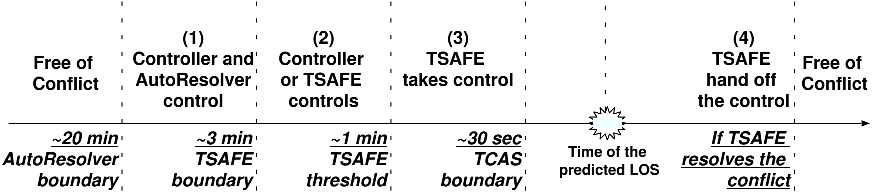
MAX-SAT for Temporal Logics

Kristin Yvonne Rozier
Iowa State University

Banff International Research Station for Mathematical Innovation and Discovery

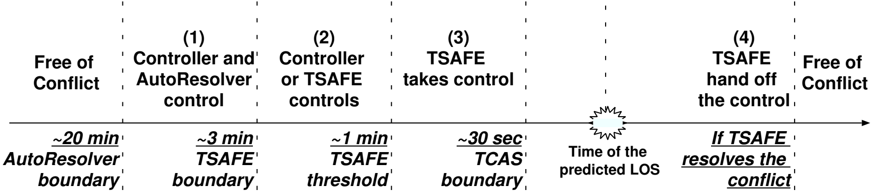
Casa Mathematica Theoretical Foundations of SAT Solving Workshop
August 30, 2018

AAC Operational Concept[1]



[1] H Erzberger, K Heere. "Algorithm and operational concept for resolving short-range conflicts." Proc. IMechE G J. Aerosp. Eng. 224 (2) (2010) 225–243.

AAC Operational Concept[2]

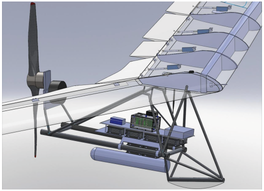


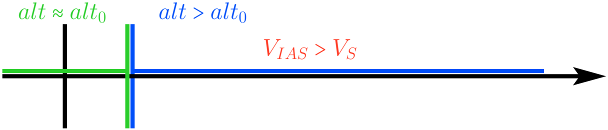
Formal verification triggered system design changes[1]

[1] Y. Zhao and K.Y. Rozier. "Formal Specification and Verification of a Coordination Protocol for an Automated Air Traffic Control System." SCP Journal, vol-96, no-3, pg 337-353, 2014.

[2] H Erzberger, K Heere. "Algorithm and operational concept for resolving short-range conflicts." Proc. IMechE G J. Aerosp. Eng. 224 (2) (2010) 225–243.

Operational Concept for the Swift UAS

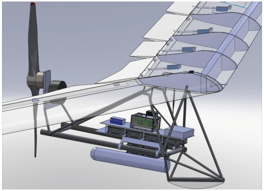


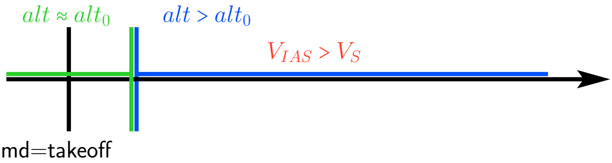


Whenever the Swift UAS is in the air, its indicated airspeed (V_{IAS}) must be greater than its stall speed V_S. The UAS is considered to be air-bound when its altitude alt is larger than that of the runway alt_0.[3]

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

Operational Concept for the Swift UAS



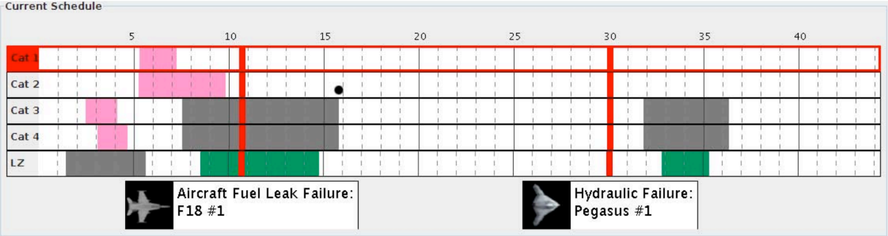


Whenever the Swift UAS is in the air, its indicated airspeed (V_{IAS}) must be greater than its stall speed V_S. The UAS is considered to be air-bound when its altitude alt is larger than that of the runway alt_0.[3]

