What's in a Name?
Linear Temporal Logic Literally Represents Time Lines

Visualization of Linear Temporal Logic

Runming Li[†]
j.w.w. Keerthana Gurushankar[†], Marijn Heule[†], and Kristin-Yvonne Rozier[‡]

Carnegie Mellon University[†] Iowa State University[‡]

October 2, 2023

Overview

Goal

Understand the challenges in Linear Temporal Logic model checking, and how *visualization* can help mitigate the challenges.

1. Linear Temporal Logic (LTL)
2. LTL Model Checking
3. LTL Visualization
4. Conclusion

Linear Temporal Logic (LTL)

What is LTL?

Linear Temporal Logic

Linear Temporal Logic (LTL) is a modal temporal logic that allows ability to reason about events that happen over time.

What is LTL?

Linear Temporal Logic

Linear Temporal Logic (LTL) is a modal temporal logic that allows ability to reason about events that happen over time.

Example scenarios in software systems where we want to reason about time:

- Every request will eventually lead to a response.
- Events a and b cannot happen at the same time.

Inductive Definition of LTL

$$
\begin{aligned}
\Phi ::= \; & a, b, c && \text{(atoms)} \\
\mid \; & \neg\Phi && \text{(negation)} \\
\mid \; & \Phi \vee \Phi && \text{(disjunction)} \\
\mid \; & \Phi \wedge \Phi && \text{(conjunction)} \\
\mid \; & \Phi \rightarrow \Phi && \text{(implication)} \\
\mid \; & \Box\Phi && \text{(always)} \\
\mid \; & \Diamond\Phi && \text{(eventually)} \\
\mid \; & \mathcal{X}\Phi && \text{(next)} \\
\mid \; & \Phi\,\mathcal{U}\,\Phi && \text{(until)}
\end{aligned}
$$

Inductive Definition of LTL

$$
\begin{aligned}
\Phi ::= {} & a, b, c && \text{(atoms)} \\
| \; & \neg\Phi && \text{(negation)} \\
| \; & \Phi \vee \Phi && \text{(disjunction)} \\
| \; & \Phi \wedge \Phi && \text{(conjunction)} \\
| \; & \Phi \rightarrow \Phi && \text{(implication)} \\
| \; & \Box\Phi && \text{(always)} \\
| \; & \Diamond\Phi && \text{(eventually)} \\
| \; & \mathcal{X}\Phi && \text{(next)} \\
| \; & \Phi \,\mathcal{U}\, \Phi && \text{(until)}
\end{aligned}
$$

$\Box\Phi = \Phi$ is Always true
$\Diamond\Phi = \Phi$ is Eventually true
$\mathcal{X}\Phi = \Phi$ is true in the Next time step
$\Phi \,\mathcal{U}\, \Psi = \Phi$ is true Until Ψ is true

Example LTL Formula

Example: Liveness

$$\Box(\text{request} \rightarrow \Diamond\text{response})$$

Example LTL Formula

Example: Liveness

$$\square(\text{request} \rightarrow \lozenge\text{response})$$

"Every request will eventually lead to a response."

Example LTL Formula

Example: Liveness

$$\Box(\text{request} \rightarrow \Diamond\text{response})$$

"Every request will eventually lead to a response."

