Principles of Robot Autonomy II

Specifications, Model Checking, and Reactive Synthesis
Today’s itinerary

• Safety from the lens of formal methods

• Model checking & Reactive Synthesis

• Linear Temporal Logic

• Reactive synthesis from Linear Temporal Logic

• Limitations
Today’s itinerary

• Safety from the lens of formal methods

• Model checking & Reactive Synthesis

• Linear Temporal Logic

• Reactive synthesis from Linear Temporal Logic

• Limitations
What does safety mean for Robotics?
Mathematically based technique for specification, development, and verification of software and hardware systems.

**Verification**: Timing and Energy analysis  
**Software**: Embedded code

Timing Analysis of Interrupt-Driven Programs under Context Bounds  
Jonathan Kotker, Dorsa Sadigh, Sanjit A. Seshia  
*Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD), October 2011*
Formal Methods

Mathematically based technique for specification, development, and verification of software and hardware systems.

Verification (model checking):

- Specification
- Model
- Verifier

Model satisfies specification?
Yes / No

Robots?
What does safety mean for robots?

Formal methods *(verification)* + Control Theory, Robotics, Machine Learning

Can we prove that autonomous cars are *safe*?
Church’s Problem

• Alonzo Church at the “Summer Institute of Symbolic Logic”, Cornell University, 1957:

“Given a requirement which a circuit is to satisfy, we may suppose the requirement expressed in some suitable logistic system which is an extension of restricted recursive arithmetic. The synthesis problem is then to find recursion equivalences representing a circuit that satisfies the given requirement (or alternatively, to determine that there is no such circuit).”
Formal Methods

Mathematically based technique for specification, development, and verification of software and hardware systems.

Verification (model checking):
Formal Methods

Mathematically based technique for specification, development, and verification of software and hardware systems.

Reactive Synthesis (Church 1957):
Formal Methods

Mathematically based technique for specification, development, and verification of software and hardware systems.

Reactive Synthesis (Church 1957):

Specification → Reactive Synthesis → Model
What does safety mean for robots?

Formal methods (verification) + Control Theory, Robotics, Machine Learning

Can we *synthesize* controller for autonomous cars are provably *safe*?
Reactive Synthesis

- Automatically constructing correct systems from specifications from scratch.

**Specification:** Temporal behavior specified over input and output sequences

**Reactive System:**
- Every step it reads inputs and produces outputs synchronously
- Non-terminating behavior
Reactive Synthesis

Determine

• If there is a finite state system that realizes the specification (realizability).

• If a specification is realizable, how do you construct the system?
Reactive Synthesis

Church 1957:
• Introduced the reactive synthesis problem

Buchi + Landweber 1969:
• Realizability is decidable.
• If a winning strategy exists, then a finite state winning strategy exists.
• Introduced a game view of the problem.

Rabin 1972:
• Simpler solution to the game with Rabin Tree Automaton.

Pnueli + Rosner 1989:
• Linear Temporal Logic Synthesis
• GR1 Synthesis (Piterman, Pnueli, Sa’ar 2006)
Today’s itinerary

• Safety from the lens of formal methods

• Model checking & Reactive Synthesis

• Linear Temporal Logic

• Reactive synthesis from Linear Temporal Logic

• Limitations
Linear Temporal Logic

\[ \varphi ::= \text{true} \mid a \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid X \varphi \mid \phi_1 U \varphi_2 \]

**a:** atomic proposition

**X:** \( \varphi \) is true at next step.

**U:** \( \varphi_1 \) is true until \( \varphi_2 \) at some point becomes true.

Atomic proposition:  \( a \)

Next step:  \( Xa \)

Until:  \( a U b \)
Additional Operators

\[ F \varphi := \text{true} U \varphi \quad \text{\( \varphi \) will become true at some point in the future.} \]

\[ G \varphi := \neg F \neg \varphi \quad \text{\( \varphi \) is always true.} \]

Eventually/Future: \( F \ a \)

\hspace{1cm} \begin{array}{cccccc}
\neg a & \rightarrow & \neg a & \rightarrow & \neg a & \rightarrow & a & \rightarrow & \ldots \\
\end{array}

Always/Globally: \( G \ a \)

\hspace{1cm} \begin{array}{cccccc}
a & \rightarrow & a & \rightarrow & a & \rightarrow & a & \rightarrow & a & \rightarrow & \ldots \\
\end{array}

Other combinations:

- \( GF \ a \): infinitely often
- \( FG \ a \): notion of stability
- \( G(a \rightarrow F \ b) \): response
Linear Temporal Logic

‘Temporal’ refers to the underlying nature of time.

- **Linear temporal logic (LTL):** each moment in time has a well-defined successor moment.
- **Branching temporal logic:** reason about multiple possible time courses (parallelism), e.g., computation tree logic (CTL).
Today’s itinerary

- Safety from the lens of formal methods
- Model checking & Reactive Synthesis
- Linear Temporal Logic
- Reactive synthesis from Linear Temporal Logic
- Limitations
Reactive Synthesis from Linear Temporal Logic

$I$ : input variables,  $O$ : output variables

**Game:**
System chooses from $2^O$.
Environment chooses from $2^I$.

**Infinite Play:**
$i_0, i_1, i_2, ...$
$o_0, o_1, o_2, ...$

**Infinite Behavior:** $i_0 \cup o_0, i_1 \cup o_1, i_2 \cup o_2, ...$

**Win:** Behavior $\models$ spec

**Specifications:** LTL formula on $I \cup O$

**Strategy:** function $f : (2^I)^* \rightarrow 2^O$
The Coffee Machine Problem:

“If the start button is pressed, then the system will immediately start grinding for the next two cycles and, then brewing for the next two cycles after that, coffee will be produced.”

