Learning Mission-time Linear Temporal Logic (MLTL) from Positive and Negative Data

Zili Wang[1], Luke Marzen[1], Nhan Tran[1], and Swaminathan Jayaraman[1]

Iowa State University, Ames, IA 50011, USA
{ziliw1, ljmarzen, nhtran, swamjay}@iastate.edu

Abstract. Mission-time Linear Temporal Logic (MLTL) is a discrete time, finite interval bounded variant of the popular Linear Temporal Logic (LTL). Given a set of positive traces, characterizing desirable behaviors of a system, and a set of negative traces, characterizing undesirable behaviors, the goal is to learn a MLTL formula succinctly capturing the system behavior. In this technical report, we describe five differing approaches to MLTL inference: genetic programming, informed beam search, deep learning via transformers, template-driven search, and Bayesian network structure learning.

1 Introduction

The task of learning temporal logic specifications from data has wide applications in a variety of domains, such as specification mining, model checking, runtime verification, and inferring reinforcement learning objectives. Given a collection of desirable and undesirable behaviors of a system, i.e., positive and negative examples, we study the problem of learning a temporal logic formula that succinctly captures the system behavior.

In this technical report, we focus on learning Mission-time Linear Temporal Logic (MLTL) formulas from positive and negative examples. MLTL is a discrete time, finite interval bounded variant of the popular Linear Temporal Logic (LTL) [19], and we provide a brief overview of MLTL in Section 3. We describe five differing approaches to MLTL inference: grammar evolution, informed beam search, deep learning via transformers, template-driven search, and Bayesian network structure learning. We also evaluate the performance of these approaches on a set of synthetic datasets of varying complexity. Grammar evolution and informed beam search yielded promising results across all datasets, while template-driven search was only able to learn simple formulas. Deep learning via transformers encountered significant challenges in training, and Bayesian network structure learning led to theoretical results that does not yet yield a practical algorithm for MLTL inference, but has use in other applications.

2 Related Work

Work in learning temporal logic formulas have also recently been explored, with approaches ranging from symbolic learning algorithms [22] [3] to deep learning algorithms [12],[13]. There is no published work on learning MLTL formulas, and the goal of this project is to explore various approaches to this problem, and to compare and contrast their performance.

3 MLTL

MLTL is a discrete time, finite interval bounded temporal logic that has found numerous recent applications. For example, MLTL was the specification logic for NASA's Robonaut2 verification project [10], as well as for the design-time and runtime verification of the NASA Lunar Gateway Vehicle System Manager [5]. Other applications of MLTL include autonomous satellite [18], UAV Traffic Management [8], and more [9], [15], [14], [2].

3.1 Syntax and Semantics

The syntax of MLTL for a finite set of atomic propositions $\mathcal{AP}$ is defined as:

$$\xi := true \mid false \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathcal{F}_{[a,b]}\varphi \mid \mathcal{G}_{[a,b]}\varphi \mid \varphi\mathcal{U}_{[a,b]}\psi \mid \varphi\mathcal{R}_{[a,b]}\psi,$$

where $p \in \mathcal{AP}$, $a, b \in \mathbb{Z}$ such that $0 \leqslant a \leqslant b$, and φ, ψ, ξ are MLTL formulas. The symbols $\mathcal{F}, \mathcal{G}, \mathcal{U}, \mathcal{R}$ denote the temporal operators Future, Globally, Until, and Release, respectively.

Definition 1. *A **trace** over a finite set of atomic propositions $\mathcal{AP}$ is a finite sequence $\pi = \pi[0], \pi[1], ..., \pi[k-1]$, where each $\pi[i] \subseteq \mathcal{AP}$.*

A trace represents an assignment to the truth values of each atomic proposition over time, where $p \in \mathcal{AP}$ holds at time i if and only if $p \in \pi[i]$. We denote the length of a trace π as $|\pi|$, and denote the suffix of a trace π starting at, and including, i as π_i. Thus note that $\pi_0 = \pi$.

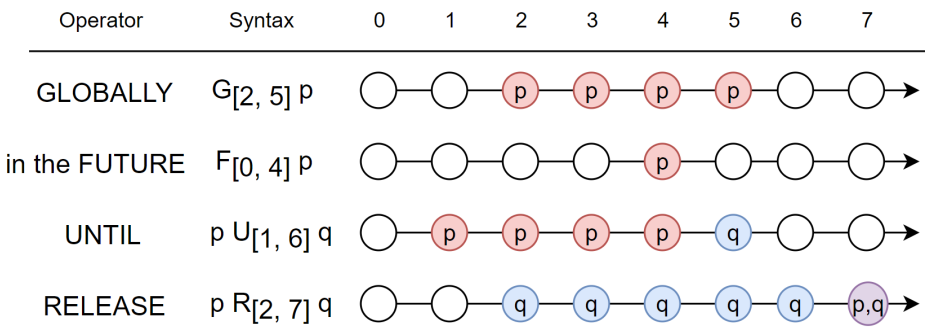


Fig. 1: "Intuitive" semantics of the temporal operators. The presence of a variable indicates that it is true at that time step, and the absence indicates that it is false.

