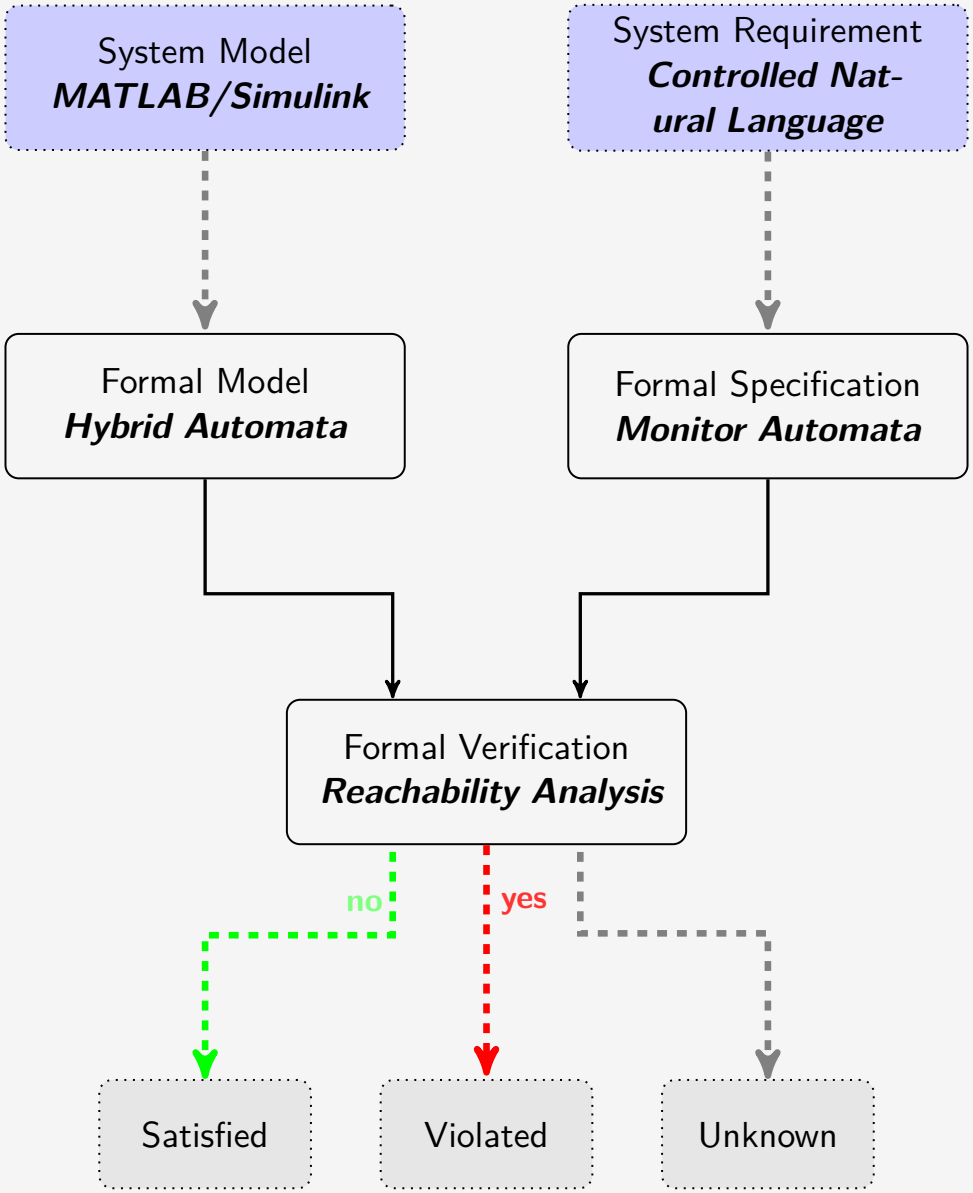
Industrial Verification Workflow - Challenges



Industrial Verification Workflow - Challenges

- Industrial models: described and analyzed in simulation environments; often unverified, unsound numerical computations.
- Industrial requirements: described in natural language $\Rightarrow$ ambiguous, translation errors.
- Formal models: described via formal languages with clearly defined semantics; often seen as complex, impractical or restricted.
- Formal requirements: described in Temporal Logic $\Rightarrow$ unintuitive language for engineers, steep learning curve.

Industrial Verification Workflow - Our Goal



Introduction

Formalizing Natural Language Requirements

- How to describe the requirements?
- How to verify the requirements?

Constructing Formal Verification Models

Toolchain

Case Studies

Conclusions

Describing Natural Language Requirements

Need to find a subset of natural language which has the following features:

- Intuitive and close to natural language.

[1]Konrad, Sascha, and Betty HC Cheng. "Real-time specification patterns." ICSE Conference. ACM, 2005.

Describing Natural Language Requirements

Need to find a subset of natural language which has the following features:

- Intuitive and close to natural language.
- Expressive enough to capture common industrial requirements.

[1] Konrad, Sascha, and Betty HC Cheng. "Real-time specification patterns." ICSE Conference. ACM, 2005.

Describing Natural Language Requirements

Need to find a subset of natural language which has the following features:

- Intuitive and close to natural language.
- Expressive enough to capture common industrial requirements.
- Understood by both experts and non-experts.

[1] Konrad, Sascha, and Betty HC Cheng. "Real-time specification patterns." ICSE Conference. ACM, 2005.

Describing Natural Language Requirements

Need to find a subset of natural language which has the following features:

- Intuitive and close to natural language.
- Expressive enough to capture common industrial requirements.
- Understood by both experts and non-experts.
- Retain formal and rigorous semantics and avoid ambiguities.

[1] Konrad, Sascha, and Betty HC Cheng. "Real-time specification patterns." ICSE Conference. ACM, 2005.

Describing Natural Language Requirements

Need to find a subset of natural language which has the following features:

- Intuitive and close to natural language.
- Expressive enough to capture common industrial requirements.
- Understood by both experts and non-experts.
- Retain formal and rigorous semantics and avoid ambiguities.

[1] Konrad, Sascha, and Betty HC Cheng. "Real-time specification patterns." ICSE Conference. ACM, 2005.

Describing Natural Language Requirements

Need to find a subset of natural language which has the following features:

- Intuitive and close to natural language.
- Expressive enough to capture common industrial requirements.
- Understood by both experts and non-experts.
- Retain formal and rigorous semantics and avoid ambiguities.

Template-based languages cover these criteria but should be extended to include timed relations.
$\Rightarrow$ **Pattern Templates**[1]

[1] Konrad, Sascha, and Betty HC Cheng. "Real-time specification patterns." ICSE Conference. ACM, 2005.

Describing Requirements with Pattern Templates

Pattern Template: natural language + placeholders for predicates

Let's look at the *Maximum Duration* pattern:

- Template: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*

Describing Requirements with Pattern Templates

Pattern Template: natural language + placeholders for predicates

Let's look at the *Maximum Duration* pattern:

- Template: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*
- Informally: a state predicate always holds for less than a specified amount of time.

Describing Requirements with Pattern Templates

Pattern Template: natural language + placeholders for predicates

Let's look at the *Maximum Duration* pattern:

- Template: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*
- Informally: a state predicate always holds for less than a specified amount of time.
- STL formula: $\square(q \rightarrow \square(p \vee (\neg p \; \mathcal{W} \; (p \wedge \Diamond_{[0,T]}(\neg p)))))$.

Describing Requirements with Pattern Templates

Pattern Template: natural language + placeholders for predicates

Let's look at the *Maximum Duration* pattern:

- Template: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*
- Informally: a state predicate always holds for less than a specified amount of time.
- STL formula: $\square(q \rightarrow \square(p \vee (\neg p \; \mathcal{W} \; (p \wedge \Diamond_{[0,T]}(\neg p)))))$.
- Industrial example: "The maximal waiting time for an ACK message should be 20 ms."

Describing Requirements with Pattern Templates

Pattern Template: natural language + placeholders for predicates

Let's look at the *Maximum Duration* pattern:

- Template: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*
- Informally: a state predicate always holds for less than a specified amount of time.
- STL formula: $\square(q \rightarrow \square(p \vee (\neg p \; \mathcal{W} \; (p \wedge \Diamond_{[0,T]}(\neg p)))))$.
- Industrial example: "The maximal waiting time for an ACK message should be 20 ms."

Describing Requirements with Pattern Templates

Pattern Template: natural language + placeholders for predicates

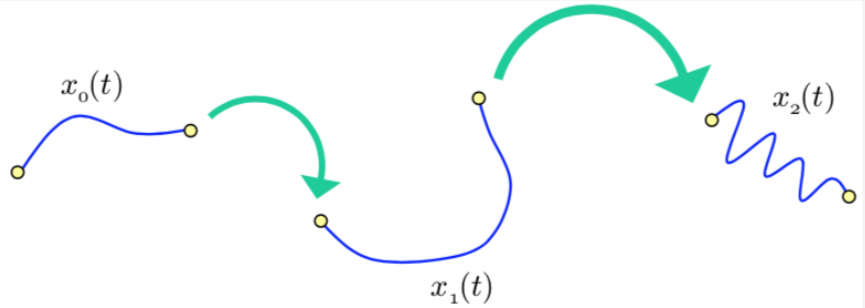
Let's look at the *Maximum Duration* pattern:

- Template: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*
- Informally: a state predicate always holds for less than a specified amount of time.
- STL formula: $\square(q \rightarrow \square(p \vee (\neg p \; \mathcal{W} \; (p \wedge \Diamond_{[0,T]}(\neg p)))))$.
- Industrial example: "The maximal waiting time for an ACK message should be 20 ms."

Long list of pattern templates available but not defined for hybrid systems.

Formalizing Pattern Templates for Hybrid Automata

Pattern templates should be defined in a formalism suitable for hybrid automata $\rightarrow$ define based on the *runs* of the hybrid automaton.