$$\text{ALWAYS}((alt > alt_0) \rightarrow (V_{IAS} > V_S))$$

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

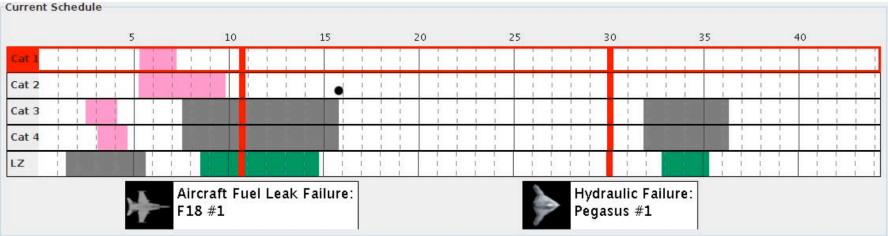
There is a Pattern Here. . .



Air Force aircraft carrier deck scheduling: deck resource timeline displaying three failures[4]

[4] J.C.Ryan, M.L.Cummings, N.Roy, A Banerjee, A.Schulte. "Designing an Interactive Local and Global Decision Support System for Aircraft Carrier Deck Scheduling." AIAA Infotech, 2011.

There is a Pattern Here. . .



Air Force aircraft carrier deck scheduling: deck resource timeline displaying three failures[4]

Aerospace Operational Concepts Are Often Specified With Timelines

[4] J.C.Ryan, M.L.Cummings, N.Roy, A Banerjee, A.Schulte. "Designing an Interactive Local and Global Decision Support System for Aircraft Carrier Deck Scheduling." AIAA Infotech, 2011.

A Natural Logic for Operational Timelines: Linear Temporal Logic

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:

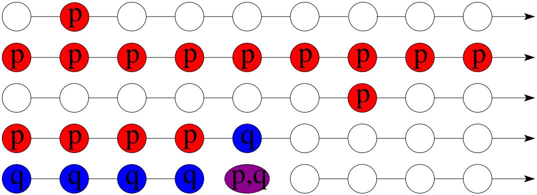
$\mathcal{X}p$ NEXT TIME

$\Box p$ ALWAYS

$\Diamond p$ EVENTUALLY

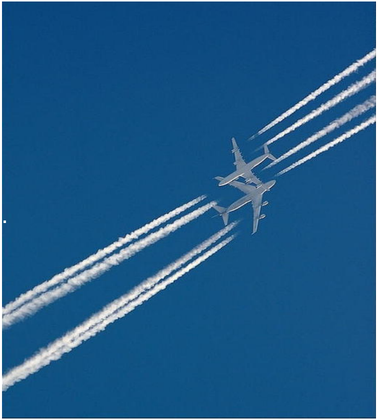
$p\mathcal{U}q$ UNTIL

$p\mathcal{R}q$ RELEASE



Formal Verification Via Model Checking

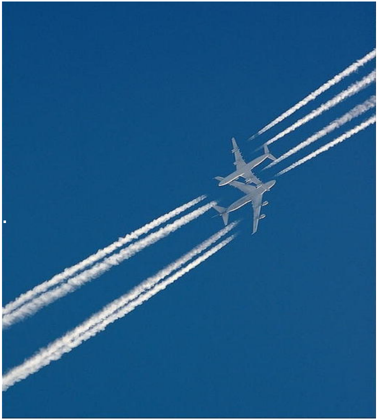
1. Describe system requirements in a formal specification, φ.
2. Create a system model with formal semantics, M.
3. Check that M satisfies φ.



Model checking finds disagreements between the system model and the formal specification.

