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

Runming Li[§]
Carnegie Mellon University
Pittsburgh, PA, USA
runmingl@andrew.cmu.edu

Keerthana Gurushankar[§]
Carnegie Mellon University
Pittsburgh, PA, USA
kgurusha@andrew.cmu.edu

Marijn J.H. Heule
Carnegie Mellon University
Pittsburgh, PA, USA
marijn@cmu.edu

Kristin Yvonne Rozier
Iowa State University
Ames, IA, USA
kyrozier@iastate.edu

***Abstract*—Linear Temporal Logic (LTL) is arguably the most popular specification language for formal verification of safety-critical systems. However, LTL formulas can be unintuitive and error-prone for human practitioners to specify and validate. Meanwhile, drawing timelines remains one of the most popular methods for specifying and validating requirements for industrial system designs, such as in aerospace operational concepts. Therefore, we provide a new timeline tool for visualizing LTL specifications as timelines, providing provably-correct, intuitive equivalents between these two specification formats. Our tool generates timeline visualizations by translating LTL formulas to intermediate representations as Büchi automata and then regular expressions, and finally simplifying and visualizing the expressions. We provide an algorithm for this visualization, a theoretical soundness analysis, and an implementation.**

***Index Terms*—Modal and Temporal Logics, Logic and Verification, Regular languages**

I. Prologue

Requirement specification is a central step in the development of safety-critical systems. For example, here is a real-world requirement specification from an air traffic control system [1]

> "If a TSAFE command is sent to an aircraft, controller/AutoResolver should then hand off the control of this aircraft"

Such natural language requirements are often ambiguous and not amenable to formal analysis, so we must instead specify such requirements using rigorous formalized semantics for the purpose of verification and validation. The above requirement, when translated to Linear Temporal Logic (detailed in Section II-A), may be expressed as follows:

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

Though, as [1] demonstrates, validating that the resulting formula represents the intended requirement is difficult, even for professionals; in this case study a subtle misconception in the initial formalization of a requirement hid an important safety concern present in the early design-stage requirements. Using the play-on-words "Little Tricky Logic," [2] points to endemic misconceptions in the understanding of LTL.

The need for validation of requirements formalizations has been widely recognized, and tools for several different formalisms have contributed strategies for breaking up the formalisms into patterns and adding GUIs to make validation easier. For example, the web-based modeling tool piStar-ext adds a GUI with text, symbols, and diagrams to enable users to better validate iStar-language models [3]. Hanfor, a GUI, web-based tool supported formalization of system requirements into an LTL-like structured natural language patterns for several industry projects in the automotive and railway domains [4]. The ForeMoSt framework enables validation of safety assurance cases by translating structured arguments into formal strategies and validating them automatically using the Lean theorem prover [5]. AutoTap provides users with a GUI utilizing structured English to write and edit LTL formulas representing Trigger-Action Programs (another form of assume-guarantee contracts) for smart devices and cloud services [6].

There has been a recent boom in validation capabilities for the related, but arguably simpler, specification logic Mission-time LTL (MLTL) [7], which adds finite, integer-bounded, closed intervals to the temporal operators of LTL. The runtime verification engine R2U2 includes a web-based GUI enabling specifiers to visualize properties of (sets of) MLTL runtime monitors, such as their shared subformulas and resource usage on embedded systems [8]. The FRET tool's GUI color-codes segments of structured natural language to elicit more accurate MLTL formulas from system designers [9]. The WEST tool provides an interactive GUI for MLTL formula validation via regular expressions and truth-table-like visualizations [10]. Now we need tools like these for LTL.

The common and intuitive technique of validating LTL formulas via sets of positive and negative examples (i.e., demonstrating traces that both satisfy and violate a given LTL formula) is insufficient for understanding and explainability of LTL specification [11]. Thus, a significant hurdle remains: how

[§]These authors contributed equally to this work

can we convincingly demonstrate to the humans in the loop, from system designers to certifiers, that the analyzed formulas truly represent the desired system requirements?

We take a simple idiomatic approach to address this problem. Since Linear Temporal Logic formulas differentiate temporal information, we devise a tool, `ltl2timeline`, which can draw timelines to depict the satisfying traces of the LTL formula, and thereby represent the specified behavior in a more natural, human-intelligible form.

Our tool, `ltl2timeline`, uses a remarkably simple algorithm: transforming an input LTL formula through a sequence of automata and regular expression-based intermediate representations. Yet, it is surprisingly effective at providing small representations for a range of LTL formulas. It visualizes the above air traffic control requirement behavior with the timeline shown in Example V.1.

We make the following contributions in our paper:

- We provide the tool, `ltl2timeline`, which takes LTL formulas and synthesizes timeline visualizations for them.
- We prove correctness in Section IV. We show that for every LTL formula, the timeline outputted indeed represents the set of satisfying traces.
- We showcase the results of our tool on a range of example formulas from real-life industrial verification efforts in Section V and Section VI, including discussing the air traffic control example above.

II. Setting the stage

We begin by setting up the prerequisite definitions involved in our work. First, we provide the definition and semantics for LTL. In the following subsections, we define (state-based) Büchi automata and ω-regular expressions, which are generalizations of the well-known deterministic finite automata (DFA) and regular expressions, for the case of infinite words. Lastly, we describe the graphic visualizations we will use to depict timelines.

A. Linear Temporal Logic (LTL)

Definition II.1 (Linear Temporal Logic (LTL))**.** The syntax of an LTL formula over a set of atomic propositions $\mathcal{AP}$, where $p \in \mathcal{AP}$ is a propositional variable, consists of the following grammar:

$$\varphi = p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \Box\varphi \mid \Diamond\varphi \mid \mathcal{X}\varphi \mid \varphi \,\mathcal{U}\, \psi \mid \varphi \,\mathcal{R}\, \psi$$

Remark II.1. Intuitively, $\Box\varphi$ says that formula φ is true at every time step; $\Diamond\varphi$ says that formula φ is true either now or at some time in the future; $\mathcal{X}\varphi$ says formula φ is true at the next time step immediately after the current one; $\varphi \,\mathcal{U}\, \psi$ says that formula φ is true *until* (strictly before) such time formula ψ becomes true (and ψ is either true now or in the future); and $\varphi \,\mathcal{R}\, \psi$ says that formula ψ must be true now and remain true unless formula φ becomes true, after which point we can *release* ψ (if formula φ is never true, then formula ψ must remain true at all times).

Remark II.2. The `ltl2timeline` tool uses a concrete input syntax of ASCII symbols to represent those logical connectives, in order to simplify typesetting connectives such as $\Box$ and $\Diamond$. For reference, the concrete syntax appears in Appendix A.

Definition II.2 (Semantics of LTL)**.** Let $\pi : \omega \rightarrow 2^{\mathcal{AP}}$ be a *computation* or trace that stores the truthhood and falsehood of every atomic proposition at every time step, where ω is the set of natural numbers that label the time step. Then we say $\pi, i \models \varphi$, that is, computation π starting from time $i \in \omega$ models LTL formula φ whenever the result of this evaluation satisfies φ; see Rozier [12] for the full LTL semantics.

B. State-based Büchi automata (BA)

Definition II.3 (ω-word)**.** Let Σ be an alphabet. An ω-word or infinite run of Σ, is an infinite string $s = (s_0, s_1, s_2, \dots)$ where each $s_i \in \Sigma$.

