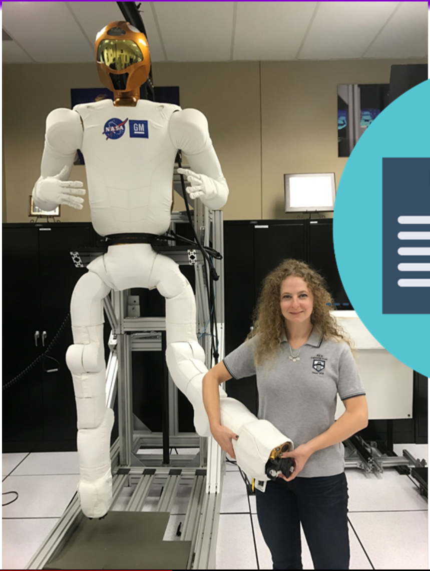
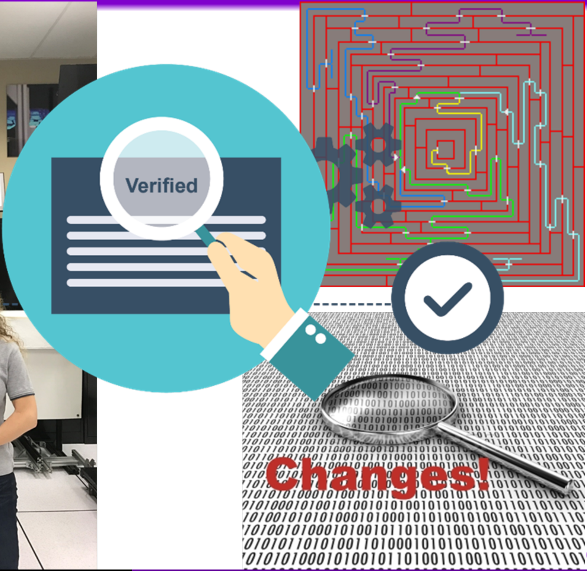
Temporal Logic Satisfiability From Specification Debugging to Benchmark Generation

Kristin Yvonne Rozier
Iowa State University

SAT, SMT, and Propositional $\rightarrow$ Temporal Logics



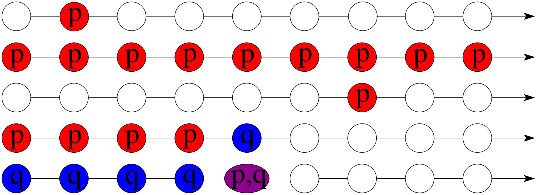


Temporal Logic Behavior Properties Over Infinite Traces

Linear Temporal Logic (LTL) formulas reason about linear timelines:

- finite set of atomic propositions {p q}
- Boolean connectives: $\neg$, $\wedge$, $\vee$, and $\rightarrow$
- temporal connectives:

$\mathcal{X}p$	NEXT TIME
$\Box p$	ALWAYS
$\Diamond p$	EVENTUALLY
$p\mathcal{U}q$	UNTIL
$p\mathcal{R}q$	RELEASE

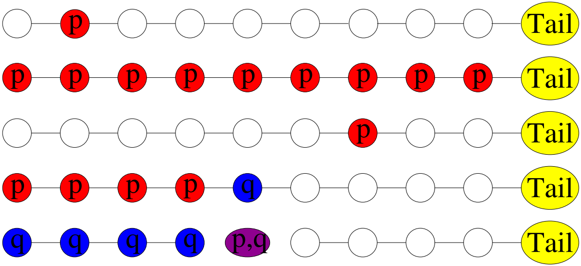


LTLf: Linear Temporal Logic on Finite Traces[1]

LTLf formulas reason about *finite* linear timelines *terminating at Tail*:

- finite set of atomic propositions {p q}
- Boolean connectives: $\neg$, $\wedge$, $\vee$, and $\rightarrow$
- temporal connectives:

$\mathcal{X} p$	NEXT TIME
$\Box p$	ALWAYS
$\Diamond p$	EVENTUALLY
$p \mathcal{U} q$	UNTIL
$p \mathcal{R} q$	RELEASE



[1] G. De Giacomo, M.Y. Vardi. "Linear temporal logic and linear dynamic logic on finite traces." IJCAI 2013.

