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

Sagar Chaki, SEI/CMU

Stanley Bak, AFRL

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0003816

AFRL case number: 88ABW-2016-2806

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

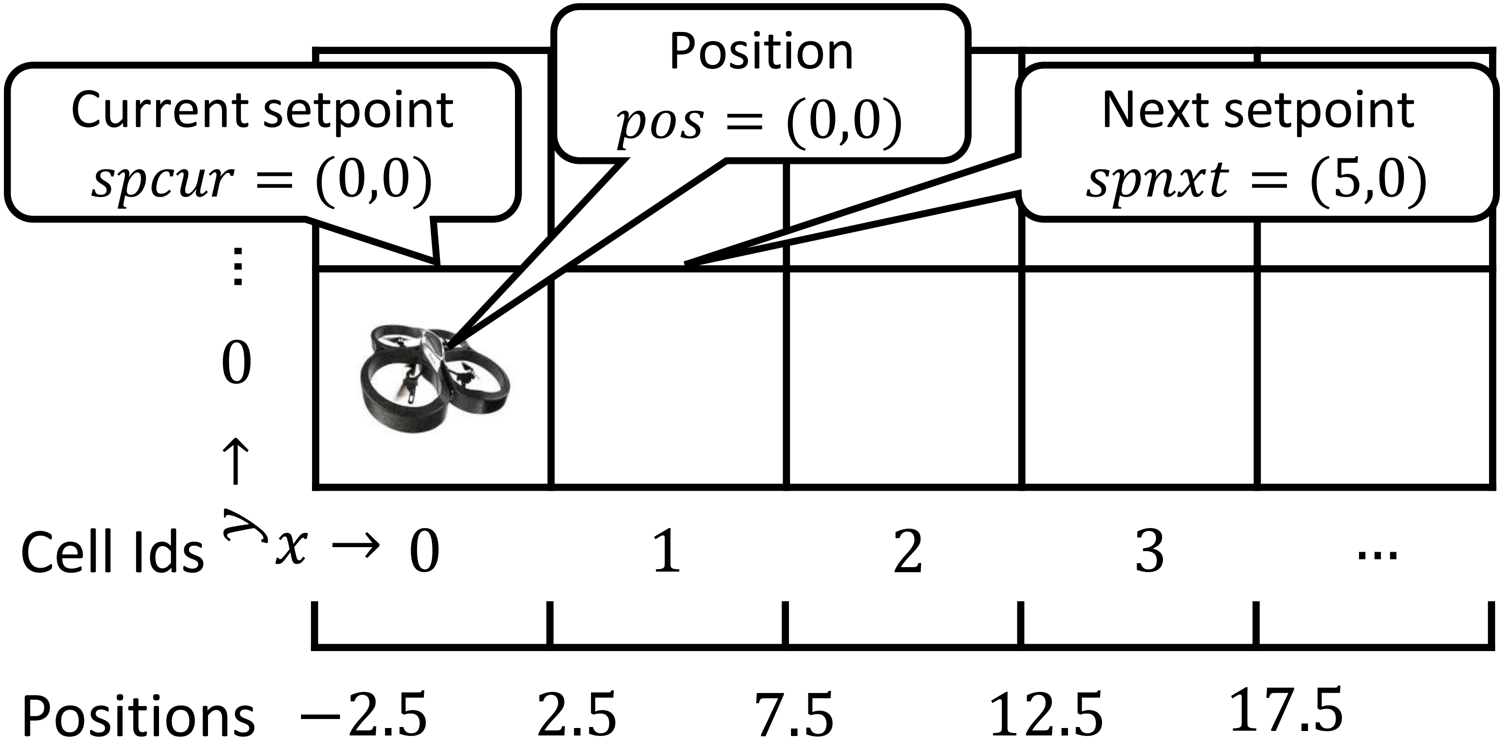
System composed of application A and controller C

- 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 = controller + plant (from control theory perspective)

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

- Formally, $S \vDash \Phi$ where Φ is an invariant expressing the safety property of interest