Definition II.4 (Büchi automaton (BA))**.** A Büchi automaton is a 5-tuple, $(Q, \Sigma, \delta, s, F)$ consisting of

1) a finite set of states Q
2) a finite alphabet of input symbols Σ
3) a transition relation $\delta \subseteq (Q \times \Sigma) \times Q$
4) an initial or start state called $s \in Q$
5) a set of accepting states $F \subseteq Q$

A BA accepts an infinite run iff at least one of its infinitely visited states is in F.

C. ω-regular expressions

Definition II.5 (Regular expression)**.** Let a be a symbol in Σ, A be a regular expression, and ϵ be the empty expression. We define regular expressions using the following grammar:

$$A = \emptyset \mid \epsilon \mid a \mid AA \mid A + A \mid A^*$$

Definition II.6 (Semantics of regular expressions)**.** Let $\mathscr{L}(A)$ denote the set of propositional formulas accepted by regular expression A, also called the language accepted by A. We define $\mathscr{L}(A)$, inductively as:

$$\begin{aligned}
\mathscr{L}(\emptyset) &= \emptyset \\
\mathscr{L}(\epsilon) &= \{\epsilon\} && (\epsilon \text{ denotes the empty string}) \\
\mathscr{L}(a) &= \{a\} \\
\mathscr{L}(A_1 A_2) &= \{s_1 s_2 \mid s_1 \in L(A_1) \text{ and } s_2 \in L(A_2)\} \\
\mathscr{L}(A_1 + A_2) &= L(A_1) \cup L(A_2) \\
\mathscr{L}^{(0)}(A) &= \{\epsilon\} \\
\mathscr{L}^{(i+1)}(A) &= \{s_1 s_2 \mid s_1 \in L(A) \text{ and } s_2 \in L^{(i)}(A)\} \\
\mathscr{L}(A^*) &= \bigcup_{i \geq 0} L^{(i)}(A)
\end{aligned}$$

Remark II.3. For the purpose of our tool, we define Σ as the set of propositional logic formulas:

$$a \in \Sigma = p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi$$

Definition II.7 (ω-regular expressions)**.** Our definition II.5 of regular expressions concerns only finite-length strings. However, since LTL formulas reason about events that happen over an infinite-length timeline, we need to model them using infinite regular expressions (i.e., ω-regular expressions), which we define via the following grammar:

$$B = A^\omega \mid AB \mid B + B$$

Definition II.8 (Semantics of ω-regular expressions)**.** Let Σ^ω denote the set of infinite-length strings over fixed alphabet Σ. Let $\mathscr{L}_\omega(B)$ denote the ω-language accepted by ω-regular expression B. Then we define $\mathscr{L}_\omega(B)$ inductively as:

$$\mathscr{L}_\omega(A^\omega) = \{s_1 s_2 s_3 \cdots \mid s_i \in \mathscr{L}(A) \text{ and } i \geq 1\} \quad (\epsilon \notin \mathscr{L}(A))$$
$$\mathscr{L}_\omega(AB) = \{s_1 s_2 \mid s_1 \in \mathscr{L}(A) \text{ and } s_2 \in \mathscr{L}_\omega(B)\}$$
$$\mathscr{L}_\omega(B_1 + B_2) = \mathscr{L}_\omega(B_1) \cup \mathscr{L}_\omega(B_2)$$

Remark II.4. Definitions II.5 and II.7 are the standard definitions of regular expressions and ω-regular expressions, respectively. We include them here for completeness. For the remainder of the paper, we denote arbitrary regular expressions using A and ω-regular expressions by B.

D. Timelines

We present timelines as graphic visualizations containing the following features:

- Every timeline starts with a node named "start."
- Every node represents one time step, and each node has a propositional logic formula ψ, which specifies the behavior of atomic propositions at that time step. The formula ψ must be true at that time step. If $\psi = 1$, that means all atomic propositions can behave arbitrarily.
- An egg-shaped node with a caption "repeats 0 - ∞" means to repeat the (one time step) current node for arbitrarily finitely many times.
- A node with label "$\cdots$" means to repeat the pattern prior to it and after it arbitrarily finitely many times.
- The grey box means repeat infinitely. Once we reach the end of a timeline in the grey box, we must reenter the same grey box from any of its (left) starting points.
- Timelines can occur in parallel, signifying that any of the parallel timelines could happen.

The manifestation of timelines we use here is one of possibly many timeline representations that we devised for the purpose of visualizing LTL formulas. We show how to construct such timelines from LTL formulas in Section IV-C.

Example II.1. In Figure 1, one can reason about two parallel timelines: the upper timeline starts with p holding in the first time step, followed by entering the grey box with one step of $\neg p$ and one step of p. Then at the end of the grey box, we reenter the box, with the next time step being $\neg p$, and so on. The lower timeline starts with one step of $\neg p$ and one step of p outside the grey box, and then enters the infinite run of $\neg p$ and p repeating.

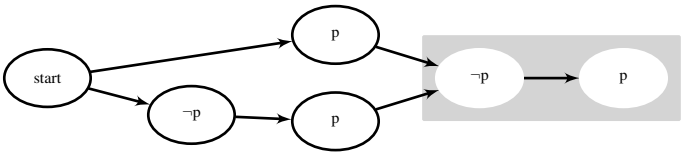


Fig. 1: Example of one possible timeline for the specification "p oscillates every time step."

Example II.2. In Figure 2, one can reason about one timeline: the atomic proposition a is false for finitely many time steps as signified by the second node (note that this could be zero time steps); followed by one node with a that substantiates the specification of "a is eventually true." Once a is true at some point, the later time steps can behave arbitrarily as signified by the infinite run of $\top$ (true) in the grey box.

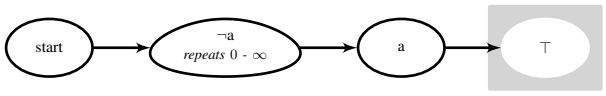


Fig. 2: Example of a timeline for the specification "a is eventually true."

III. Algorithm: "Though she be but little, she is fierce[§]"

On a high level, our algorithm for converting LTL formulas into timeline visualizations (depicted in Figure 3) works as follows.

1) Convert the given LTL formula to its corresponding Büchi automaton.
2) Derive the ω-regular expression corresponding to the Büchi automaton.
3) Simplify the derived ω-regular expression. (Note that this step represents a stylistic choice balancing size, complexity, and clarity.)
4) Visualize the ω-regular expression as a timeline according to its structure.

A. LTL to BA

The automata-theoretic approach [13] to evaluating LTL (e.g., via reducing LTL formulas to their corresponding Büchi automata) has been well-studied [14], [15], [16], [17], [18]. Our tool `ltl2timeline` uses SPOT [19][1] for this step, which provides a provably-correct implementation of the translation from LTL formulas to minimally-sized Büchi automata.

B. BA to ω-regex

We translate Büchi automata to ω-regular expressions by first finding the regular expressions for paths from the start state to some final state (say r_{sf}), and for those for paths looping from the final state back to itself (say r_{ff}), and finally combining those to form the ω-expression for satisfying

[§]William Shakespeare, *A Midsummer Night's Dream*

[1]We used SPOT version 2.11.