Formal Verification Via LTL Model Checking

1. Describe system requirements in a formal LTL specification, φ.
2. Create a system model with formal semantics, M.
3. Check that M satisfies φ.



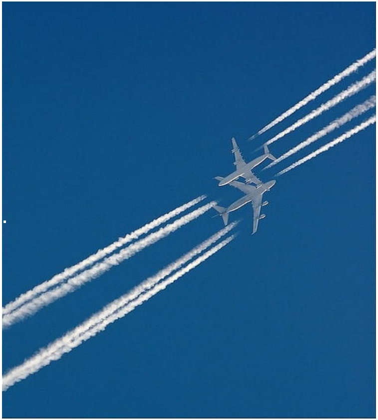
Model checking finds disagreements between the system model and the formal specification.

Formal Verification Via LTL Model Checking

1. Describe system requirements in a formal LTL specification, φ.

2. Create a system model with formal semantics, M.

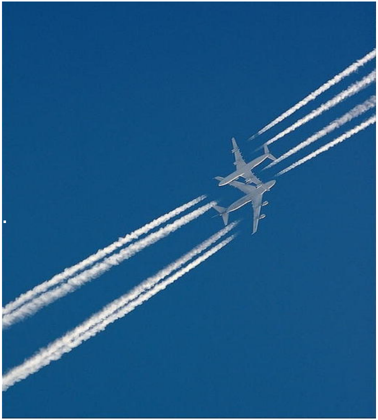
3. Check that M satisfies φ.
   - Graph-search-based
   - BDD-based
   - BMC-based
   - IC3-based



Model checking finds disagreements between the system model and the formal specification.

Formal Verification Via LTL Model Checking

1. Describe system requirements in a formal LTL specification, φ.
   - Only works if the formula is correct!
2. Create a system model with formal semantics, M.
3. Check that M satisfies φ.
   - Graph-search-based
   - BDD-based
   - BMC-based
   - IC3-based



Model checking finds disagreements between the system model and the formal specification.

Property Assurance: We Propose Satisfiability Checking

$M \models \varphi$ 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 counterexample.
- Model checkers will always return "success."
- φ is probably wrong.

Property Assurance: We Propose Satisfiability Checking

$M \models \varphi$ 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 counterexample.
- Model checkers will always return "success."
- φ is probably wrong.

If φ is not satisfiable, then

- There is always a counterexample.
- Model checkers will always return "failure."
- 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-to-Automaton Complexity

- LTL property f of size $|\varphi|$
- System model M of size $|M|$
- LTL satisfiability checking takes time $|M| \cdot 2^{\mathcal{O}(|\varphi|)}$.

LTL Satisfiability Checking is PSPACE-Complete!

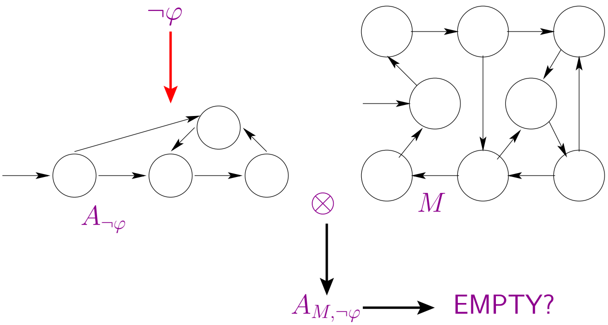
LTL-to-Automaton Complexity

- LTL property f of size $|\varphi|$
- System model M of size $|M|$
- LTL satisfiability checking takes time $|M| \cdot 2^{\mathcal{O}(|\varphi|)}$.

LTL Satisfiability Checking is PSPACE-Complete!

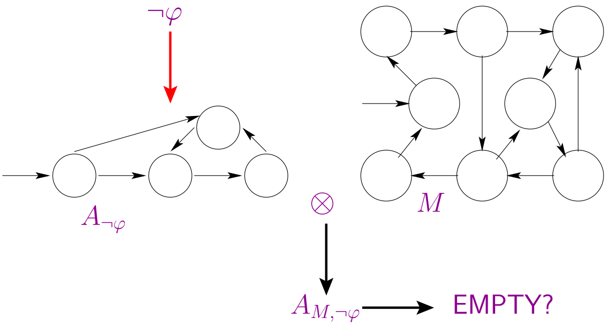
We have to be smart about encoding the problem!

