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

Stanley Bak, AFRL
Sagar Chaki, SEI/CMU

May 12, 2017

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Motivation

Cyber-Physical System (CPSs) play safety-critical roles in day-to-day lives

- Avionics, automotive, healthcare, energy

High-level of assurance of safe and secure behavior desired

- As close to the executable as possible

Formal verification provides high confidence in principle, but

- **Issue1:** Application and controller algorithms analysed by different techniques – each with their own specialized tools
- **Issue2:** In practice plagued by scalability issues
- Can **compositional reasoning** address both issues?

We present a compositional approach to verify CPS software

- Software model checking + hybrid system reachability
- Validated on a multi-agent collision avoidance protocol

Paper appeared 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 $V_C$ written by $A$ and read by $C$
  - Physical variables $V_P$ written by $C$ and read by $A$
- Accessed by $A$ via API functions
- 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 $sp_{cur} = (0,0)$

Position $pos = (0,0)$

Next setpoint $sp_{nxt} = (5,0)$

Cell Ids $x \rightarrow 0$, 1, 2, 3, ...

Positions $-2.5$, $2.5$, $7.5$, $12.5$, $17.5$
Example: Target Property

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

\[ (\Phi_{hover} \land spnxt = spcur) \]

\[ \lor \]

\[ (\Phi_{move} \land (|spnxt - spcur| = (5,0) \lor |spnxt - spcur| = (0,5))) \]
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`.
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 $A \parallel C \models \Phi$
  - 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 $\text{update\_setpoint}(x, y)$, with arguments that satisfy the condition $|(x, y) - \text{spcur}| = (5, 0) \lor |(x, y) - \text{spcur}| = (0, 5)$.

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

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

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

$$\Phi_{\text{move}} \equiv \min(\text{spcur}_x, \text{spnxt}_x) - 1.5 \leq \text{pos}_x \leq \max(\text{spcur}_x, \text{spnxt}_x) + 1.5$$

$$\land \min(\text{spcur}_y, \text{spnxt}_y) - 1.5 \leq \text{pos}_y \leq \max(\text{spcur}_y, \text{spnxt}_y) + 1.5$$
Example: Contract Automaton

\( C_1 \) and \( C_2 \) are enforced by the possible transitions and the function calls labeling them.

\( C_3 \) and \( C_4 \) are enforced by the invariants labeling the locations.
Contract Automaton Invariant = Target Property

\[ spnxt = spcur \]

\[ \Phi_{hover} \]

\[ f: update_setpoint(x, y) \]

\[ req: |(x, y) - spcur| = (5,0) \]

\[ \lor |(x, y) - spcur| = (0,5) \]

\[ \text{grd: true} \]

\[ A: \langle spnxt := (x, y) \rangle \]

\[ rv: \diamond \]

\[ f: has\_arrived() \]

\[ req: \text{true} \]

\[ \text{grd: } |pos - spnxt| \leq (0.1, 0.1) \]

\[ A: \langle spcur := spnxt \rangle \]

\[ rv: \text{true} \]

\[ |spnxt - spcur| = (5,0) \lor |spnxt - spcur| = (0,5) \]

\[ \Phi_{move} \]

\[ f: has\_arrived() \]

\[ req: \text{true} \]

\[ \text{grd: } |pos - spnxt| > (0.1, 0.1) \]

\[ A: \langle \rangle \]

\[ rv: \text{false} \]

CA Invariant = disjunction of state invariants

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

Target Property
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)

**Theorem 1 (Compositional Refinement).**

\[
\frac{A \preceq M \quad C \preceq M}{A \parallel C \preceq M}
\]

Conclusion: System satisfies all invariants of the contract automaton $M = \text{target safety property}$

\[
(\Phi_{\text{hover}} \land spnxt = spcur) \lor (\Phi_{\text{move}} \land
(\lvert spnxt - spcur \rvert = (5, 0) \lor \lvert spnxt - spcur \rvert = (0, 5)))
\]
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
Discharging Premise 1

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

\[ spnxt \equiv spcur \]

\[ f: \text{update\_setpoint}(x, y) \]

\[ \text{req: } |(x, y) - spcur| = (5, 0) \]
\[ \lor |(x, y) - spcur| = (0, 5) \]
\[ \text{grd: true} \]
\[ A: \langle spnxt := (x, y) \rangle \]
\[ rv: \Diamond \]

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

\[ \text{enum Loc \{hover, wait\};} \]
\[ \text{Loc loc = hover;} \]

\[ \text{void update\_setpoint(double x, double y) \{} \]
\[ \text{\quad pos = *; //-- assign non-deterministic value} \]
\[ \text{\quad if (loc == hover) \{} \]
\[ \text{\quad\quad assume(\text{INV\_hover}); assert(\text{REQ\_hover\_wait});} \]
\[ \text{\quad\quad spnxt = (x, y); assert(\text{INV\_wait});} \]
\[ \text{\quad\quad loc = wait; return;} \]
\[ \text{\quad\}} \]
\[ \text{\quad assert(0);} \]
\[ \text{\}} \]

Verification Stub for \text{update\_setpoint()}

Verifying Cyber-Physical Systems: Chaki, Bak
October 4, 2016
© 2016 Carnegie Mellon University
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.
Discharging Premise 1

Verification Stubs for API functions

Application Source code that calls API functions

CBMC Software Model Checker

Verification Result

SUCCESS

void A1() {
    for (int n=1;;++n) {
        update_setpoint(n,0);
        while(!has_arrived());
    }
}
Discharging Premise 1

Verification Stubs for API functions

Application Source code that calls API functions

CBMC Software Model Checker

Verification Result

FAILURE

void A2() {
    for (int n=1;;++n) {
        update_setpoint(n,0);
        while(has_arrived());
    }
}

More details in paper
Discharging Premise 2

Hybrid Automaton extracted from contract automaton

Hybrid automaton for controller dynamic

SpaceEX Hybrid Reachability Tool

Verification Result
Discharging Premise 2

SpaceEX Hybrid Reachability Tool

Success

Used symmetry to reduce statespace (dimensions, time horizon)

More details in 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 $cell_{curr}$ and $cell_{next}$ values
  - $\forall i \neq j \in [0,9]. cell_{curr}[i] \neq cell_{curr}[j] \land cell_{curr}[i] \neq cell_{next}[j]$

• Property 2. Setpoints are 5 times cell values
  - $sp_{curr} = 5 \times cell_{curr}$ and $sp_{next} = 5 \times cell_{next}$

• 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 that 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?