Definition 2. *A trace π **satisfies** MLTL formula φ, denoted $\pi \vDash \varphi$, is defined as follows:*

$\pi \vDash p$ iff $p \in \pi[0]$

$\pi \vDash \neg\alpha$ iff $\pi \nvDash \alpha$

$\pi \vDash \alpha \wedge \beta$ iff $\pi \vDash \alpha$ and $\pi \vDash \beta$

$\pi \vDash \alpha \vee \beta$ iff $\pi \vDash \alpha$ or $\pi \vDash \beta$

$\pi \vDash \mathcal{F}_{[a,b]}\alpha$ iff $|\pi| > a$ and $\exists i \in [a, b]$ such that $\pi_i \vDash \alpha$

$\pi \vDash \mathcal{G}_{[a,b]}\alpha$ iff $|\pi| \leqslant a$ or $\forall i \in [a, b]$ $\pi_i \vDash \alpha$

$\pi \vDash \alpha\,\mathcal{U}_{[a,b]}\beta$ iff $|\pi| > a$ and $\exists i \in [a, b]$ such that $\pi_i \vDash \beta$ and $\forall j \in [a, i-1]$ $\pi_j \vDash \alpha$

$\pi \vDash \alpha\,\mathcal{R}_{[a,b]}\beta$ iff $|\pi| \leqslant a$ or $(\forall i \in [a, b]$ $\pi_i \vDash \beta)$ or $(\exists j \in [a, b-1]$ such that $\pi_j \vDash \alpha$ and $\forall a \leqslant k \leqslant j$ $\pi_k \vDash \beta)$

Illustrations of the intuitive meanings of the temporal operators appear in Figure 1. Two MLTL formulas φ and ψ are semantically equivalent if for any trace π, $\pi \models \varphi \iff \pi \models \psi$. As in LTL, the usual dual equivalence relationships hold between the Finally and Globally operators, and the Until and Release operators. That is, $\mathcal{F}_{[a,b]}\alpha \equiv \neg\mathcal{G}_{[a,b]}\neg\alpha$ and $\alpha\,\mathcal{U}_{[a,b]}\beta \equiv \neg(\neg\alpha\,\mathcal{R}_{[a,b]}\neg\beta)$.

MLTL reasons over a finite set of atomic propositions, and thus we may assume without loss of generality that $\mathcal{AP} = \{p_0, p_1, \ldots, p_{n-1}\}$. This imposes a natural ordering on the atomic propositions, which we use to define the string encoding of a trace.

Definition 3. *The **string encoding**, as defined in [24], of a finite trace π of length $m = |\pi|$ over $n = |\mathcal{AP}|$ atomic propositions is the string $w_\pi \in \{\mathbf{0}, \mathbf{1}\}^{mn}$, i.e., a binary string of length mn, such that $p_k \in \pi[i]$ if and only if the $(i * n + k)$-th character (indexing from 0) of w_π is **1**.*

As an example, for $\mathcal{AP} = p0, p1$, the string encoding of the trace $\pi = p0, p0, p1,$ is the binary string $w_\pi = \mathbf{10}, \mathbf{11}, \mathbf{00}$. The string encoding of traces gives a convenient way to represent traces as binary strings, and also lends MLTL to analysis using formal language theory.

Definition 4. *The **future reach** of an MLTL formula ξ, denoted $FR(\xi)$, is recursively defined as:*

$$FR(\xi) = \begin{cases} 1 & \text{if } \xi \text{ is an atomic proposition } p \in \mathcal{AP} \\ FR(\varphi) + FR(\psi) & \text{if } \xi = \varphi \wedge \psi \text{ or } \xi = \varphi \vee \psi \\ b + FR(\varphi) & \text{if } \xi = \mathcal{F}_{[a,b]}\varphi \text{ or } \xi = \mathcal{G}_{[a,b]}\varphi \\ \xi = \max(FR(\varphi), FR(\psi)) & \text{if } \xi = \varphi\mathcal{U}_{[a,b]}\psi \text{ or } \xi = \varphi\mathcal{R}_{[a,b]}\psi \end{cases}$$

Note that the future reach of an MLTL formula follows the same definition as the worst propagation delay (wpd) of an LTL formula [10]. Intuitively, the future reach of an MLTL formula is the maximum number of time steps that the formula can specify behavior over.

