A Tutorial on Runtime Verification and Assurance

Ankush Desai
EECS 219C
Outline

1. Background on Runtime Verification

2. Challenges in Programming Robotics System (Drona).

3. Solution 1: Combining Model Checking and Runtime Verification.

Background

Formal Verification (e.g., Model checking):
• Formal, sound, provides guarantees.
• Doesn’t scale well - state explosion problem.
• Checks a model, not an implementation.
• Most people avoid it - too much effort.

Testing (ad-hoc checking):
• Most widely used technique in the industry.
• Scales well, usually inexpensive.
• Test an implementation directly.
• Informal, doesn’t provide guarantees.
Runtime Verification

Attempt to bridge the gap between formal methods and ad-hoc testing.

• A program is monitored while it is running and checked against properties of interest.
• Properties are specified in a formal notation (LTL, RegEx, etc.).
• Dealing only with finite traces.

Considered as a light-weight formal method technique.

• Testing with formal “flavour”.
• Still doesn’t provide full guarantees.
How to monitor a program?

- Need to extract events from the program while it is running.
- Code instrumentation.

---

**Runtime Verification, cont’d**

![Diagram of runtime verification process]

1. **Static phase**
   - **Specifications**

2. **Dynamic phase**
   - **Runtime Checker**
   - **Stream of events**
   - **Code Instrumentation**
   - **Instrumented Program**
Still, What is Runtime Verification?

There are three interpretations of what runtime verification is, in contrast with formal verification discussed in this course.

1. RV as lightweight verification, non-exhaustive simulation (testing) plus formal specifications
2. RV as getting closer to implementation, away from abstract models.
3. RV as checking systems after deployment while they are up and running.
RV as Lightweight Formal Methods

- Verification is glorious and romantic but practically hard beyond certain complexity.
- Simulation/testing is here to stay with or without attempts to guarantee some coverage.
- So let us add to this practice some formal properties and property monitors that check the simulation traces.
- Instead of language inclusion $L_s \subseteq L_\varphi$ as in verification, we check membership $w \in L_\varphi$, one trace at a time.
- Monitoring is less sensitive to system complexity. It does not require a mathematical model of the system, a program or a black box is sufficient.
- In fact, it does not care who generates the simulation traces, it could be measurements of a real physical process.
Main Challenge: Efficient monitoring

1. Low instrumentation + communication overhead.

2. An efficient monitor should have the following properties:
   • No backtracking.
   • Memory-less: doesn’t store the trace.
   • Space efficiency.
   • Runtime efficiency.
     • A monitor that runs in time exponential in the size of the trace is unacceptable.
     • A monitor that runs in time exponential in the size of the formula is usable, but should be avoided.
PROGRAMMING SAFE ROBOTICS SYSTEMS
A major challenge in autonomous mobile robotics is programming robots with *formal guarantees* and *high assurance* of correct operation.
“If you are able to verify a robotics system then most likely something is wrong”

-Russ Tedrake, MIT
Surveillance Application

Workspace in Gazebo Simulator

Obstacle Map and Drone Trajectory
Robotics Software Stack

- Surveillance Protocol
  - e.g., reached target location
  - e.g., next target location

- Motion Planner
  - OMPL
  - Solvers
  - ..
  - e.g., next waypoint

- Motion Primitives
  - Untrusted Controllers
  - ..

- State Estimators (Multi-Sensor Data Fusion)
Robotics Software for such an autonomous drone is highly complex, reactive and concurrent.

Modes of Operation of the Surveillance Drone
Our Approach

1. Programming Framework for Reactive Systems:
   - P Programming Language.

2. Scalable Analysis of Robotics Software using Model Checking.
   - Using discrete abstractions of the robot behavior.

3. Use Runtime Monitoring to ensure that the assumption hold.
Motion Planner

- **Motion Primitive: goto(location)**
Motion Planner

• Verify that the plans generated by the motion planner are always \( \epsilon \) distance away from all obstacles.

• We used constraint solver (and RRTStar) to implement the motion planner.
Signal Temporal Logic (STL)

Syntax
- **Real-time and real-valued** temporal logic formulas

Semantic
- Qualitative (Boolean): Is the formula True or False?
- Quantitative (Real): How **robustly** is the formula True or False?

\[
\begin{align*}
\rho^\mu(x, t) & = f(x_1[t], \ldots, x_n[t]) \\
\rho^{\neg\varphi}(x, t) & = -\rho^\varphi(x, t) \\
\rho^{\varphi_1 \land \varphi_2}(x, t) & = \min(\rho^{\varphi_1}(x, t), \rho^{\varphi_2}(w, t)) \\
\rho^{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}(x, t) & = \sup_{\tau \in t+[a,b]} \min(\rho^{\varphi_2}(x, \tau), \inf_{s \in [t,\tau]} \rho^{\varphi_1}(x, s))
\end{align*}
\]
Quantitative Semantics

Example

- STL requirement: Avoid an obstacle
- \( f := G_{[0,5]}( x < 10 ) \)
- Both the paths satisfy the requirement
- But \( p1 \) more robustly than \( p2 \)

\[
\begin{align*}
B(f,p1) &= T \\
R(f,p1) &= 5 \\
B(f,p2) &= T \\
R(f,p2) &= 1 
\end{align*}
\]
Assumptions as STL Formulas

\[ \text{goto}(q_g, t, \epsilon) := \text{tube}((q_i, q_g), \epsilon) \cup_{[0,t]} \text{close}(q_g, \epsilon) \]
Assumptions as STL Formulas

\[
\text{traj}(\xi, t, \epsilon) := \begin{cases} 
\text{tube}(q_{g1}, q_{g2}, \epsilon_1) \cup [0, t_1] \land \text{close}(q_{g2}, \epsilon_1) & \text{if } n = 2 \\
\text{tube}(q_{g1}, q_{g2}, \epsilon_1) \cup [0, t_1] \land (\text{close}(q_{g2}, \epsilon_1) \land \text{traj}(\xi', t', \epsilon')) & \text{otherwise}
\end{cases}
\]

where \( \xi' = (q_{g2}, \ldots, q_{g_n}) \), \( t' = (t_2, \ldots, t_{n-1}) \), and \( \epsilon' = (\epsilon_2, \ldots, \epsilon_{n-1}) \).
Parameter Learning

Question: What value of $\epsilon$, $t$ to use?

