Temporal Logic Satisfiability
From Specification Debugging
to Benchmark Generation

Kristin Yvonne Rozier
Iowa State University
SAT, SMT, and Propositional → 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: \(\neg, \land, \lor, \text{ and } \rightarrow\)
- temporal connectives:
  - \(\chi p\): NEXT TIME
  - \(\square p\): ALWAYS
  - \(\diamond p\): EVENTUALLY
  - \(p U q\): UNTIL
  - \(p R q\): RELEASE

\[
\chi p \quad \square p \quad \diamond p \quad pUq \quad pRq
\]

\[
\begin{align*}
\chi p & \quad \text{NEXT TIME} \\
\square p & \quad \text{ALWAYS} \\
\diamond p & \quad \text{EVENTUALLY} \\
pUq & \quad \text{UNTIL} \\
pRq & \quad \text{RELEASE}
\end{align*}
\]
**LTLf: Linear Temporal Logic on Finite Traces**

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

- finite set of atomic propositions \{p, q\}
- Boolean connectives: \(\neg, \land, \lor, \text{ and } \rightarrow\)
- temporal connectives:
  - \(\chi p\) NEXT TIME
  - \(\Box p\) ALWAYS
  - \(\Diamond p\) EVENTUALLY
  - \(p U q\) UNTIL
  - \(p R q\) RELEASE

---


Laboratory for Temporal Logic

Kristin Yvonne Rozier

Temporal Logic Satisfiability
Property Assurance: We Propose Satisfiability Checking

Let $M$ be a temporal model that assigns values to the propositions in $\varphi$.

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

Recall that a property $\varphi$ 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).
- $\varphi$ is probably wrong.
Property Assurance: We Propose Satisfiability Checking

Let $M$ be a temporal model that assigns values to the propositions in $\varphi$.

$M \models \varphi$ 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 $\varphi$ 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).
- $\varphi$ is probably wrong.

If $\varphi$ is not satisfiable, then
- There is always a violation (e.g., model-checking counterexample).
- $\varphi$ is probably wrong.
Specification Debugging: LTL Satisfiability Checking

For each property \( \varphi \) and \( \neg \varphi \) we should check for satisfiability.
Specification Debugging: LTL Satisfiability Checking

For each property $\varphi$ 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\textsuperscript{2}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Total Processing Time on 2-variable Linear Counter Formulas - Correct Results}
\end{figure}

Many tools cannot check 8-bit binary counter formulas

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.

We correct these by establishing rigorous benchmarks.

Random Formulas: 60,000

Counter Formulas: \(~60\) (4 types)

\[
\begin{array}{cccccc}
00 & 01 & 10 & 11 & \ldots \\
000 & 001 & 010 & 011 & 100 & \ldots \\
0000 & 0001 & 0010 & 0011 & 0100 & 0101 & \ldots \\
00000 & 00001 & 00010 & 00011 & 00100 & 00101 & 00110 & \ldots \\
\vdots
\end{array}
\]

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

Implementation is Hugely Influential

Run Times for U-class Scalable Formulas

Better Encoding Can Lead to Exponential Improvement!  

\[ R_2(n) = \left( \ldots (p_1 \, \mathcal{R} \, p_2) \, \mathcal{R} \, \ldots \right) \, \mathcal{R} \, p_n. \]

Even for Very Hard Formulas! 7

\[ U(n) = (\ldots (p_1 \cup p_2) \cup \ldots) \cup p_n. \]

We Need to Sustain a Runtime Verification Competition

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 from 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).
Mission-Bounded Linear Temporal Logic

