Lazy
Abstraction-Based Control Design

Rupak Majumdar
Joint work with
Kyle Hsu, Kaushik Mallik, Anne-Kathrin Schmuck

Max Planck Institute for Software Systems
Kaiserslautern, Germany
The Crisis of Software

Our society runs on software

Software mediates and controls interactions with the physical world

Software a major component of physical systems:
Boeing 787 flight software: ~50ECUs, 8M LOC
Large Hadron collider: 50M LOC
High end cars: ~70-100CPUs, ~100M LOC

Energy distribution grids, medical devices, ...
The Crisis of Software

Our society runs on software

Software mediates and controls interactions with the physical world

The Value is in the Software
The Crisis of Software

Our society runs on software

Software mediates and controls interactions with the physical world

The Challenge is in the Software

- Expensive, brittle
- Low productivity, High dev cost

Software is among the most complex artifacts we build routinely. Can we do it better?
Software is Unreliable
Something Reliable

Uptime: 85 years
Why don’t Bridges Crash?

Abstraction

Mathematical Model

Physical Artifact

Build, Test

Analyze, Design

Abstract

Predict

Research Question:
What is a science of rigorous CPS design?
The Ingredients

1. A framework for reasoning about *discrete evolution of state* (Temporal) Logic and Automata Theory

2. A framework for reasoning about *continuous evolution of state* Dynamical systems and Control

A Computational Lens: *Algorithmic Design & Analysis of Systems*
Quick Intro to Control Systems

Design of dynamics by integrating sensing, computation, and actuation via feedback
Principles of Control

*Design goals:* Robustness (system stabilizes under uncertainty) or regulation (system tracks a reference signal)

*Emerging View:* Control as a collection of tools and techniques for designing and implementing complex, interconnected, dynamical systems

1. Dynamics $\rightarrow$ Systems
2. Abstraction and layering
A Reliable CPS Stack

Human-CPS Interaction

1. Programming Environments and Tools for Autonomous CPS
2. Programming Models for Co-ordinating CPS
3. Verification of Controller Implementations
4. Design of Controller Algorithms

Applications

Hierarchical Abstract Control

Temporal Logic Control: LQR/PID Control

You Are Here
A Reliable CPS Stack

Programming Environments and Tools for Autonomous CPS
Programming Models for Co-ordinating CPS
Verification of Controller Implementations
Design of Controller Algorithms

Abstraction-based Control Design
The Controller Synthesis Problem

Controller: \( x(t) \mapsto u(t) \)

Specification: LTL formula over states
Reach-Avoid Controller

Reach-avoid specification: Reach the target but avoid obstacles
Safety Controller

Nominal/ unperturbed dynamics

\[ \frac{dx(t)}{dt} \in f(x(t), u(t)) + W \]

Bounded external perturbation

Controller: \( x(t) \mapsto u(t) \)

Safety specification: Always avoid bad states
Finite State Games

- Finite set of states $S$
- Set of control actions $U$
- Transition Function $F(s, u) \subseteq S$

- Intuition: Two players play on a graph
- At state $s$:
  - Control picks an action $u$ from $U$,
  - Disturbance picks the next state from $F(s,u)$
Finite State Games: One Step

The Controllable Predecessor:

\[ C_{\text{pre}}(T) = \{ s \mid \text{there is a control } u \text{ s.t.} \]
\[ \text{for all } w \in W. \ F(s, u) \subseteq T \}\]

Every state in \( C_{\text{pre}}(T) \) can be forced into \( T \) in one step by the controller
Finite State: Fix Point Algorithms

Reach-Avoid Game with target $T$, obstacle $O$:
\[ \mu X. T \cup (S \setminus O \cap \text{Cpre}(X)) \]

Safety game with Bad states $B$:
\[ \nu Y. S \setminus \text{Bad} \cap \text{Cpre}(Y) \]
Continuous system
\[ \dot{x} \in f(x, u) + W \]

Abstraction

Control input u

Refinement

Discrete controller

Automata-theoretic reactive synthesis

ABCD:
Abstraction-Based Control Design
ABCD: Abstraction-Based Control Design

Continuous system
\[ \dot{x} \in f(x, u) + W \]

