IOWA STATE UNIVERSITY
Department of Mathematics
Department of Aerospace Eng.

Using the FORCE with R2U2: Adapting Robust Intervals to MLTL

Christopher Barbosa [1] Jonas Carew [2] Jason Hao [3] Gustavo Luzzi [4]
[1]City College of New York [2]Oregon State University [3]The Cooper Union [4]Iowa State University

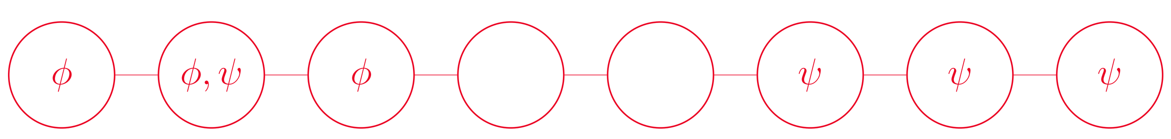
Abstract

Formal verification of complex autonomous systems is difficult; runtime verification is an effective method for monitoring specified system behavior and bridging the gap between a system's model and its real-life counterpart. Robustness in Signal Temporal Logic (STL) is one metric monitored by some runtime verification frameworks, but STL is notoriously difficult to implement. The Realializable Responsive Unobtrusive Unit (R2U2) runtime monitor is a Mission-time Linear Temporal Logic MLTL) based flight-certified framework that provides the capabilities necessary for safe online monitoring of specifications, but robustness is not supported in MLTL. We present the new semantics and proofs required for implementing robustness in R2U2. We modify robust satisfaction intervals (RoSIs) for implementation in a stream-based runtime monitor using the new Fast Online Robustness Calculation Engine (FORCE) for R2U2. Finally, we compare our results with the existing tools.

Overview of MLTL

Mission-time Linear Temporal Logic, or MLTL, is a flight-proven way to verify autonomous systems while they're running. MLTL extends standard boolean logic to also have a time component and introduces several more operators such as globally $\Box\phi$, future $\Diamond\phi$, until $\phi\mathcal{U}\psi$, and next $\mathcal{X}\phi$. These allow us to reason about *when* a formula is true or false, creating an error-free way to validate that systems are running correctly when more complex reasoning is needed.

We do this reasoning using *traces*(π), which are finite sequences of time steps that determine which propositions are true or false at a given time.



For an MLTL formula ϕ, we write $\pi, t \models \phi$ to denote that the trace π accepts our formula at that time. For formula ϕ we define the semantics:

- $\phi := \top \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1\mathcal{U}_{[a,b]}\phi_2$
  $p \in \mathcal{AP} :=$ the simplest propositions that we work with
- $\pi_t \models \top$
- $\pi_t \models p$ iff $p \in \pi[t]$
- $\pi_t \models \neg\phi_1$ iff $\pi_t \not\models \phi_1$
- $\pi_t \models \phi_1 \wedge \phi_2$ iff $\pi_t \models \phi_1$ and $\pi_t \models \phi_2$
- $\pi_t \models \phi_1\mathcal{U}_{[a,b]}\phi_2$ iff $\exists t' \in [t+a, t+b], \pi_{t'} \models \phi_2$ and $\forall t'' \in [t+a, t'), \pi_{t''} \models \phi_1$

If $\mathcal{AP}$ is all of our atomic propositions and a boolean signal is a map $\mathbb{N} \to \mathbb{B}$, we can also view a trace as an assignment of a boolean signal to each atomic proposition, i.e., a map $\mathcal{AP} \to \mathbb{B}^{\mathbb{N}}$.

Overview of Robustness

Motivation:
For safety critical temporal logic based systems, it becomes important to not just know if we are satisfying the logical specification but by how much we satisfy it, or how far are we from satisfying it. Robustness gives a clear metric that measures "how true" or "how false" our system is.
How do we use Robustness?