Fig. 3: `ltl2timeline` algorithm outline

runs $\left(+_{f\in F} r_{\mathrm{sf}} r_{\mathrm{ff}}^{\omega}\right)$. Our visualization algorithm finds regular expressions for finite paths by iteratively deleting interior nodes in the digraph of the automaton.

The algorithm to convert Büchi automata to ω-regular expressions `ba2wregex` combines Algorithms 1–4 below.

Algorithm 1 reduce_nfa

Input: (G, v): NFA G with state v that is not initial or final
Output: Deletes state v from G while ensuring $\mathscr{L}(G)$ remains unchanged
  for every $u \xrightarrow{r_{\mathrm{in}}} v, v \xrightarrow{r_{\mathrm{out}}} w$ **do**
    if v has a self-edge $v \xrightarrow{r_{\mathrm{loop}}} v$ **then**
      replace edges $u \xrightarrow{r_{\mathrm{in}}} v, v \xrightarrow{r_{\mathrm{out}}} w$ with $u \xrightarrow{r_{\mathrm{in}} r_{\mathrm{loop}}{}^* r_{\mathrm{out}}} w$
    else
      replace edges $u \xrightarrow{r_{\mathrm{in}}} v, v \xrightarrow{r_{\mathrm{out}}} w$ with $u \xrightarrow{r_{\mathrm{in}} r_{\mathrm{out}}} w$
    end if
  end for
  delete node v from G

Algorithm 2 nfa2regex

Input: (G, s, f): NFA G with initial state s and final state f
Output: The regular expression corresponding to all paths from s to f
  while there exists an interior vertex v **do**
    reduce_nfa(G, v)
    combine multi-edges, i.e., convert $r1 : u \to w, r2 : u \to w$ to $r1|r2 : u \to w$
  end while
  return $(r_{\mathrm{ss}}|r_{\mathrm{sf}} r_{\mathrm{ff}}^* r_{\mathrm{fs}})^* r_{\mathrm{sf}} r_{\mathrm{ff}}$

Algorithm 3 nfa2regex_firstvisit

Input: (G, s, f): NFA G with initial state s and final state f
Output: The regular expression of all paths from s reaching f for the first time
  delete all out edges from f in G
  return nfa2regex(G, s, f)

Algorithm 4 ba2wregex

Input: G, a Büchi automaton
Output: The ω-regular expression recognized by G
  return $\bigcup_{f\in F}$ (nfa2regex_firstvisit(G, s, f)nfa2regex(G, f, f))

C. ω-regex simplification

The ω-regular expression generated in Section III-B may not be the "simplest" for the purpose of visualizing the timeline. We have observed multiple patterns in the resulting ω-regular expression that could be simplified. For example, an ω-regular expression of the form of $r^* r^\omega$ represents the same timeline as r^ω, but the latter is more intuitive and concise. For this purpose, we devised some simplification rules in our tool, based on our observation of common patterns in the generated ω-regular expressions.

There is no agreed-upon canonical form for regular expressions representing LTL formulas, hence we do not hope to find the shortest possible regular expression for visualization. Regular expression simplification comes down to regular expression equivalence checking, which is computationally hard [20]. For purpose of simplification, one could perform a search over equivalent regular expressions and decide which one is simpler to use. However, this strategy is expensive in terms of the running time, and for the purpose of visualization, we did not use this strategy in our tool. We consider simplification for the purpose of finding the most intuitive regular expression for a given LTL formula to be an interesting direction for future work.

Rule-based simplification: Here we show a demonstrating subset of the simplification rules we encoded. In theory one could add more rules to the tool, so long as they are sound; here we only choose to encode the rules that represent common patterns we have observed in the generated ω-regular expressions.

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

Result of simplification: These simplification rules lead to more intuitive representations of timelines. Here we demonstrate their effects using an example.

Example III.1. Using our algorithm, the LTL formula $\varphi = \square(a \to \Diamond(\neg a))$ generates the un-simplified ω-regular expression $((\neg a)|(aa^*(\neg a)))((\neg a)|(aa^*(\neg a)))^\omega$, and the simplified version $((\neg a)|(aa^*(\neg a)))^\omega$, which correspond to the two timelines in Figure 4 and Figure 5, respectively.

Remark III.1. Both simplified and un-simplified ω-regular expressions could generate correct timeline representations that faithfully represent the set of satisfying traces of the original LTL formula φ. Nonetheless, we, as human users,

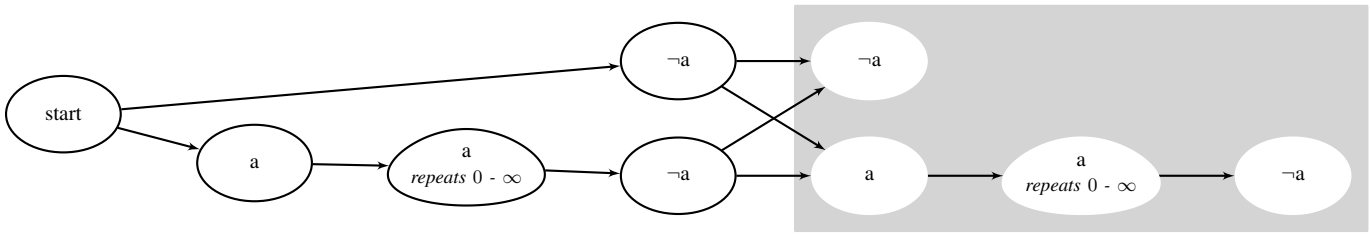


Fig. 4: Timeline visualization for ω-regular expression $((\neg a)|(aa^*(\neg a)))((\neg a)|(aa^*(\neg a)))^\omega$.

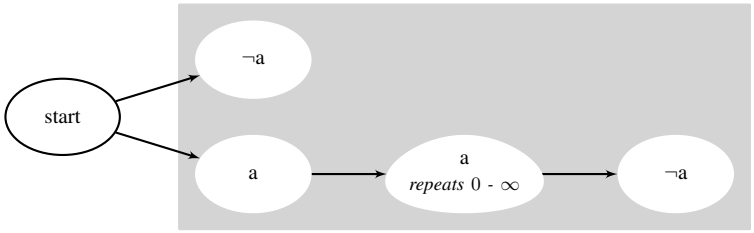


Fig. 5: Timeline visualization for simplified ω-regular expression $((\neg a)|(aa^*(\neg a)))^\omega$.

oftentimes find the simplified version more intuitive to reason about.

D. ω-regex to timeline

Our tool uses Graphviz [21] to achieve the timeline visualization step. By construction of our algorithm, every ω-regular expression is in the form of

$$A_1A_2^\omega + A_3A_4^\omega + \cdots + A_{2n-1}A_{2n}^\omega$$

where A_i are regular expressions, A_{2i-1} could be ϵ, and $\epsilon \notin \mathscr{L}(A_{2i})$. At a high level, we visualize each regular expression A_i as a set of accepted inputs. We view each union operator as a set of parallel timelines. We denote that each A_{2i-1} gets concatenated with A_{2i}, which then repeats infinitely many times by surrounding these repeating final nodes with grey boxes. Figure 6 presents a generic timeline resulting from this construction pattern.