3.2 MLTL Inference

We formally define the MLTL inference problem and show that it is well-defined. [1] Fix some integer $n > 0$, and let Π^+ be a set of positive traces, and Π^- be a set of negative traces, where $\Pi^+ \cap \Pi^- = \varnothing$. Then the goal is to learn an MLTL formula φ over n atomic propositions that separates $\Pi = \Pi^+ \cup \Pi^-$, meaning for all $\pi \in \Pi^+$ we have $\pi \models \varphi$ and for all $\pi \in \Pi^-$ we have $\pi \not\models \varphi$.

Definition 5. *An instance of the MLTL inference problem is a triplet (Π^+, Π^-, n) where Π^+ and Π^- are sets of positive and negative traces, respectively, and $n \geqslant 1$ is some integer.*

Theorem 1. *For any instance of the MLTL inference problem, (Π^+, Π^-, n), there exists MLTL formula φ over n atomic propositions to separate $\Pi = \Pi^+ \cup \Pi^-$.*

[1] This is work done from the previous semester.

Proof. For any trace $\pi = \pi[0], \pi[1], ..., \pi[m]$, we can construct a MLTL formula φ_π such that $\mathcal{L}(\varphi_\pi) = \{\pi\}$ by taking $\varphi_\pi = \bigwedge_{i=0}^{m} G_{[i,i]}(\ell_{i,0} \wedge ... \wedge \ell_{i,n-1})$, where $\ell_{i,j} = p_j$ if $p_j \in \pi[i]$, otherwise $\ell_{i,j} = \neg p_j$. Then the formula $\varphi = \bigwedge_{\pi \in \Pi^+} \varphi_\pi$ separates Π.

However, this construction of brute force enumeration of all positive traces is not useful. For any practical application, we should strive to learn formulas of size sublinear with respect to the total size of example traces Π, i.e. formulas that are good high-level descriptors of Π. Thus rather than striving for a formula φ that separates Π exactly, we maximize the accuracy of φ over Π, defined as

$$\text{acc}(\varphi, \Pi) = \frac{|\{\pi \in \Pi^+ : \pi \models \varphi\}| + |\{\pi \in \Pi^- : \pi \not\models \varphi\}|}{|\Pi|}.$$

4 Approaches

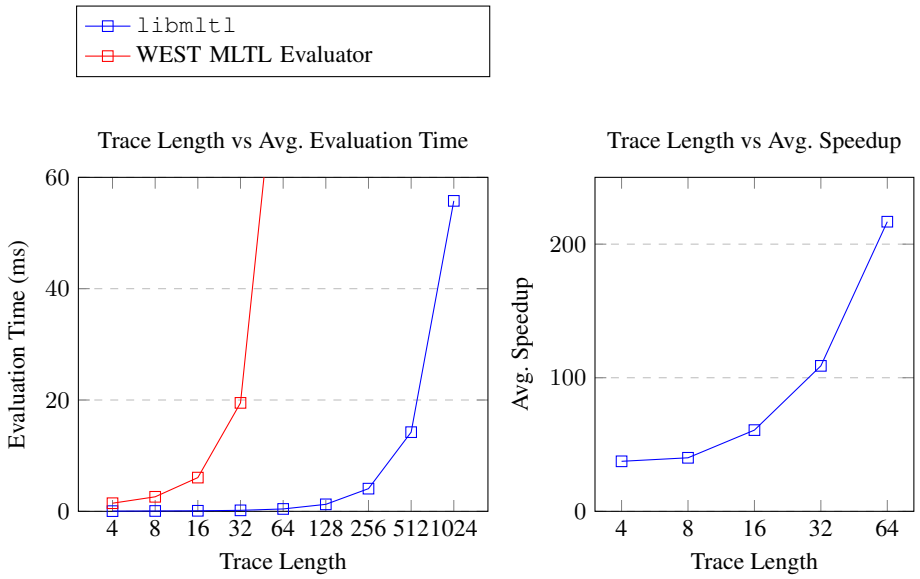
Previously, Zili Wang developed and evaluated a Genetic Programming (GP) based approach to learning MLTL formulas. Details of the previous work may be found here. Approaches in this report builds on the following components of the previous work:

1. A parser (Python) for MLTL formulas, that can compute various properties of the formula, such as the number of atomic propositions, the syntax tree, worst propagation delay (wpd) (defined in [10]), and various other useful functions.
2. An MLTL interpreter (C++): on input trace π and a MLTL formula φ, determines if $\pi \models \varphi$.
3. A dataset generator (Python) that given an MLTL formula, uses either random trace sampling or regular expression sampling (see [6]) to generate a set of positive and negative examples.

As a group, we explore four additional approaches to MLTL inference: informed beach search, template-driven search, sequence to sequence deep neural networks using Transformer models, and a correspondence between MLTL and dynamic Bayesian networks.