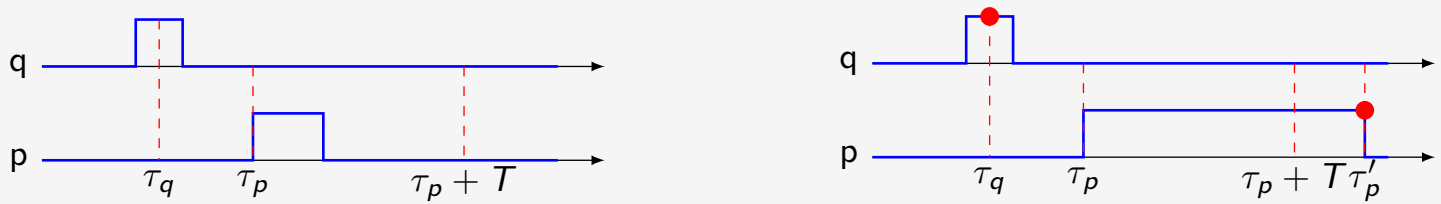


- A run *r*is given by locations, continuous states, trajectories, and durations.
- We use *event-times* $\tau = (i, t)$ to capture the timing of the states and uniquely identify discrete and continuous states on the run.

Formalizing Pattern Templates for Hybrid Automata

Maximum duration: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*
$r \models \phi$ iff p following q implies that for all $\tau_p, \tau_p' \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$ one of the following holds:

1. $d(\tau_p, \tau_p') < T$ (τ_p' is early enough, covering the $\tau_p = \tau_p'$ case), or
2. there is a $\tau_{\bar{p}}$ such that $\tau_p < \tau_{\bar{p}} < \tau_p'$ (p is false in between).

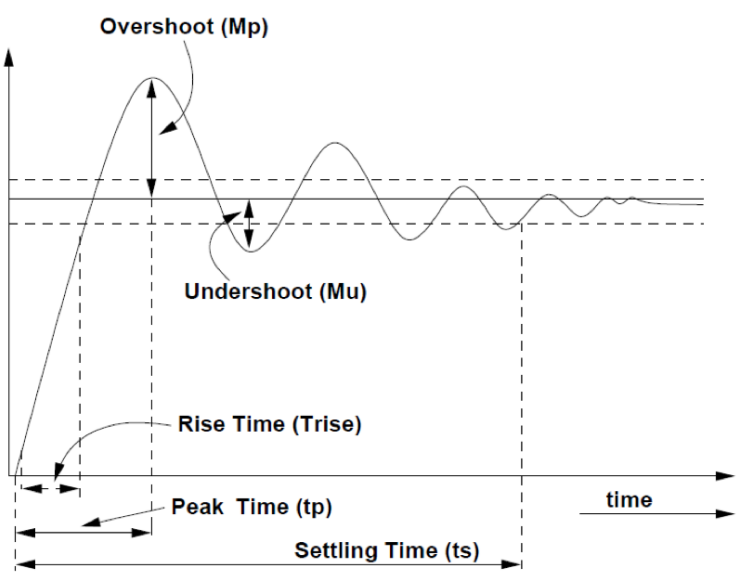


Satisfied (left), violated (right).

There are 13 more pattern definitions.

Application to control specifications

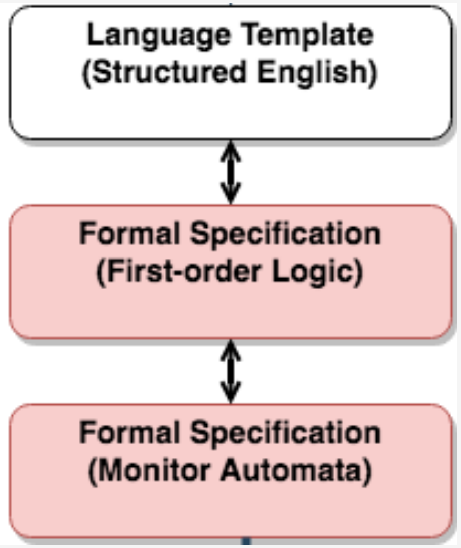
Control specifications can be expressed with the formalized pattern templates.



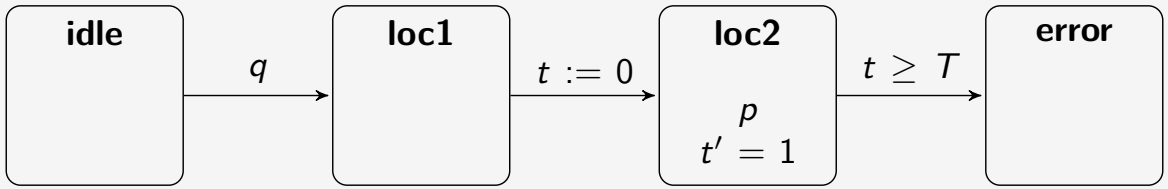
Rise Time: *The state x of the system should reach 90% of the reference x_{ref} at time T_{rise}.*
bounded response pattern: where $p :=$true, $T := T_{rise}$, and $s := \{x >= 0.9 * x_{ref}\}$.

From Formalized Patterns to Monitor Automata

- The formalized patterns need to be transformed in a representation that can be used in verification tools ⇒ *monitor automata*.
- Monitors can be represented in graphical and machine readable format.
- Originally used for discrete systems[2]



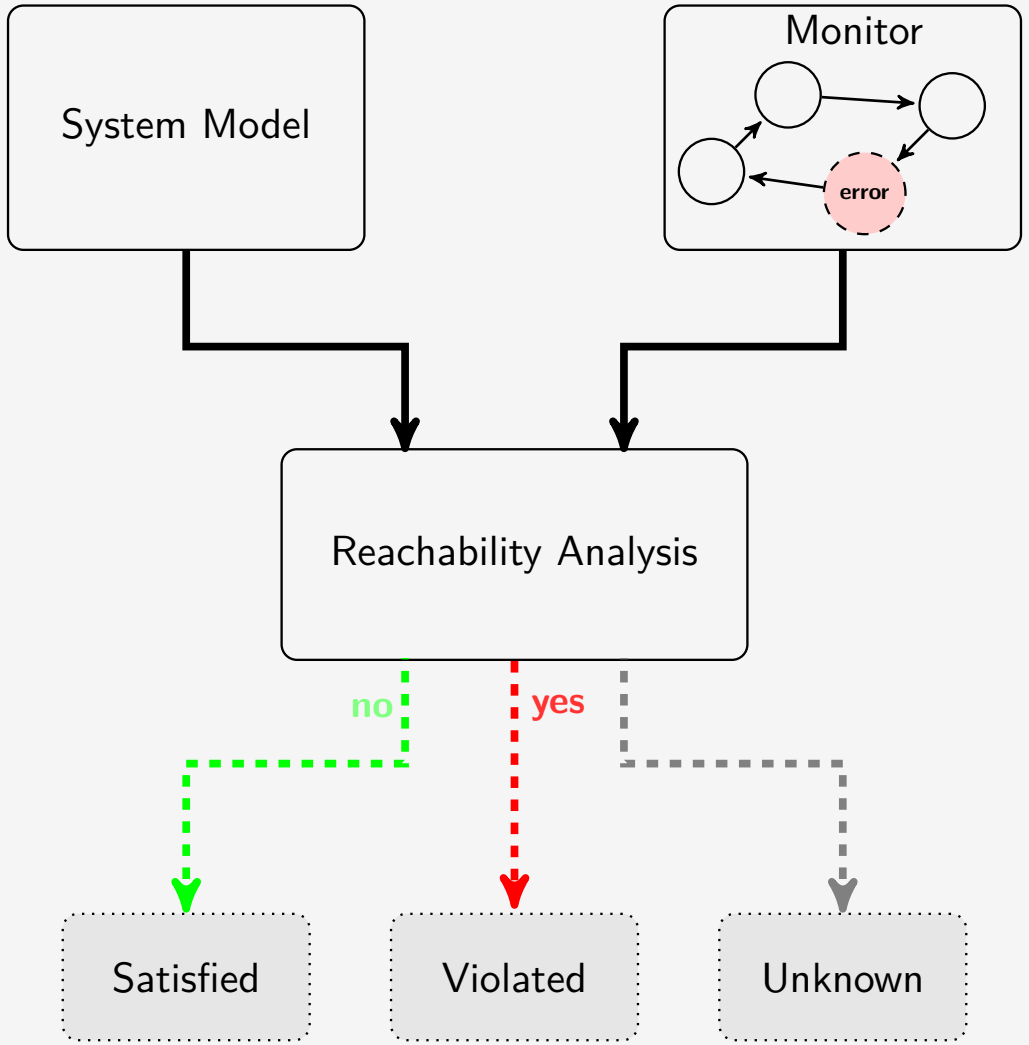
Maximum duration: *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*



[2] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In AMAST Conference, 1993.

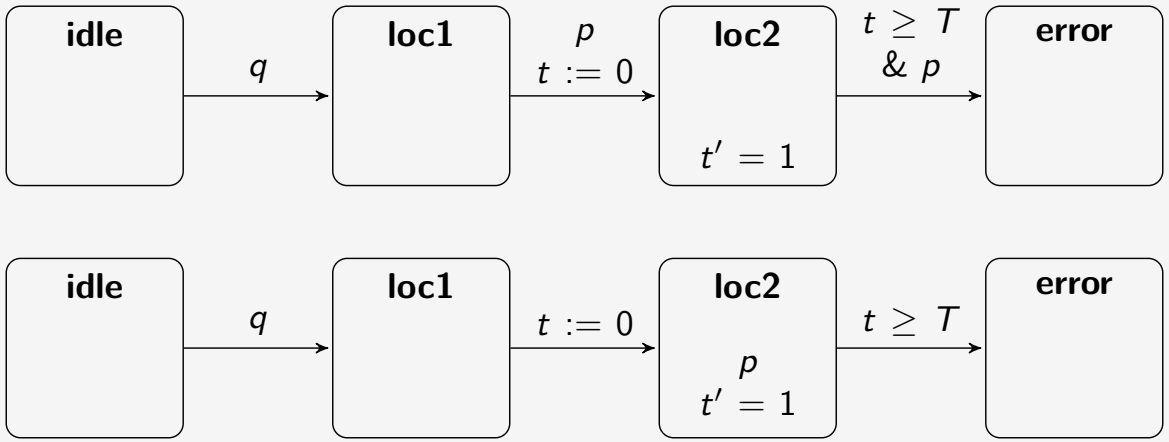
Formal Verification with Monitor Automata

These monitor automata, once composed with the system under test, encode the requirements as reachability properties.