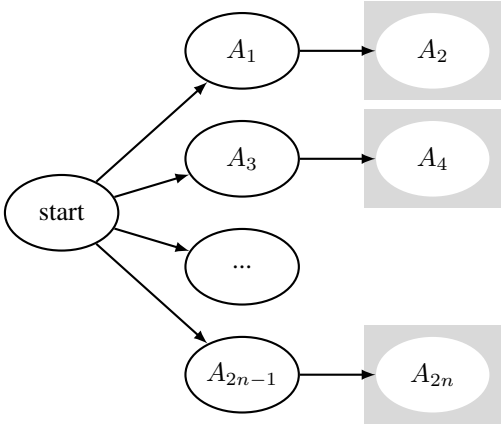


Fig. 6: Generic timeline construction created from $A_1A_2^\omega + A_3A_4^\omega + \cdots + A_{2n-1}A_{2n}^\omega$.

IV. Theoretical Analyses: "If we are true to ourselves, we can not be false to anyone[§]"

We prove the correctness of our translation pipeline. The correctness of the translation from LTL formulas to Büchi automata stems from using SPOT [19]. Below, we prove the correctness of (i) the translation from Büchi automata to ω-regular expressions, and (ii) the (ω-)regular expression rewrite rules we apply. Lastly, we outline the correctness of our timeline visualizations.

A. Correctness of Regex Translation

Lemma IV.1. *For any NFA G and any state v in G that is neither a start state or a final state,* ***reduce_nfa****(G, v) preserves the regular language accepted by G.*

Proof. Let G' be the graph of G post reduction by the application of **reduce_nfa**(G, v). We show that the trace of every path accepted by G is also accepted by G'. Suppose a path accepted by G does not pass through v, clearly the lemma holds. Otherwise, suppose the path passes through v; since v is an interior node, v cannot be the first or last in the path. Thus, for every pass through v, let $u \neq v$ be the last node passed before entering v, and likewise w be the first node after exiting v. We show that the regular language of sub-traces from u to w in G (shown in Figure 7) is identical to that in G'. Suppose

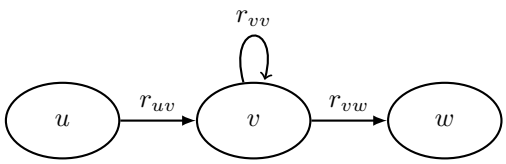


Fig. 7: A path from u to w in G.

there is no loop at v, the path from u to w must simply be

[§]William Shakespeare, *Hamlet*

$u \xrightarrow{r_{uv}} v \xrightarrow{r_{vw}} w$, with the trace $r = r_{uv}r_{vw}$. If there is a loop edge r_{vv}, this edge can be traveled any number of times before exiting v, thus the regular expression is $r = r_{uv}r_{vv}^*r_{vw}$. This edge $u \xrightarrow{r} w$ was added to G'. Thus, for every path accepted by G, for every subpath entering and exiting v from some u to w, there exists a corresponding subpath from u to w in G', and thus a corresponding path accepted by G'. □

Lemma IV.2. *For any NFA G with start state s and (exactly one) final state f, **nfa2regex**(G, s, f) outputs the regular expression capturing all paths from s to f (i.e., $\mathscr{L}(G)$, the language accepted by G).*

Proof. The **while** loop clearly terminates since the number of interior nodes is strictly decreasing, and upon termination, G is an NFA containing only the nodes s and f. Now, every non-terminal visit from s to f and back to s, can be replaced by an edge $s \xrightarrow{r_{\mathrm{sf}}r_{\mathrm{ff}}^*r_{\mathrm{fs}}} s$, instead of the edge $f \to s$. This NFA yields the regular language $(r_{\mathrm{ss}}|r_{\mathrm{sf}}r_{\mathrm{ff}}^*r_{\mathrm{fs}})^*r_{\mathrm{sf}}r_{\mathrm{ff}}$, as produced by the algorithm. □

Lemma IV.3. *For any NFA G with start state s and (exactly one) final state f, **nfa2regex_firstvisit**(G, s, f) outputs the regular expression of all paths from s to f, visiting f for the first time.*

Proof. Let G' be G with all out-edges (including loops) from f deleted. We claim that $\mathscr{L}(G')$ is equal to the language $\mathscr{L}'$ of all paths from s reaching f for the first time.

First we show $\mathscr{L}(G') \subseteq \mathscr{L}'$, i.e., every path accepted by G' starts in s and ends by reaching f for the first time. Since s and f are the initial and final states of G' respectively, clearly every path accepted by G must start in s and end in f. Further, as f has no out-edges in G, any accepted path must end immediately once it reaches f. Thus it must end once it reaches f for the first time.

Also, we show $\mathscr{L}' \subseteq \mathscr{L}(G')$, i.e., every path starting in s and ending by reaching f for the first time is accepted by G'. If a path contains an out-edge of f, it cannot be in $\mathscr{L}'$ as the trace continues beyond the first time step at f. Thus, every trace in $\mathscr{L}'$ only uses edges in G', and is thus accepted by G iff it is accepted by G'. □

Theorem IV.1. *For any Büchi automaton G with start state s and final states F, **ba2wregex**(G) outputs the ω-regular language accepted by G.*

Proof. We show that

$$\mathscr{L} = \bigcup_{f \in F} \textbf{nfa2regex}(G, s, f)\,(\textbf{nfa2regex\_firstvisit}(G, f, f))^\omega$$

and the ω-language accepted by G, $\mathscr{L}_\omega(G)$ are identical.

First, $\mathscr{L} \subseteq \mathscr{L}_\omega(G)$ since for every infinite run in $\mathscr{L}$, by definition of $\mathscr{L}$, there is some $f \in F$ such that passes through f infinitely often. The converse is true as well: for every infinite run $q_0, q_1, \ldots$ accepted by G, then $q_0 = s$ and by definition of the accepting language, there is some accepting state f, passed through infinitely often; thus the run is in **nfa2regex**(G, s, f) $(\textbf{nfa2regex\_firstvisit}(G, f, f))^\omega$. □

B. Correctness of Rewrite Rules

Lemma IV.4. *For all regular expressions r_1, r_2, r, each of the following rewrite rules preserve the regular or ω-regular language represented by the expressions:*

$$r_1 + r_1r_2^* \implies r_1r_2^* \quad (1)$$
$$r + r \implies r \quad (2)$$
$$r_1 + r_2^*r_1 \implies r_2^*r_1 \quad (3)$$
$$(r^*)^\omega \implies r^\omega \quad (4)$$
$$(r_1r_2^*)r_2^\omega \implies r_1r_2^\omega \quad (5)$$
$$(r_1r_2)r_2^\omega \implies r_1r_2^\omega \quad (6)$$
$$r^*r^\omega \implies r^\omega \quad (7)$$
$$rr^\omega \implies r^\omega \quad (8)$$

The soundness of each of these translations follows directly from the semantics detailed in Definition II.8.