- From STL, we take signals as our atomic propositions in the form $\sigma_p(t) \geq 0$
- Robustness is then defined as distance from "ideal" signal conditions
- ISSUE: Robustness requires completed traces
- So, for real-time systems we use an estimation, Robust Satisfaction Intervals (RoSI) denoted $[\rho]$

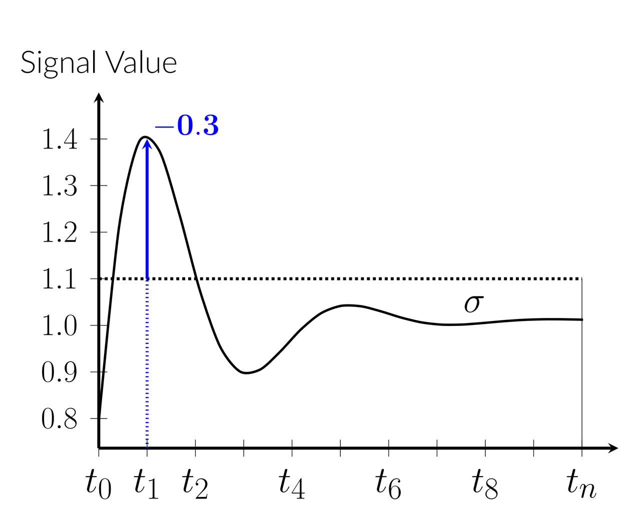


Figure 1. False robustness of signal for the formula $\varphi \equiv \Box(f(t) \leq 1.1)$

What do we want from Robustness?

We want the $\rho(\Pi, \phi, t)$ value to not only tell us the truth-ness of ϕ but also *how* true it is.

New Notation and Consistency Result

Let T denote the set of times for our mission. We define the signal map $\Pi : \mathcal{AP} \to \mathbb{R}^T$ which defines for every atomic proposition a signal over the time set of the mission.

Example: for $p \in \mathcal{AP}$ at some $t \in T$, we define $[\Pi(p)](t) = \sigma_p(t)$, This formulation goes hand in hand with the usual notion of a trace:

we then define the induced trace π by the following map

$$\forall t \in T \quad \pi(t) := \{p \in \mathcal{AP} \mid \sigma_p(t) \geq 0\}$$

Recall that robustness gives us a measurement of how well our system is doing, so we want there to be some sort of result that correlates the exact robustness value with the truth value of the formula:

Consistency: Let $\Pi : \mathcal{AP} \to \mathbb{R}^{\mathbb{N}}$ be an arbitrary choice of signals and π the induced trace. For a given MLTL specification ϕ and time t, we have the following properties:

$$\rho(\Pi, \phi, t) > 0 \implies \pi_t \models \phi \qquad \pi_t \models \phi \implies \rho(\Pi, \phi, t) \geq 0$$
$$\rho(\Pi, \phi, t) < 0 \implies \pi_t \not\models \phi \qquad \pi_t \not\models \phi \implies \rho(\Pi, \phi, t) \leq 0$$

Key Theoretical Results

We first give the definition for the robustness estimate of a formula:

$$\rho(\Pi, p, t) = \sigma_p(t)$$
$$\rho(\Pi, \neg\phi, t) = -\rho(\Pi, \phi, t)$$
$$\rho(\Pi, \phi_1 \wedge \phi_2, t) = \min(\rho(\Pi, \phi_1, t), \rho(\Pi, \phi_2, t))$$
$$\rho(\Pi, \Diamond_{[a,b]}\phi, t) = \max_{t' \in t+[a,b]} \rho(\Pi, \phi, t')$$
$$\rho(\Pi, \phi_1\mathcal{U}_{[a,b]}\phi_2, t) = \max_{t' \in t+[a,b]} \min\left(\rho(\Pi, \phi_2, t'), \min_{t'' \in [t+a, t')} \rho(\Pi, \phi_1, t'')\right)$$

We see, however, that this requires us to know all of the values of σ_p to calculate, which we often don't have for a long time. To combat this, we define robust satisfaction intervals, or RoSIs.

If our signals are only defined up to some time $t \leq \tau$, we can define signals up to τ that are extensions of our original signals, and examine the robustness of these extensions. The RoSI is then defined to be the smallest interval containing all such robustnesses of these extensions.

