Verifying Cyber-Physical Systems by Combining Software Model Checking with Hybrid Systems Reachability

Sagar Chaki, SEI/CMU
Stanley Bak, AFRL
Motivation

Cyber-Physical System (CPSs) play safety-critical roles in day-to-day lives
- Avionics, automotive, healthcare, energy

High-level of assurance desired in terms of safe and secure behavior

Formal verification provides high confidence in principle
- In practice plagued by scalability issues
- Compositional reasoning known to help scale

We present a compositional approach to verify CPS
- Combine software model checking with hybrid system reachability
- Validated on a multi-agent collision avoidance protocol

Paper to appear at EMSOFT’16
CPS Model Of Computation

System composed of application $A$ and controller $C$

- Execute concurrently: $S = A \parallel C$
- Communicate via shared variables
  - Cyber variables written by $A$ and read by $C$
  - Physical variables written by $C$ and read by $A$
- Application $A$ available as source code
- Controller $C$ available as a hybrid automaton
  - $C = \text{controller } + \text{ plant}$ (from control theory perspective)

Want to verify that $S$ satisfies a safety property (something bad never happens)

- Formally, $S \models \Phi$ where $\Phi$ is an invariant expressing the safety property of interest
Example: 2D Quadcopter Movement

Current setpoint \( spcur = (0,0) \)

Next setpoint \( spnxt = (5,0) \)

Position \( pos = (0,0) \)

Cell Ids

\[ x \rightarrow 0, 1, 2, 3, \ldots \]

Positions

\[ -2.5, 2.5, 7.5, 12.5, 17.5 \]
Example: Target Property

\[
(\Phi_{\text{hover}} \land \text{spnxt} = \text{spcur}) \lor (\Phi_{\text{move}} \land \\
(|\text{spnxt} - \text{spcur}| = (5, 0) \lor |\text{spnxt} - \text{spcur}| = (0, 5)))
\]

\[
\Phi_{\text{hover}} \equiv |\text{pos} - \text{spcur}| \leq (1.5, 1.5)
\]

\[
\Phi_{\text{move}} \equiv \min(\text{spcur}_x, \text{spnxt}_x) - 1.5 \leq \text{pos}_x \\
\leq \max(\text{spcur}_x, \text{spnxt}_x) + 1.5 \\
\land \min(\text{spcur}_y, \text{spnxt}_y) - 1.5 \leq \text{pos}_y \\
\leq \max(\text{spcur}_y, \text{spnxt}_y) + 1.5
\]
Example: 2D Quadcopter Movement

Periodically invokes API functions
`update_setpoint(x, y)` and `has_arrived()`
that update `spcur` and `spnxt` to interact with the controller.

Continuously executes a control algorithm to move/hover the platform based on values of `spcur` and `spnxt`. Updates `pos`.

Shared Variables
Cyber: `spcur, spnxt`
Physical: `pos`

API Function Parameters
`x, y`
Verification Approach

No existing tools to verify (source code + hybrid automata)

- But each domain has its own specialized tools: software model checkers and hybrid reachability checkers
- Developing such a tool that combines the statespace $A$ and $C$ in a brute-force way will not scale

Insight: application and controller make assumptions about each other to achieve overall safe behavior

Approach:

- Use “contract automaton” to express inter-dependency between $A$ and $C$
- Separately verify that $A$ and $C$ implement desired behavior under the assumption that the other party does so as well
- Use an “assume-guarantee” style proof rule to show the $A \parallel C \models \Phi$
Benefits of Verification Approach

Use “contract automaton” to express inter-dependency between $A$ and $C$

- Explicit formal understanding between teams developing $A$ and $C$

Separately verify that $A$ and $C$ implement desired behavior under the assumption that the other party does so as well

- Compositional $\Rightarrow$ more scalable
- Use domain-specific tools $\Rightarrow$ build on progress in each area

Use an “assume-guarantee” style proof rule to show the $A \parallel C \Vdash \Phi$

- Proof-rule formally proven to be sound $\Rightarrow$ amortized proof cost
- Other variants can be developed to manage tradeoff between completeness and verification complexity
Example: Assumptions between $A$ and $C$

(C1) The application always calls $update\_setpoint(x, y)$, with arguments that satisfy the condition $|(x, y) - spcur| = (5, 0) \lor |(x, y) - spcur| = (0, 5)$.

(C2) Once the application calls $update\_setpoint(x, y)$, it can keep calling $has\_arrived()$ until it gets a return value of $\text{TRUE}$; once $has\_arrived()$ returns $\text{TRUE}$, the application can only then start to call $update\_setpoint(x, y)$ again.

(C3) When the quadcopter is hovering (i.e., $spnxt = spcur$), the controller must maintain the following invariant: $\Phi_{\text{hover}} \equiv |pos - spcur| \leq (1.5, 1.5)$.