C. Correctness of Timeline Visualizations

Inductively, we show how to faithfully capture each form of regular and ω-regular expressions as timeline representations.

a) Regular expressions:

1) $A = \emptyset$. Visualized as no timeline. An empty regular expression signifies nothing is true at any time.
2) $A = \epsilon$. Omitted in the visualization.
3) $A = a$. Visualized as a single timeline node with a. A regular expression of a propositional logic formula signifies that the atomic proposition is true at the current time step.
4) $A = A_1A_2$. Visualized as the timeline of A_1 followed by the timeline of A_2. A concatenation of two regular expressions signifies that the first regular expression must be true before the second regular expression can be true in the time step.
5) $A = A_1 + A_2$. Visualized as the timelines of A_1 and A_2 in parallel. A union of two regular expressions signifies that either of the regular expressions can be true, which `ltl2timeline` represents by parallel timelines.
6) $A = A_1^*$. If A_1 is a propositional logic formula, then its visualization is an egg-shaped node with the caption "repeats 0 - ∞." If A_1 is a regular expression involving concatenation or a star ($*$) then its visualization is the timeline of A_1 followed by a node with label "$\cdots$," and that timeline pattern again. A Kleene star of a regular expression signifies that the regular expression can be repeated any number of times, including 0 times, which matches the meaning of our representation of an egg-shaped node and a "$\cdots$" node as described in Section II-D.

b) ω-regular repressions:

1) $B = A^\omega$. Visualized as a grey box enclosing the timeline of A. An ω in a regular (sub-)expression signifies that the regular expression repeats infinitely many times, which matches the meaning of our grey box representation, because every time a timeline traversal reaches the end

of the grey box, it must reenter the same grey box again, hence designating an infinite repeat.

2) $B = AB$. Visualized as the timeline of A followed by the timeline of B. A concatenation of a regular expression and an ω-regular expression signifies that the first must be true before the second ω-regular expression can be true in the following time steps.
3) $B = B_1 + B_2$. Visualized as the timelines of B_1 and B_2 in parallel. A union of two ω-regular expressions signifies that either of the ω-regular expressions can be true, which we represent by parallel timelines.

V. Tool Showcase: "Be great in act, as you have been in thought[§]"

To demonstrate using `ltl2timeline`, we present three examples, one from a real-world model checking exercise from Zhao and Rozier [1], one from a randomly generated LTL formula, and one that specifically shows the applicability of `ltl2timeline` to specification validation.

Example V.1. Zhao and Rozier [1] presents a model verification specification, "[i]f a TSAFE command is sent to an aircraft, controller/AutoResolver should then hand off the control of this aircraft," which corresponds to the LTL formula

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

For simplicity, we swap the concrete atomic proposition to a, b and get

$$\square(a \wedge b \rightarrow \mathcal{X}(\neg b)).$$

For this LTL formula, `ltl2timeline` generates the timeline representation in Figure 8.

Example V.2. SPOT [22] includes a command-line tool for random LTL formula generation called `randltl` [23]. We used it to generate the following LTL formula.

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

This formula is reasonably complicated, and hard for humans to reason about directly. For this LTL formula, our tool generates the timeline representation in Figure 9.

Remark V.1. These two examples show that our tool can generate reasonably intuitive diagrams for both real-world and randomly-generated formulas.

Example V.3. Suppose we have the specification "p oscillates every time step." Human specifiers often write one of the following two LTL formulas to describe this specification.

- $\square((p \wedge \mathcal{X}(\neg p)) \vee ((\neg p) \wedge \mathcal{X} p))$
- $\square((p \wedge \mathcal{X}(\neg p)) \wedge ((\neg p) \wedge \mathcal{X} p))$

It may be hard, without rigorous analysis, to distinguish which one faithfully represents the specification. However, `ltl2timeline` generates the two timelines for these two formulas shown in Figures 10 and 11, which clearly show the first formula is correct and the second formula is equivalent to $\bot$.

[§]William Shakespeare, *King John*

VI. Playing it out

We quantify the usability and performance of `ltl2timeline` by compiling an extensive set of LTL formulas used in the analysis of real-life systems, defining timeline metrics, and characterizing the result of visualizing the set of real-life LTL formulas over those metrics. Our experimental evaluation conclusively demonstrates that `ltl2timeline` scales to provide helpful visualizations of LTL formulas written by humans.

A. Timeline Metrics

We use two measures of complexity for timeline visualizations: timeline length and star height. We use timeline length as a proxy for size of our timeline visualization graphs. Since Kleene stars are the trickiest operator to depict in timelines, we also use star height, a well-studied measure for the structural complexity of regular expressions, to measure the complexity of our visualizations.

Definition VI.1 (Timeline Length)**.** We define the timeline length of a regular expression $\text{tllen}(A)$ recursively as follows.

$$\begin{aligned}
\text{tllen}(\emptyset) &\quad \text{undefined} \\
\text{tllen}(\epsilon) &= 0 \\
\text{tllen}(a) &= 1 \\
\text{tllen}(A_1 A_2) &= \text{tllen}(A_1) + \text{tllen}(A_2) \\
\text{tllen}(A_1 + A_2) &= \max(\text{tllen}(A_1), \text{tllen}(A_2)) \\
\text{tllen}(A^*) &= \text{tllen}(A)
\end{aligned}$$

We also extend this definition to ω-regular expressions as follows.

$$\begin{aligned}
\text{tllen}(A^\omega) &= \text{tllen}(A) \\
\text{tllen}(AB) &= \text{tllen}(A) + \text{tllen}(B) \\
\text{tllen}(B_1 + B_2) &= \max(\text{tllen}(B_1), \text{tllen}(B_2))
\end{aligned}$$

Then tllen captures the length of the longest path in our timeline graph visualization.

Definition VI.2 (Star height)**.** We define the star height of regular and ω-regular expressions to be the star-nesting depth in the (unsimplified) expression.

$$\begin{aligned}
h(\emptyset) = h(\epsilon) = h(a) &= 0 \\
h(A_1 A_2) = h(A_1 + A_2) &= \max(h(A_1), h(A_2)) \\
h(A^*) &= 1 + h(A) \\
h(A^\omega) &= h(A) \\
h(AB) &= \max(h(A), h(B)) \\
h(B_1 + B_2) &= \max(h(B_1), h(B_2))
\end{aligned}$$

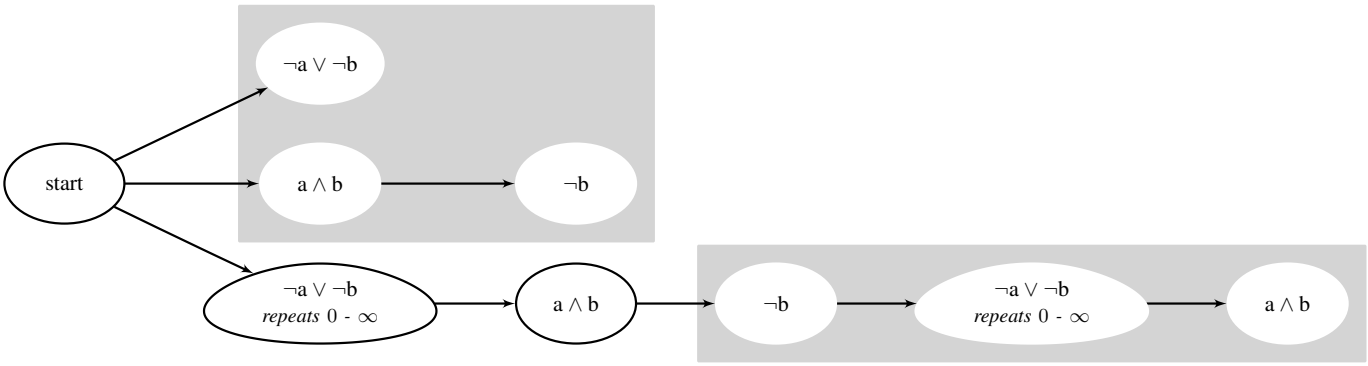


Fig. 8: Timeline for $\square(a \wedge b \rightarrow \mathcal{X}(\neg b))$.

