Formal Verification of Cyber-Physical Systems

Matthias Althoff

Technische Universität München

November 15, 2013
Examples of Cyber-Physical Systems

automated driving
source: Carnegie Mellon University

human-robot collaboration
source: Rethink Robotics

Smart grids
source: Siemens

automated farming
source: Kesmac

surgical robots
source: daVinci

Air traffic control
source: NASA

Emerging technologies are increasingly safety- or operation-critical!
Examples of Insufficiently Verified Systems

Intel Bug (1994)
- Error in the floating point division.
- costs: $500 Mio

Explosion of the Ariane 5 rocket (1996)
- Variable overflow due to software reuse from Ariane 4.
- costs: $500 Mio

Unintended acceleration in Toyota cars (2010)
- Several investigations, no mistake found (NHTSA and NASA).
- PR disaster
Need for Formal Verification

<table>
<thead>
<tr>
<th>Testing</th>
<th>Formal Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not much expert knowledge required</td>
<td>Expert knowledge required</td>
</tr>
<tr>
<td>Applicable to large systems</td>
<td>Not (yet) applicable to large systems</td>
</tr>
<tr>
<td>No guarantee of correctness</td>
<td>Guarantees correctness</td>
</tr>
</tbody>
</table>

Critical components: Proof of correctness outweighs advantages of testing.

Case study:
Unmanned aerial vehicle (UAV) flight control verified by two teams of equal size.

Lockheed Martin Aero
- Used testing
- Found no bugs

Rockwell Collins
- Used Model Checking
- Found 12 bugs

"Model-checking was more cost effective than testing at finding design errors."
(Dr. Steven P. Miller, Rockwell Collins, CMACS Industry Workshop, 2011)
Interaction of Discrete and Continuous Dynamics

Formal verification has first been developed for discrete systems. However:

**Cyber-physical systems interact with the physical world.**

### Discrete Dynamics
- discrete-event control
- communication protocols
- discrete sensors/actuators
- discrete signal processing
- scheduling algorithms

### Continuous Dynamics
\[
\begin{align*}
\dot{x}(t) &= f(x(t), u(t)) \\
y(t) &= g(x(t), u(t))
\end{align*}
\]

- $x$: state vector
- $u$: input vector
- $y$: output vector
- continuous control
- physics
- continuous sensors/actuators
- continuous signal processing
Outline

Modeling of Cyber-Physical Systems with Hybrid Automata

Definition of the Verification Problem

Reachability Analysis

- Linear continuous systems
- Nonlinear continuous systems
- Hybrid systems
- Applications (automated driving, phase-locked loops, smart grids)

Future Research Proposals

- Scalable verification algorithms using compositional design
- Model-based design using abstract models
- Fault-tolerant systems
Hybrid Automaton (HA)

\( HA = (Z, z^0, \mathcal{X}, \mathcal{X}^0, U, \text{inv}, T, \text{guard}, \text{jump}, f) \)

- **set of locations** \( Z = \{z_1, \ldots, z_m\} \) with initial location \( z^0 \),
- **continuous state space** \( \mathcal{X} \subset \mathbb{R}^n \) with initial continuous set \( \mathcal{X}^0 \),
- **discrete transitions** \( T \subseteq Z \times Z \),
- **invariants** \( \text{inv} : Z \rightarrow 2^{\mathcal{X}} \),
- **guard sets** \( \text{guard} : T \rightarrow 2^{\mathcal{X}} \),
- **jump functions** \( \text{jump} : T \times \mathcal{X} \rightarrow \mathcal{X} \)
- **flow functions** \( f : \mathcal{X} \times U \times Z \rightarrow \mathbb{R}^n \) (\( U \): continuous input space)

**Continuous evolution**

- Start at \( z^0 \) and \( x^0 \in \mathcal{X}^0 \)
- \( x(t) \) is the solution of

\[ \dot{x}(t) = f(x(t), u(t), z(t)) \]
Hybrid Automaton (HA)

\[ HA = (Z, z^0, X, X^0, U, inv, T, guard, jump, f) \]

- set of locations \( Z = \{z_1, \ldots, z_m\} \) with initial location \( z^0 \),
- continuous state space \( X \subset \mathbb{R}^n \) with initial continuous set \( X^0 \),
- discrete transitions \( T \subseteq Z \times Z \),
- invariants \( inv : Z \rightarrow 2^X \),
- guard sets \( guard : T \rightarrow 2^X \),
- jump functions \( jump : T \times X \rightarrow X \)
- flow functions \( f : X \times U \times Z \rightarrow \mathbb{R}^n \) (\( U \): continuous input space)