4.1 Formula Evaluator

Each of the explored techniques require an MLTL evaluator to decide whether a trace satisfies or dissatisfies an given MLTL formula. We initially planned to use the WEST Evaluator[6] which is a high performance C++ MLTL evaluator that has been validated against WEST. However, we eventually switched to using `libmltl`, which was actively being developed as a project for *COM S 512 Formal Methods in Software Engineering*, as it offers significant performance improvements and more advanced capabilities. `libmltl` provides robust and extensible interfaces for both C++ and Python that enable MLTL parsing, AST manipulation and evaluation capabilities. The AST manipulations enabled by `libmltl` allow for performant analysis and transformation of MLTL formulas. `libmltl` was validated to a high degree of confidence against WEST and includes a regression test suite to mitigate the risk of bugs in future versions. As shown in Figure 2a, `libmltl` can evaluate traces more than 16 times longer than MLTL Evaluator in the same period of time. Figure 2b shows that `libmltl` achieves a speedup of 37–217$\times$ over the previous state-of-the-art when evaluating traces between length 4 and 64.



(a) Average MLTL formula evaluation time in milliseconds for traces of increasing length.

(b) Average speed up of `libmltl` over the WEST MLTL Evaluator.

Fig. 2: Performance comparison between `libmltl` and the WEST MLTL Evaluator.

4.2 Sequence to Sequence Neural Network

The neural network approach involves a supervised learning model with randomly generated formulas and randomly generated positive and negative traces for the formula. The goal is for the neural network model to infer the correct MLTL formula for a randomly generated group of traces.

Neural Network Models For the task of learning MLTL formulas from examples, three neural network models will be applied:

Sequence to Sequence is a neural network architecture often used for translation tasks. There are two parts to a Sequence to Sequence model: encoder and decoder. The encoder encodes the input sequence, and the decoder decodes the encoding to match the target sequence.

Transformer is a neural network used to process sequences. Transformers are popular in Natural Language Processing for their ability to identify context in text and parallel processing. Transformer self-attention mechanism is useful for its ability to identify relationship between tokens in a sequence.

Long-Short Term Memory (LSTM) is a recurrent neural network used to process sequences. LSTM is useful for encoding sequences with long term dependencies.

Input Ordering Problem A group of traces may be arranged in any order, and the model should evaluate to the same formula for all orderings. In other words, a group of traces is order invariant. However, the order of each boolean proposition in each trace matters in producing the formula, and a different ordering of boolean proposition in a

trace may produce a different formula. In other words, boolean propositions in a trace are order dependent. A neural network solution must reflect this property of the input traces.

Implementation For this problem, the input to the model consists of 128 positive traces and 128 negative traces. The output of the model is the MLTL formula that satisfies those traces. Each of the traces is fed into an LSTM for encoding the order of traces. The positive traces are fed into a group of LSTMs that encode only positive traces, while the negative traces are fed into a separate group of LSTMs that encode only the negative traces. In total, there are 256 LSTMs. Then, the output of the LSTMs is fed into a Transformer Encoder to further encode the traces and their relationships to each other. Finally, the output of the Transformer Encoder is fed into the Transformer Decoder to decode and output an MLTL formula. After each training batch, to maintain order invariant among the traces, the weights of the positive LSTMs are averaged and redistributed among all the LSTMs, and the same procedure is followed for the negative LSTMs.

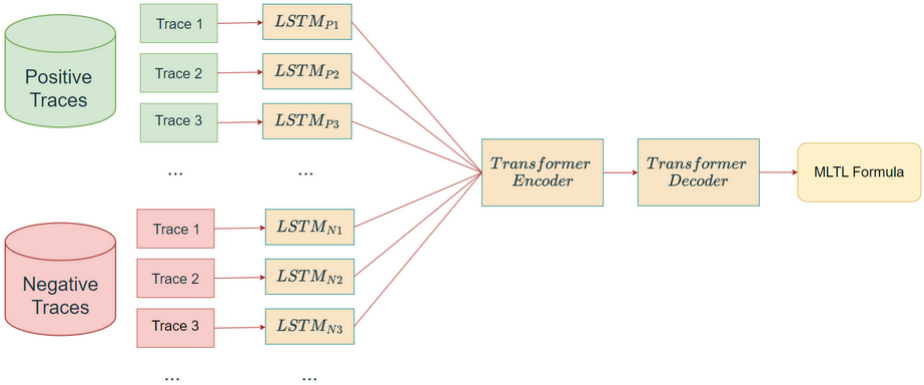


Fig. 3: Model diagram of the Sequence to Sequence architecture

Results Unfortunately, due to limited computational and storage resources, the model was only able to train on 1000 examples for 100 epochs. The results from the training show that the model tends to overfit the data, implying that more data are necessary for a better model. The results also reveal another issue of repeating tokens in the output formula, known as text degeneration, indicating that more research is required in this area to improve the model for longer training.