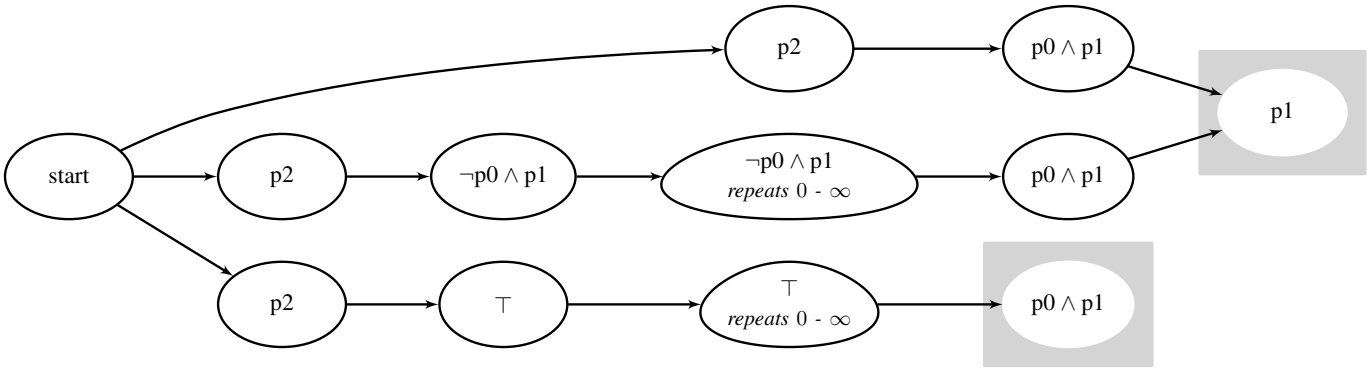


Fig. 9: Timeline for $p_2 \wedge (\Diamond\square p_0 \,\mathcal{U}\, \mathcal{X}(\square p_1 \wedge (((p_0 \rightarrow p_2) \wedge (p_2 \rightarrow p_0)) \,\mathcal{U}\, \Diamond p_0)))$.

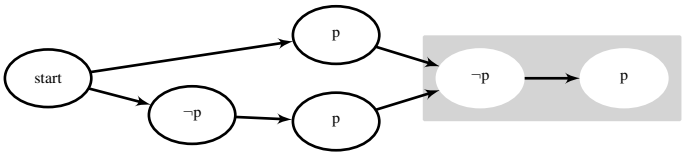


Fig. 10: Timeline for $\square((p \wedge \mathcal{X}(\neg p)) \vee ((\neg p) \wedge \mathcal{X}p))$.

Fig. 11: Timeline for $\square((p \wedge \mathcal{X}(\neg p)) \wedge ((\neg p) \wedge \mathcal{X}p))$ (a single "start" means every time step is $\bot$).

B. Experimental Analysis

Feasibility. We study the feasibility of our tool on a benchmark suite of LTL formulas gathered from real world use cases. We gather a test suite totalling 91 formulas from two real-world requirement specification applications. We take 6 specifications from NASA's Automated Airspace Concept (AAC) [24]. We also collect formulas from the Acacia suite of examples [16]: 85 formulas extracted from the suite of 23 collections of formulas. We choose to visualize each formula individually, rather than visualize a complete set of (conjuncted) specifications, as a complete specification set may describe a large, complex system, and we only target the use case of validating individual formulas.

We run `ltl2regex` on these 91 examples: 87 of the 91 input formulas complete execution within a timeout of 20 seconds for the feasibility tests. Within the 87, two examples have regular expressions of star height 8 and are intractable to generate graph visualizations of. We are able to generate tractable visualization on the remaining 85, or over 93% of benchmarks tested. The plots of visualization complexity on these formula appear below in Figure 12.

Scalability. We study the scalability of our tool on a benchmark suite of LTL formulas that scales in size. According to Rozier and Vardi [16], the LTL formula required to encode a n-bit binary counter scales in size as n gets larger. Therefore the benchmark for scalability consists of LTL formulas that describe n-bit counters. Figure 13 shows that the timeline length grows approximately exponentially with the number of bits in the counter, which matches the result in Rozier and Vardi [16].

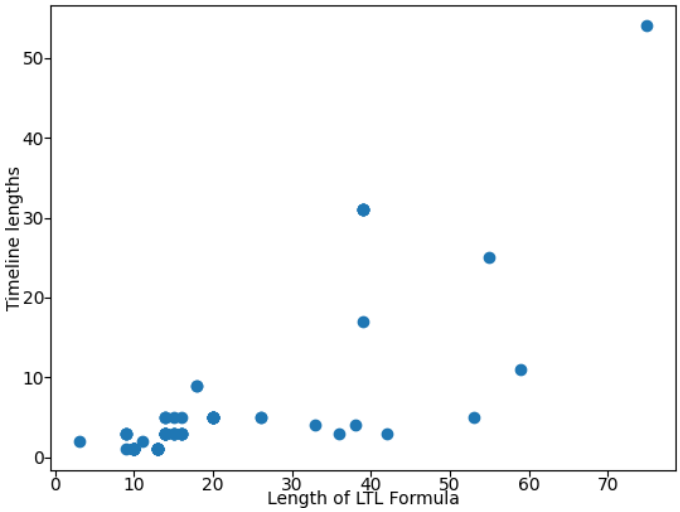
In Figure 13, the x-axis ranges from 1 to 6, after which point it hits the 30 seconds timeout we set for the scalability tests, as the timeline length grows exponentially fast.

VII. Artifact Availability: "Thought is free[§]"

The tool we present in this paper is available at

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

[§]William Shakespeare, *The Tempest*



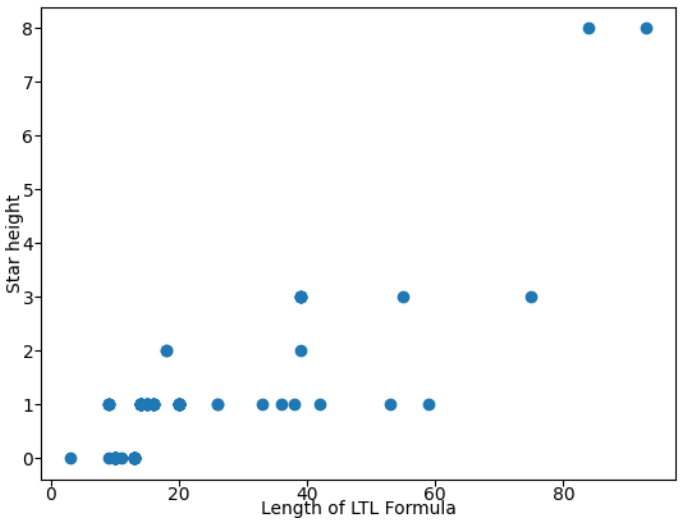


Fig. 12: Visualization complexity metrics for formulas collected from Acacia [16] and NASA's AAC [24].

