Generating System-Agnostic Runtime Verification Benchmarks from MLTL Formulas

Josh Wallin & Kristin Yvonne Rozier
Iowa State University

September 29, 2018
Midwest Verification Day 2018

Runtime Verification for Robonaut 2[1]

R2U2

REALIZABLE
RESPONSIVE
UNOBTRUSIVE
UNIT

[1] https://robonaut.jsc.nasa.gov/R2/

Runtime Verification for Robonaut 2

How can we debug/validate our monitor specifications?

Runtime Verification for Robonaut 2

How can we debug/validate our monitor specifications?
$\hookrightarrow$ Satisfiability checking!

Runtime Verification for Robonaut 2

How can we debug/validate our monitor specifications?
$\hookrightarrow$ Satisfiability checking!

How can we test our monitors?

Runtime Verification for Robonaut 2

How can we debug/validate our monitor specifications?
$\hookrightarrow$ Satisfiability checking!

How can we test our monitors?
$\hookrightarrow$ Benchmark generation!

Runtime Verification for Robonaut 2

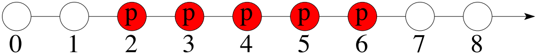
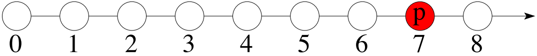
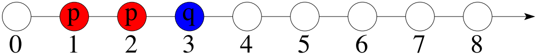
How can we debug/validate our monitor specifications?
$\hookrightarrow$ Satisfiability checking!

How can we test our monitors?
$\hookrightarrow$ Benchmark generation!

We need a procedure to check satisfiability for properties, and return a satisfying assignment

Mission-Time Linear Temporal Logic [2]

Mission-Time Linear Temporal Logic (MLTL) reasons about *finite, integer-bounded* timelines:

Symbol	Operator	Timeline
$G_{[2,6]}\,p$	ALWAYS$_{[2,6]}$	
$F_{[0,7]}\,p$	EVENTUALLY$_{[0,7]}$	
$p\,\mathcal{U}_{[1,5]}\,q$	UNTIL$_{[1,5]}$	

[2] T. Reinbacher, K.Y. Rozier, J. Schumann. "Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems." TACAS 2014.

Mission-Time Linear Temporal Logic

Why?

Naturally aligns with (some) real mission applications

e.g. actual UAS flights are predictably bounded

Bounded logics may provide faster procedures for determining SAT

Can we just use BMC?

MLTL Benchmarks

An MLTL benchmark is a 3-tuple consisting of:

MLTL Benchmarks

An MLTL benchmark is a 3-tuple consisting of:

- Trace, or computation, π
  e.g. $\pi = \langle 0, \{a, \neg b, \neg c\}\rangle, \langle 1, \{a, b, \neg c\}\rangle, \langle 2, \{\neg a, \neg b, \neg c\}\rangle \ldots$

MLTL Benchmarks

An MLTL benchmark is a 3-tuple consisting of:

- Trace, or computation, π
  e.g. $\pi = \langle 0, \{a, \neg b, \neg c\}\rangle, \langle 1, \{a, b, \neg c\}\rangle, \langle 2, \{\neg a, \neg b, \neg c\}\rangle \ldots$
- MLTL Formula, φ
  e.g. $\varphi = G_{[0,1]}(a \vee b)$

MLTL Benchmarks

An MLTL benchmark is a 3-tuple consisting of:

- Trace, or computation, π
  e.g. $\pi = \langle 0, \{a, \neg b, \neg c\}\rangle, \langle 1, \{a, b, \neg c\}\rangle, \langle 2, \{\neg a, \neg b, \neg c\}\rangle \ldots$
- MLTL Formula, φ
  e.g. $\varphi = G_{[0,1]}(a \vee b)$
- Oracle, $\mathcal{O}$
  e.g. $\mathcal{O} = \langle 0, T\rangle, \langle 1, F\rangle, \ldots$