Proving Formal Correctness of Monitor Automata

A monitor automaton is correct if its error location is reachable exactly when the system H violates the property. We prove correctness of M by showing that every violating run of H has a corresponding run in $H||M$ that reaches the error location, and vice versa. Designing correct monitors is not straightforward!
Two seemingly equivalent monitors for the maximum duration pattern

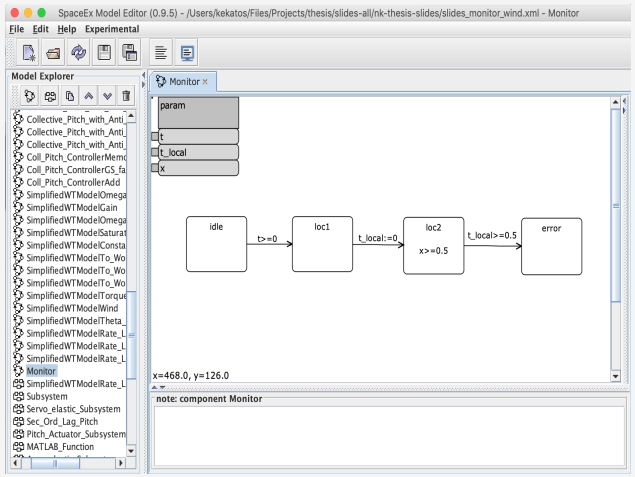


Incorrect monitor (above) vs. Correct (below)

Tool Support for Monitor Automata

- formalSpec[3] tool assists with the instantiation of monitor automata and the reduction of manual, translation errors.
- Inputs: templates and user-defined predicates
- Output: instantiated monitor in SpaceEx

- After t>=0, it is always the case that once x>=0.5 becomes satisfied it holds for less than 0.5 time units.
- Globally, it is always the case that y<ymax holds.
- ...



[3] Axel Busboom, Simone Schuler, and Alexander Walsch. formalspec - semi-automatic formalization of system requirements for formal verification. In ARCH, 2016.

Introduction

Formalizing Natural Language Requirements

Constructing Formal Verification Models

- Addressing scalability issues.
- Addressing semantic issues.
- Addressing syntactic, modeling and other issues.

Toolchain

Case Studies

Conclusions

From simulation models to formal models

There are three main challenges:

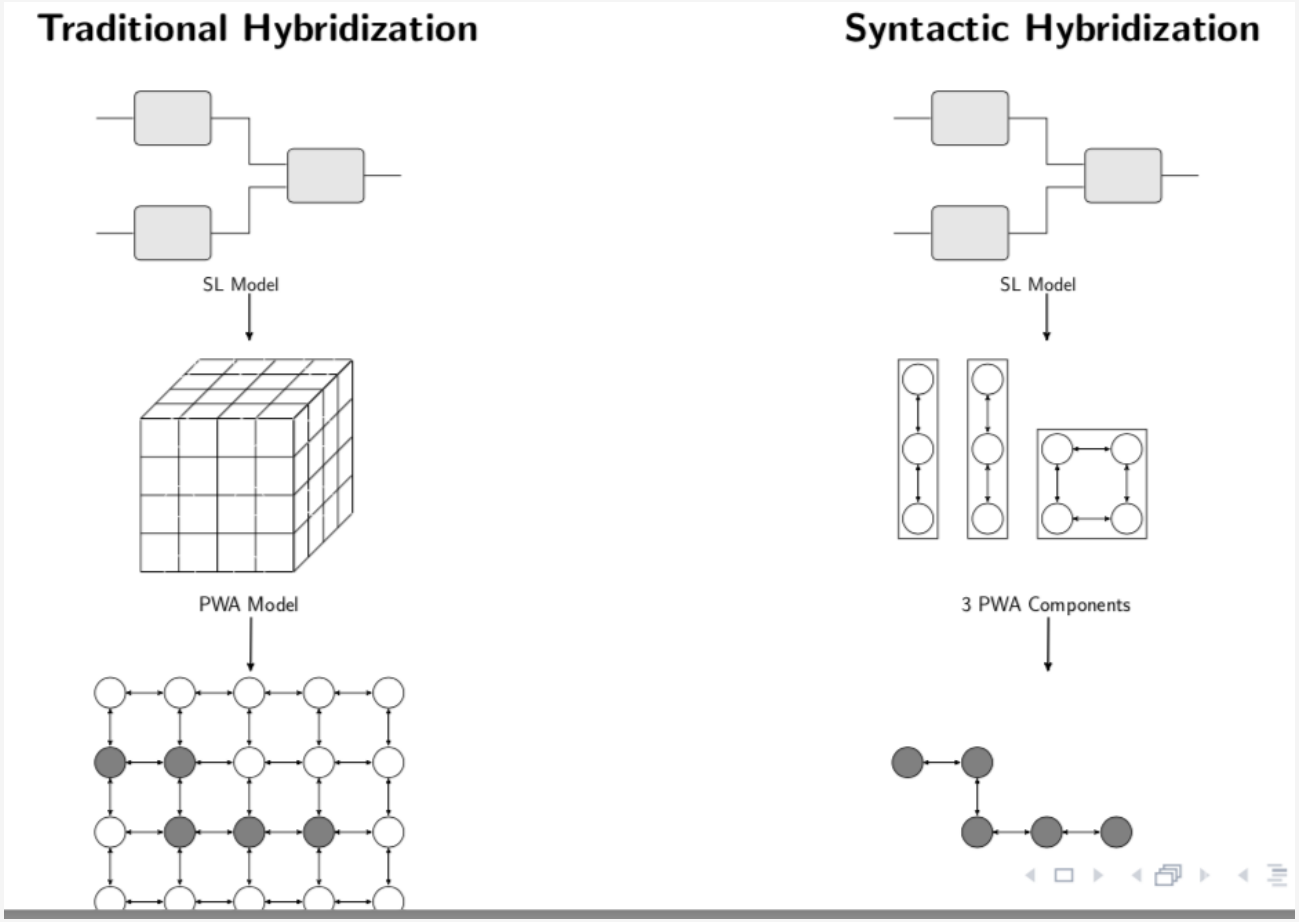
- scalability $\rightarrow$ formal model amenable to existing reachability algorithms,
- semantics $\rightarrow$ *may* vs *must (urgent)* semantics
- syntax $\rightarrow$ conversion to suitable formalism.

Transformation Challenge no. 1 - Scalability

Simulation models can be high-dimensional and nonlinear. Formal verification via reachability analysis scales better for linear dynamics.

- Need to approximate nonlinear dynamics by linear or piecewise linear ones.
- Hybridization is an abstraction method to obtain PWA over-approximations of nonlinear dynamics.
- The state-space is partitioned into domains and for each domain the dynamics are approximated by simple ones plus the errors.

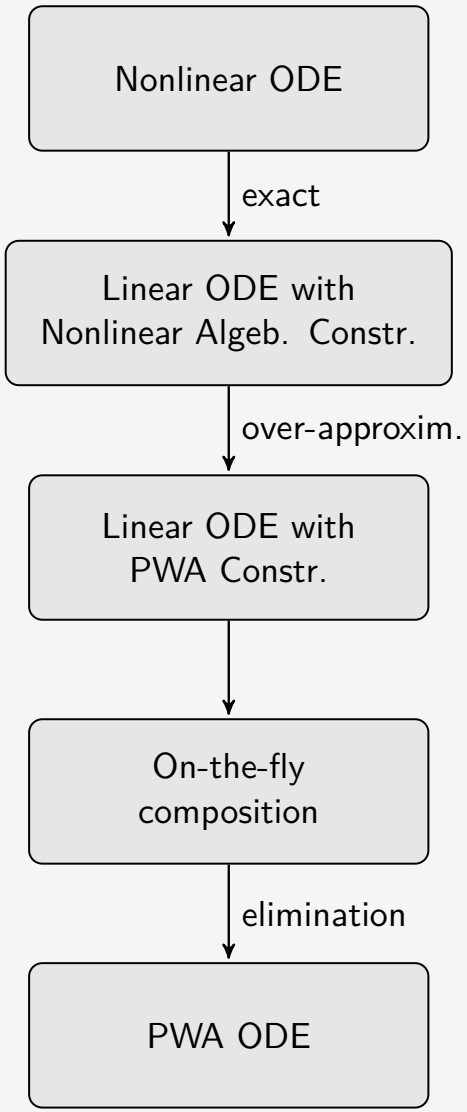
Traditional vs Proposed Hybridization Method



Compositional Syntactic Hybridization - Scheme

The proposed scheme includes 4 steps:

- Syntactic Decomposition
  - Construct a new model where nonlinear terms are replaced by auxiliary variables.
- PWA Approximation
  - State space partitioning (estimation of the signal ranges leads to approximation with fewer locations).
  - For each domain, use Taylor's formula and bound approximation errors through the Lagrange Remainder.
- Construct SpaceEx Model
  - Each hybrid automaton corresponds to the PWA approximation of one nonlinearity.
- Composition
  - SpaceEx builds the product-automaton and eliminates the algebraic equations.



Compositional Syntactic Hybridization - Example

Example 1

Consider the 9-dim. polynomial vector field

$$f = \begin{bmatrix} 3x_3 - x_1 \cdot x_6 \\ x_4 - x_2 \cdot x_6 \\ x_1 \cdot x_6 - 3x_3 \\ x_2 \cdot x_6 - x_4 \\ 3x_3 + 5x_1 - x_5 \\ 5x_5 + 3x_3 + x_4 - x_6 \cdot (x_1 + x_2 + 2x_8 + 1) \\ 5x_4 + x_2 - 0.5x_7 \\ 5x_7 - 2x_6 \cdot x_8 + x_9 - 0.2x_8 \\ 2x_6 \cdot x_8 - x_9 \end{bmatrix}$$

.

Instead of partitioning a 9-dimensional space, we only need to partition each element of h which has dimension 2.