Learn functions: $f_t, f_\epsilon : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

duration $t = f_t(q_i, q_g)$ and overshoot $\epsilon = f_\epsilon(q_i, q_g)$
Validating Low-Level Controllers
Online STL Monitoring

For each of the assumption about the low-level components:

1. An STL formula is generated corresponding.

2. An online monitor is created dynamically to monitor the STL specification and take preemptive action based on the robustness value.
Drona Tool
Results: Reference Trajectory
Results: Obstacle Avoidance

\[ \varphi_{obs} := \bigwedge_{j=1}^{n} \neg F_{[0,120]}(d(q, obs_j) < 0.5) \]
Demo Video
RUNTIME ASSURANCE FOR SAFE ROBOTICS
Robotics Software Stack

(1) Obstacle Avoidance ($\phi_{\text{obs}}$): The drone must never collide with any obstacle.

(2) Battery Safety ($\phi_{\text{bat}}$): The drone must never crash because of low battery, instead, when the battery is low it must land safely.
Simplex Architecture

Decision Module (DM) (sampling period Δ)

Advanced Controller (AC)

Safe Controller (SC)

Plant or Robot

Switch control if $\phi$ can be violated in Δ time.

RTA module to ensure property $\phi$

Sensor data (state of the system)
A RTA Module

A RTA Module is a tuple \((Q_{ac}, Q_{sc}, \phi_{safe}, \phi_{safer}, \Delta)\)

- \(Q_{ac}\) is the Advanced controller.
- \(Q_{sc}\) is the Safe controller.
- \(\phi_{safer} \subset \phi_{safe}\) is a set of states.
- \(\Delta\) is the sampling rate of the DM.

```plaintext
1 if (mode=SC \land s_t \in \phi_{safer}) mode = AC /*switch control to AC*/
2 elseif (mode=AC \land \text{Reach}(s_t,*,2*\Delta) \not\subseteq \phi_{safe}) mode = SC /*switch control to SC*/
3 else mode = mode /* No mode switch */
```
A RTA machine is well-formed

A RTA Machine \((Q_{ac}, Q_{sc}, \phi_{safe}, \phi_{safer}, \Delta)\) is well-formed:

- Outputs of \(Q_{ac}\) and \(Q_{sc}\) are the same.
- \(Q_{ac}\) and \(Q_{sc}\) have same period (\(<= \Delta\)).
- The \(Q_{sc}\) satisfies the following properties:
  1. \(\text{Reach}(\phi_{safe}, Q_{sc}, \ast) \subseteq \phi_{safe}\)
  2. \(\forall s \in \phi_{safe}, \exists s', T \text{ s.t. } s' \in \text{Reach}(s, Q_{sc}, T)\)  
     and \(\text{Reach}(s', Q_{sc}, \Delta) \subseteq \phi_{safer}\)

1. \(\text{Reach}(\phi_{safer}, \ast, 2\Delta) \subseteq \phi_{safe}\). Note that this condition is stronger.
Definition 4.1 (Regions). Let $R(\phi, t) = \{ s \mid s \in \phi \land \text{Reach}_{M}(s, *, t) \subseteq \phi \}$. For example, $R(\phi_{\text{safe}}, \Delta)$ represents the region or set of states in $\phi_{\text{safe}}$ from which all reachable states in time $\Delta$ are still in $\phi_{\text{safe}}$. 

R1 (Unsafe Region): 
$\neg \phi_{\text{safe}}$

R2: $\phi_{\text{safe}}$

R3: $R(\phi_{\text{safe}}, \Delta)$

R4: $R(\phi_{\text{safe}}, 2\Delta)$

R5: $R(\phi_{\text{safe}}, \Delta)$

Switch to AC

Switching Control Region

Switch to SC
Theorem 4.1 (Runtime Assurance). For a well-formed RTA module $M$, let $\phi_{\text{Inv}}(\text{mode}, s)$ denote the predicate $(\text{mode}=\text{SC} \land s \in \phi_{\text{safety}}) \lor (\text{mode}=\text{AC} \land \text{Reach}_M(s, *, \Delta) \subseteq \phi_{\text{safety}})$. If the initial state satisfies the invariant $\phi_{\text{Inv}}$, then every state $s_t$ reachable from $s$ will also satisfy the invariant $\phi_{\text{Inv}}$. 
Theorem 4.2 (Compositional RTA System). Let \( S = \{M_0, \ldots M_n\} \) be an RTA system. If for all \( i, M_i \) is a well-formed RTA module satisfying the safety invariant \( \phi_{\text{inv}}^i \) (Theorem 4.1) then, the RTA system \( S \) satisfies the invariant \( \bigwedge_i \phi_{\text{inv}}^i \).
Untrusted Motion Primitive
RTA Protected Motion Primitive

Only AC = 10 secs
RTA (AC + SC) = 14 secs
Only SC = 23 secs
Demo