MLTL Benchmarks

An MLTL benchmark is a 3-tuple consisting of:

- Trace, or computation, π
  e.g. $\pi = \langle 0, \{a, \neg b, \neg c\}\rangle, \langle 1, \{a, b, \neg c\}\rangle, \langle 2, \{\neg a, \neg b, \neg c\}\rangle \ldots$
- MLTL Formula, φ
  e.g. $\varphi = G_{[0,1]}(a \vee b)$
- Oracle, $\mathcal{O}$
  e.g. $\mathcal{O} = \langle 0, T\rangle, \langle 1, F\rangle, \ldots$

t	a	b	c	$\mathcal{O}$
0	T	F	F	T
1	T	T	F	F
2	F	F	F	...
...	...	...	...	

MLTL Benchmarks

An MLTL benchmark is a 3-tuple consisting of:

- Trace, or computation, π

  e.g. $\pi = \langle 0, \{a, \neg b, \neg c\}\rangle, \langle 1, \{a, b, \neg c\}\rangle, \langle 2, \{\neg a, \neg b, \neg c\}\rangle \ldots$
- MLTL Formula, φ

  e.g. $\varphi = G_{[0,1]}(a \vee b)$
- Oracle, $\mathcal{O}$

  e.g. $\mathcal{O} = \langle 0, T\rangle, \langle 1, F\rangle, \ldots$

t	a	b	c	$\mathcal{O}$
0	T	F	F	T
1	T	T	F	F
2	F	F	F	...
...	...	...	...	

MLTL Benchmarks

An MLTL benchmark is a 3-tuple consisting of:

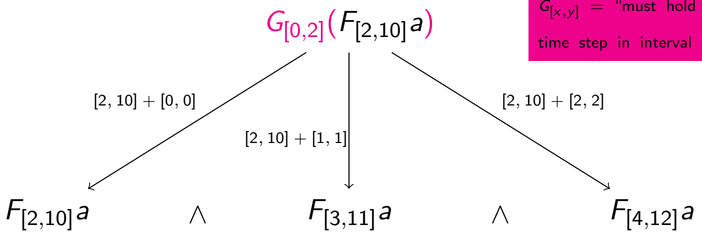
- Trace, or computation, π
  e.g. $\pi = \langle 0, \{a, \neg b, \neg c\}\rangle, \langle 1, \{a, b, \neg c\}\rangle, \langle 2, \{\neg a, \neg b, \neg c\}\rangle \ldots$
- MLTL Formula, φ
  e.g. $\varphi = G_{[0,1]}(a \vee b)$
- Oracle, $\mathcal{O}$
  e.g. $\mathcal{O} = \langle 0, T\rangle, \langle 1, F\rangle, \ldots$

t	a	b	c	$\mathcal{O}$
0	T	F	F	T
1	T	T	F	F
2	F	F	F	...
...	...	...	...	

MLTL Peculiarities

The bounded nature of MLTL formulas permits application of certain transformations.

Nested temporal operators can be eliminated



$G_{[x,y]}$ = "must hold at each time step in interval $[x, y]$"

MLTL Peculiarities

The bounded nature of MLTL formulas permits application of certain transformations.

Each temporal operator can be encoded in terms of *Globally*

$F_{[x,y]}$ = "must hold at some time step in interval $[x, y]$"

$$F_{[2,4]}c$$

$$G_{[2,2]}c \quad \vee \quad G_{[3,3]}c \quad \vee \quad G_{[4,4]}c$$

$$\equiv F_{[2,2]}c \qquad \equiv F_{[3,3]}c \qquad \equiv F_{[4,4]}c$$

Naive Encoding

We can explicitly generate a benchmark by "expanding" a formula:

$$\varphi = F_{[1,2]}(G_{[0,2]}(a \vee b))$$

Naive Encoding

We can explicitly generate a benchmark by "expanding" a formula:

$$\varphi = F_{[1,2]}(G_{[0,2]}(a \vee b))$$

$$\downarrow$$

$$\varphi' = G_{[1,2]}(a \vee b) \vee G_{[2,3]}(a \vee b)$$

Naive Encoding

We can explicitly generate a benchmark by "expanding" a formula:

$$\varphi = F_{[1,2]}(G_{[0,2]}(a \vee b))$$
$$\downarrow$$
$$\varphi' = G_{[1,2]}(a \vee b) \vee G_{[2,3]}(a \vee b)$$
$$\downarrow$$
$$\varphi' = ((a_1 \vee b_1) \wedge (a_2 \vee b_2)) \vee ((a_2 \vee b_2) \wedge (a_3 \vee b_3))$$

Naive Encoding

We can explicitly generate a benchmark by "expanding" a formula:

$$\varphi = F_{[1,2]}(G_{[0,2]}(a \vee b))$$

$$\downarrow$$

$$\varphi' = G_{[1,2]}(a \vee b) \vee G_{[2,3]}(a \vee b)$$

$$\downarrow$$

$$\varphi' = ((a_1 \vee b_1) \wedge (a_2 \vee b_2)) \vee ((a_2 \vee b_2) \wedge (a_3 \vee b_3))$$

$$\downarrow$$

SMT Solver*

Naive Encoding

We can explicitly generate a benchmark by "expanding" a formula:

$$\varphi = F_{[1,2]}(G_{[0,2]}(a \vee b))$$

$$\downarrow$$

$$\varphi' = G_{[1,2]}(a \vee b) \vee G_{[2,3]}(a \vee b)$$

$$\downarrow$$

$$\varphi' = ((a_1 \vee b_1) \wedge (a_2 \vee b_2)) \vee ((a_2 \vee b_2) \wedge (a_3 \vee b_3))$$

$$\downarrow$$

SMT Solver*

$$\downarrow$$

$$\{SAT, UNSAT\}$$

Naive Encoding

We can explicitly generate a benchmark by "expanding" a formula:

$$\varphi = F_{[1,2]}(G_{[0,2]}(a \vee b))$$
$$\downarrow$$
$$\varphi' = G_{[1,2]}(a \vee b) \vee G_{[2,3]}(a \vee b)$$
$$\downarrow$$
$$\varphi' = ((a_1 \vee b_1) \wedge (a_2 \vee b_2)) \vee ((a_2 \vee b_2) \wedge (a_3 \vee b_3))$$
$$\downarrow$$
$$\text{SMT Solver*}$$
$$\downarrow$$
$$\{SAT, UNSAT\}$$

*a, b could resolve to FO properties with respect to some theories

Problems

- Doesn't distinguish SAT from Benchmark Generation
  A benchmark *must* be generated to check SAT

Problems

- Doesn't distinguish SAT from Benchmark Generation
  A benchmark *must* be generated to check SAT
- Simple formulas over long intervals can blow up query
  e.g. $G[0, 10000]a \rightarrow$ really just need to check a itself once