The specification is unrealizable, because the environment can produce input that makes it impossible to satisfy both requirements at the same time.
Synthesizing Systems

1. **Specification**
   - Logical formulas + Partitioning

2. **Construct Game**

3. **Solve Game**
   - Finding Controller

4. **Correct System**

5. **System with Freedom**

   - Plant
$G(r \rightarrow Xg)$

Game

Strategy / Mealy machine

Word automaton

Specification
Controller Synthesis from Logic Specifications

Given a formal *specification*, encoding the
- objective,
- environment model,

*synthesize* a controller that is guaranteed to satisfy
the specification.
Controller Synthesis from Logic Specifications

Temporal Logic Specifications:

- At all times (something is true)
- In the future (something is true)
- safety
- some version of stability
- surveillance (infinitely often)
- fairness
- response (if then else)
Controller Synthesis from Logic Specifications

Goal
Environment Model

Controller Synthesis

Temporal Logic Specifications
Controller Synthesis from Logic Specifications

Goal
Environment Model

Controller Synthesis

Temporal Logic Specifications

Non-deterministic Buchi Automaton

ExpTime
Controller Synthesis from Logic Specifications

Goal

Environment Model

Controller Synthesis

Temporal Logic Specifications

Determinize it

Non-deterministic Buchi Automaton

ExpTime
Controller Synthesis from Logic Specifications

Goal Environment Model → Controller Synthesis

Temporal Logic Specifications ExpTime

Determine it ExpTime

Non-deterministic Buchi Automaton

2-player game: system (what we control) & environment (all the thing that can go wrong)
Controller Synthesis from Logic Specifications

Goal Environment Model → Controller Synthesis

Temporal Logic Specifications

Determinize it

Non-deterministic Buchi Automaton

ExpTime

2-player game: system (what we control) & environment (all the thing that can go wrong)

Polynomial Time

ExpTime

ExpTime
Controller Synthesis from Logic Specifications

Goal → Environment Model → Controller Synthesis

Temporal Logic Specifications → ExpTime
Determinize it → ExpTime

Non-deterministic Buchi Automaton → ExpTime
2-player game: system (what we control) & environment (all the thing that can go wrong)

Provably correct strategy → Polynomial Time
Counter-strategy
Given a formal *specification*, encoding the
• objective,
• environment model,
*synthesize* a controller that is guaranteed to satisfy
the specification.
Controller Synthesis from Logic Specifications

Given a formal specification, encoding the
- objective,
- environment model,
- human model

synthesize a controller that is guaranteed to satisfy the specification.
Temporal Logic Spec.

- Goal
- Environment Model
- Human Model
Unprotected Left Turn

Temporal Logic Spec.

- Goal
- Environment Model
- Human Model

\[ \psi^{new} := (\varphi \land \psi^{env}) \implies \psi^{sys} \]

Mine Assumptions

Controller Synthesis

Counterstrategy Graph

\[ \varphi \]

Assumption Monitoring

\[ auto = \text{false} \]

Realizable

Unrealizable

Year: 2014
Today’s itinerary

• Safety from the lens of formal methods

• Model checking & Reactive Synthesis

• Linear Temporal Logic

• Reactive synthesis from Linear Temporal Logic

• Limitations
Controller Synthesis from Logic Specifications

Temporal Logic Specifications:

1) Need discrete time and *discrete space*. 

Goal Environment Model  \rightarrow  Controller Synthesis
Controller Synthesis from Logic Specifications

Temporal Logic *Specifications:*

1) Need discrete time and *discrete space.*

**Solution:**
- Use Signal Temporal Logic Instead!
- Translate to Mixed-Integer Optimizations
Controller Synthesis from Logic Specifications

Temporal Logic *Specifications*:

1) Need discrete time and *discrete space*.
2) Our human models were totally made-up!
Standard Critique

• **Impractical:**
  
  2EXPTIME is a **horrible** complexity!

• **Response:**
  
  2EXPTIME is just worst-case complexity.
Real Critique

- Algorithms not ready for practical implementation.

- Complete Specification- unrealistic.

- Construction from scratch- unrealistic.
Response

• Better algorithms.

• Incremental synthesis - with specification added incrementally.

• Compositional synthesis – synthesis from components.
The Temporal Logic Planning (TuLiP) toolbox

The Temporal Logic Planning (TuLiP) tool embedded control software.

1. Download the current release.
2. To install it, try

   $ python setup.py install

   Detailed instructions are available.
3. Explore the demonstrations under example.

source repository:

---

Robotics Synthesis Tools: TuLiP (Caltech), LTLMoP (Cornell), RATSY, Lily (TU Graz), etc.
Program Synthesis Tools: Sketch (MIT), Rosette (UW)
Translation Tools: LTL2BA, Rabinizer (TU Munich)
Today’s itinerary

• Safety from the lens of formal methods
• Model checking & Reactive Synthesis
• Linear Temporal Logic
• Reactive synthesis from Linear Temporal Logic
• Limitations
Lots of stuff left out

• Metric temporal logic
• Combining optimization with reactive synthesis
• Probabilistic specifications
• Connection to task and motion planning
Next time

• More in depth discussion of model checking
• More in depth discussion of temporal logic
• Verification of Neural Networks