Example: 2D Quadcopter Movement



Current setpoint
$spcur = (0,0)$

Position
$pos = (0,0)$

Next setpoint
$spnxt = (5,0)$

$\vdots$

0

$y \uparrow$

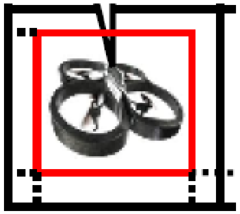
Cell Ids $x \rightarrow$ 0 1 2 3 …

Positions -2.5 2.5 7.5 12.5 17.5

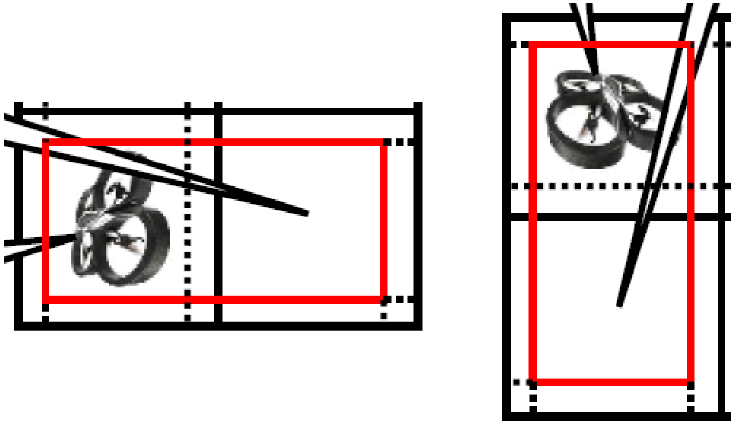
Example: Target Property

$$(\Phi_{hover} \wedge spnxt = spcur) \vee (\Phi_{move} \wedge \\ (|spnxt - spcur| = (5,0) \vee |spnxt - spcur| = (0,5)))$$

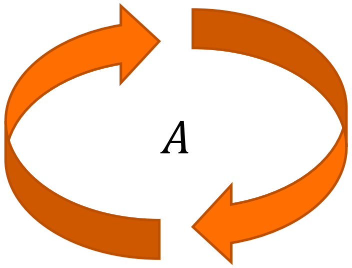
$$\Phi_{hover} \equiv |pos - spcur| \leq (1.5, 1.5)$$



$$\begin{aligned} \Phi_{move} \equiv\ & min(spcur_x, spnxt_x) - 1.5 \leq pos_x \\ & \leq max(spcur_x, spnxt_x) + 1.5 \\ \wedge\ & min(spcur_y, spnxt_y) - 1.5 \leq pos_y \\ & \leq max(spcur_y, spnxt_y) + 1.5 \end{aligned}$$

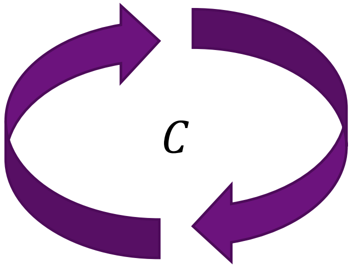


Example: 2D Quadcopter Movement



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

API Function Parameters
x, y



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 $A \parallel C \vDash \Phi$

Benefits of Verification Approach

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

- Explicit formal understanding between teams developing A and C

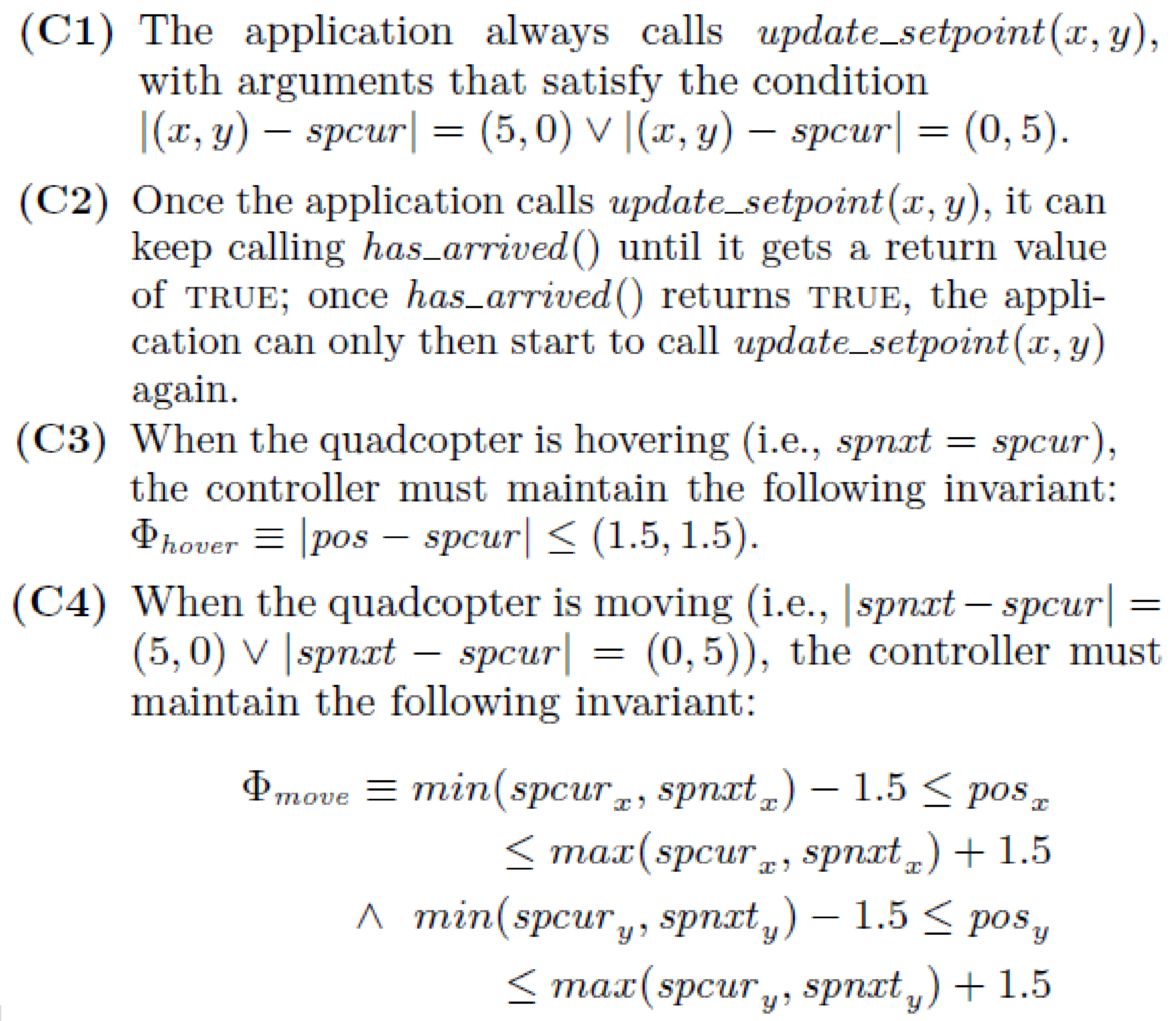
Separately verify that A and C implement desired behavior under the assumption that the other party does so as well