Ex: Automata-Theoretic Approach to Model Checking: One of the PSPACE-Complete Algorithms for LTL-SAT

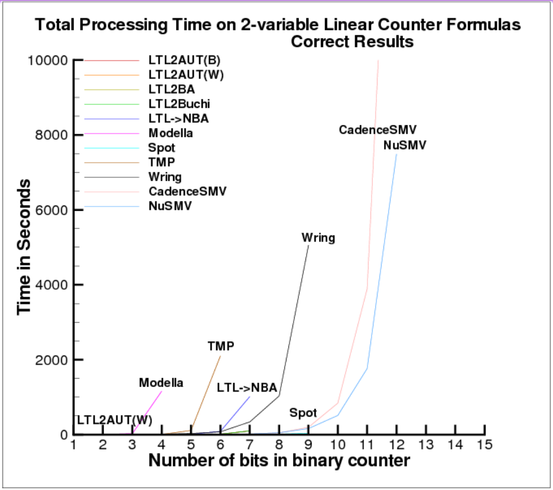


Ex: Automata-Theoretic Approach to Model Checking: One of the PSPACE-Complete Algorithms for LTL-SAT

Requires efficient LTL-to-automaton translation.



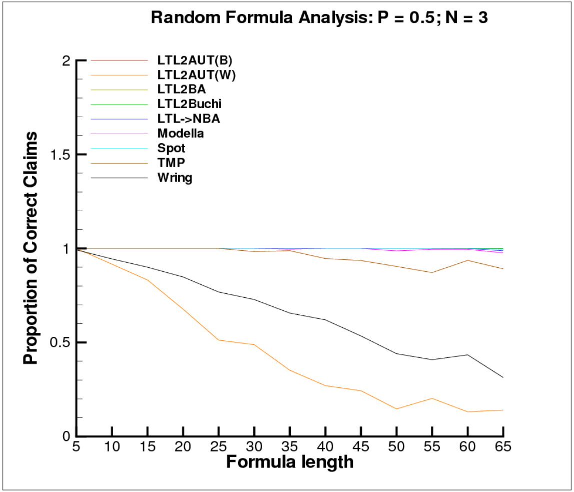
LTL Satisfiability is Hard to Scale[5]



Many tools cannot check 8-bit binary counter formulas

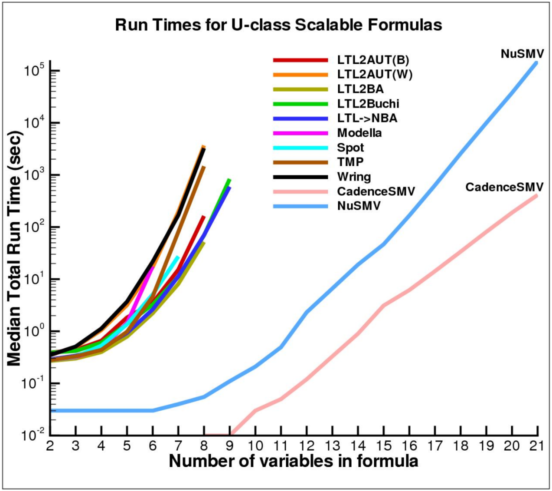
[5] K.Y.Rozier, M.Y.Vardi. "LTL Satisfiability Checking." STTT Journal, pg. 123–137, 2010.