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

- finite set of atomic propositions \{p, q\}
- Boolean connectives: \(\neg, \land, \lor,\) and \(\to\)
- temporal connectives with time bounds:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>□[2,6]p</td>
<td><strong>ALWAYS</strong>[2,6]</td>
<td><img src="" alt="Timeline Diagram" /></td>
</tr>
<tr>
<td>♦[0,7]p</td>
<td><strong>EVENTUALLY</strong>[0,7]</td>
<td><img src="" alt="Timeline Diagram" /></td>
</tr>
<tr>
<td>pU[1,5]q</td>
<td><strong>UNTIL</strong>[1,5]</td>
<td><img src="" alt="Timeline Diagram" /></td>
</tr>
<tr>
<td>pR[3,8]q</td>
<td><strong>RELEASE</strong>[3,8]</td>
<td><img src="" alt="Timeline Diagram" /></td>
</tr>
</tbody>
</table>

What is an (MLTL) RV Benchmark?

An MLTL Runtime Benchmark is a 3-tuple:

- Input stream, or computation, $\pi$
- MLTL formula, $\varphi$, over $n$ propositional variables
- Oracle $O$, of $\langle \text{time}, \text{verdict} \rangle$

MLTL formula $\varphi$ evaluated over system trace $\pi$:

$$\forall i : 0 \leq i \leq \text{Mission Time} \pi, i \models \varphi.$$
MLTL Runtime Benchmark Generation: An Example\(^9\)

<table>
<thead>
<tr>
<th>Time:</th>
<th>a</th>
<th>¬a</th>
<th>¬a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

MLTL formula \(\varphi\) evaluated over system trace \(\pi\):
\[
\forall i : 0 \leq i \leq \text{Mission Time } \pi, i \models \varphi.
\]

MLTL Runtime Benchmark Example:
- \(\pi = a, \neg a, \neg a, a, a, a, a, a, a, a\)
- \(\varphi = \text{ALWAYS}_5(a)\)
- \(O = \langle 0, F \rangle, \langle 1, F \rangle, \langle 2, F \rangle, \langle 3, T \rangle, \langle 4, T \rangle, \ldots\)

Example: Benchmark Generation via Formula Progression

Inputs: \( \varphi = \mathcal{F}[2,3]a \)
\( \pi = \neg a, \neg a, a \)

Figure: The schema of \( \text{prog}(F_{[2,3]}a, \pi = \{\neg a\}{\neg a\}{a}) \).
Example: Benchmark Generation via Formula Progression

\[ \text{Inputs: } \varphi = F_{[2,3]}a \]
\[ \pi = \neg a, \neg a, a \]

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

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

---

Example: Benchmark Generation via Formula Progression\(^{10}\)

\[\text{Inputs : } \varphi = F_{[2,3]}a \]
\[\pi = \neg a, \neg a, a\]

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

\[
\begin{align*}
\text{prog}(\varphi, \pi^1(= \{\neg a\})) &= F_{[1,2]}a \\
\text{prog}(\varphi, \pi^2(= \{\neg a\}\{\neg a\})) &= F_{[0,1]}a
\end{align*}
\]

\(^{10}\)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

Inputs: \( \varphi = F_{[2,3]}a \)
\( \pi = \neg a, \neg a, a \)

\( F_{[2,3]}a \) \( \rightarrow \) \( \neg a \) \( F_{[1,2]}a \) \( \rightarrow \) \( \neg a \) \( F_{[0,1]}a \) \( \rightarrow \) \( a \) \( F_{[0,1]}a \)

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

\[
\begin{align*}
prog(\varphi, \pi^1(= \{\neg a\})) &= F_{[1,2]}a \\
prog(\varphi, \pi^2(= \{\neg a\} \{\neg a\})) &= F_{[0,1]}a \\
prog(\varphi, \pi^3(= \{\neg a\} \{\neg a\} \{a\})) &= \text{TRUE}
\end{align*}
\]

\( JIANwen Li \) and Kristin Yvonne Rozier. “MLTL Benchmark Generation via Formula Progression.” In Runtime Verification (RV18), Springer-Verlag, 2018.
We can generate MLTL RV benchmarks with an MLTL Satisfiability Solver\textsuperscript{11}

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 . . .**