Since robust satisfaction intervals are defined rather abstractly, we define the estimator function $[\rho]$ recursively to be

$$[\rho](\Pi, p, t) = \begin{cases} [\sigma_p(t), \sigma_p(t)] & \text{if } t \text{ is known} \\ [-\infty, \infty] & \text{if } t \text{ is unknown} \end{cases}$$

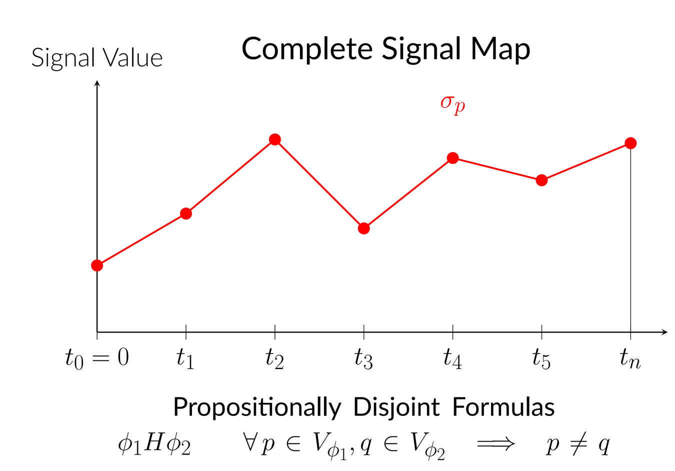
and, in all other cases of ϕ, $[\rho]$ is defined the same as ρ but on the bounds of the intervals. It is natural to believe that the RoSI of a formula ϕ coincides with the interval that $[\rho]$ calculates, but this is not true in general!

Counterexample and Solution

We have that the interval calculated ($[\rho]$) contains the RoSI, but this may be a strict containment, even for formulas as simple as $\Diamond_{[1,1]}(p \wedge \neg p)$.

To move toward equality, we provide several restrictions:

- A signal is *complete* if it is defined as on the timeset $T = [0, n] \cap \mathbb{N}$ for some n.
- A *Propositionally Disjoint* formula ϕ is such that, if no atomic proposition appears on both sides of the same binary operation.



The main result of these definitions is the following:

Given a propositionally disjoint formula and complete signals, the function $[\rho]$ calculates the exact robust satisfaction interval.

Motivating Example

Let's consider an example of an MLTL formula and compare the robust intervals with the existing true/false verdicts that tools such as R2U2 gives us. We will notice RoSIs provide the user with more information at each time step, making the intervals a powerful tool for autonomous systems.