Compositional Syntactic Hybridization - Example

Example 1

Introducing the auxiliary variables $y_1 = x_1 x_6$, $y_2 = x_2 x_6$ and $y_3 = x_6 x_8$, then $g(x, y)$ is a linear ODE and $h(x, y) = [x_1 \cdot x_6, x_2 \cdot x_6, x_6 \cdot x_8]$ is a nonlinear algebraic equation.

$$g = \begin{bmatrix} 3x_3 - \mathbf{y_1} \\ x_4 - \mathbf{y_2} \\ \mathbf{y_1} - 3x_3 \\ \mathbf{y_2} - x_4 \\ 3x_3 + 5x_1 - x_5 \\ 5x_5 + 3x_3 + x_4 - \mathbf{y_1} - \mathbf{y_2} - 2\mathbf{y_3} - x_6 \\ 5x_4 + x_2 - 0.5x_7 \\ 5x_7 - 2\mathbf{y_3} + x_9 - 0.2x_8 \\ 2\mathbf{y_3} - x_9 \end{bmatrix}$$

Instead of partitioning a 9-dimensional space, we only need to partition each element of h which has dimension 2.

Transformation Challenge no. 2 - Semantics

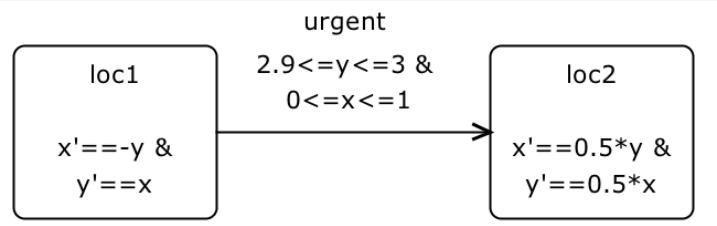
Simulation models are *deterministic* and respect *may* semantics.

Formal models are *non-deterministic* and respect *must (urgent)* semantics.

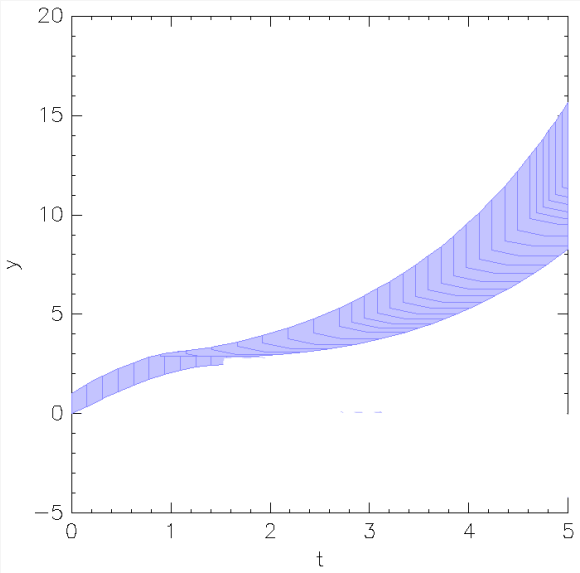
Problematic when working with sets instead of points!

Must vs. May Semantics

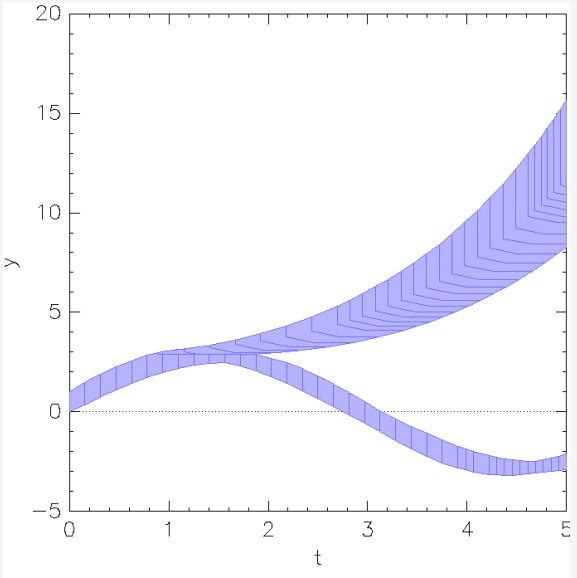
Let's look at a hybrid automaton model with 2 locations and 1 urgent guard.



Correct (intended) behavior



Obtained behavior - Reach. Tool



Handling Must Semantics

Idea: Model with urgent transitions $\rightarrow$ new model with may transitions.

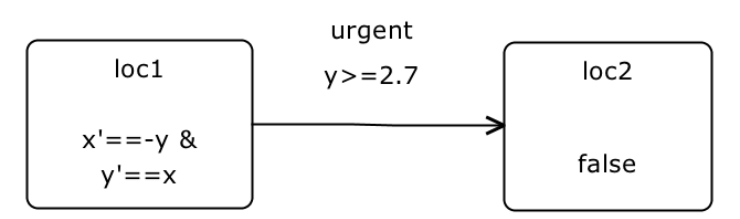
Approach[4]:

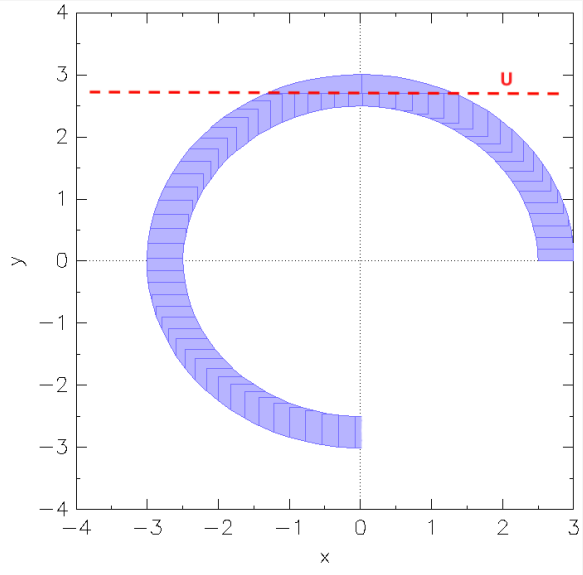
- Transform model by describing guard constraints as invariant constraints.
- Add complement of the negation of the guard in the invariant.
- Partition non-convex invariant into convex subsets.

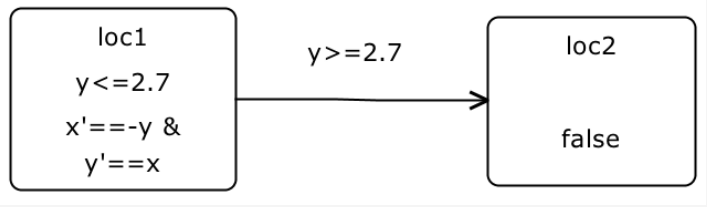
[4] Minopoli, Stefano, and Goran Frehse. "From simulation models to hybrid automata using urgency and relaxation." HSCC, 2016.

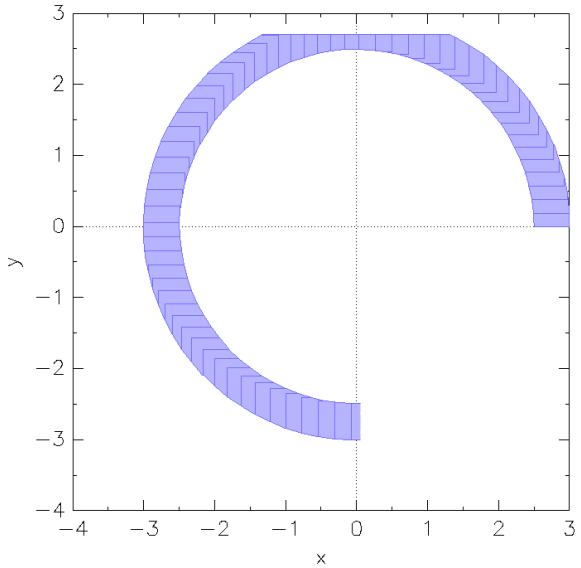
Handling Must Semantics

Idea: Model with urgent transitions $\rightarrow$ new model with may transitions.