Abstraction

Feedback refinement relation

Control input \( u \)

Refinement

Sound closed loop behavior

Feedback reactive synthesis

Automata-theoretic reactive synthesis

Discrete controller
Feedback Refinement Relation

Concrete state $x \rightarrow$ Abstract state $s$

$s = \text{FRR}(x)$ if

1. $x$ and $s$ have the same observations
2. Every control action available at $s$ is also available at $x$
3. For each $u$, the abstraction of the one-step reach set is contained in the abstract one-step reach set:
   \[ \text{FRR}(\text{Step}(x,u,\tau)) \subseteq F(s, u) \]

Ensures: *If there is a controller in the abstraction, there is a controller in the concretization*
Alternating Simulation

Alternating Simulation [AlurHenzingerKupfermanVardi]:
- Generalizes simulation relation to games

Feedback refinement relations are stronger:
Allow refinement of controllers by removing the existential choice of control
Abstraction

Pick a time discretization $\tau$ and a space discretization $\eta$

Partition the state and input spaces into hypercells of size $\eta$

$\rightarrow$ Abstract states

For each control, simulate the dynamics for time $\tau$

$\rightarrow$ Abstract transitions

The growth bound $B()$ can be obtained as the solution of a related dynamical system
Abstraction

Pick a time discretization $\tau$ and a space discretization $\eta$.

Partition the state and input spaces into hypercells of size $\eta$.

$\Rightarrow$ Abstract states

Simulate the dynamics for time $\tau$.

$\Rightarrow$ Abstract transitions

Finite state, two-player game:
Controller picks $u$, plant resolves non-determinism
ABCD: Abstraction-Based Control Design

1. Quality of abstraction depends on $\eta$ and $\tau$
2. Abstraction is exponential in system dimension
3. Can we choose $\eta$ and $\tau$ lazily? On-the-fly?

Continuous system

\[ \dot{x} \in f(x, u) + W \]

Control input $u$

Tabuada, Girard, Pappas, ReissigWeberRungger2017
Motivating Example

Consider a robot in a 2-D workspace and a reachability specification. How should we choose the grid-size?

- Large: less states but small controller domain
- Small: large state space but large controller domain

Can we design a synthesis algorithm which automatically adapts based on the structure of the game graph?
Multi-layered ABCD

Continuous system

\[ \dot{x} \in f(x, u) + W \]
Multi-layered ABCD

Continuous system
\[ \dot{x} \in f(x, u) + W \]

Control input \( u \)

Refinement

Multi-layered Synthesis

\[ \tau_1, \eta_1 \]

\[ \tau_2, \eta_2 \]

\[ \tau_3, \eta_3 \]

\[ c_1, c_2, c_3 \]
Multi-layered ABCD

MASCOT: http://mascot.mpi-sws.org/

*Adaptively* pick a sequence of $\tau$ and $\eta$

- Guided by the exploration
- Non-uniform abstractions

Challenges:

1. Different abstractions are not related to each other: proof requires combining controller domains
2. Abstraction is computed forward, but games are solved backward
Reach-Avoid Controller

Nominal/unperturbed dynamics

Time-derivative of state trajectory

\[ \frac{dx(t)}{dt} \in f(x(t), u(t)) + W \]

Controller: \( x(t) \mapsto u(t) \)

Reach-avoid specification: Reach the target but avoid obstacles
Multi-Layered Reachability

**Input:** Reachability Game
Parameter **L:** Number of layers
\{((τ, η), (2τ, 2η), ... (2^Lτ, 2^Lη))\}

Layer L: Coarsest Layer
Layer 1: Finest Layer

Parameter **m:** Local iteration bound
Multi-Layered Reachability

**REACH:** $W^0 = T$ and $W^{i+1} = \text{CPre}(W^i) \cup T$

- **START**
- Execute **REACH** for $S_L$ and $T$, return $T = T \cup W$
- $l = L - 1$
- Execute **REACH** for $S_l$ and $T$ for $m$ iterations, return $T = T \cup W$
Multi-Layered Reachability

**START**

1. Execute \textbf{REACH} for $S_L$ and $T$, return $T = T \cup W$

2. $l = L - 1$

3. Execute \textbf{REACH} for $S_l$ and $T$ for \textbf{m} iterations, return $T = T \cup W$

4. 
   - **Fixed point reached?**
     - Yes: $l = l - 1$
     - No: $l = 0$?