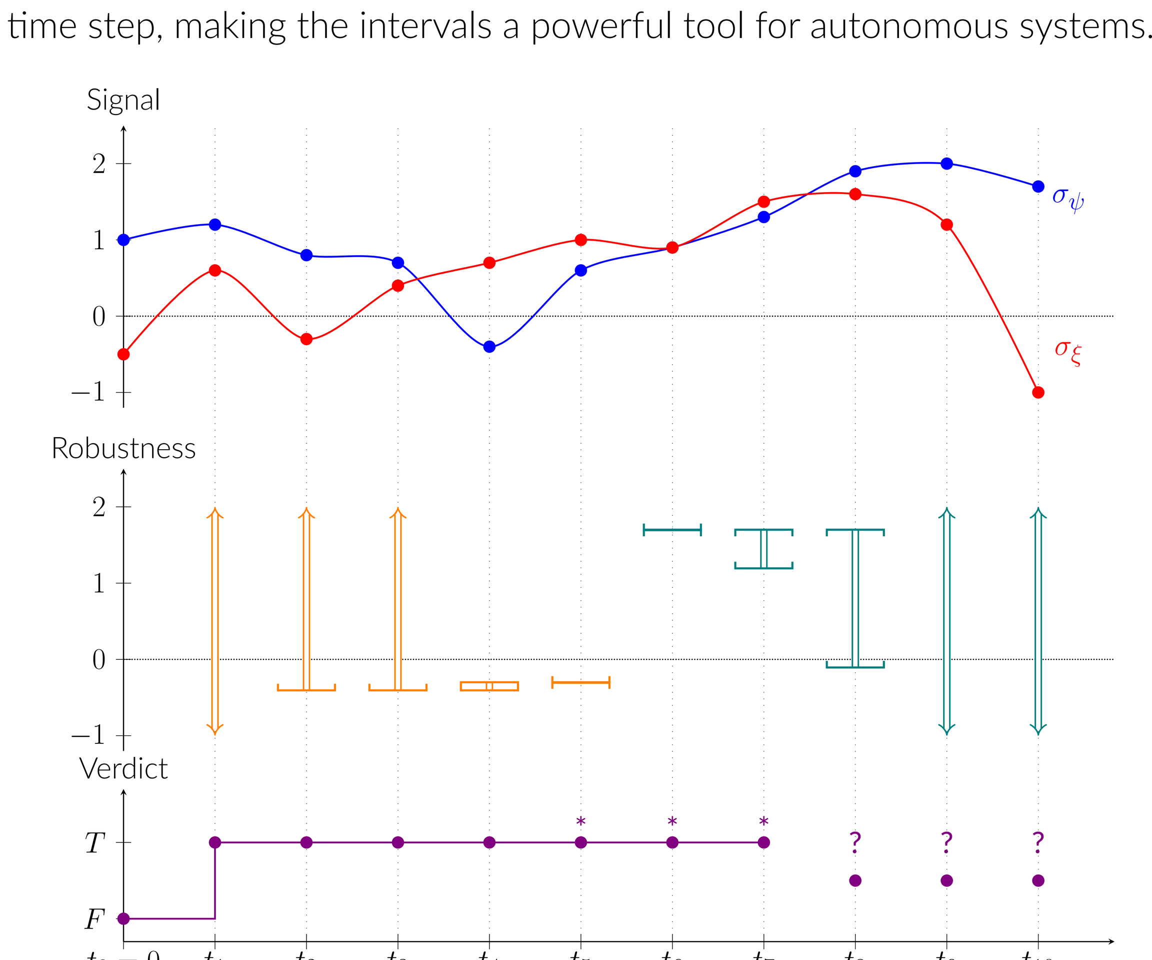


Figure 2. Calculated truth values (purple) and robust intervals (orange and teal) for the MLTL formula $\phi \equiv (\Box_{[0,3]}\psi)\,\mathcal{U}_{[2,4]}\xi$ based off of the signals given.

The verdicts (whether ϕ is true or false) at each time (in purple) are relatively easy to find, and at times $t = t_5, t_6, t_7$ (denoted with a $*$) we see that although we don't have the entire trace that the proposition is defined over, it is still known to be true no matter the completion of the trace.

Now, the RoSI shows two separate concepts. On the left (in orange) we see that calculation of $[\rho](\Pi, \phi, t_0)$ as more data points are revealed. This allows us to see that even at time t_4 we know that our entire interval is negative, and thus ϕ is false at t_0. Furthermore, even though to calculate the robustness at time t_0 we would need up to time t_6, the interval becomes singular at t_5, and thus we know our precise robustness value earlier than anticipated.

On the right (in teal), we see the values of $[\rho](\Pi, \phi, t)$ for $t = \{t_6, \ldots, t_{10}\}$ when we have the data up to time t_{10}. We see that the RoSI at t_6 has already become singular earlier than anticipated and at t_7 is entirely positive, so we already have lots of directly accessible information. We also see that at time t_8 our RoSI is only barely negative, which tells us that even if we fail our specification, we will not fail it by a lot.

R2U2 and New Engines

R2U2 is a realtime, online monitor for evaluating MLTL and past -time MLTL specifications for a given trace. Within R2U2, a temporal logic engine receives the verdicts for each atomic proposition and calculates the verdicts for each specification. To help with calculations, a type of modified ring buffer called a shared connection queue (SCQ) is used to store intermediate verdicts. For calculations on raw signal values, R2U2 also provides a booleanizer engine for converting signals into verdicts for each atomic proposition.