Example: Mutal Exclusion

$$\Box(\neg(a \wedge b))$$

Example LTL Formula

Example: Liveness

$$\Box(\text{request} \rightarrow \Diamond\text{response})$$

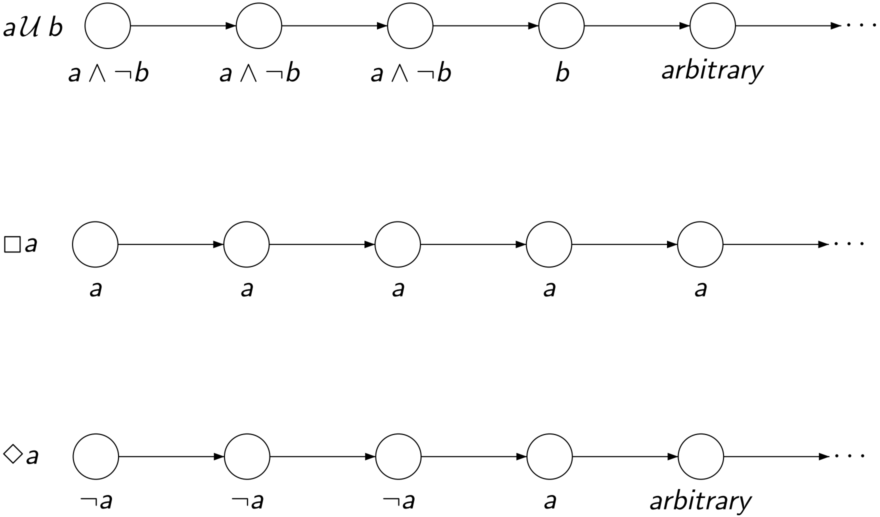
"Every request will eventually lead to a response."

Example: Mutal Exclusion

$$\Box(\neg(a \wedge b))$$

"Events a and b cannot happen at the same time."

LTL formulas are commonly thought as time lines

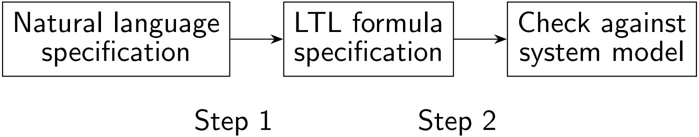


LTL Model Checking

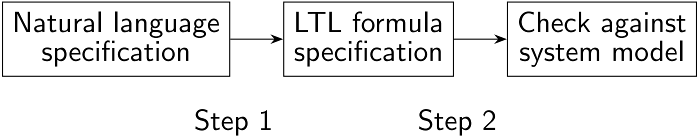
Model Checking

- A software system can be described as a finite state model $\mathcal{M}$.
- A specification can be described as an LTL formula Φ.
- Question to answer: does model $\mathcal{M}$ satisfies specification Φ?

Model Checking

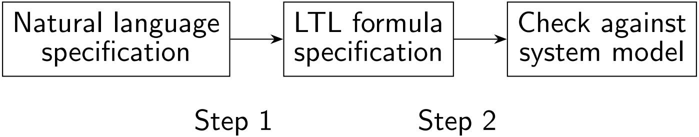


Model Checking



- Step 2 has elegant algorithms, proven correct[3].

Model Checking



- Step 2 has elegant algorithms, proven correct[3].
- Step 1 remains a human effort.

Can you tell the difference?

A NASA Rocket Scientist was given this English specification:
"**p oscillates every time step**"
She wrote two possible LTL formulas:

Can you tell the difference?

A NASA Rocket Scientist was given this English specification:
"**p oscillates every time step**"
She wrote two possible LTL formulas:

$$\Phi = \text{Always}\ ((p \wedge \text{Next}\ \neg p) \vee (\neg p \wedge \text{Next}\ p))$$

$$\Psi = \text{Always}\ ((p \wedge \text{Next}\ \neg p) \wedge (\neg p \wedge \text{Next}\ p))$$

Which one is correct?

A canonical approach

p	$\mathcal{X}p$	$\neg p$	$\mathcal{X}\neg p$	$p \wedge \mathcal{X}\neg p$	$\neg p \wedge \mathcal{X}p$	Φ	Ψ
T	T	F	F	F	F	F	F
T	F	F	T	T	F	T	F
F	T	T	F	F	T	T	F
F	F	T	T	F	F	F	F

A canonical approach

p	$\mathcal{X}p$	$\neg p$	$\mathcal{X}\neg p$	$p \wedge \mathcal{X}\neg p$	$\neg p \wedge \mathcal{X}p$	Φ	Ψ
T	T	F	F	F	F	F	F
T	F	F	T	T	F	T	F
F	T	T	F	F	T	T	F
F	F	T	T	F	F	F	F

What can we do better?

What can we do better?

If only there is a way to intuitively know whether the LTL formula matches up the timeline we have in mind.

LTL Visualization

Which one is correct?