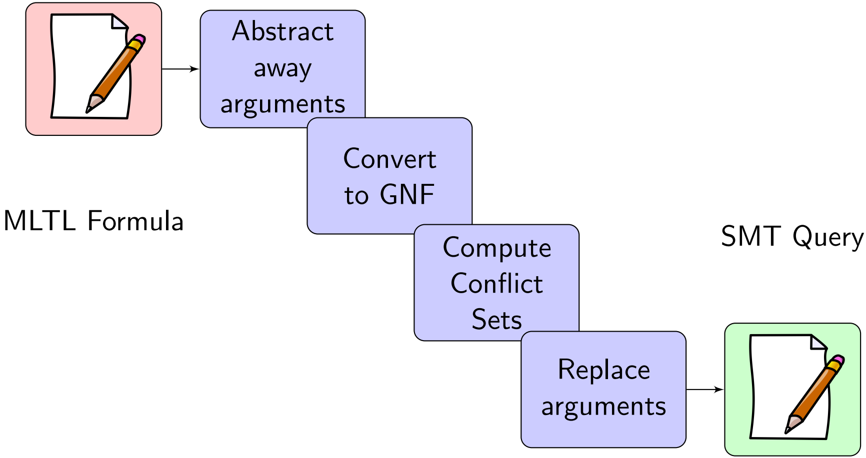
Problems

- Doesn't distinguish SAT from Benchmark Generation
  A benchmark *must* be generated to check SAT
- Simple formulas over long intervals can blow up query
  e.g. $G[0, 10000]a \rightarrow$ really just need to check a itself once
- Doesn't utilize intervals beyond expanding formulas
  e.g. $G[0, 10]a \wedge G[20, 30]b \rightarrow$ can check a and b separately

Reducing Our Encoding

How can we use the explicitly bounded nature of MLTL *effectively* to support checking satisfiability and generating benchmarks?

Interval-Aware Encoding



Procedure

1. Abstract away arguments

$$\varphi = (G_{[0,4]}(altitude > 1000ft \vee !airborne) \vee G_{[5,10]}(altitude > 1000ft \vee !airborne)) \wedge G_{[0,10]}(AMS1.valid \ \wedge \ AMS2.valid) \ \wedge F_{[1,3]}(received\_takeoff\_command)$$

$$\downarrow$$

$$\varphi' = (G_{[0,4]}a \vee G_{[5,10]}a) \wedge G_{[0,10]}b \wedge F_{[1,3]}c$$

Procedure

2. Convert to Globally Normal Form (GNF)

   i. Convert all temporal operators to *Globally*

$$\varphi = (G_{[0,4]}a \vee G_{[5,10]}a) \wedge G_{[0,10]}b \wedge F_{[1,3]}c$$

$$\downarrow$$

$$\varphi' = (G_{[0,4]}a \vee G_{[5,10]}a) \wedge G_{[0,10]}b \wedge (G_{[1,1]}c \vee G_{[2,2]}c \vee G_{[3,3]}c)$$

Procedure

2. Convert to Globally Normal Form (GNF)

   ii. Rewrite as a disjunction of conjunctions

$$\varphi' = (G_{[0,4]}a \vee G_{[5,10]}a) \wedge G_{[0,10]}b \wedge (G_{[1,1]}c \vee G_{[2,2]}c \vee G_{[3,3]}c)$$

$$\downarrow$$

$$\begin{aligned}\varphi'_{GNF} = \; & (G_{[0,4]}a \wedge G_{[0,10]}b \wedge G_{[1,1]}c) \vee \\ & (G_{[0,4]}a \wedge G_{[0,10]}b \wedge G_{[2,2]}c) \vee \\ & (G_{[0,4]}a \wedge G_{[0,10]}b \wedge G_{[3,3]}c) \vee \\ & (G_{[5,10]}a \wedge G_{[0,10]}b \wedge G_{[1,1]}c) \vee \\ & (G_{[5,10]}a \wedge G_{[0,10]}b \wedge G_{[2,2]}c) \vee \\ & (G_{[5,10]}a \wedge G_{[0,10]}b \wedge G_{[3,3]}c)\end{aligned}$$

Procedure

3. For each clause, compute the overlap between sub-formula intervals (*conflict set*)

$$C_1 = (G_{[0,4]}a \wedge G_{[0,10]}b \wedge G_{[1,1]}c)$$
$$\dots$$
$$C_6 = (G_{[5,10]}a \wedge G_{[0,10]}b \wedge G_{[3,3]}c)$$
$$\downarrow$$
$$S_{C_1} = \{a \wedge b, a \wedge c, b \wedge c, b\}$$
$$\dots$$
$$S_{C_6} = \{a \wedge b, b \wedge c, b\}$$

If, for any clause, $C_i \in \varphi'_{GNF}$, every formula in S_{C_i} is satisfiable, then φ is satisfiable.

