Specification, Verification and Synthesis of Networked Control Systems

Richard M. Murray
Control & Dynamical Systems
California Institute of Technology

Tichakorn (Nok) Wongpiromsarn
Ministry of Science and Technology
Thailand

Ufuk Topcu
Electrical & System Engineering
U. Pennsylvania

IFAC World Congress
Cape Town, South Africa
29 August 2014

Research support by AFOSR, Boeing, DARPA (FCRP), IBM and United Technologies Corp.

Alice
- 300+ miles of fully autonomous driving
- 8 cameras, 8 LADAR, 2 RADAR
- 12 Core 2 Duo CPUs + Quad Core
- 3 Gb/s data network
- ~75 person team over 18 months (x 2)

Software
- 25 programs with ~200 exec threads
- 237,467 lines of executable code

How should we design systems of this complexity? How do we make sure they function as desired?
Networked control system
- 12 computers, 25 processes, 200+ threads
- 1-3 Gb/s raw data rates from sensors

Application of existing controls technology
- Receding horizon control
- Multi-layer sensor fusion
- Real-time trajectory generation
- PID w/ anti-windup
# Abstractions for Networked Control System Design

Continuous: \[ \dot{x} = f_\alpha(x, u, d) \]

Discrete: \[ g(x, \alpha) \implies \alpha' = r(x, \alpha) \]

\[ \min J = \int_0^T L(x, u, \alpha) dt + V(x(T)) \]

if X then Y, never Z, always W, ...

### Outline for remainder of today’s talk
- Formal specification using temporal logic (LTL, STL)
- “Design then verify”: modeling checking and abstraction
- “Correct-by-construction” synthesis of controllers
- Final thoughts: where we have been, where we might go
(Selected) Prior Results to Build On

Discrete Event Systems

- Ramadge and Wonham, “Supervisory control of a class of discrete event processes”, SIAM J. on Control and Optimization, 1987

Hybrid Systems

- Bemporad, Morari, “Control of systems integrating logic, dynamics, and constraints”, *Automatica*, 1999 [IFAC High Impact ’14]

Model Checking


Reactive Synthesis (of control protocols)

Specifying Discrete Behavior Using Temporal Logic

**Linear temporal logic (LTL)**

- ◊ “eventually” - a property is satisfied at some point in the future
- □ “always” - a property is satisfied now and forever into the future
- ◯ “next” - true at next step

- p → ◊q  p implies eventually q (response)
- p → q U r  p implies q until r (precedence)
- □◊p  always eventually p (progress)
- ◯□p  eventually always p (stability)
- ◯p → ◯q  eventually p implies eventually q (correlation)

**Signal temporal logic (STL)**

- Allow predicates that compare values (via subsets of state space)
- Allow bounds on temporal operators

- V < V_{max}  V(t) less than threshold (V_{max})
- □[t_1,t_2] p  p true for all time in [t_1, t_2]
- p → ◯ [0,t] q  if p occurs, q will occur within time t

---

Example: Traffic Light

Ordering specifications

• Liveness: “traffic light is green infinitely often”
  \[\Box \Diamond \text{green}\]

• Chronological ordering: “once red, the light cannot become green immediately”
  \[\Box (\text{red} \rightarrow \neg \Diamond \text{green})\]

• More detailed: “once red, the light always becomes green eventually after being yellow for some time”
  \[\Box (\text{red} \rightarrow (\Diamond \text{green} \land (\neg \text{green} \lor \Diamond \text{yellow})))\]
  \[\Box (\text{red} \rightarrow \Diamond (\text{red} \lor \Diamond (\text{yellow} \land \Diamond (\text{yellow} \lor \Diamond \text{green}))))\]

Progress property

• Every request will eventually lead to a response
  \[\Box (\text{request} \rightarrow \Diamond \text{response})\]
Discrete Abstractions for (Hybrid) Dynamical Systems

Continuous models to discrete abstractions

\[ \dot{x} = f_\alpha(x, u) \]
\[ g_\alpha(x, u, z) \leq 0 \]

Formal tools available to create abstractions

- Use reachability analysis (trajectory generation) to compute regions, transitions
- Account for disturbances, uncertainty, failures (using, for example, MPT toolbox)

\[ \min J = \int_0^T L_\alpha(x, u) \, dt + V(x(T)) \]
Model Checking: Design and Verify

Approach: enumeration of all possible execution sequences (!)
- Can test systems with up to $10^{11}$ states

$\phi_{\text{init}} \land \Box \phi_{\text{env}} \implies (\Box \phi_{\text{safe}} \land \Box \Diamond \leq T \phi_{\text{live}})$