Property Assurance: We Propose Satisfiability Checking

Let M be a temporal model that assigns values to the propositions in φ.

$M \models \varphi$ may not mean the system has the intended behavior

Recall that a property φ is *valid* iff $\neg\varphi$ is *unsatisfiable*.

If $\neg\varphi$ is not satisfiable, then

- There can never be a violation (e.g., model-checking counterexample).
- φ is probably wrong.

Property Assurance: We Propose Satisfiability Checking

Let M be a temporal model that assigns values to the propositions in φ.

$M \models \varphi$ may not mean the system has the intended behavior

$M \not\models \varphi$ may not mean the system does not have the intended behavior

Recall that a property φ is *valid* iff $\neg\varphi$ is *unsatisfiable*.

If $\neg\varphi$ is not satisfiable, then

- There can never be a violation (e.g., model-checking counterexample).
- φ is probably wrong.

If φ is not satisfiable, then

- There is always a violation (e.g., model-checking counterexample).
- φ is probably wrong.

Specification Debugging: LTL Satisfiability Checking

For each property φ and $\neg\varphi$ we should check for satisfiability.

Specification Debugging: LTL Satisfiability Checking

For each property φ and $\neg\varphi$ we should check for satisfiability.

We need to check the conjunction of all properties for satisfiability.

LTL Satisfiability Is Hard

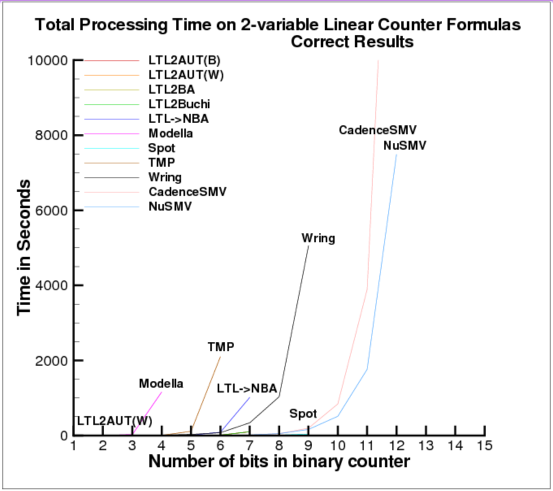
LTL Satisfiability Checking is PSPACE-Complete!

LTL Satisfiability Is Hard

LTL Satisfiability Checking is PSPACE-Complete!

We have to be smart about encoding the problem!

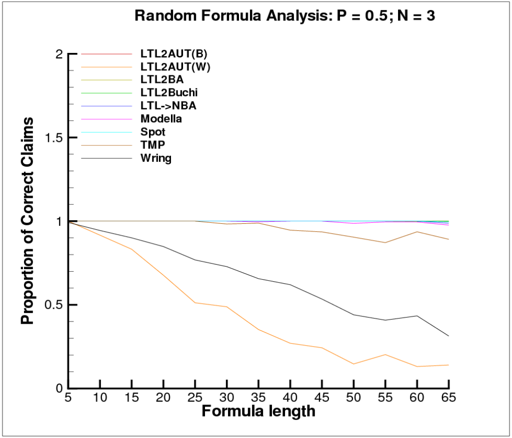
LTL Satisfiability is Hard to Scale[2]



Many tools cannot check 8-bit binary counter formulas

[2] K.Y.Rozier, M.Y.Vardi. "LTL Satisfiability Checking." STTT Journal, pg. 123–137, 2010.

LTL Satisfiability Is Hard to Code Correctly[3]



Common Problems:

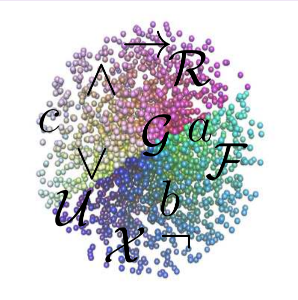
- Reporting SAT when a formula is UNSAT and vice versa.
- No difference between empty automata (indicating UNSAT) and error cases.

Most LTL encoding tools do not behave robustly and die gracelessly.