5. 
   - **Fixed point reached?**
     - Yes: $l = l - 1$
     - No: $l = L - 1$

\[ \textbf{REACH: } W^0 = T \text{ and } W^{i+1} = \text{CPre}(W^i) \cup T \]
Multi-Layered Reachability

**REACH:** $W^0 = T$ and $W^{i+1} = \text{CPre}(W^i) \cup T$

**Algorithm:***

1. **START**
2. Execute **REACH** for $S_L$ and $T$, return $T = T \cup W$
3. $l = L - 1$
4. **l = L?**
   - No: Execute **REACH** for $S_l$ and $T$ for $m$ iterations, return $T = T \cup W$
   - Yes: $l = 0$
5. **l = 0?**
   - No: $l \leftarrow l + 1$
   - Yes: Fixed point reached?
6. **Fixed point reached?**
   - No: $l \leftarrow l - 1$
   - Yes: **STOP**
Multi-Layered Reachability

**REACH:** \( W^0 = T \) and \( W^{i+1} = CPre(W^i) \cup T \)

START

Execute **REACH** for \( S_L \) and \( T \), return \( T = T \cup W \)

\( l = L - 1 \)

\( l = L? \)

No

\( l = 0? \)

No

\( l \leftarrow l + 1 \)

No

Fixed point reached?

Yes

\( l \leftarrow l - 1 \)
Multi-Layered Reachability

\textbf{REACH: } \( W^0 = T \) and \( W^{i+1} = \text{CPre}(W^i) \cup T \)
Multi-Layered Reachability

**REACH:** \(W^0 = T\) and \(W^{i+1} = \text{CPre}(W^i) \cup T\)

- Start
  - Execute **REACH** for \(S_L\) and \(T\), return \(T = T \cup W\)
  - \(l = L - 1\)
  - \(l = L\)? (No) \(l = l + 1\)
  - l = L? (No) \(l = 0\)? (No) Fixed point reached? (Yes) \(l = l - 1\)
  - Execute **REACH** for \(S_l\) and \(T\) for \(m\) iterations, return \(T = T \cup W\)

Robot Example (\(m = 2\))
ML\textsubscript{REACH}\textsubscript{m}:

1. Execute \texttt{REACH} for $S_L$ and $T$, return $T = T \cup W$

2. If $l = L$, STOP

3. If $l = 0$, return $T = T \cup W$

4. Execute \texttt{REACH} for $S_i$ and $T$ for $m$ iterations, return $T = T \cup W$

5. $l = l - 1$

6. $l = l + 1$

\texttt{REACH}: $W^0 = T$ and $W^{i+1} = \text{CPre}(W^i) \cup T$
MASCOT: http://mascot.mpi-sws.org/
Lazy Computation of Transitions
Safety Control

Controller: $x(t) \mapsto u(t)$

Safety specification: Always avoid bad states

Time-derivative of state trajectory
Nominal/unperturbed dynamics
Bounded external perturbation

$$\frac{dx(t)}{dt} \in f(x(t), u(t)) + W$$
Coarse vs Fine in Safety Games

Coarser Partition
↓
Smaller Abstraction
↓
Faster Computation

Finer Partition
↓
More Accurate Approximation
↓
Larger Controller Domain
Single-Layered Synthesis for Safety

Mark all safe states

Unmark the uncontrollable states

Stop when converged

ν Y. Good ∩ Cpre(Y)
Attempt 1: Multi-Layered Safety

\[ v \ Y. \text{ Good} \cap \text{Cpre}(Y) \cup \text{CoarseSafe} \]