- Compositional ⇒ more scalable
- Use domain-specific tools ⇒ 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 ⇒ amortized proof cost
- Other variants can be developed to manage tradeoff between completeness and verification complexity

Example: Assumptions between A and C



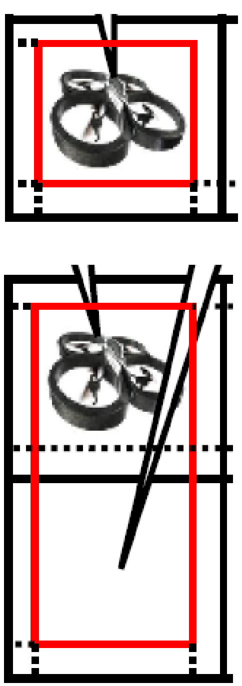
(C1) The application always calls $update\_setpoint(x, y)$, with arguments that satisfy the condition $|(x, y) - spcur| = (5, 0) \vee |(x, y) - spcur| = (0, 5)$.

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

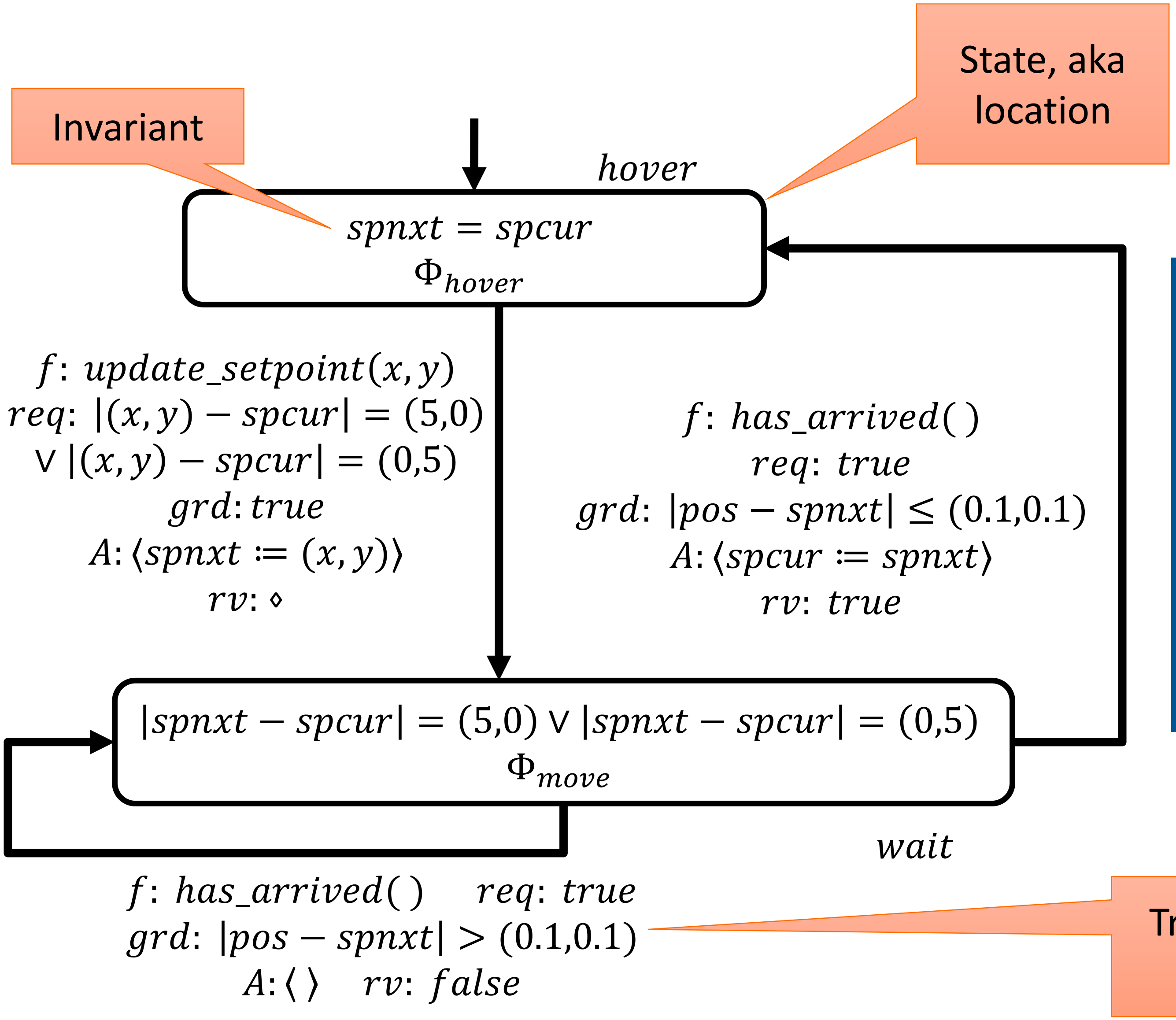
(C3) When the quadcopter is hovering (i.e., $spnxt = spcur$), the controller must maintain the following invariant: $\Phi_{hover} \equiv |pos - spcur| \leq (1.5, 1.5)$.

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

$$\begin{aligned}\Phi_{move} \equiv\ & min(spcur_x, spnxt_x) - 1.5 \leq pos_x \\ & \leq max(spcur_x, spnxt_x) + 1.5 \\ \wedge\ & min(spcur_y, spnxt_y) - 1.5 \leq pos_y \\ & \leq max(spcur_y, spnxt_y) + 1.5\end{aligned}$$



Example: Contract Automaton



Invariant

State, aka location

$hover$

$spnxt = spcur$

Φ_{hover}

$f\colon update\_setpoint(x,y)$
$req\colon |(x,y) - spcur| = (5,0)$
$\vee\ |(x,y) - spcur| = (0,5)$
$grd\colon true$
$A\colon \langle spnxt \coloneqq (x,y)\rangle$