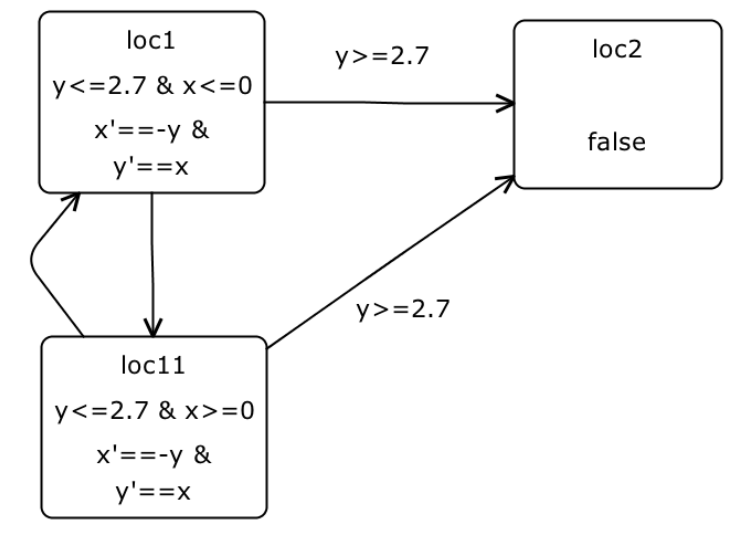


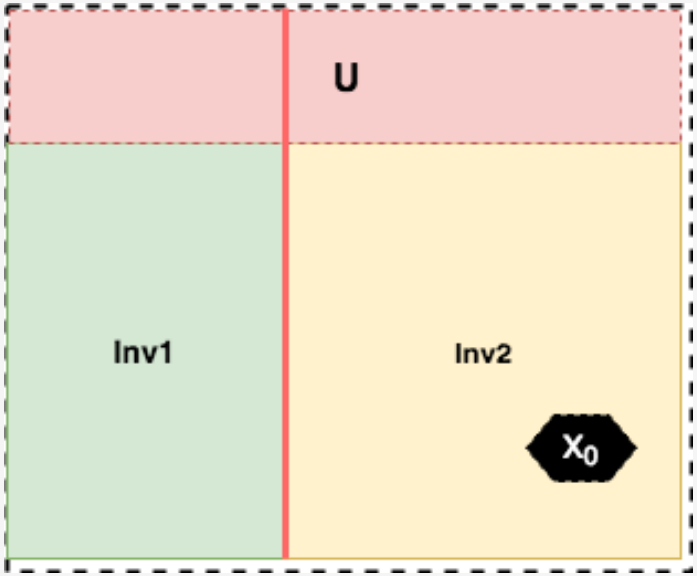


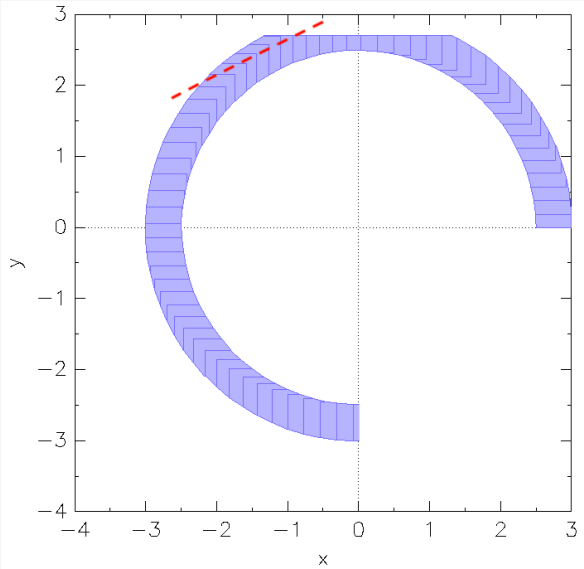


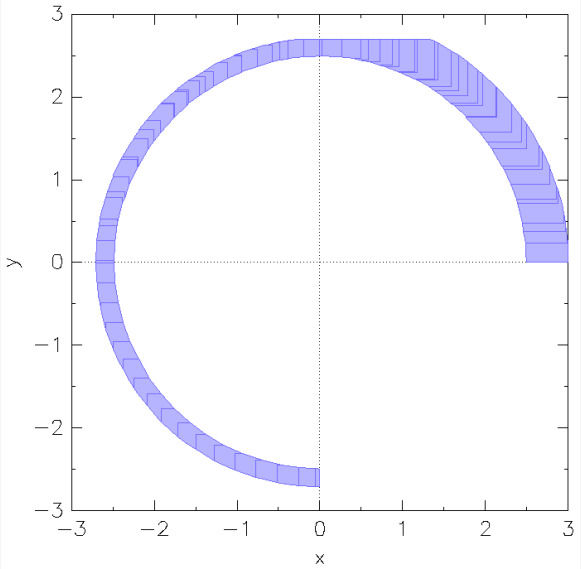


Proposed Approach to Reduce Approximation Error

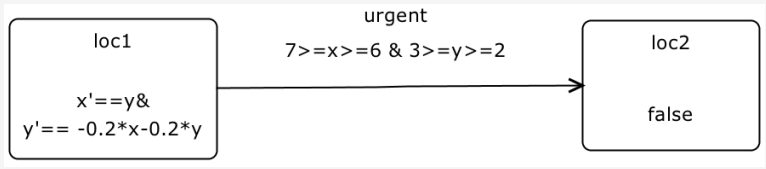


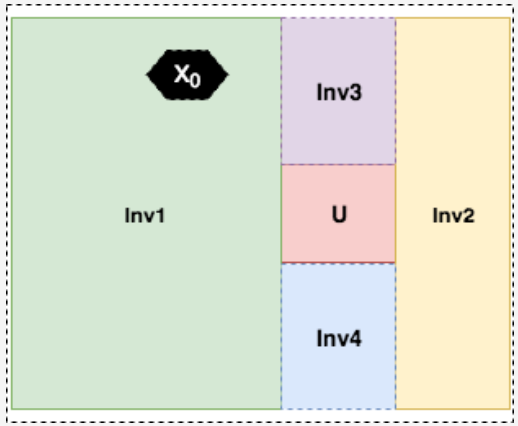


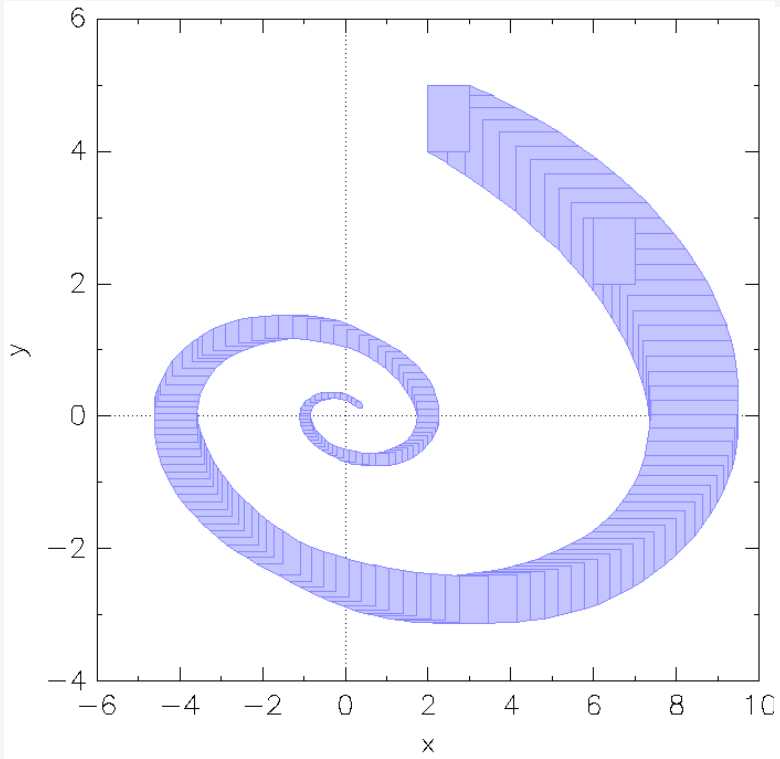


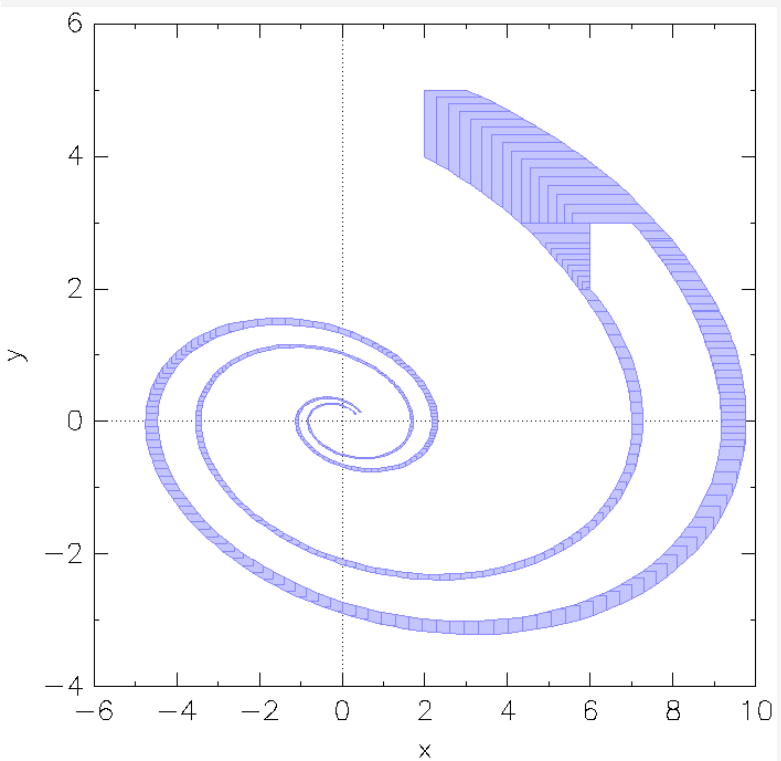


Proposed Approach to Reduce Approximation Error



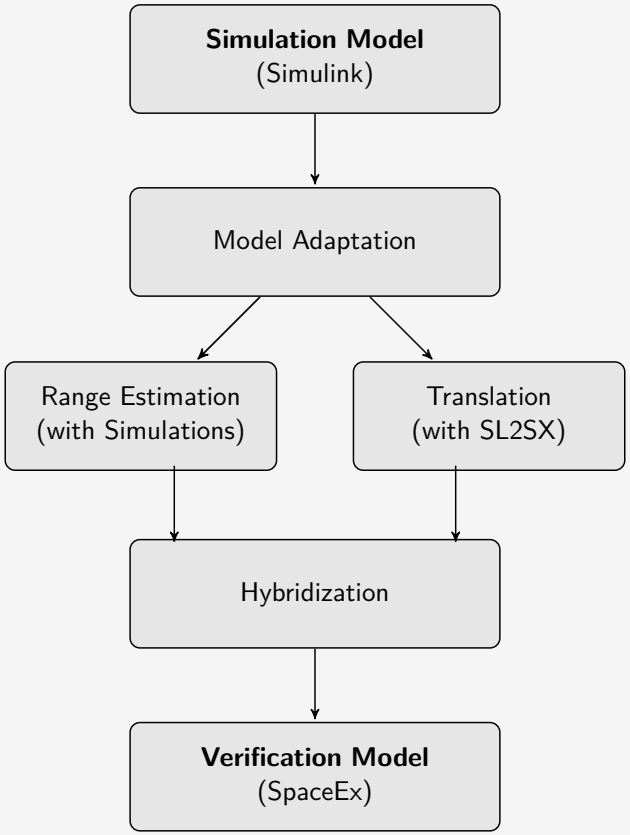






Transformation Challenge no. 3 - Syntax
MATLAB/Simulink

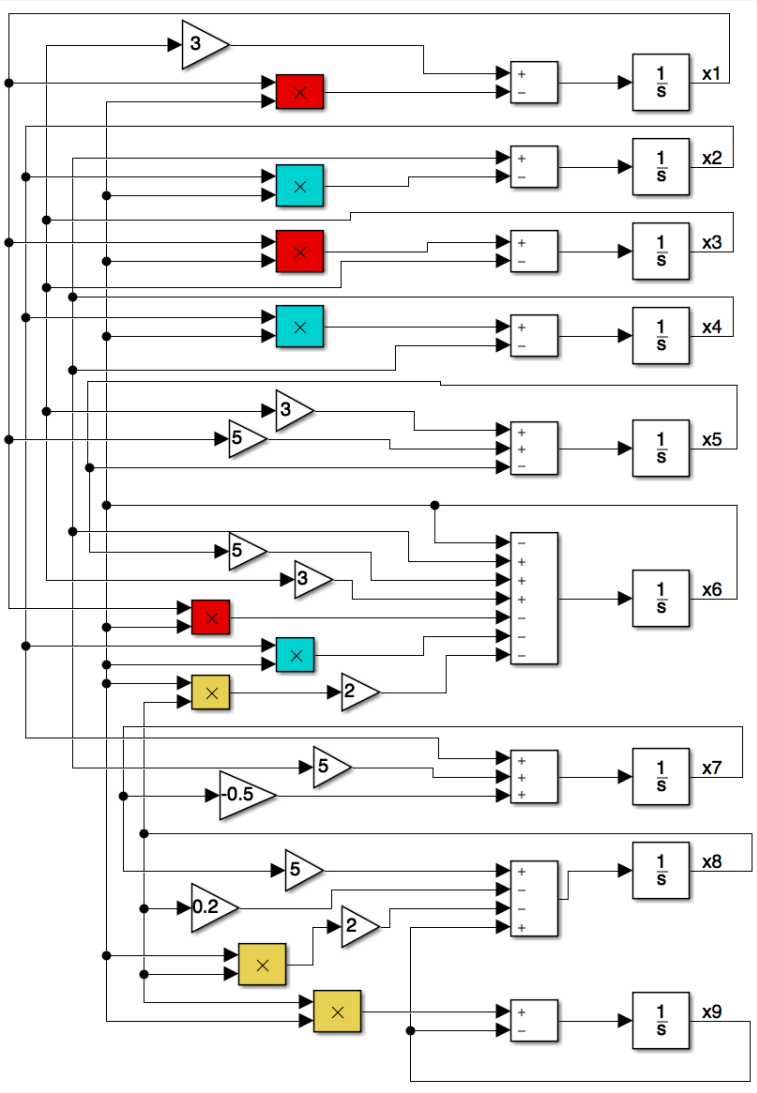
Matlab/Simulink is a widely used tool for modeling, control and code generation of hybrid and embedded systems.



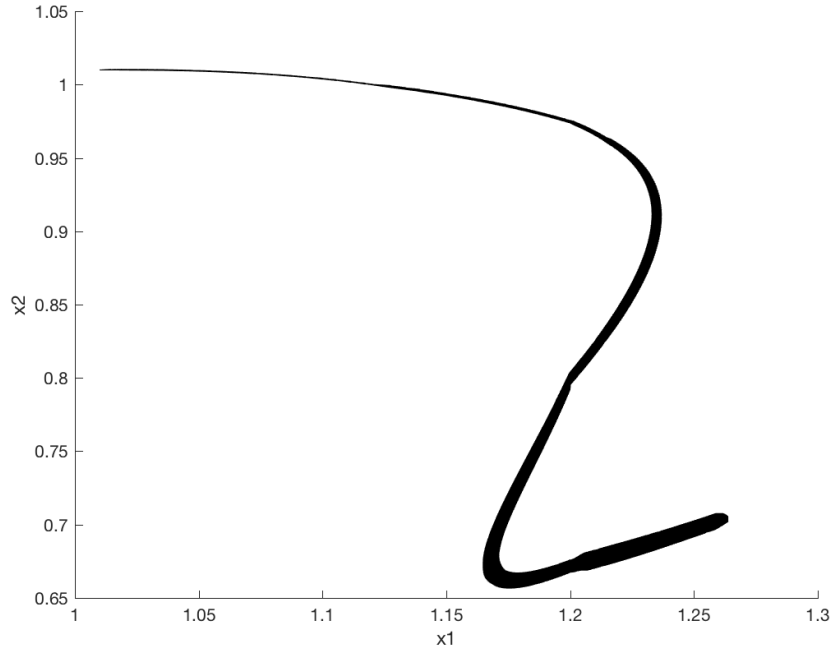
Prototype tool: SynLin

From MATLAB/Simulink to Hybrid Automata

Reusable abstractions are practical for Simulink models.



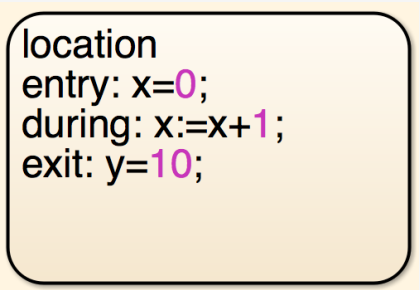
Reachability analysis with SpaceEx for a 9-dim. nonlinear genetic model.