4.3 Template Driven Search

Template Driven Search approaches the problem of synthesizing Mission Time Linear Temporal Logic (MLTL) formulas from given positive and negative traces using predefined formula templates. Synthesizing MLTL formulas from positive and negative traces is a challenging task that requires exploring a large search space of possible formulas. In this approach, we employ a CEGIS-based technique that combines inductive synthesis, iterative deepening search, and counterexample-guided abstraction refinement to efficiently synthesize MLTL formulas.

Formula Templates The approach utilizes a set of predefined templates to generate candidate formulas. The templates are based on common specification patterns observed in aerospace system telemetry [14]. They include:

- Existence: $F[a, b]\varphi$
- Universality: $G[a, b]\varphi$
- Disjunction: $F[a, b](\varphi_1 \vee \varphi_2)$
- Conjunction: $G[a, b](\varphi_1 \wedge \varphi_2)$
- Until: $\varphi_1 U[a, b]\varphi_2$
- Release: $\varphi_1 R[a, b]\varphi_2$

where φ, φ_1, and φ_2 are propositional formulas, and a and b are time bounds.

Inductive Synthesis The inductive synthesis component generates candidate MLTL formulas based on predefined templates. The templates capture common temporal patterns and are instantiated with propositions and time bounds. The candidate formulas are ranked using a consistency scoring heuristic that measures their consistency with the positive and negative traces.

Consistency Scoring Heuristic The consistency scoring heuristic evaluates the quality of candidate formulas by measuring their consistency with the positive and negative traces. The heuristic assigns higher scores to formulas that correctly classify a larger number of traces. The ranked formulas are then pruned based on a threshold to select the most promising candidates.

Iterative Deepening Search To explore the search space efficiently, we employ an iterative deepening search strategy. The search starts with a shallow depth and gradually increases the depth until a consistent formula is found or a maximum depth is reached. This allows the approach to find simpler formulas first before exploring more complex ones.

Counterexample-Guided Abstraction Refinement (CEGAR) The CEGAR loop iteratively refines the abstraction based on counterexamples. If a candidate formula is found to be inconsistent with a counterexample trace, the abstraction is refined by splitting the relevant propositions. The refined abstraction allows for more precise formula generation in subsequent iterations.

The refinement step $Refine(\mathcal{M}_i, c_i)$ takes the current abstract model $\mathcal{M}_i$ and the spurious counterexample c_i and produces a refined abstract model $\mathcal{M}_{i+1}$ that eliminates the spurious behavior [4].

The refinement can be formally defined as:

$$\mathcal{M}_{i+1} = \mathcal{M}_i \backslash \{s \in \mathcal{M}_i \mid s \models c_i \wedge s \not\models \mathcal{C}\} \tag{1}$$

where $\mathcal{C}$ represents the concrete model.

Counterexample-Guided Inductive Synthesis (CEGIS) The CEGIS approach iteratively refines the candidate formula space based on counterexamples. Let $\mathcal{S}$ be the set of all possible candidate formulas, and φ be the specification that the synthesized formula must satisfy.

The CEGIS approach can be defined as follows:

Algorithm 1 CEGIS

1: Initialize the candidate formula space: $\mathcal{S}_0 = \mathcal{S}$
2: **for** each iteration $i = 0, 1, 2, \ldots$ **do**
3: Choose a candidate formula: $s_i \in \mathcal{S}_i$
4: Verify if s_i satisfies the specification: $s_i \models \varphi$
5: **if** $s_i \models \varphi$ **then**
6: **return** s_i as the synthesized formula
7: **else**
8: Find a counterexample c_i such that $s_i \not\models c_i$
9: Refine the candidate formula space: $\mathcal{S}_{i+1} = \mathcal{S}_i \backslash \{s \in \mathcal{S}_i \mid s \not\models c_i\}$

The CEGIS loop terminates when a candidate formula satisfies the specification or when the candidate formula space becomes empty [1]. Since this approach is highly template dependant using the basic templates we were able to synthesize all the basic formulas with 100 percent accuracy.

4.4 Beam Search

Beam Search is a variant of best-first search that uses breadth-first search to explore the next layer of the search tree, but limits the number of successors to a predetermined amount b, known as the *beam width*. Beam search with an infinite beam width would be equivalent to best-first search. At each level, all successor nodes are generated and sorted by a heuristic cost function, h. Only the b candidate nodes with the lowest cost are kept, and all other nodes are discarded. This limits memory usage and maintains tractability for large state space problems, such as MLTL-inference. The primary disadvantage of beam search is the possibility of pruning all paths to the optimal goal state.

Search Problem Formulation This section details the search problem formulation we used to reduce MLTL-inference to a search problem.