$rv\colon \diamond$

$f\colon has\_arrived(\,)$
$req\colon\ true$
$grd\colon\ |pos - spnxt| \leq (0.1,0.1)$
$A\colon \langle spcur \coloneqq spnxt\rangle$
$rv\colon\ true$

$|spnxt - spcur| = (5,0) \vee |spnxt - spcur| = (0,5)$

Φ_{move}

$wait$

$f\colon\ has\_arrived(\,) \quad req\colon\ true$
$grd\colon\ |pos - spnxt| > (0.1,0.1)$
$A\colon\langle\,\rangle \quad rv\colon\ false$

Transition label

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

Premise1: Application A refines the contract automaton M (calls API functions in the right order and with proper arguments)

Premise2: Controller C refines the contract automaton M (keeps the physical state within required bounds)

THEOREM 1 (C MPOSITIONAL R FINEMENT).

$$\frac{A \preceq M \qquad C \preceq M}{A \parallel C \preceq M}$$

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

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

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 $cellcur$ and $cellnext$ values
    - $\forall i \neq j \in [0,9]. cellcur[i] \neq cellcur[j] \wedge cellcur[i] \neq cellnext[j]$
- Property 2. Setpoints are 5 times cell values
    - $spcur = 5 \times cellcur$ and $spnxt = 5 \times cellnext$
- 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 ⇒ 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?