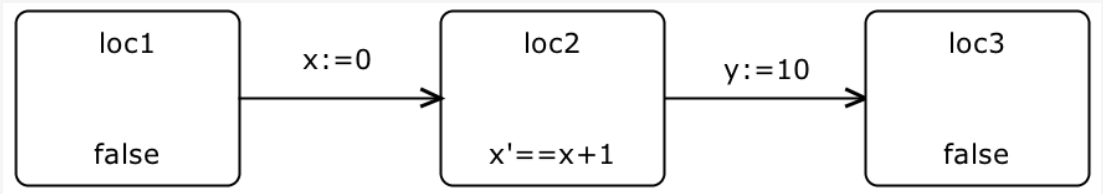


Supporting a Subset of Stateflow Diagrams

Stateflow is a toolbox suitable for describing hybrid behavior, switches and logic.
Relying existing work[5] we can handle:

- Syntax
- Parallel composition
- Hierarchy
- Synchronization

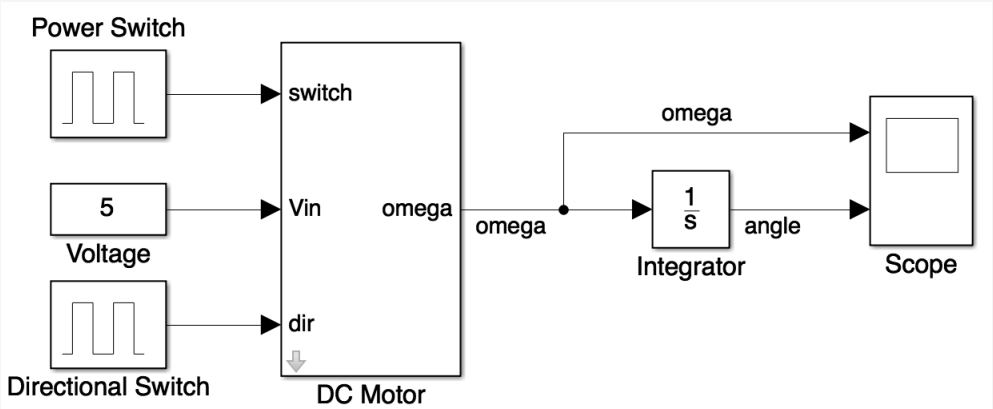


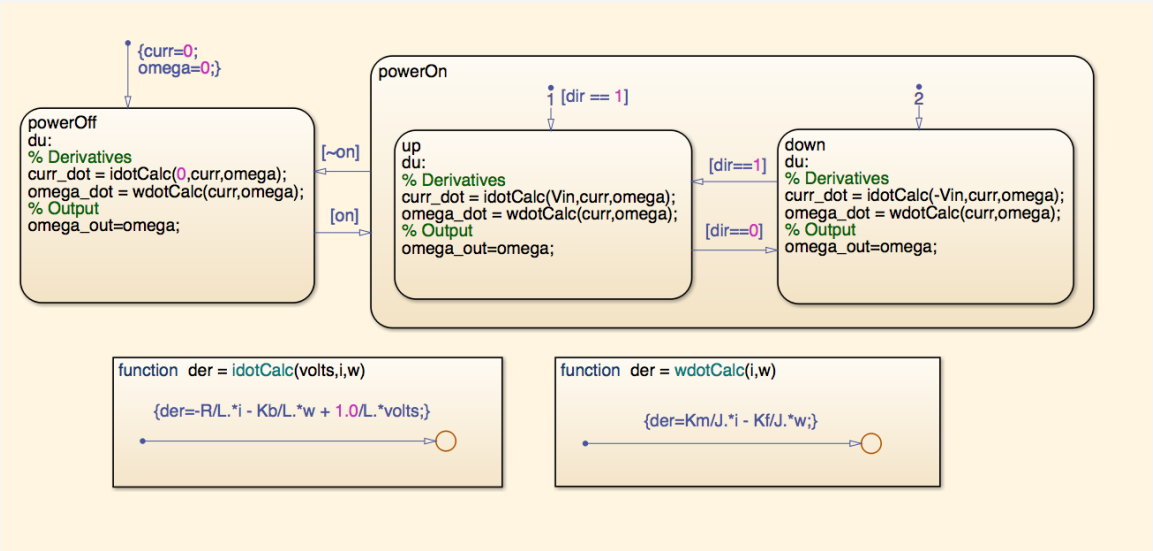


[5] Manamcheri, K., Mitra, S., Bak, S., & Caccamo, M. A step towards verification and synthesis from Simulink/Stateflow models. HSCC, 2011

Supporting a Subset of Stateflow Diagrams - an Example

Hierarchical model. A Simulink model from Mathworks that describes a permanent-magnet DC motor and employs hierarchy.





Supporting a Subset of Stateflow Diagrams - an Example

HA model. The existence of hierarchy necessitates the use of synchronization labels. Self-loops are critical to avoid deadlocks due to the synchronization. Also, transitions from locations `up` and `down` back to the `powerOff` are required since this location corresponds to a superstate.