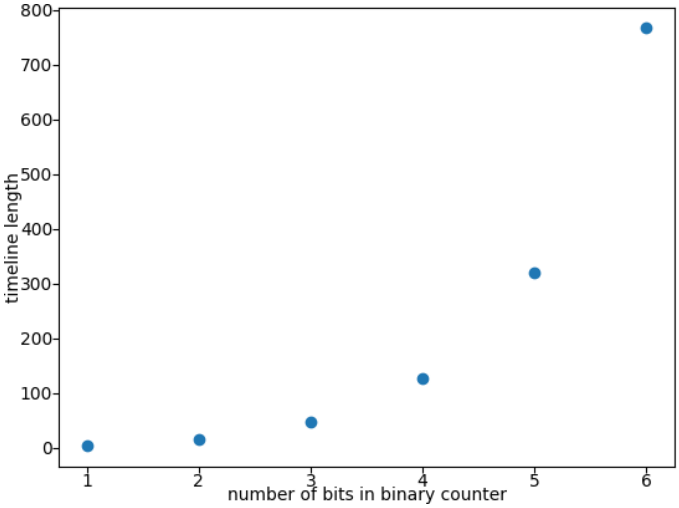


Fig. 13: Scalability measured with binary counter examples from Rozier and Vardi [16].

which comes with two command-line tools, `ltl2regex` and `ltl2timeline`. The specific usage of the tool can be found in the artifact repository.

The set of LTL formulas we used to evaluate our tool in Section VI can be found in the artifact repository under `ltl-formulas` directory.

VIII. Denouement

Based on our results, we propose the following questions for discussions and future work:

- Can there be more simplification methods as described in Section III-C? The answer is definitely positive. For example, we can add more rule-based simplifications by finding more intuitive patterns in the generated ω-regular expressions. Other simplifications are also possible; for example, if the generated ω-regular expression is $r_1 + r_2$ where r_1, r_2 are not syntactically equivalent (or equivalent up to algebraic rules) ω-regular expressions, but they accept the same set of infinite-length strings, then we can simplify this down to just r_1. This reduces the problem to equivalence testing on ω-regular expressions, which is decidable. In a similar vein, many more simplifications may apply, but the question is how *useful* are they? In the end, we have to balance the efficiency, simplicity, and understandabillity of each output visualization.
- Can we define timelines more formally so that one can do this process reversely (i.e., a tool that converts timeline representation to its corresponding LTL formula)? That would be very useful in practice. However, a direct reverse of the algorithm we presented in this paper may not be viable.

Going forward, it will become important to conduct a user survey to gather data from a representative audience of system engineers regarding what timeline visualizations help most with formula validation. Specifically, we list the following questions regarding the intuitive aesthetics of timelines.

- **Can lines in timelines be curved or do they need to be straight?** If a timeline has a curved line is that still visually, intuitively a line? Does the placement of the line matter as to whether it is intuitive for the line to be straight or curved (e.g., to save space or reduce possible visual clutter)?
- **Can there be back edges in a limited fashion (i.e., to simplify formulas) in timelines?** Most of our formulas have a star-height of 1, so if we limit the back-edge representation to only surrounding a single node, that would simplify most formulas. We can then use the expanded notation for stared formulas larger than a single proposition, thus navigating the trade-off between having a small representation and limiting the visual complexity. One criticism of this approach may be that back edges cause confusion between timeline and automata representations, and that with back edges timelines stop being intuitive with respect to the linearity of time.

- **What are intuitive representations of star-formulas and do they change depending on the formula?** Common regular expression visualizers use two ways to represent stars in regular expressions. We can represent a stared formula inside a box [25] with related caption to indicate the repetition (similar to the design decision we made in `ltl2timeline`); or with back edges [26]. Does using the box cause confusion with respect to the LTL $\Box$ operator? What is the most intuitive way to remind users of the corner case where the starred formula occurs zero times?
- **When do we choose to unroll for clarity? When do we merge parallel parts of timelines?** To merge parts of lines, we have to check for logical equivalence of the parts of the timelines we want to merge. By our algorithm, we generate parallel lines only when the formulas are not syntactically equivalent, so we would be guaranteed to run the more complex check for logical equivalence. In some cases, this could provide a clarifying simplification, with the cost of a repetitive computationally-expensive check. Would it make sense to offer an "optimizing" compilation option to users that takes longer to create a timeline representation but checks for smaller representations via logical equivalence?

IX. Epilogue

The Achilles heel of formal verification is specification; formal methods are only as effective at verification as their specifications are at describing the essential properties to verify. Yet, specification remains the biggest bottleneck to the use of formal methods [27]. LTL is one of the most popular specification logics for industrial-scale critical systems; in the space domain alone, it is currently encapsulating specifications for the development of the NASA Lunar Gateway [28], [29], the Air Force Research Laboratory/Collins Aerospace Spacecraft Collision Avoidance system [30], Space Systems Finland's Attitude and Orbit Control Systems (AOCS) [31], and NASA/JPL's Europa Lander Mission Concept [32] just to name a few. Yet the humans that need to use formal verification tools and deeply understand their results struggle to validate that LTL formulas capture the specifications they are meant to capture. A major contributor to LTL's popularity is the propensity of humans to think of requirements in terms of timelines. This inspired the creation of LTL in the first place, as a logic that "intuitively" represents timelines. Our work serves to reinforce that connection, enabling visualization of most realistic LTL formulas as timelines. By releasing `ltl2timeline`, we contribute to better validation capabilities for LTL specifications and aid the effort to make formal verification more accessible and wide-spread.

Future extensions of this work include optimizations to the algorithm and implementation to improve performance and scalability. While we have chosen visual elements that succinctly represent timelines, it would be informative to conduct a study on different possible visualizations and which of the many ways of representing different timeline elements humans find most intuitive. It is possible that factors of context, such as the type of requirement an LTL formula describes, change its optimal timeline representation.

Acknowledgements

This work was partially supported by the National Science Foundation (NSF) under grants CCF-2015445, CAREER-1664356, and CCRI-2016592.

Appendix

Recall from II.1 that we define an LTL formula as

$$\varphi = p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \Box\varphi \mid \Diamond\varphi \mid \mathcal{X}\varphi \mid \varphi\, \mathcal{U}\, \psi \mid \varphi\, \mathcal{R}\, \psi$$

where $p \in \mathcal{AP}$ and φ and ψ are LTL formulas. In our tool `ltl2timeline`, we use the following concrete input syntax to represent LTL formulas in ASCII text.

	abstract syntax	*concrete syntax*	*description*
φ =	p	`p`	atomic prop.
|	$\neg\varphi$	`!`φ	negation
|	$\varphi \wedge \psi$	φ `&` ψ	conjunction
|	$\varphi \vee \psi$	φ `|` ψ	disjunction
|	$\varphi \rightarrow \psi$	φ `->` ψ	implication
|	$\Box\varphi$	`G`φ	globally
|	$\Diamond\varphi$	`F`φ	in the future
|	$\mathcal{X}\varphi$	`X`φ	next
|	$\varphi\, \mathcal{U}\, \psi$	φ `U` ψ	until
|	$\varphi\, \mathcal{R}\, \psi$	φ `R` ψ	release

References

[1] Y. Zhao and K. Y. Rozier, "Formal specification and verification of a coordination protocol for an automated air traffic control system," *Science of Computer Programming Journal*, vol. 96, no. 3, pp. 337–353, December 2014.