Activation of discrete transition

- Transition \((z_j, z_i)\) is activated when \( x(t) \in guard((z_j, z_i)) \)
- Transition has to be taken when \( x(t) \) at the border of \( inv(z_i) \)
Hybrid Automaton (HA)

\[ HA = (Z, z^0, \mathcal{X}, \mathcal{X}^0, U, \text{inv}, T, \text{guard}, \text{jump}, f) \]

- **set of locations** \( Z = \{z_1, \ldots, z_m\} \) with initial location \( z^0 \),
- **continuous state space** \( \mathcal{X} \subseteq \mathbb{R}^n \) with initial continuous set \( \mathcal{X}^0 \),
- **discrete transitions** \( T \subseteq Z \times Z \),
- **invariants** \( \text{inv} : Z \rightarrow 2^\mathcal{X} \),
- **guard sets** \( \text{guard} : T \rightarrow 2^\mathcal{X} \),
- **jump functions** \( \text{jump} : T \times \mathcal{X} \rightarrow \mathcal{X} \)
- **flow functions** \( f : \mathcal{X} \times U \times Z \rightarrow \mathbb{R}^n \) (\( U \): continuous input space)

**Discrete transition and jump of continuous state**

- Location changes from \( z_i \) to \( z_j \)
- Continuous state may jump:
  \[ x' = \text{jump}((z_j, z_i), x) \]
  \((x')\): cont. state after jump
Hybrid Automaton (HA)

\[ HA = (\mathcal{Z}, z^0, \mathcal{X}, \mathcal{X}^0, \mathcal{U}, \text{inv}, T, \text{guard}, \text{jump}, f) \]

- **set of locations** \( \mathcal{Z} = \{z_1, \ldots, z_m\} \) with initial location \( z^0 \),
- **continuous state space** \( \mathcal{X} \subseteq \mathbb{R}^n \) with initial continuous set \( \mathcal{X}^0 \),
- **discrete transitions** \( T \subseteq \mathcal{Z} \times \mathcal{Z} \),
- **invariants** \( \text{inv} : \mathcal{Z} \rightarrow 2^\mathcal{X} \),
- **guard sets** \( \text{guard} : T \rightarrow 2^\mathcal{X} \),
- **jump functions** \( \text{jump} : T \times \mathcal{X} \rightarrow \mathcal{X} \)
- **flow functions** \( f : \mathcal{X} \times \mathcal{U} \times \mathcal{Z} \rightarrow \mathbb{R}^n \) (\( \mathcal{U} \): continuous input space)
Reachability Analysis

Informal Definition

A reachable set is the set of states that can be reached by a dynamical system in finite or infinite time for a

- set of initial states,
- uncertain inputs,
- and uncertain parameters.
Verification Task

Check if a set of unsafe states is never reached.

- Exact reachable set only for special classes computable → overapproximation computed for consecutive time intervals.
- Overapproximation might lead to spurious counterexamples.
- Simulation cannot prove correctness.
Linear Systems: Overview of Reachable Set Computation

\[ \dot{x}(t) = Ax(t) + u(t), \quad A \in \mathbb{R}^{n \times n}, \quad x(t) \in \mathbb{R}^n, \quad x(0) \in \mathcal{R}(0), \quad u(t) \in u_c \oplus \mathcal{U} \]

1. Compute reachable set \( \mathcal{H}(r) = e^{Ar}\mathcal{R}(0) \oplus \int_{t=0}^{r} e^{A(r-t)} \text{d}t \space u_c \) at time \( r \) neglecting the uncertain input (\( \mathcal{C} \oplus \mathcal{D} := \{ c + d | c \in \mathcal{C}, d \in \mathcal{D} \} \)).

2. Obtain convex hull of initial set \( \mathcal{R}(0) \) and \( \mathcal{H}(r) \).

3. Enlarge reachable set to account for (1) uncertain inputs, (2) curvature of trajectories.

4. Continue with further time intervals \([kr, (k+1)r], \ k \in \mathbb{N}\).

Known algorithm, similar to work of A. Girard at HSCC’05.
Nonlinear Reachability Analysis: Overall Algorithm

1. initial set $\mathcal{R}(0)$, input set $\mathcal{U}$, time step $k = 1$
2. linearize system
3. compute reachable set $R_{\text{lin}}$ without linearization error
4. obtain set of linearization errors $L$ based on $R_{\text{lin}}$ and $\overline{L}$ ($\overline{L}$: set of admissible linearization errors)
5. $L \subseteq \overline{L}$?
   - yes: compute reachable set $R_{\text{err}}$ due to $L$
   - no: enlarge $\overline{L}$
6. $R = R_{\text{lin}} \oplus R_{\text{err}}$
Overall Algorithm: Animation