Nilsson et al.
Hsu M. Mallik Schmuck
Multi-Layered Safety

Does not allow “coming back”
Lazy Multi-layered Algorithm for Safety

1. Mark all safe states
2. Pass uncontrollable states to next layer
3. Unmark remaining uncontrollable states
4. Stop when converged

Unmark the uncontrollable states

Bad states
A DC-DC Boost Converter

\[ \frac{d}{dt} \begin{bmatrix} i_l(t) \\ v_c(t) \end{bmatrix} \in A_p \begin{bmatrix} i_l(t) \\ v_c(t) \end{bmatrix} + b + W \]

\( p \in \{1,2\} \) is controlled by the switch

Experimental Results: DC-DC Boost Converter

<table>
<thead>
<tr>
<th>No. of layers</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>375</td>
</tr>
<tr>
<td>3</td>
<td>250</td>
</tr>
<tr>
<td>4</td>
<td>125</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
</tbody>
</table>

- **Abstraction time**
  - Single layered: 114%
  - Synthesis time: 22%

- **Synthesis time**
  - No. of layers: 1, 2, 3, 4, 5, 6

- **Abstraction time**
  - No. of layers: 1, 2, 3, 4, 5, 6
A DC-DC Converter Example
What Next?

Abstract view of control: Action Primitives

Compact representation of dynamic controllers (Precondition, Effect, Postcondition)

Design of Controller Algorithms

Abstraction-based Control Design

Hierarchical Abstract Control

Applications

Programming Models for Co-ordinating CPS

Verification of Controller Implementations

Programming Environments and Tools for Autonomous CPS

Applications

Applications
A Reliable CPS Stack

Human-CPS Interaction

Programming Environments and Tools for Autonomous CPS

Programming Models for Co-ordinating CPS

Verification of Controller Implementations

Design of Controller Algorithms

Hierarchical Abstract Control

Temporal Logic Control

Turtlebot

LQR/PID Control

Applications

Systems Services

You Are Here

Applications
Concurrency & Control

System = Concurrent Execution of Components

- Each component can run an action primitive
- Components can synchronize with each other
Example: Handover
Example: Handover (Meet)
Example: Handover (Grab)
Example: Handover (Fold)
Example: Handover (Done)
How Many Robots?
How Many Robots?

Cart & Arm: Two robots attached together that act as “one” robot (communication)
How Many Robots?

Arm

Cart

Carrier

World
Current Direction: Program Logics for Concurrency + Control

M.PirronZufferey19,M.PirronYoshidaZufferey19

Formally proving safety properties
(e.g., Collision freedom)

Compositional approach using session types
while true do
  receive (Cart, idle)
  grab(loc) ⇒
  grip(loc); send(cart, ok)
  fold ⇒
  move(origin); send(cart, ok)
  done ⇒
  break

Motion primitive
Communication
Handover: Global Specification

Verification = Session Type Checking + Additional checks on action primitives
Human-CPS Interaction

- Programming Environments and Tools for Autonomous CPSs
- Programming Models for Co-ordinating CPSs
- Verification of Controller Implementations
- Computational fabrication
- Models of Concurrency with dynamics and geometry, Program Logics
- Implementation, verification, and testing of controllers
- Supervisory control theory & reactive synthesis
Thank You

http://www.mpi-sws.org/~rupak
http://mascot.mpi-sws.org

CPEC: Center for Perspicuous Computing
http://perspicuous-computing.science
(A Collaborative Research Center between Saarland University, TU Dresden, and the Max Planck Institutes)
1. Phil Rogaway. The Moral Character of Cryptographic Work. 2015

2. Stuart Russell. Artificial Intelligence: Implications for Autonomous Weapons
http://www.cs.berkeley.edu/~russell/talks/russell-ccw15-autonomy.pptx
See also:
http://futureoflife.org/Al/open_letter_autonomous_weapons
What about Machine Learning?

- Learning for perception
  - No alternatives at this point!
- Learning for control
  - Not an either-or choice

Research challenge I: Develop interfaces between them to allow end-to-end reasoning for safety/correctness

Research challenge II: Explain ML!
(DFG TRR Saarland, Dresden, MPI: up to 12 yrs)