3 K.Y.Rozier, M.Y.Vardi. "LTL Satisfiability Checking." STTT Journal, pg. 123–137, 2010.

We Correct These By Establishing Rigorous Benchmarks [4]

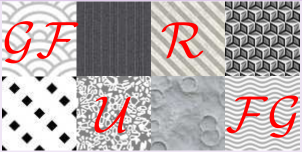
Random Formulas: 60,000



Counter Formulas: ~60 (4 types)

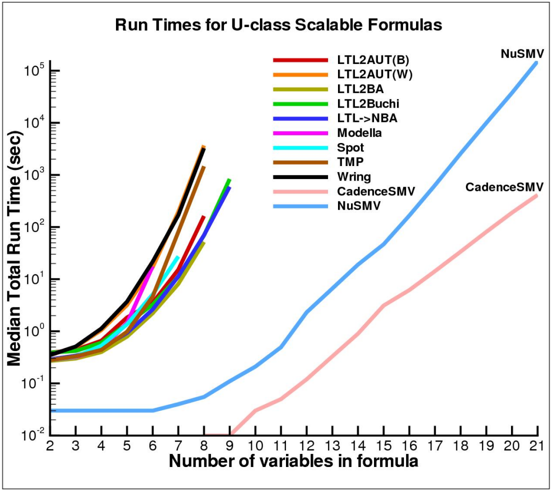
00	01	10	11 ...				
000	001	010	011	100 ...			
0000	0001	0010	0011	0100	0101 ...		
00000	00001	00010	00011	00100	00101	00110	...
⋮							

Pattern Formulas: ~$8,000$ (9 patterns)



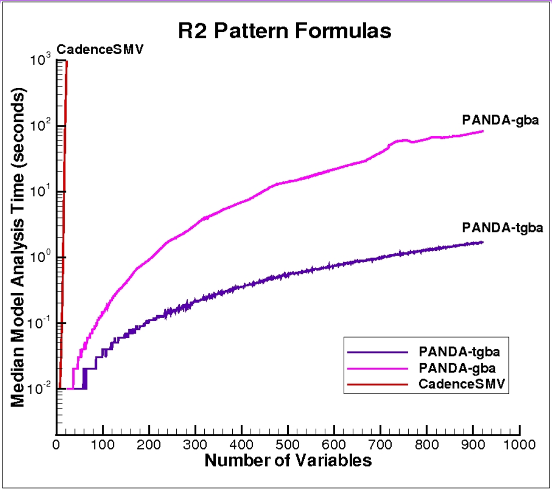
[4] K.Y. Rozier and M.Y. Vardi. "LTL Satisfiability Checking." SPIN'07.

Implementation is Hugely Influential[5]



5 K.Y.Rozier, M.Y.Vardi. "LTL Satisfiability Checking." STTT Journal, pg. 123–137, 2010.

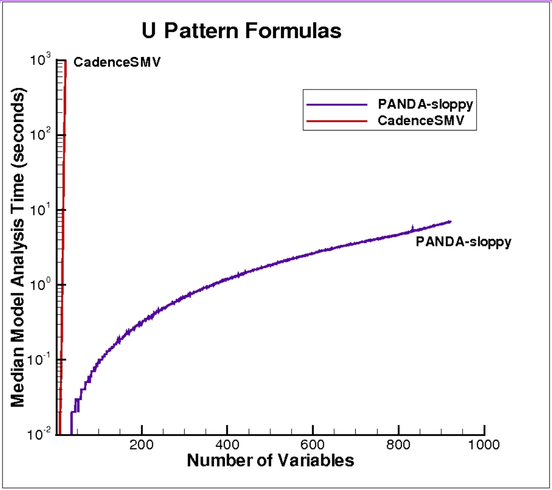
Better Encoding Can Lead to Exponential Improvement! [6]



$$R_2(n) = (..(p_1 \ \mathcal{R} \ p_2) \ \mathcal{R} \ \ldots) \ \mathcal{R} \ p_n.$$

[6] K.Y. Rozier and M.Y. Vardi. "A Multi-Encoding Approach for LTL Symbolic Satisfiability Checking." FM'11.

Even for Very Hard Formulas! [7]



$$U(n) = (\ldots(p_1 \ \mathcal{U} \ p_2) \ \mathcal{U} \ \ldots) \ \mathcal{U} \ p_n.$$

[7] K.Y. Rozier and M.Y. Vardi. "A Multi-Encoding Approach for LTL Symbolic Satisfiability Checking." FM'11.