1. Linearize system
2. \( \mathcal{R}(0) \)
3. Linearize system
Overall Algorithm: Animation

\[ \mathcal{R}_{lin}([0, r]) \]

compute reachable set \( \mathcal{R}_{lin} \) without linearization error
Overall Algorithm: Animation

\[ R_{lin}([0, r]) \oplus \overline{R}_{err}([0, r]) \]

\( \overline{R}_{err} \): reachable set due to \( \mathcal{L} \)

obtain set of linearization errors \( \mathcal{L} \) based on
\[ R_{lin}([0, r]) \oplus \overline{R}_{err}([0, r]) \]
Overall Algorithm: Animation

\[ \mathcal{R}([0, r]) = \mathcal{R}_{\text{lin}}([0, r]) \oplus \mathcal{R}_{\text{err}}([0, r]) \]

\[ \mathcal{L} \subseteq \overline{\mathcal{L}} ? \]
Overall Algorithm: Animation

$\mathcal{R}([r, 2r])$

reachable set of next time interval
Overall Algorithm: Animation

The reachable set of the complete time horizon $t_f$ is $\mathcal{R}([0, t_f])$. The possible trajectories are shown by the black lines.
Linearization and Lagrange Remainerder

\[ \dot{x} = f(z(t)), \quad z^T := [x^T, u^T] \text{ and } t \in [kr, (k+1)r]: \]

**Taylor series**

\[ \dot{x}_i \in f_i(z^*) + \left. \frac{\partial f_i(z)}{\partial z} \right|_{z=z^*} (z - z^*) \oplus \]

1\textsuperscript{st} order Taylor series: \( A\Delta x + B\Delta u + f_i(z^*) \)

\[ \left\{ \frac{1}{2} (z - z^*)^T \left. \frac{\partial^2 f_i(z)}{\partial z^2} \right|_{z=\xi} (z - z^*) \bigg| \xi, z \in \mathcal{R}([kr, (k+1)r]) \times U \right\} \]

Lagrange remainder \( L_i \)

Possible computation of the Lagrange remainder:
Enclose reachable set by a multidimensional interval and apply interval arithmetic.
Scalability of the Linearization Approach

Water tank system.

Projected reachable set 
\((n = 6)\).

Complexity with respect to the number of continuous state variables \(n\): \(O(n^3)\).

<table>
<thead>
<tr>
<th>Dimension (n)</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU-time [sec]</td>
<td>1.19</td>
<td>1.73</td>
<td>3.11</td>
<td>11.59</td>
<td>35.78</td>
</tr>
</tbody>
</table>
Reachability Analysis of Hybrid Systems

Hybrid systems additionally require intersections of guard sets:

\[ \mathcal{R}(0) \cap \mathcal{R}([t_k, t_{k+1}]) \]

\[ \mathcal{R}_g(t_\eta) \]

\( t_\eta \): last point in time before intersecting the hyperplane.
\( \mathcal{R}_g \): Overapproximation of the guard set intersection.
Scalability of the Mapping-Based Approach

Powertrain with backlash.

Complexity with respect to the number of continuous state variables $n$: $O(n^5)$.

<table>
<thead>
<tr>
<th>Dimension $n$</th>
<th>11</th>
<th>21</th>
<th>31</th>
<th>41</th>
<th>51</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU time [sec]</td>
<td>8.122</td>
<td>14.31</td>
<td>23.72</td>
<td>31.83</td>
<td>53.74</td>
<td>1550</td>
</tr>
</tbody>
</table>
Comparison With SpaceEx

SpaceEx: state of the art tool for reachability analysis of hybrid systems.

- Uses geometric guard intersection.
- Example sensitive to overapproximation → comparison for initial set with 5% of initial size and $n = 7$.

Computational times: 10023 s (new approach: 0.133 s).
Further Work

Reachability Analysis
- Hybrid systems with differential-algebraic equations
- Consideration of uncertain parameters
- Improved algorithms for nonlinear systems using non-convex sets

Stochastic Verification
- Abstraction of stochastic hybrid systems to Markov chains
- Probabilistic inevitable collision states
Online Verification Of Automated Driving

Test site

Test vehicle

lane change maneuver B
lane change maneuver A

initial occupancy

final occupancy

ego vehicle

reference trajectory

ego vehicle (braking part)

other vehicle

M. Althoff

CPS Verification

November 15, 2013 19 / 29
Test Drive Results

\[ s_x, s_y \, [m] \quad \text{x- and y-position} \]
\[ \Psi \, [\text{rad}] \quad \text{orientation} \]
\[ \beta \, [\text{rad}] \quad \text{slip angle at center of mass} \]
\[ \delta \, [\text{rad}] \quad \text{front wheel angle} \]
\[ v \, [\text{m/s}] \quad \text{velocity} \]

\[ \dot{\Psi} \, [\text{rad/s}] \]
\[ I_c \, B \quad I_c \, A \]

computation time: \( \approx 1.8 \) times faster than maneuver time (Intel i7, 1.6GHz)
Verification of a Phase-Locked Loop (PLL)