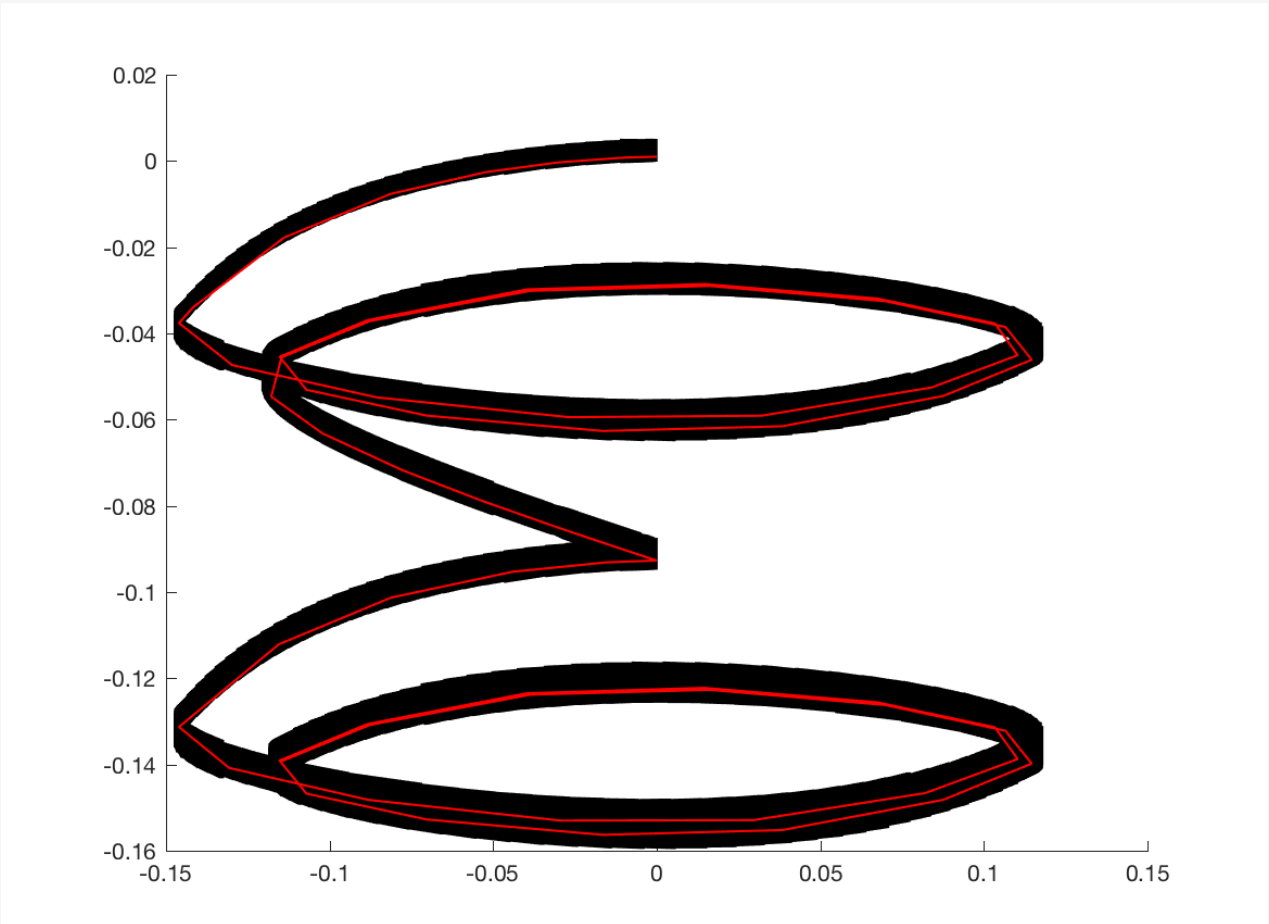
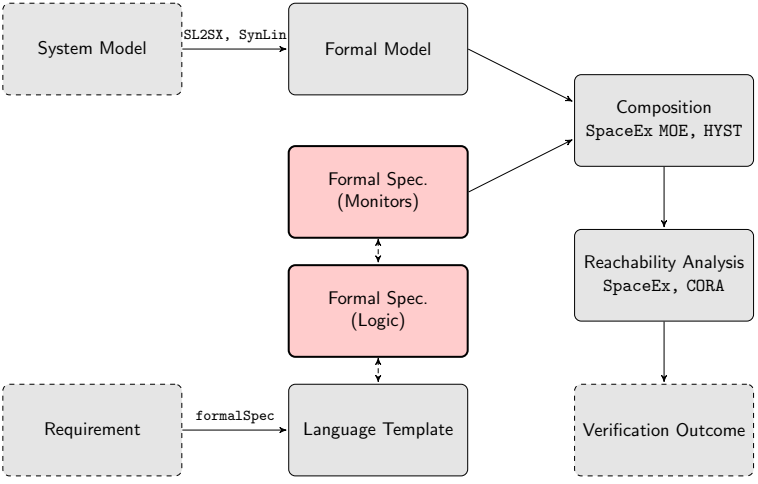
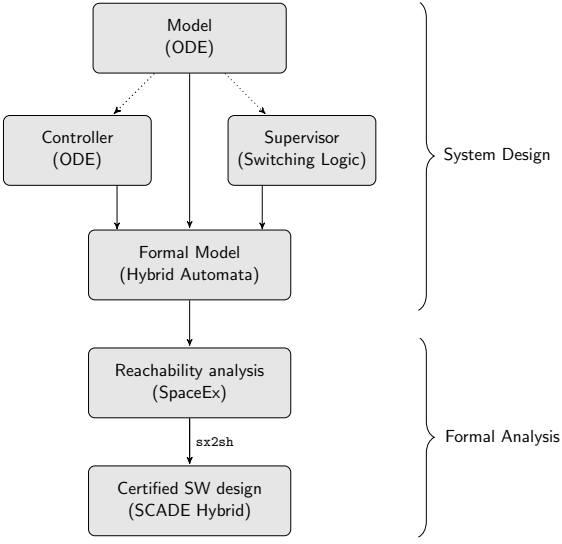


Table of Contents

Toolchain


System Model
SL2SX, SynLin
Formal Model
Composition
SpaceEx MOE, HYST
Formal Spec.
(Monitors)
Reachability Analysis
SpaceEx, CORA
Formal Spec.
(Logic)
Requirement
formalSpec
Language Template
Verification Outcome

Toolchain



Proposed workflow and toolchain for the construction of verified controllers using SpaceEx and SCADE

Toolchain

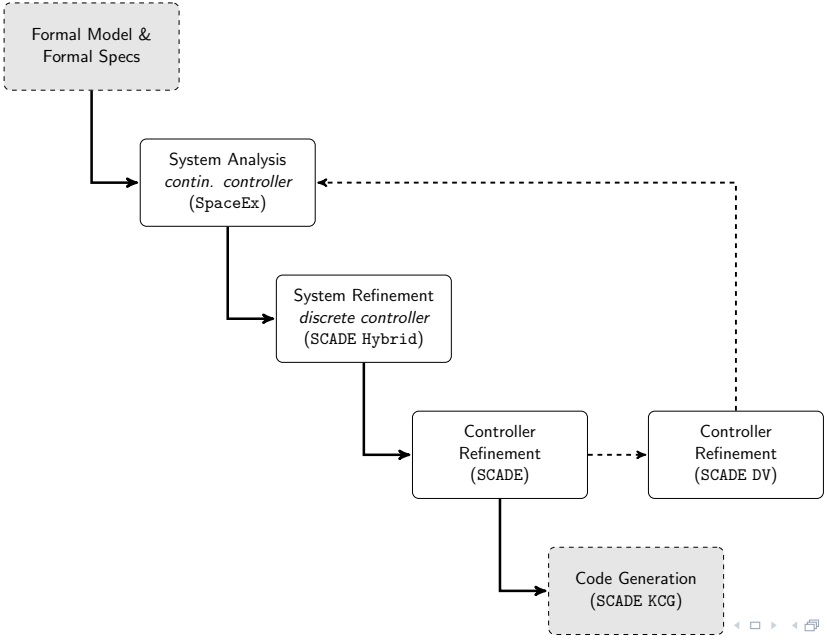
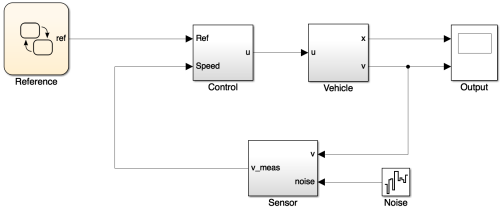

Formal Model &
Formal Specs
System Analysis
contin. controller
(SpaceEx)
System Refinement
discrete controller
(SCADE Hybrid)
Controller
Refinement
(SCADE)
Controller
Refinement
(SCADE DV)
Code Generation
(SCADE KCG)

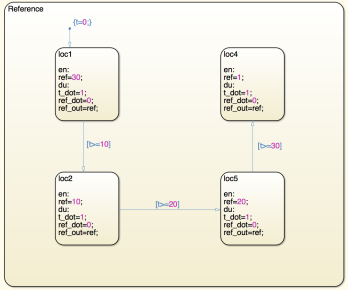
Table of Contents

Case Studies

Cruise Controller (work with Esterel/ANSYS)

Simulink Model. We assume a linear plant and a PID controller. The speed uncertainty is modeled as a bandwidth-limited white noise with a noise power of 0.01 and correlation time of 0.1s. The reference signal is time-varying and is modeled with Stateflow.

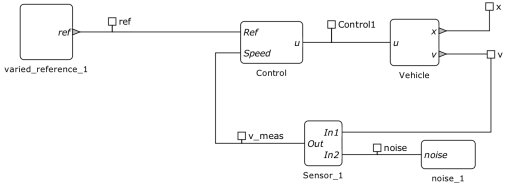


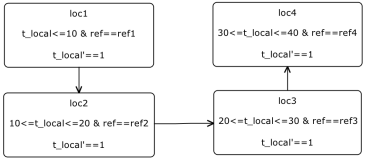


Case Studies

Cruise Controller

Formal Model. Using the SL2SX tool, we obtain a PWA SpaceEx model with 13 base components (single HA) and 3 network components (networks of HA).