State Space – The space of possible MLTL formulas is constructed recursively from the given alphabet of propositions and operators.

$$\varphi, \psi := true \mid false \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathcal{F}_{[a,b]}\varphi \mid \mathcal{G}_{[a,b]}\varphi \mid \varphi\mathcal{U}_{[a,b]}\psi \mid \varphi\mathcal{R}_{[a,b]}\psi$$

Initial State – The empty formula.

Goal State – A formula that best describes the given positive and negative traces, according to evaluation criteria that include accuracy and simplicity.

Actions – Operations can be applied to a current formula to generate new candidate formula. This includes adding a new operator, modifying an existing operator, or combining multiple formulas.

Cost Function – The cost of a node is considered to be $1 - accuracy$, where accuracy is evaluated over all input traces in the training dataset.

Termination Criteria – The search should terminate once a predetermined depth, accuracy, or time-limit is reached.

Implementation Developing heuristics to explore the enormous state space in an efficient manner proved to be very challenging. This sub-section discuss our implementation of beam search for MLTL-inference and outlines the techniques we employed to address these challenges.

Our first observation was that constructing formulas one operator and propositional variable at a time was wasteful. Consider that temporal operators often contain simple boolean function, for example, $\mathcal{F}_{[0,10]}\ (p_0 \wedge p_1) \vee (p_1 \wedge p_2)$. If we construct this formula by adding one additional operator and propositional variable at each level of the search then constructing this formula would require several levels of search to be completed. Instead, we try to pre-generate *interesting* boolean functions that can be considered for inclusion at any point in the search tree. Generating boolean functions is a hard problem, since for each variable there are two possible assignments and for each unique assignment of all variables there are two possible corresponding truth values. Thus the run time to enumerate all boolean function for n propositional variables is double exponential, $O(2^{2^n})$. The number of boolean functions reaches the capacity of a 32-bit integer with just 5 variables. To keep this problem a reasonable size and we limit the size of boolean functions we generate to contain at most 2-4 unique variables. It is trivial to construct a boolean function in *Conjunctive Normal Form* (CNF) or *Disjunctive Normal Form* (DNF) given a truth table, however, this creates long formulas which counters the goal of finding concise representations. To address this we pass each boolean function through the Quine-McCluskey algorithm[20,21,16] which returns an equivalent boolean function as a simplified DNF.

To determine whether a boolean function may be useful as part of the construction of MLTL formulas, we begin by evaluating each function as the operand to the Finally and Globally operators over all training traces. If the boolean function as an operand to one of these two unary temporal operators achieves an accuracy of over 50% then we consider it to be *interesting*, otherwise we will discard it. We also always consider all variables are on their own to be *interesting* since this allows us to construct functions that we may have prematurely dismissed. We also discard boolean expressions that even when simplified are very long.

Iterating over every possible combination of bounds for each operator can quickly make this problem intractable, so we fix the amount that the bounds can be stepped to some fraction of the average trace length. In general using approximate bounds can still result in accurate formulas as long as the structure and operators of the formula are correct. By giving up precision on the operator bounds we can identify the correct structure and operators in a practical amount of time. After identifying the correct formula structure another algorithm could be used to optimize the operator bounds.

In our implementation each depth of the search tree is defined by the number of nested temporal operators. As a result most practical MLTL formulas will be found at depths 1 to 3. To build formulas of the next level we use already explored formulas of the highest accuracy as operands to new formulas. This heuristic tends to lead towards good solutions, but can sometimes make the optimal solution unreachable, hence this is an inadmissible heuristic.

After all the above heuristics and simplifying assumptions, the problem can now be computed in a reasonable amount of time. Exploring each subsequent level of the search tree is now mainly limited by evaluation time. Since the evaluation of each formula can be done independently of any other formula, this is embarrassingly parallel. As such we have paralyzed the algorithm to run across all available CPU cores.

4.5 MLTL and Dynamic Bayesian Networks

The original motivation for exploring the relationship between MLTL and Bayesian came from the recent advancements in structural learning algorithms for Bayesian networks from data. For instance, [23] describes over 74 algorithms for structure learning in Bayesian networks, while easily accessible Python libraries, such as `pgmpy` and `bnlearn`, implements many state-of-the-art algorithms such as hill-climbing search, Chow-Liu, Tree-augmented Naive Bayes, and constraint-based methods.

The idea to apply structural learning algorithms to MLTL inference as follows:

1. Define a correspondence between MLTL formulas and Bayesian networks, in which network structure corresponds to formula syntax, and network parameters corresponds to formula semantics.
2. Apply Bayesian network structure learning algorithms to learn a DAG structure from data, and use this to infer a most likely MLTL formula.