[2] B. Greenman, S. Saarinen, T. Nelson, and S. Krishnamurthi, "Little tricky logic: Misconceptions in the understanding of ltl," *arXiv preprint arXiv:2211.01677*, 2022.

[3] E. J. T. Gonçalves, G. Rodrigues, P. Miranda, J. Pimentel, J. Araujo, and J. Castro, "pistar-ext: Supporting the creation of istar extensions with the pistar tool." in *iStar*, 2020, pp. 31–36.

[4] S. Becker, D. Dietsch, N. Hauff, E. Henkel, V. Langenfeld, A. Podelski, and B. Westphal, "Hanfor: Semantic requirements review at scale." in *REFSQ Workshops*, vol. 2857, 2021.

[5] T. Viger, L. Murphy, A. Di Sandro, C. Menghi, R. Shahin, and M. Chechik, "The foremost approach to building valid model-based safety arguments," *Software and Systems Modeling*, pp. 1–22, 2022.

[6] L. Zhang, W. He, J. Martinez, N. Brackenbury, S. Lu, and B. Ur, "Autotap: synthesizing and repairing trigger-action programs using ltl properties," in *2019 IEEE/ACM 41st international conference on software engineering (ICSE)*. IEEE, 2019, pp. 281–291.

[7] J. Li, M. Y. Vardi, and K. Y. Rozier, "Satisfiability checking for Mission-time LTL," in *Proceedings of 31st International Conference on Computer Aided Verification (CAV)*, ser. LNCS, vol. 11562. New York, NY, USA: Springer, July 2019, pp. 3–22.

[8] C. Johannsen, P. Jones, B. Kempa, K. Y. Rozier, and P. Zhang, "R2U2 Version 3.0: Re-Imagining a Toolchain for Specification, Resource Estimation, and Optimized Observer Generation for Runtime Verification in Hardware and Software," in *Computer Aided Verification*, C. Enea and A. Lal, Eds. Cham: Springer Nature Switzerland, 2023, pp. 483–497.

[9] D. Giannakopoulou, T. Pressburger, A. Mavridou, and J. Schumann, "Generation of formal requirements from structured natural language," in *International working conference on requirements engineering: Foundation for software quality*. Springer, 2020, pp. 19–35.

[10] J. Elwing, L. Gamboa-Guzman, J. Sorkin, C. Travesset, Z. Wang and K. Y. Rozier, "Mission-time ltl (mltl) formula validation via regular expressions," in *Proceedings of the 18th International Conference on integrated Formal Methods (iFM 202 3)*. Leiden, the Netherlands: Springer, November 2023.

[11] D. Neider and R. Roy, "Expanding the horizon of linear temporal logic inference for explainability," in *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2022, pp. 103–107.

[12] K. Rozier, "Linear Temporal Logic Symbolic Model Checking," *Computer Science Review Journal*, vol. 5, no. 2, pp. 163–203, May 2011. [Online]. Available: http://dx.doi.org/10.1016/j.cosrev.2010.06.002

[13] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Proceedings of the First Symposium on Logic in Computer Science*. IEEE Computer Society, 1986.

[14] M. Daniele, F. Giunchiglia, and M. Y. Vardi, "Improved automata generation for linear temporal logic," in *Computer Aided Verification*, N. Halbwachs and D. Peled, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 249–260.

[15] C. Fritz, "Constructing büchi automata from linear temporal logic using simulation relations for alternating büchi automata," in *Implementation and Application of Automata*, O. H. Ibarra and Z. Dang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 35–48.

[16] K. Rozier and M. Vardi, "LTL satisfiability checking," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 12, no. 2, pp. 123 – 137, March 2010.

[17] ——, "A multi-encoding approach for LTL symbolic satisfiability checking," in *17th International Symposium on Formal Methods (FM2011)*, ser. Lecture Notes in Computer Science (LNCS), vol. 6664. Springer-Verlag, 2011, pp. 417–431.

[18] A. Duret-Lutz, "Ltl translation improvements in spot 1.0," *International Journal of Critical Computer-Based Systems 5*, vol. 5, no. 1-2, pp. 31–54, 2014.

[19] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. Lauko, "From Spot 2.0 to Spot 2.10: What's new?" in *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, ser. Lecture Notes in Computer Science, vol. 13372. Springer, Aug. 2022, pp. 174–187.

[20] T. S. Wim Martens, Frank Neven, "Complexity of decision problems for simple regular expressions," *International Symposium on Mathematical Foundations of Computer Science*, pp. 889–900, 2004.

[21] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz - open source graph drawing tools," in *International Symposium Graph Drawing and Network Visualization*, 2001.

[22] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. Lauko, "Spot: a platform for LTL and omega-automata manipulation," Online: https://spot.lre.epita.fr/, 2023.

[23] A. 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)*, ser. Lecture Notes in Computer Science, vol. 8172. Hanoi, Vietnam: Springer, Oct. 2013, pp. 442–445.

[24] M. Gario, A. Cimatti, C. Mattarei, S. Tonetta, and K. Y. Rozier, "Model checking at scale: Automated air traffic control design space exploration," in *Proceedings of 28th International Conference on Computer Aided Verification (CAV 2016)*, ser. LNCS, vol. 9780. Toronto, ON, Canada: Springer, July 2016, pp. 3–22.

[25] Bowen, "Regex visualizer and editor," https://github.com/Bowen7/regex-vis, 2022.

[26] J. Avallone, "regexper-static," https://gitlab.com/javallone/regexper-static, 2020.

[27] K. Y. Rozier, "Specification: The biggest bottleneck in formal methods and autonomy," in *Proceedings of 8th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2016)*, ser. LNCS, vol. 9971. Toronto, ON, Canada: Springer-Verlag, July 2016, pp. 1–19.

[28] J. B. Dabney, J. M. Badger, and P. Rajagopal, "Adding a verification view for an autonomous real-time system architecture," in *AIAA SciTech Forum*. AIAA, 2021, p. 0566.

[29] ——, "Trustworthy autonomy for gateway vehicle system manager," in *2023 IEEE Space Computing Conference (SCC)*. IEEE, 2023, pp. 57–62.

[30] K. L. Hobbs, J. Davis, L. Wagner, and E. Feron, "Formal specification and analysis of spacecraft collision avoidance run time assurance requirements," in *2021 IEEE Aerospace Conference (50100)*. IEEE, 2021, pp. 1–16.

[31] D. Ilić, L. Laibinis, T. Latvala, E. Troubitsyna, and K. Varpaaniemi, "Deployment in the space sector," in *Industrial Deployment of System Engineering Methods*. Springer, 2013, pp. 45–62.

[32] S. Chien, J.-P. de la Croix, J. Russino, C. Wagner, G. Rabideau, D. Wang, and G. Lim, "Onboard scheduling and execution to address uncertainty for a planetary lander," in *Proceedings of 16th Symposium on Advanced Space Technologies in Robotics and Automation*. European Space Agency, 2022.