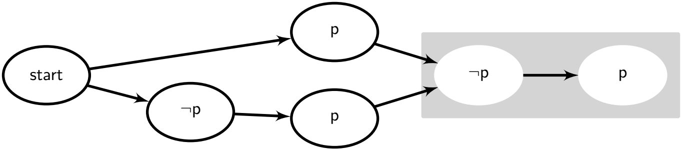


Figure: Timeline for $\Phi = \Box((p \wedge \mathcal{X}\neg p) \vee (\neg p \wedge \mathcal{X} p))$.

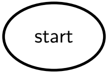


Figure: Timeline for $\Psi = \Box((p \wedge \mathcal{X}\neg p) \wedge (\neg p \wedge \mathcal{X} p))$ (a single "start" means every time step is $\perp$).

LTL to Timeline Conversion

Our contribution: an algorithm and a tool that converts any LTL formula into its corresponding timeline visualization

1. LTL to state-based Nondeterministic Buchi Automata (NBA)
2. NBA to ω-regular expression
3. Heuristics based ω-regular expression simplification
4. ω-regular expression to timeline

LTL to state-based Buchi Automata (BA)

Büchi Automata: the normal automata you know, except for the accepting condition

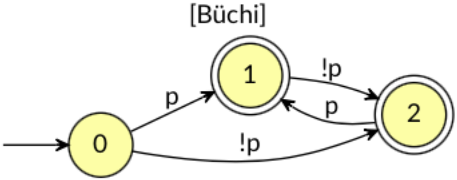


Figure: Example Buchi Automata for $\square((p \wedge \mathcal{X}\neg p) \vee (\neg p \wedge \mathcal{X}p))$

This step is very well-studied [1] and our tool uses *Spot*.

BA to ω-regular expression

Definition

Regular expression and ω-regular expression

$$A ::= \epsilon \mid \emptyset \mid p(\in \Sigma) \mid AA \mid A + A \mid A^*$$
$$B ::= A^\omega \mid AB \mid B + B$$

Our Σ is the set of propositional logic formula.

BA to ω-regular expression

Definition

Regular expression and ω-regular expression

$$A ::= \epsilon \mid \emptyset \mid p(\in \Sigma) \mid AA \mid A + A \mid A^*$$
$$B ::= A^\omega \mid AB \mid B + B$$

Our Σ is the set of propositional logic formula.

Example

$$ab^\omega$$

represents an ω-regular expression whose first word is a and all (infinite) remaining words are b.

BA to ω-regular expression

Definition

$A^0_{(s,f)}$ represents the regular expression that corresponds to all paths from state s reaching state f for the first time

$A^1_{(f,f)}$ represents the regular expression that corresponds to all paths from state f to itself

BA to ω-regular expression

Definition

$A^0_{(s,f)}$ represents the regular expression that corresponds to all paths from state s reaching state f for the first time
$A^1_{(f,f)}$ represents the regular expression that corresponds to all paths from state f to itself

$$B = \sum_{f \in F} A^0_{(s,f)} \left(A^1_{(f,f)}\right)^\omega$$

BA to ω-regular expression

$$B = \bigplus_{f \in F} A^0_{(s,f)} (A^1_{(f,f)})^\omega$$

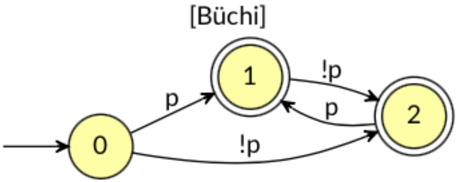


Figure: Example Büchi Automata for $\square((p \wedge \mathcal{X}\neg p) \vee (\neg p \wedge \mathcal{X} p))$