LTL Satisfiability is Hard to Code Correctly[6]



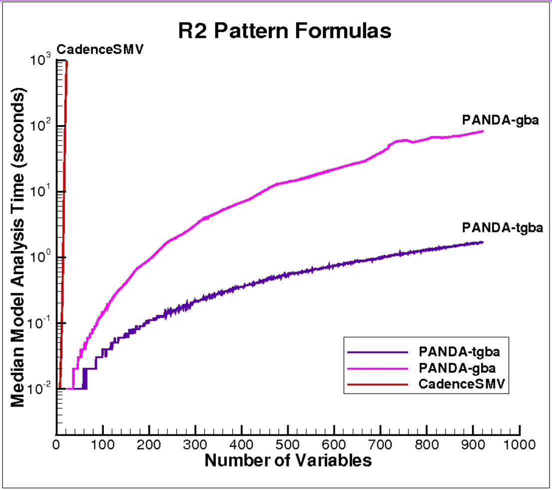
6 K.Y.Rozier, M.Y.Vardi. "LTL Satisfiability Checking." STTT Journal, pg. 123–137, 2010.

Implementation is Hugely Influential[7]



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

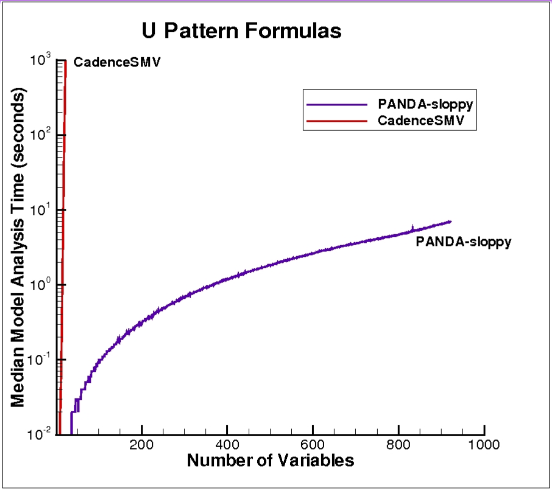
Better Encoding Can Lead to Exponential Improvement! [8]



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

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

Even for Very Hard Formulas! [9]



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

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

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.

Specification Debugging: LTL Satisfiability Checking

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

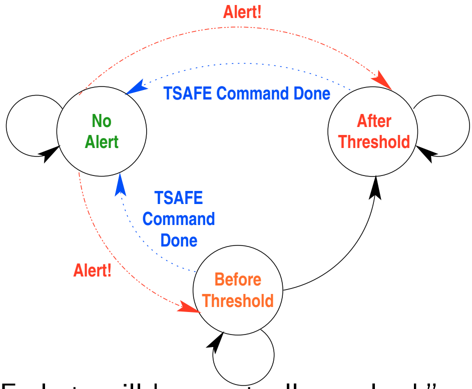
We need to check the conjunction of all properties for satisfiability.
Is this actually required in real life?

LTL Satisfiability Checking Found A Specification Bug

LTL safety requirement φ_0
LTL fairness constraint φ_1

ALWAYS EVENTUALLY $\varphi_1 \rightarrow \varphi_0$

An overstrict φ_1 can effectively cause φ_0 to be valid!



Example:

Safety Requirement: "All TSAFE alerts will be eventually resolved."
Fairness Constraint: Progress between TSAFE alerts

Wrong: `FAIRNESS (TSAFE_Alert = Non);`
Right: `FAIRNESS (TSAFE_Alert != AT);`

Problem Overview

- **Specification Debugging:** If the conjunction of all properties is not satisfiable, where is the problem?

Problem Overview

- **Specification Debugging:** If the conjunction of all properties is not satisfiable, where is the problem?
- **Requirements Engineering:** If the conjunction of all requirements is UNSAT, how many can I have? What's the closest you can give me to what I want?

Problem Overview

- **Specification Debugging:** If the conjunction of all properties is not satisfiable, where is the problem?
- **Requirements Engineering:** If the conjunction of all requirements is UNSAT, how many can I have? What's the closest you can give me to what I want?
- **XAI:** "I could not solve this because ... This (smallest subset of) requirement(s) is not compatible with the rest of the set"

Problem Overview

- **Specification Debugging:** If the conjunction of all properties is not satisfiable, where is the problem?
- **Requirements Engineering:** If the conjunction of all requirements is UNSAT, how many can I have? What's the closest you can give me to what I want?
- **XAI:** "I could not solve this because ... This (smallest subset of) requirement(s) is not compatible with the rest of the set"

These are all MAX-SAT!