We Need to Sustain a Runtime Verification Competition

RV-CuBES | RV 2017 - Mozilla Firefox

RV-CuBES | ...

rv2017.cs.manchester.ac.uk/rv-cubes/

Most Visited · Red Hat · Customer Portal · Documentation · Red Hat Network

Position papers

Initial submissions should use a minimum of 2 pages to explore a particular position related to the evaluation, comparison or standardisation of Runtime Verification tools (and benchmarks). Topics may include, but are not limited to:

- What should a RV benchmark look like?
- Can we have a common specification language for RV? If so, what should it look like?
- Is execution time the most important performance criteria? What might be more important?
- How can we evaluate hardware monitoring tools?
- What are we doing wrong in evaluation? Can we fix this?
- What can be borrowed form other communities?

A selection of position papers will be used to structure a discussion panel to be held at RV 2017. Again, there is no upper page limit however the number of pages used should reflect the level of detail given. There will be an opportunity to update the paper based on discussions before inclusion in post-proceedings (see below).

RV2017
@RV_2017

The RV'17 poster is now o

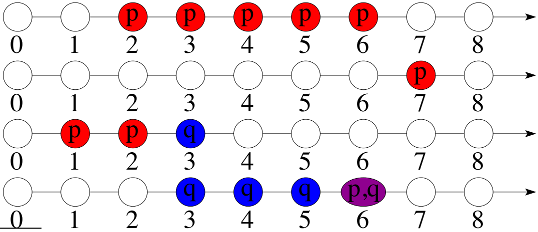
Mission-Bounded Linear Temporal Logic [8]

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

- finite set of atomic propositions {p q}
- Boolean connectives: $\neg$, $\wedge$, $\vee$, and $\rightarrow$
- temporal connectives *with time bounds*:

Symbol	Operator
$\Box_{[2,6]} p$	ALWAYS$_{[2,6]}$
$\Diamond_{[0,7]} p$	EVENTUALLY$_{[0,7]}$
$p \mathcal{U}_{[1,5]} q$	UNTIL$_{[1,5]}$
$p \mathcal{R}_{[3,8]} q$	RELEASE$_{[3,8]}$

Timeline



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

What is an (MLTL) RV Benchmark?

Time:										
	0	1	2	3	4	5	6	7	8	9

MLTL formula φ evaluated over system trace π:

$$\forall i : 0 \leq i \leq \text{Mission Time } \pi, i \models \varphi.$$

An MLTL Runtime Benchmark is a 3-tuple:

- Input stream, or computation, π
- MLTL formula, φ, over n propositional variables
- Oracle $\mathcal{O}$, of $\langle time, verdict \rangle$

MLTL Runtime Benchmark Generation: An Example[9]

Time:	a	¬a	¬a	a	a	a	a	a	a	a
	0	1	2	3	4	5	6	7	8	9

MLTL formula φ evaluated over system trace π:

$$\forall i : 0 \leq i \leq \text{Mission Time } \pi, i \models \varphi.$$

MLTL Runtime Benchmark Example:

- $\pi = a, \neg a, \neg a, a, a, a, a, a, a, a$
- $\varphi = \textsc{Always}_{[5]}(a)$
- $\mathcal{O} = \langle 0, F \rangle, \langle 1, F \rangle, \langle 2, F \rangle, \langle 3, T \rangle, \langle 4, T \rangle, \ldots$

9 J.Wallin and K.Y.Rozier. "Generating System-Agnostic Runtime Verification Benchmarks from MLTL Formulas via SAT." Under Submission.

Example: Benchmark Generation via Formula Progression[10]

$$Inputs: \quad \varphi = \mathcal{F}_{[2,3]} a$$
$$\pi = \neg a,\ \neg a,\ a$$

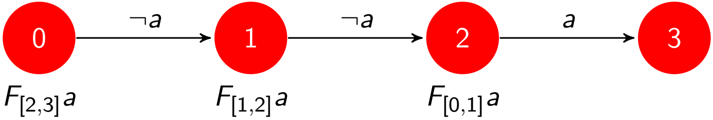


Figure: The schema of $prog(F_{[2,3]} a, \pi = \{\neg a\}\{\neg a\}\{a\})$.

[10] Jianwen Li and Kristin Yvonne Rozier. "MLTL Benchmark Generation via Formula Progression." In Runtime Verification (RV18), Springer-Verlag, 2018.