The first step was successful, and the translation from MLTL to Bayesian networks is detailed below. However, we were not able to formulate a well defined structure learning problem that would allow us to extract a MLTL formula from a learned Bayesian network. The most significant hurdle lies in modeling latent variables in the data, and we previously thought this was not doable with Bayesian networks. Upon further research, however, we found the Structural Expectation Propagation (SEP) algorithm for this exact problem [11]. Thus, there is much work to be done in exploring this avenue of research, and the second step is not necessarily a failure.

MLTL and Bayesian Networks The correspondence between MLTL formulas and Bayesian Networks is recursively defined. Given a MLTL formula φ, φ is said to be **temporal** if φ contains a $\mathcal{F}, \mathcal{G}, \mathcal{U}, \mathcal{R}$ operator, and **non-temporal** otherwise. As one might expect, non-temporal formulas correspond to classical Bayesian networks while temporal formulas correspond to Dynamic Bayesian networks. Additionally, all nodes Bayesian networks are assumed to be Boolean nodes taking values in $\{0, 1\}$.

For non-temporal formulas, the structure of the Bayesian network is exactly that of the abstract syntax tree. For temporal formulas, the nodes of the Bayesian network creates copies of the subformulas over time, as well as copies of the final output node over time. When a node φ is duplicated over time, $\varphi^{(t)}$ denotes the evaluation of the node at time t. The DAG structures and conditional probabilities tables (CPTs) of each case is outlined below.

- If $\psi = \neg\varphi$, then figure 4 shows the DAG structure, and the CPT) for ψ is defined $P(\psi = 1|\varphi = 0) = 1$.
- If $\psi = \varphi_1 \wedge \varphi_2$, then figure 5 shows the DAG structure, and the CPT for $P(\psi|\varphi_1, \varphi_2)$ is defined as follows:

φ_1	0		1	
φ_2	0	1	0	1
$P(\psi = 0)$	1	1	1	0
$P(\psi = 1)$	0	0	0	1

– If $\psi = \varphi_1 \vee \varphi_2$, then figure 5 shows the DAG structure, and the CPT for $P(\psi|\varphi_1, \varphi_2)$ is defined as follows:

φ_1	0		1	
φ_2	0	1	0	1
$P(\psi = 0)$	0	1	1	1
$P(\psi = 1)$	1	0	0	0

– $\psi = \mathcal{F}_{[a,b]}\varphi$, then figure 6 shows the DAG structure. For all $t < a$ and $t > b$, we have that $P(\psi^{(t+1)} = 1|\psi^{(t)} = 1) = 1$, i.e., the value of the next node is exactly the value of the previous node. Additionally, $P(\psi^{(0)=0} = 1$. For $t \in [a, b]$, the CPT for $P(\psi^{(t+1)}|\psi^{(t)}, \varphi^{(t+1)})$ is as follows:

$\varphi^{(t+1)}$	0		1	
$\psi^{(t)}$	0	1	0	1
$P(\psi^{(t+1)} = 0)$	1	0	0	0
$P(\psi^{(t+1)} = 1)$	0	1	1	1

– $\psi = \mathcal{G}_{[a,b]}\varphi$, then figure 6 shows the DAG structure. For all $t < a$ and $t > b$, we have that $P(\psi^{(t+1)} = 1|\psi^{(t)} = 1) = 1$, i.e., the value of the next node is exactly the value of the previous node. Additionally, $P(\psi^{(0)=1} = 1$ (note the difference from the $\mathcal{F}$ case). For $t \in [a, b]$, the CPT for $P(\psi^{(t+1)}|\psi^{(t)}, \varphi^{(t+1)})$ is as follows:

$\varphi^{(t+1)}$	0		1	
$\psi^{(t)}$	0	1	0	1
$P(\psi^{(t+1)} = 0)$	1	1	1	0
$P(\psi^{(t+1)} = 1)$	0	0	0	1

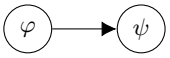


Fig. 4: Bayesian Network structure for $\psi = \neg\varphi$

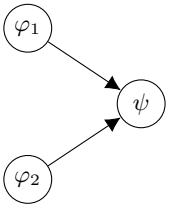


Fig. 5: Bayesian Network structure for $\psi = \varphi_1 \wedge \varphi_2$ and $\psi = \varphi_1 \vee \varphi_2$

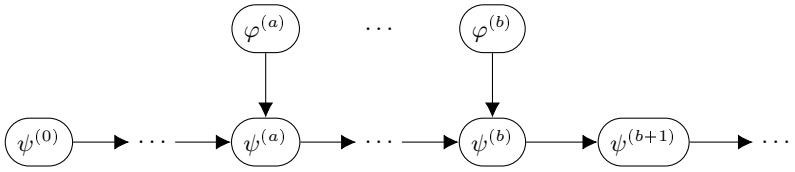


Fig. 6: Bayesian Network structure for $\psi = \mathcal{F}_{[a,b]}\varphi$ and $\psi = \mathcal{G}_{[a,b]}\varphi$