Procedure

4. Substitute the original arguments back in

$$S_{C_1} = \{a \wedge b, a \wedge c, b \wedge c, b\}$$
$$\ldots$$
$$S_{C_6} = \{a \wedge b, b \wedge c, b\}$$

$$\text{for } a = (altitude > 1000ft \vee\ !airborne),$$
$$b = (AMS1.valid \ \wedge \ AMS2.valid),$$
$$c = (received\_takeoff\_command)$$

Check the corresponding sets of formulas for satisfiability

Procedure (for Benchmark Generation)

(5.) Conjunct the satisfying clause, increment bounds, and repeat

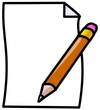
$$S_{C_6} = \{a \wedge b, b \wedge c, b\}$$

$$\downarrow$$

$$\begin{aligned}\varphi'_{GNF} = \; & (G_{[5,10]}a \wedge G_{[0,10]}b \wedge G_{[3,3]}c) \wedge \\ & ((G_{[6,11]}a \wedge G_{[1,11]}b \wedge G_{[2,2]}c) \vee \\ & (G_{[6,11]}a \wedge G_{[1,11]}b \wedge G_{[3,3]}c) \vee \\ & (G_{[6,11]}a \wedge G_{[1,11]}b \wedge G_{[4,4]}c))\end{aligned}$$

An MLTL Front End (SAT)

Conflict sets need only be computed once (not *online*)



→ MLTL Front End

$S_{C_1} = \{a \wedge b \wedge c, d \wedge e, \ldots\}$

$\vdots$

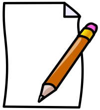
$S_{C_i} = \{a, c \wedge f, d \wedge e, \ldots\}$

↓

$\{SAT, UNSAT\}$ ← Standard SMT Solver

An MLTL Front End (Benchmark Generation)

A satisfiable conflict set can be backpropagated to generate a valid benchmark



t	alt	$airborne$	$speed$	$\mathcal{O}$
0	$1040m$	T	$812m/s$	T
⋮	⋮	⋮	⋮	⋮

MLTL Benchmark Generator

$S_{C_2} = \{a, d \wedge e, \ldots\}$ → Standard SMT Solver

Questions?

What have we proposed?

- Rewrite rules
- SAT procedure
- System-Agnostic Benchmark Generation
- Integration with existing SMT infrastructure

Open Questions

1. Can we perform this technique with CNF instead of DNF?
2. How can we avoid recomputing the same conflicts multiple times?
3. How do our assumptions on formulas hold up in the literature? e.g. Is nesting operators deeply really uncommon?
4. How might data structures be altered to support the problem?

Eliminating Nested Temporal Operators

Let $\circ^1, \circ^2$ be either of the temporal operators G or F. The appropriate rewrite rule for $\circ^1, \circ^2 = U$ follows intuitively from the other two.

$$F_{[x,y]}(\circ^1_{[a,b]}\varphi_1) = \bigvee_{i=x}^{y} \circ^1_{[a+i,b+i]}\varphi_1$$

$$G_{[x,y]}(\circ^1_{[a,b]}\varphi_1) = \bigwedge_{i=x}^{y} \circ^1_{[a+i,b+i]}\varphi_1$$

$$\circ^1_{[a,b]}\varphi_1 \;\; U_{[x,y]} \;\; \circ^2_{[c,d]}\,\varphi_2 =$$
$$\circ^2_{[c+x,d+x]}\varphi_2 \vee \bigvee_{i=x+1}^{y} \circ^1_{[a+i-1,b+i-1]}\varphi_1 \wedge \circ^2_{[c+i,c+i]}\varphi_2$$

Converting Temporal Operators to Globally

$$F_{[x,y]}\varphi_1 = \bigvee_{i=x}^{y} G_{[i,i]\varphi_1}$$

$$\varphi_1 U_{[x,y]}\varphi_2 = G_{[x,x]}\varphi_2 \vee \bigvee_{i=x+1}^{y}(G_{[x,i-1]}\varphi_1 \wedge G_{[i,i]}\varphi_2)$$