Example

Generated ω-regular expression:

$$(p(\neg pp)^\omega) + (\neg p(p\neg p)^\omega)$$

ω-regular expression to timeline

Every ω-regular we generate is the form of

$$A_1 A_2^\omega + A_3 A_4^\omega + \cdots + A_{2n-1} A_{2n}^\omega$$

ω-regular expression to timeline

Every ω-regular we generate is the form of

$$A_1 A_2^\omega + A_3 A_4^\omega + \cdots + A_{2n-1} A_{2n}^\omega$$

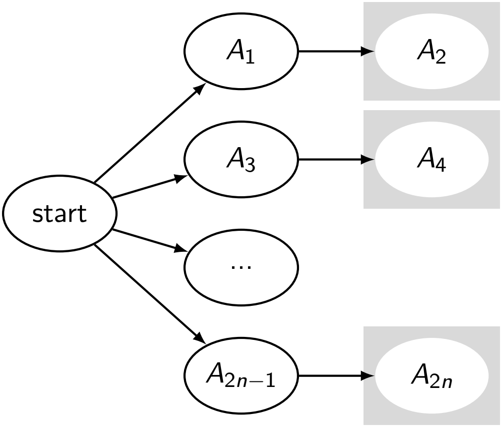


Figure: Generic timeline construction of $A_1 A_2^\omega + A_3 A_4^\omega + \cdots + A_{2n-1} A_{2n}^\omega$.

Heuristics based ω-regular expression simplification

The ω-regular expression generated may not be the "simplest" to visualize.

Idea

Regular expression equivalence forms a congruence: therefore we can replace regular expressions with equivalent "simpler" ones everywhere.

Heuristics based ω-regular expression simplification

The ω-regular expression generated may not be the "simplest" to visualize.

Idea

Regular expression equivalence forms a congruence: therefore we can replace regular expressions with equivalent "simpler" ones everywhere.

- Syntactical equivalence: equivalent by algebraic laws
- Semantical equivalence: equivalent by representing the same set of words

Heuristics based ω-regular expression simplification

$$
\begin{aligned}
r_1 + r_1 r_2^* &\implies r_1 r_2^* \\
r + r &\implies r \\
r_1 + r_2^* r_1 &\implies r_2^* r_1 \\
(r^*)^\omega &\implies r^\omega \\
(r_1 r_2^*) r_2^\omega &\implies r_1 r_2^\omega \\
(r_1 r_2) r_2^\omega &\implies r_1 r_2^\omega \\
r^* r^\omega &\implies r^\omega \\
r r^\omega &\implies r^\omega
\end{aligned}
$$

Heuristics based ω-regular expression simplification

Example

$\Phi = \Box(a \rightarrow \Diamond(\neg a))$

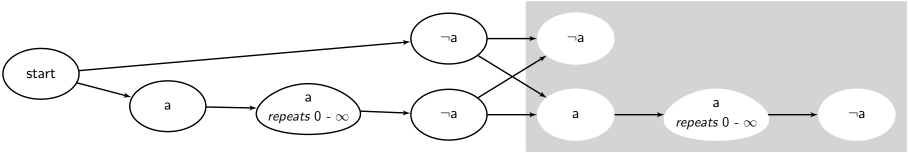


Figure: Timeline visualization for $((\neg a)|(aa^*(\neg a)))((\neg a)|(aa^*(\neg a)))^\omega$.

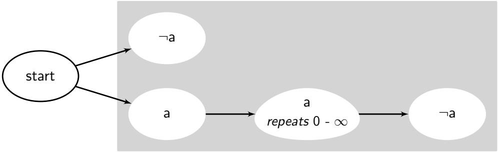


Figure: Timeline visualization for $((\neg a)|(aa^*(\neg a)))^\omega$.

Tool showcase

Example

"[i]f a TSAFE command is sent to an aircraft, controller/AutoResolver should then hand off the control of this aircraft."[4]

$$\square(\texttt{tsafe.TSAFE\_command1} \wedge \texttt{controller.CTR\_control\_1}$$
$$\rightarrow \mathcal{X}(\neg\texttt{controller.CTR\_control\_1}))$$

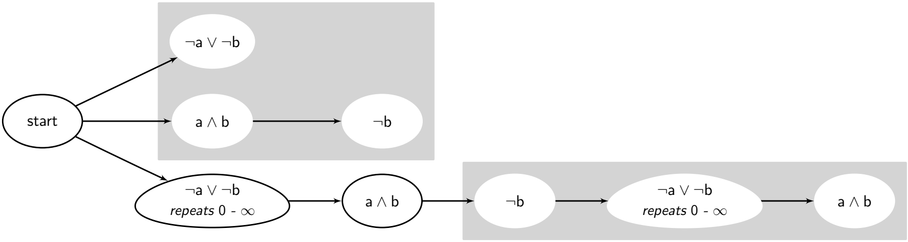


Figure: Timeline for $\square(a \wedge b \rightarrow \mathcal{X}(\neg b))$.