(C4) When the quadcopter is moving (i.e., $|spnxt - spcur| = (5, 0) \lor |spnxt - spcur| = (0, 5)$), the controller must maintain the following invariant:

$$\Phi_{\text{move}} \equiv \min(spcur_x, spnxt_x) - 1.5 \leq pos_x \leq \max(spcur_x, spnxt_x) + 1.5$$

$$\land \min(spcur_y, spnxt_y) - 1.5 \leq pos_y \leq \max(spcur_y, spnxt_y) + 1.5$$
Example: Contract Automaton

\[
\begin{align*}
\text{spnxt} &= \text{spcur} \\ 
\Phi_{\text{hover}} &= \text{hover}
\end{align*}
\]

- **Invariants**:
  - C1: update_setpoint(x, y)
  - C2: has_arrived()
- **Transition**:
  - f: has_arrived()
  - req: true
  - grd: true
  - \( |\text{pos} - \text{spnxt}| \leq (0.1, 0.1) \)
  - A: \( \langle \text{spcur} := \text{spnxt} \rangle \)
  - rv: true

- **States**:
  - **hover**
    - \( \Phi_{\text{hover}} \)
    - \( f: \text{update\_setpoint}(x, y) \)
    - req: \(|(x, y) - \text{spcur}| = (5,0) \lor |(x, y) - \text{spcur}| = (0,5)\)
    - grd: true
    - A: \( \langle \text{spnxt} := (x, y) \rangle \)
    - rv: ✷

- **Transition**:
  - f: has_arrived()
  - req: true
  - grd: \(|\text{pos} - \text{spnxt}| > (0.1, 0.1)\)
  - A: \( \langle \rangle \)
  - rv: false

- **Transition Label**
  - **wait**

C1 and C2 are enforced by the possible transitions and the function calls labeling them.

C3 and C4 are enforced by the invariants labeling the locations.
**Assume-Guarantee Proof Rule**

**Premise 1:** Application $A$ refines the contract automaton $M$ (calls API functions in the right order and with proper arguments)

**Premise 2:** Controller $C$ refines the contract automaton $M$ (keeps the physical state within required bounds)

**Conclusion:** System satisfies all invariants of the contract automaton $M$ which happens to be the target safety property

\[
\Phi_{\text{hover}} \land \Phi_{\text{move}} \land \Phi_{\text{hover}} \land \Phi_{\text{move}}
\]

\[
(\Phi_{\text{hover}} \land spnxt = spcur) \lor (\Phi_{\text{move}} \land (|spnxt - spcur| = (5, 0) \lor |spnxt - spcur| = (0, 5)))
\]
Discharging The Premises

Premise1: Application $A$ refines the contract automaton $M$ (calls API functions in the right order and with proper arguments)
  • Reduced to software model checking, discharged via CBMC
  • Manually supplied invariants and used CBMC to verify that they are inductive
  • 1700 LOC, 2.9GHz, 16GB RAM, 3.5 seconds

Premise2: Controller $C$ refines the contract automaton $M$ (keeps the physical state within required bounds)
  • Reduced to hybrid system reachability, discharged via SpaceEX
  • Required continuous approximation and symmetry argument
  • 2.3GHz, 16GB RAM, 33 seconds

More details in EMSOFT’16 paper
Verifying Distributed Collision Avoidance

We implemented a system with 10 quadcopters moving on the 2D grid using a DSL called DMPL that supports synchronous model of computation.

Verified two properties of this distributed system using software model checking:

- **Property 1.** Distinct quadcopters have disjoint \( cell_{cur} \) and \( cell_{next} \) values
  
  \[ \forall i \neq j \in [0,9]. \ cell_{cur}[i] \neq cell_{cur}[j] \land cell_{cur}[i] \neq cell_{next}[j] \]

- **Property 2.** Setpoints are 5 times cell values
  
  \[ sp_{cur} = 5 \times cell_{cur} \land sp_{nxt} = 5 \times cell_{next} \]

- 17.5KLOC, 2.9GHz, 16GB RAM, 1900 seconds

Proved that these two properties and the property of movement of a single quadcopter verified earlier using a contract automaton \( \Rightarrow \) distance between centers of distinct quadcopters is always greater than the quadcopter diameter:

- Encoded as a SMT formula and proved using Z3
- Implies physical collision avoidance of the distributed system
Conclusion

Presented a compositional approach to verify CPS consisting of an application and a controller

• Combine software model checking with hybrid system reachability and works at the source code level
• Based on a contract automaton to capture application-controller dependencies and a sounds assume-guarantee style proof rule
• Validated on a multi-agent collision avoidance protocol

Future Work

• Manual steps automated and packaged as an end-to-end tool
• Parametric verification can reason about unbounded number of quadcopters and grids
QUESTIONS?