Problem Overview

- **Specification Debugging:** If the conjunction of all properties is not satisfiable, where is the problem?
- **Requirements Engineering:** If the conjunction of all requirements is UNSAT, how many can I have? What's the closest you can give me to what I want?
- **XAI:** "I could not solve this because . . . This (smallest subset of) requirement(s) is not compatible with the rest of the set"

These are all MAX-SAT!

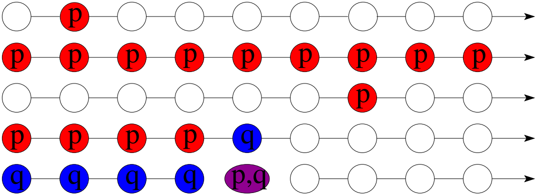
But SAT for LTL is already hard!

Linear Temporal Logic: Reasons 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

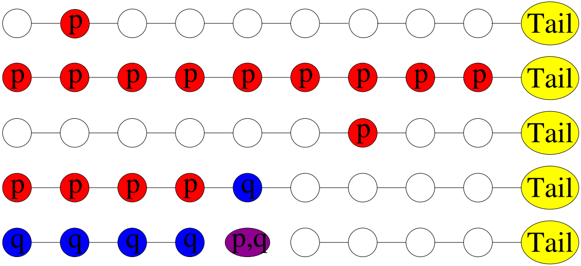


LTLf: Linear Temporal Logic on Finite Traces[10]

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



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

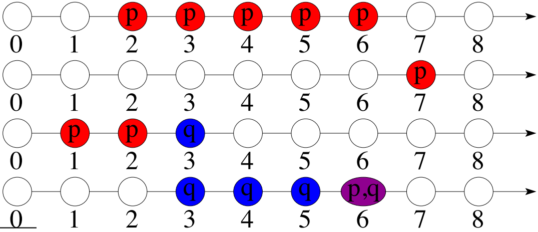
Mission-Bounded Linear Temporal Logic [11]

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

- finite set of atomic propositions {p q}
- Boolean connectives: ¬, ∧, ∨, and →
- 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



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

MLTL Runtime Benchmark Generation: An Easier Problem[12]

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

MLTL formula φ evaluated over system trace π:

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

An MLTL Runtime Benchmark is a 3-tuple:

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

[12] J.Walling and K.Y.Rozier. "Generating System-Agnostic Runtime Verification Benchmarks from MLTL Formulas via SAT." Under Submission, 2018.

MLTL Runtime Benchmark Generation: An Example[13]

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:

$$\forall i : 0 \leq i \leq \text{MissionTime}\ \pi, i \vDash \varphi.$$

MLTL Runtime Benchmark Example:

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

[13] J.Walling and K.Y.Rozier. "Generating System-Agnostic Runtime Verification Benchmarks from MLTL Formulas via SAT." Under Submission, 2018.

MLTL Runtime Benchmark Generation: An Example[13]

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:

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

MLTL Runtime Benchmark Example:

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

A SAT Encoding:

Assign a_i to a at time i.

Iteratively conjunct the satisfying assignment from i to the formula for $i + 1$. Record UNSAT as $\mathcal{O} = \langle i, F \rangle$; otherwise $\langle i, T \rangle$

13 J.Walling and K.Y.Rozier. "Generating System-Agnostic Runtime Verification Benchmarks from MLTL Formulas via SAT." Under Submission, 2018.

Open Questions

- How can we design (more) efficient MAX-SAT for MLTL?
- Can we design a MAX-SAT solver for LTL? For LTLf?
- Can we develop heuristics specific to MAX-SAT for temporal logics?
- Can we take advantage of the intuitions inherent to this domain?