Example: Benchmark Generation via Formula Progression[10]

$$\textit{Inputs}: \quad \varphi = \mathcal{F}_{[2,3]}a$$
$$\pi = \neg a,\ \neg a,\ a$$

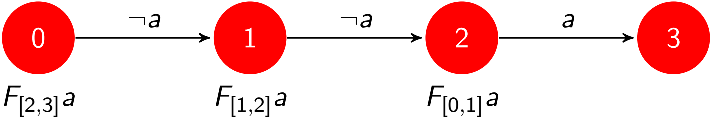


Figure: The schema of $prog(F_{[2,3]}a, \pi = \{\neg a\}\{\neg a\}\{a\})$.

$$prog(\varphi, \pi^1 (= \{\neg a\})) = F_{[1,2]}a$$

[10] Jianwen Li and Kristin Yvonne Rozier. "MLTL Benchmark Generation via Formula Progression." In Runtime Verification (RV18), Springer-Verlag, 2018.

Example: Benchmark Generation via Formula Progression[10]

$$Inputs: \quad \varphi = \mathcal{F}_{[2,3]}a$$
$$\pi = \neg a,\ \neg a,\ a$$

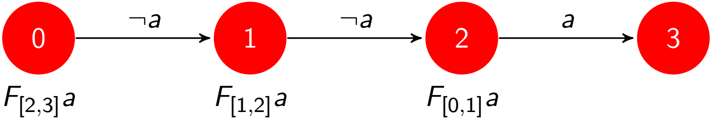


Figure: The schema of $prog(F_{[2,3]}a, \pi = \{\neg a\}\{\neg a\}\{a\})$.

$$prog(\varphi, \pi^1(= \{\neg a\})) = F_{[1,2]}a$$
$$prog(\varphi, \pi^2(= \{\neg a\}\{\neg a\})) = F_{[0,1]}a$$

[10] Jianwen Li and Kristin Yvonne Rozier. "MLTL Benchmark Generation via Formula Progression." In Runtime Verification (RV18), Springer-Verlag, 2018.

Example: Benchmark Generation via Formula Progression[10]

$$Inputs: \quad \varphi = \mathcal{F}_{[2,3]}a$$

$$\pi = \neg a,\ \neg a,\ a$$

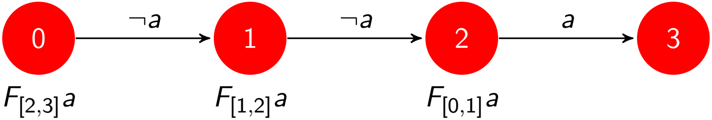


Figure: The schema of $prog(F_{[2,3]}a, \pi = \{\neg a\}\{\neg a\}\{a\})$.

$$prog(\varphi, \pi^1(=\{\neg a\})) = F_{[1,2]}a$$
$$prog(\varphi, \pi^2(=\{\neg a\}\{\neg a\})) = F_{[0,1]}a$$
$$prog(\varphi, \pi^3(=\{\neg a\}\{\neg a\}\{a\})) = \mathrm{TRUE}$$

[10] Jianwen Li and Kristin Yvonne Rozier. "MLTL Benchmark Generation via Formula Progression." In Runtime Verification (RV18), Springer-Verlag, 2018.

MLTL Satisfiability

We can generate MLTL RV benchmarks with an MLTL Satisfiability Solver[11]

[11] Jianwen Li, Moshe Vardi, and Kristin Yvonne Rozier. Satisfiability Checking for Mission-Time LTL. In CAV, Springer-Verlag, 2019.

Open Questions

- **Can we improve LTL Satisfiability (algorithms, tools, usability) further? (How?)**
  - MLTL Satisfiability? MLTL Benchmarks?
  - LTLf Satisfiability? LTLf Benchmarks?
- **All of these logics have past-time variants. . .**
  - pt-LTL-SAT? Benchmarks?
  - pt-MLTL-SAT? Benchmarks?
  - pt-LTLf-SAT? Benchmarks?
- **MaxSat for Temporal Logics?**
  - Applications in specification debugging, requirements engineering, explainability, . . .
  - RV? Essentially a MaxSat problem at each time instance?
- **Many more . . .**