Tool showcase

Example

Random LTL formula generated by [2]:

$$p_2 \wedge (\Diamond\Box p_0 \,\mathcal{U}\, \mathcal{X}(\Box p_1 \wedge (((p_0 \rightarrow p_2) \wedge (p_2 \rightarrow p_0))\,\mathcal{U}\,\Diamond p_0)))$$

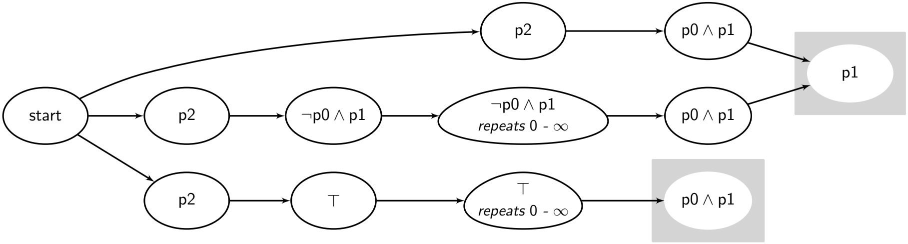


Figure: Timeline for $p_2 \wedge (\Diamond\Box p_0 \,\mathcal{U}\, \mathcal{X}(\Box p_1 \wedge (((p_0 \rightarrow p_2) \wedge (p_2 \rightarrow p_0))\,\mathcal{U}\,\Diamond p_0)))$.

Artifact

Our artifact is available online.

`https://github.com/EULIR/ltl-explainability`

Figure: Open Research Object Badge

Figure: Research Object Reviewed Badge

Conclusion

Conclusion and future work

We present an algorithm and a tool to visualize LTL formula, in attempt to make LTL-based formal verification more intuitive and accessible.

Conclusion and future work

We present an algorithm and a tool to visualize LTL formula, in attempt to make LTL-based formal verification more intuitive and accessible.

Future work

- User study to gather data from a representative audience of system engineers regarding what timeline visualizations help most with formula validation.
- More, faster implementation optimizations over simplification algorithms.

Thinking about visualization

From anonymous reviewer

This paper takes an interesting approach, being based in the essential mathematical theory of the objects being visualised, rather than just ad hoc accidental properties of software artefacts.

Acknowledgements

We thank anonymous reviewers for feedback and suggestions. This work was partially supported by the National Science Foundation (NSF) under grants CCF-2015445, CAREER-1664356, and CCRI-2016592.

References

[1] Alexandre Duret-Lutz et al. "From Spot 2.0 to Spot 2.10: What's New?" In: *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*. Vol. 13372. Lecture Notes in Computer Science. Springer, Aug. 2022, pp. 174–187. DOI: 10.1007/978-3-031-13188-2_9.

[2] Alexandre Duret-Lutz. "Manipulating LTL formulas using Spot 1.0". In: *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*. Vol. 8172. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Oct. 2013, pp. 442–445. DOI: 10.1007/978-3-319-02444-8_31.

[3] K.Y. Rozier. "Linear Temporal Logic Symbolic Model Checking". In: *Computer Science Review Journal* 5.2 (May 2011), pp. 163–203. DOI: doi:10.1016/j.cosrev.2010.06.002. URL: http://dx.doi.org/10.1016/j.cosrev.2010.06.002.

[4] Yang Zhao and Kristin Yvonne Rozier. "Formal Specification and Verification of a Coordination Protocol for an Automated Air Traffic Control System". In: *Science of Computer Programming Journal* 96.3