**System properties:**
- Hybrid dynamics with many switchings ($\approx 3000$ until convergence)
- Slow convergence to locking

**Verification Task:**
Check if for any initial condition, the phase is locked to the reference signal.

(Received IEEE/ACM William J. McCalla ICCAD Best Paper Award)

**Computation time: Verification vs. simulation**

<table>
<thead>
<tr>
<th>$\Delta \Phi(0)$</th>
<th>Reachability analysis [s]:</th>
<th>Avg. MATLAB simulation [s]:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[-1, -0.8] \pi$</td>
<td>55.0461</td>
<td>48.3297</td>
</tr>
<tr>
<td>$[-0.8, -0.6] \pi$</td>
<td>54.4418</td>
<td>47.9096</td>
</tr>
<tr>
<td>$[-0.6, -0.4] \pi$</td>
<td>53.4820</td>
<td>46.2673</td>
</tr>
<tr>
<td>$[-0.4, -0.2] \pi$</td>
<td>47.8208</td>
<td>44.4596</td>
</tr>
<tr>
<td>$[-0.2, 0] \pi$</td>
<td>42.9191</td>
<td>38.5102</td>
</tr>
</tbody>
</table>
Transient Stability in Smart Grids

System properties:
- Differential-algebraic equations
- Nonlinear dynamics
- Large system size

Verification Task:
Check if all states return to the operating point after a power drop-out.

Values at bus 3:
Conclusions

- Scalability issues in formal verification of cyber-physical systems have been successfully addressed.

- The presented approaches verified applications that were previously out of reach.

- The online verification of automated vehicles is the first approach that verifies a system with non-trivial dynamics on-the-fly.

- Results of reachability analysis can be easily interpreted.

- Unlike other approaches (e.g. barrier certificates, constraint propagation, theorem proving), reachability analysis immediately shows where a specification is violated.
Scalable Verification using Compositional Design

**Simulation**: complete system has to be considered

**Reachability analysis**: set of behaviors can be computed compositionally.

**Simplest scenario**

Assume set of inputs $\mathcal{U}_A$ and $\mathcal{U}_B$ for $A$ and $B$, respectively. If $\mathcal{Y}_A \subseteq \mathcal{U}_B$ and $\mathcal{Y}_B \subseteq \mathcal{U}_A \rightarrow$ assumptions and set of behaviors are overapproximative.

Possible application to smart grids:
Verification of Programmable Logic Controllers (PLCs) and Real-Time Constraints

Timed automata or timed Petri nets are often sufficient for PLC verification.

- Timed automata and timed Petri nets are a special class of hybrid automata.
- Algorithms for hybrid automata can be specialized.
Non-Formal Verification Approaches

Non-formal verification is often more effective for less critical components. Most approaches try to steer the system into a fault:

- Sensitivity-based simulation
- Rapidly-exploring random trees
- Cross-entropy method
- Ant-colony optimization
- etc.

Method selection is more art than science.

Falsification of the room heating benchmark problem using ant-colony optimization
Future Research Proposals

Model-Based Design using Abstract Models

Build abstract subsystems to which uncertainty is added.
→ design changes within the abstraction require no design iteration.

Continuous abstraction
\[ \dot{x} \in \{ f(x, p) + u | p \in P, u \in U \} \]
Parametric \((p \in P)\) and additive uncertainties \((u \in U)\)

Discrete abstraction
Add non-deterministic transitions

Example: Vehicle dynamics for automated driving.

source: bremarauto
Fault-Tolerant Systems

**Information redundancy:**
Check bits to detect data transfer errors.

**Physical redundancy:**
- Voter switches subsystem in case of a fault.
- Drawbacks: Faults need to be directly measurable, redundant components are expensive.

**Analytical redundancy:**
- Controller re-design based on information from model-based observers.
- Drawbacks: Control performance is degraded, challenging design.

Design of scalable and reliable observers and fault-tolerant controllers for hybrid dynamics is an open research problem.
Final Thoughts

How to build safe and reliable cyber-physical systems?

- Good balance between formal and non-formal verification
- Scalable formal verification techniques (rather provide techniques that can address a limited set of specifications, but for large systems)
- Consideration of disturbances, parameter variations, and faults in the verification process
- Model-based design supporting abstraction, modularity, and hierarchy
- Built-in fault tolerance
- Good tool support