Case Studies

Cruise Controller

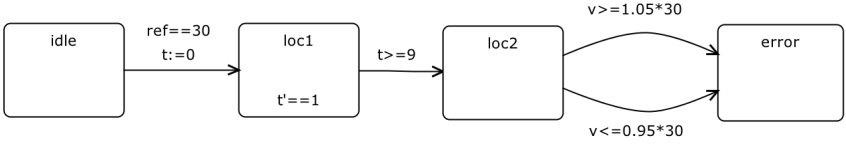
Control Objectives. Regulate the vehicle speed while respecting the following design specifications, (i) rise-time < 6s, (ii) settling-time < 9s, and (iii) overshoot < 10%.

Formal Specifications. The rise-time is mapped to the *bounded response* pattern, the settling-time to the *timed absence* pattern, and the overshoot to the *absence* pattern.

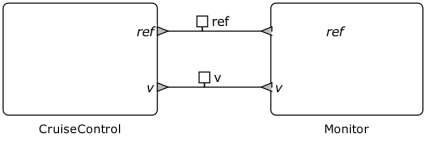
Case Studies

Cruise Controller

Monitor. Considering the first reference speed ($v = 30$) and the settling-time pattern, we employ the monitor for the timed-absence.



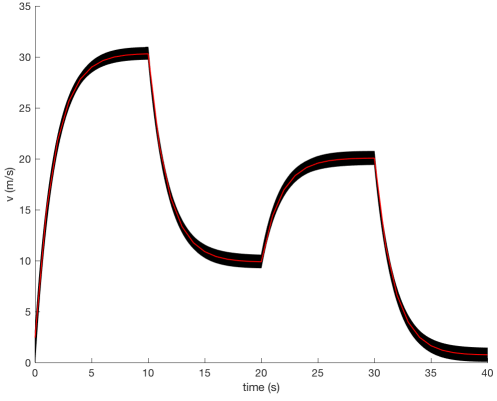
Composition. The monitor automaton, once composed with the system under test, encodes the requirement as a reachability property (shown in SpaceEx Model Editor).



Case Studies

Cruise Controller

Verification. We run SpaceEx for $T = 40$s and uncertain X_0, i.e. $0 \leq v_0 \leq 3$. The forbidden state is "loc(monitor)==error".

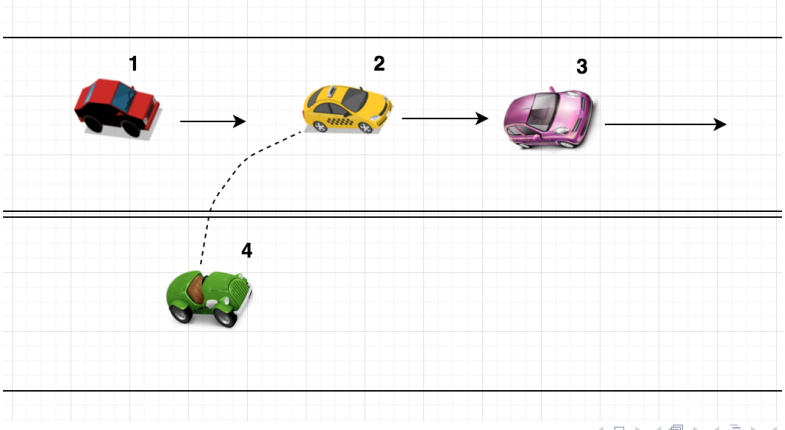


The induced computational overhead is less than 10%.

Case Studies

Lane Change Maneuver (with DLR)

Description: modeling a cooperative lane change maneuver that involves four autonomous cars; one in the right lane and three in the left lane.The maneuver should be undertaken while (i) maintaining safety margins with all the surrounding vehicles, (ii) respecting the tra□c rules, and (iii) satisfying physical and design limitations.



Lane Change Maneuver

Benchmark Requirements & Assumptions

Assumptions:

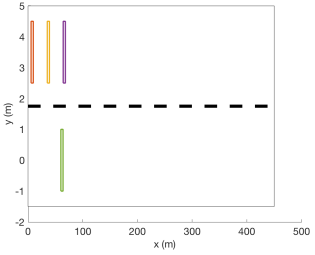
- Automated merging maneuver [**?**], i.e. move in the middle of two pre-defined vehicles.
- Maneuver requested, e.g. road infrastructure or emergency
- E□cient car to car communication
- Vehicle platooning with desired speed

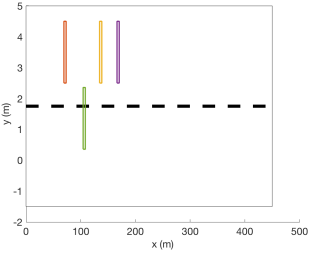
Requirements: In the spirit of [**?**], the maneuver should be undertaken while

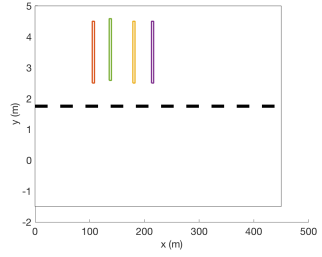
- maintaining safety margins with all the surrounding vehicles,
- respecting the tra□c rules, and
- satisfying the physical and design limitations.

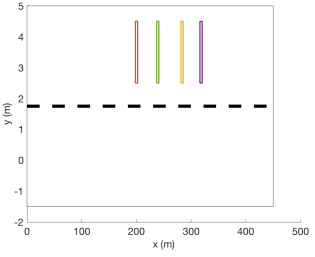
Case Studies

Lane Change Maneuver



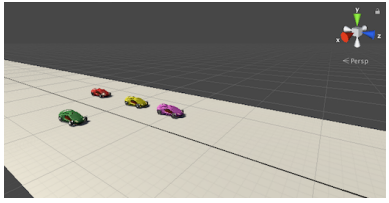




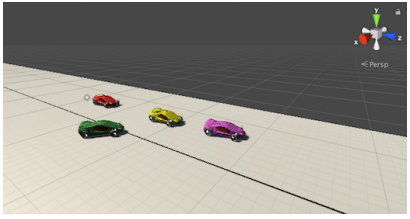


Case Studies

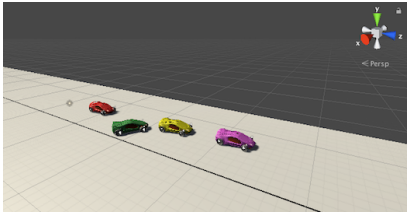
Lane Change Maneuver



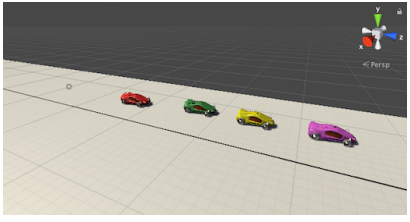
(a) Initial Phase (before maneuver)



(b) Maneuver has begun



(c) Maneuver - in progress



(d) Maneuver finished

Table of Contents

Conclusions

- Facilitate the verification of hybrid systems against rich formal requirements.
  - Define patterns in a formalism suitable for hybrid automata and applicable over bounded and unbounded time.
  - Give formally correct monitors for these patterns.
  - By composing the monitors with the system model, the verification problem is transformed into the reachability problem of an error state.
- Provide a (semi-) automated toolchain amenable to industrial systems.
  - Employ o□-the-shelf fully automated tools.
  - Applicable to industries with text-based requirements.
  - Yield risk reduction when translating requirements into formal specifications.

Future Work

- E□cient implementation of urgent transitions for hybrid automata.
- Construct monitors for other patterns that appear often.
- Application to control systems could be beneficial, e.g. identify maximum set of X_0 that satisfies a control objective.
- Extension to quantitative semantics, e.g. replace Boolean outcome, add robustness metric.

How to instantiate as few locations and transitions as possible during reachability analysis?

- On-the-fly instantiation and composition of the models.
- Compositional pre-processing of the components.
- Compositional mapping of the initial states.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 643921.

Thank you!

References

Davide Bresolin.
HyLTL: a temporal logic for model checking hybrid systems.
In Hybrid Autonomous Systems Workshop, 2013.

Axel Busboom, Simone Schuler, and Alexander Walsch.
formalspec - semi-automatic formalization of system requirements for formal verification.
In ARCH, 2016.

Alessandro Cimatti, Marco Roveri, and Stefano Tonetta.
HRELTL: A temporal logic for hybrid systems.
Inf. Comput., 2015.

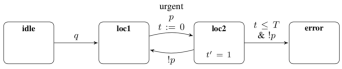
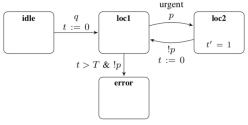
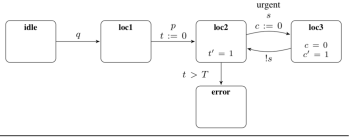
Antonio Anastasio Bruto da Costa, Pallab Dasgupta, and Goran Frehse.
Formal feature analysis of hybrid automata.
In Formal Methods and Models for System Design (MEMOCODE), 2016 ACM/IEEE International Conference on, pages 2–11. IEEE, 2016.

Nicolas Halbwachs, Fabienne Lagnier, and Pascal Raymond.
Synchronous observers and the verification of reactive systems.
In Algebraic Methodology and Software Technology (AMAST?93). Springer, 1994.

S. Konrad and B. Cheng.
Real-time specification patterns.
ICSE, 2005.

Oded Maler and Dejan Ničković.
Monitoring properties of analog and mixed-signal circuits.
STTT Journal, 2013.

Monitor Automata

absence	After q, it is never the case that p holds[a].	
absence (timed)	When T time units are measured, after q was first satisfied, it is never the case that p holds.	
minimum duration	After q, it is always the case that once p becomes satisfied, it holds for at least T time units.	
maximum duration	After q, it is always the case that once p becomes satisfied, it holds for less than T time units.	
bounded recurrence	After q, it is always the case that p holds at least every T time units.	
bounded response (persisting)	After q, it is always the case that if p holds, then s persists (holds for nonzero time) after at most T time units.	
bounded invariance	After q, it is always the case that if p holds, then s holds for at least T time units.