**Example: Verification of Safety Logic**

**Function:** respond to control commands + DARPA pause/emergency stop

**Verify the following properties**

- \( \square (estop = \text{DISABLE}) \Rightarrow \square (state = \text{DISABLED} \land acc = -1) \)
- \( \square (estop = \text{PAUSE}) \Rightarrow \square (state = \text{PAUSED} \lor estop = \text{DISABLE}) \)
- \( \square (estop = \text{RUN}) \Rightarrow \square (state = \text{RUNNING} \lor state = \text{RESUMING}) \)
- \( \square (state = \text{RESUMING}) \Rightarrow \square (state = \text{RUNNING} \lor estop = \text{DISABLE} \lor estop = \text{PAUSE}) \)
- \( \square (state \in \{\text{DISABLE, PAUSED, RESUMING, SHIFTING}\} \Rightarrow acc = -1) \)

**Verification using temporal logic (Lamport’s TLC, TLA+)**

- Model follower, Actuation Interface, DARPA, accModule, transModule in TLC
- Shared variables: state, estop, acc, acc_command, trans, trans_command
Model Checking for *Hybrid* Systems

\[ \dot{x} = f_\alpha(x, u) \]
\[ g_\alpha(x, u, z) \leq 0 \]

\( \phi_{\text{init}} \land \square \phi_{\text{env}} \) \iff \( (\square \phi_{\text{safe}} \land \square \Diamond \leq T \phi_{\text{live}}) \)
Formal Methods for System Verification & Synthesis

- **Requirements (on the system behavior)**
- **Assumptions (on the unknowns, e.g., environment behavior)**

**Formal Specifications**

**System Model**

**Verification**
- Satisfied (+certificate)
- Violated (+counterexample)

**Synthesis**
- Controller that renders the system to satisfy the spec's
- No such controller exists
“Correct-by-Construction” Controller Synthesis

Reactive Protocol Synthesis

- Find control action that insures that specification is always satisfied
- Complexity is doubly exponential (!) in size of the system specification

GR(1) synthesis for reactive protocols

- Piterman, Pnueli and Sa’ar, 2005
- Assume environment fixes action before controller (breaks symmetry)
- For certain class of specifications, get complexity cubic in # of states (!)

\[(\phi_{\text{init}}^e \land \Box \phi_{\text{safe}}^e \land \Box \Diamond \phi_{\text{prog}}^e) \rightarrow (\phi_{\text{init}}^s \land \Box \phi_{\text{safe}}^s \land \Box \Diamond \phi_{\text{prog}}^s)\]

Environment assumption  System guarantee
Example: Runner Blocker System

Simple two person game
- Runner attempts to reach goal w/out being blocked
- Blocker has limited motion
- Each player must move each turn
- Back out strategy from sequence of winning sets

A. Pnueli, 2005
Example: Autonomous Navigation in Urban Environment

Traffic rules
- No collisions with other vehicles
- Stay in the travel lane unless there is an obstacle blocking the lane
- Only proceed through an intersection when it is clear

Assumptions
- Obstacle may not block a road
- Obstacle is detected before vehicle gets too close
- Limited sensing range
- Obstacle does not disappear when the vehicle is in its vicinity
- Obstacles may not span more than a certain number of consecutive cells in the middle of the road
- Each intersection is clear infinitely often
- Each of the cells marked by star and its adjacent cells are not occupied by an obstacle infinitely often

\[
\begin{align*}
(\phi^e_{\text{init}} \land \Box \phi^e_{\text{safe}} \land \Box \Diamond \phi^e_{\text{prog}}) \\
\rightarrow (\phi^s_{\text{init}} \land \Box \phi^s_{\text{safe}} \land \Box \Diamond \phi^s_{\text{prog}})
\end{align*}
\]
Example: Autonomous Navigation in Urban Environment

Time: 104.30 s

- Solved using receding horizon temporal logic planning
- TuLiP returns 900 state FSA in about 1.5 seconds

Use response mechanism to replan if no feasible solution exists
- Trajectory planner sees blockage and fails to find strategy satisfying specification
- Trajectory planner reports failure to goal generator
- Goal generator re-computes a (high level) path to the goal state
**Temporal Logic Planning (TuLiP) toolbox**

http://tulip-control.org

**Python Toolbox**
- GR(1), LTL specs
- Nonlin dynamics
- Supports discretization via MPT
- Control protocol designed using JTLV
- Receding horizon compatible

**Applications of TuLiP**
- Autonomous vehicles - traffic planner (intersections and roads, with other vehicles)
- Distributed camera networks - cooperating cameras to track people in region
- Electric power transfer - fault-tolerant control of generator + switches + loads
Summary and Future Research

Design of abstraction layers + interfaces

Synthesis of reactive protocols for STL specs

Uncertainty and robustness (for discrete representations)

Many other directions: scaling, incremental, probabilistic, performance metrics, …

Special thanks:
- Nok Wongpiromsarn
- Ufuk Topcu