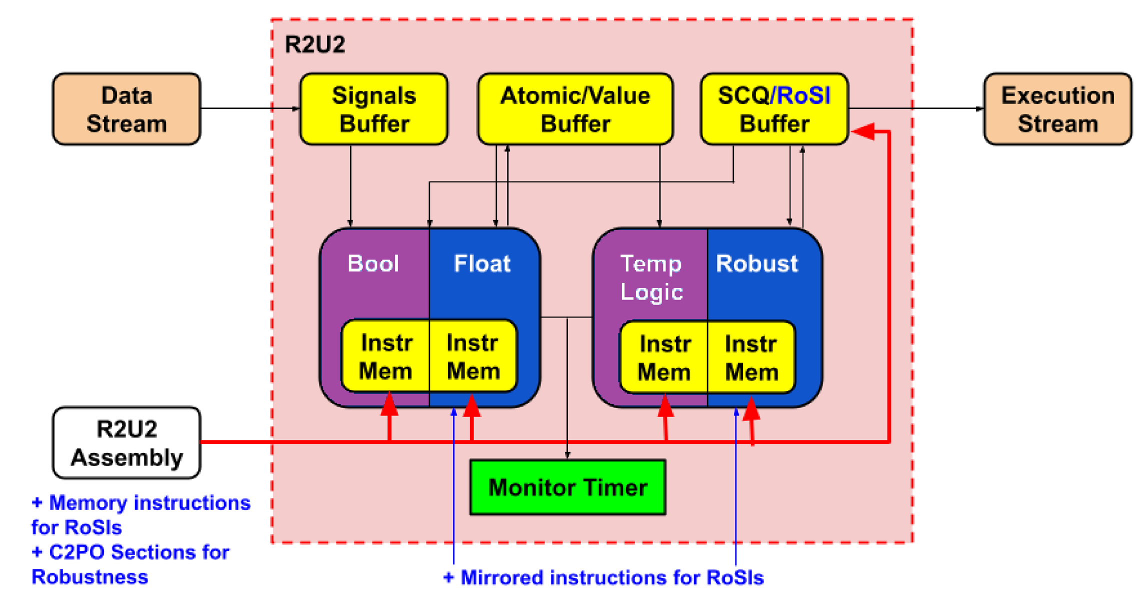


Figure 3. Functional diagram of R2U2 and new engines

However, R2U2 does not give any indication (such as robustness) of how close any trace is to passing/failing the specifications. To implement robustness in R2U2, we made a mirrored version of both engines and the SCQ buffer to process RoSIs. Instead of converting signals to verdicts, the floatizer engine evaluates the robustness of each atomic proposition from the raw signal values. The robustness engine then uses these robustness values and an analogous ring buffer to calculate the RoSI for each specification.

FORCE Implementation

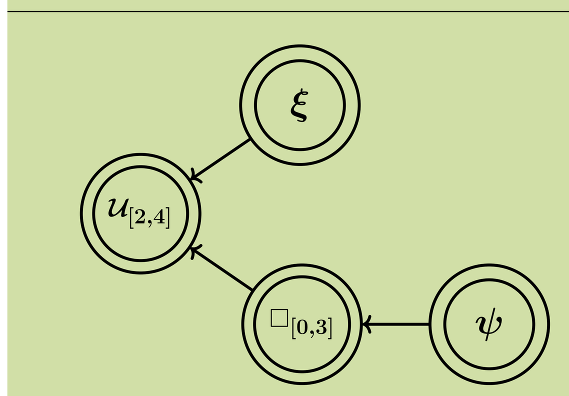


Figure 4. Abstract Syntax Tree (AST) for $\phi \equiv (\Box_{[0,3]}\psi)\,\mathcal{U}_{[2,4]}\xi$

The Fast Online Robustness Calculation Engine (FORCE) calculates RoSIs for a proposition by calculating the RoSIs for each node below the proposition in the abstract syntax tree (AST).

The algorithm first starts at the leaf nodes and propogates the calculated results up the AST. Each node of the AST also performs different operations on the RoSIs depending on which temporal/logical operator the node holds.

General Structure of Node Algorithms

- Step 1: Discard old data from RoSI buffer when unused.
- Step 2: Read appropriate RoSIs from children nodes.
- Step 3: Perform appropriate operations and store results in buffers.

Horizon

The horizon of a node v in an AST T_φ for a specification φ is defined as

$$\text{hor}(v) = \begin{cases} [0,0], & v = root(T_\varphi) \\ I \oplus \text{hor}(\text{parent}(v)), & v \neq root(T_\varphi) \text{ and parent(v)} = U_I \text{ or } R_I \\ \text{hor}(\text{parent}(v)), & \text{otherwise} \end{cases}$$

Intuitively, the horizon of a node is the timestamps needed to evaluate the RoSI at the node for the full proposition. When a RoSI in a node's buffer stops being in the horizon, we know to throw out the RoSI since no more robustness calculations for the specification needs the RoSI.

Node Operations

The operations performed at each node mirror the robustness semantics of each operator calculated for RoSIs. A table of operations for each node is provided below.

Node	Proposition	Not	And	Or	Until
Operation	Load	Negation	Min	Max	Max of Running Min on Signals

Future Work

We are currently completing a basic implementation of FORCE in R2U2, but it will still require innumerable optimizations in both memory and performance. After FORCE is implemented, we plan on comparing the performance of a variety of existing tools with R2U2 running with FORCE. After this, the next step is to apply this research to a real world scenario, such as an autonomous drones, to illustrate possible use cases.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number 1950583 and 2038903. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.