– $\psi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$, we again have that for all $t < a$ and $t > b$, we have that $P(\psi^{(t+1)} = 1 | \psi^{(t)} = 1) = 1$, and $P(\psi^{(0)=0} = 1$. The sequence of nodes $\alpha^{(t)}$, for $t \in [a, b]$, checks that φ_1^i holds for $i \in [a, t]$ using the same CPD defined for the Globally case. For $t \in (a, b]$, the CPT for $P(\psi^{(t+1)} | \psi^{(t)}, \alpha^{(t)}, \varphi_2^{t+1})$ is defined as:

$\psi^{(t)}$	0				1			
$\alpha^{(t)}$	0		1		0		1	
$\varphi_2^{(t+1)}$	0	1	0	1	0	1	0	1
$P(\psi^{(t+1)} = 0)$	1	1	1	0	0	0	0	0
$P(\psi^{(t+1)} = 1)$	0	0	0	1	1	1	1	1

– For the last case of $\psi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$, we have that for all $t < a$ and $t > b$, we have that $P(\psi^{(t+1)} = 1 | \psi^{(t)} = 1) = 1$, and $P(\psi^{(0)=0} = 1$. The sequence of nodes $\alpha^{(t)}$, for $t \in [a, b]$, checks that φ_2^i holds for $i \in [a, t]$ using the same CPD defined for the Globally case. For $t \in [a, b)$, the CPT for $P(\psi^{(t+1)} | \psi^{(t)}, \alpha^{(t+1)}, \varphi_1^{t+1})$ is defined as:

$\psi^{(t)}$	0				1			
$\alpha^{(t+1)}$	0		1		0		1	
$\varphi_2^{(t+1)}$	0	1	0	1	0	1	0	1
$P(\psi^{(t+1)} = 0)$	1	1	1	0	0	0	0	0
$P(\psi^{(t+1)} = 1)$	0	0	0	1	1	1	1	1

Potential Future Work The correspondence between MLTL formulas and Bayesian networks is a promising avenue for future research. The current translation from MLTL to Bayesian networks is represented relatively efficiently as a Dynamic Bayesian network, and the next step is to apply structural learning algorithms with latent variable support to learn a DAG structure from data.

However, such a translation from MLTL to Bayesian networks has applications beyond MLTL inference. When querying such a Bayesian network, if the evidence is provided for the propositional variables, the network is fully deterministic and simply evaluates the MLTL formula. However, if the evidence is instead provided for the output nodes, the network can be used to infer the distribution of the propositional variables at each time step. Such a distribution on the *input* variables can be used to sample traces that captures more a wider range of behaviors than the original training data that was generated by assuming each atomic proposition was i.i.d. Bernoulli with probabilty 0.5. This is a powerful tool for generating synthetic data for training MLTL inference algorithms, and can be used to generate more complex traces that are more representative of the true system behavior.

Additionally, in the domain of sensor fault tolerance analysis, the result of such a query can be used to identify the required tolerance of the system to sensor faults. For example if the system is required to maintain a certain property with a certain probability, this evidence can be used as input to the Bayesian network to determine the distribution of sensor values needed. A minimum tolerance for a sensor can then be calculated by taking the maximum values of the requirements for the sensor across all time steps.

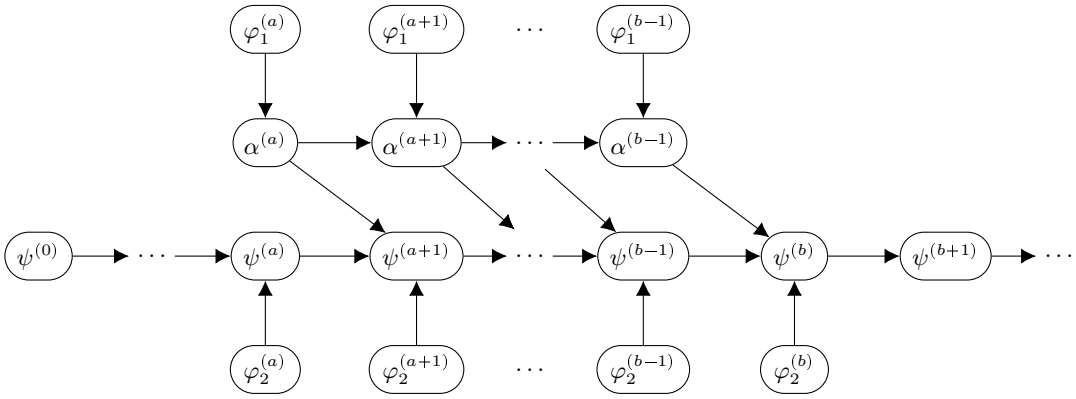


Fig. 7: Bayesian Network structure for $\psi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$

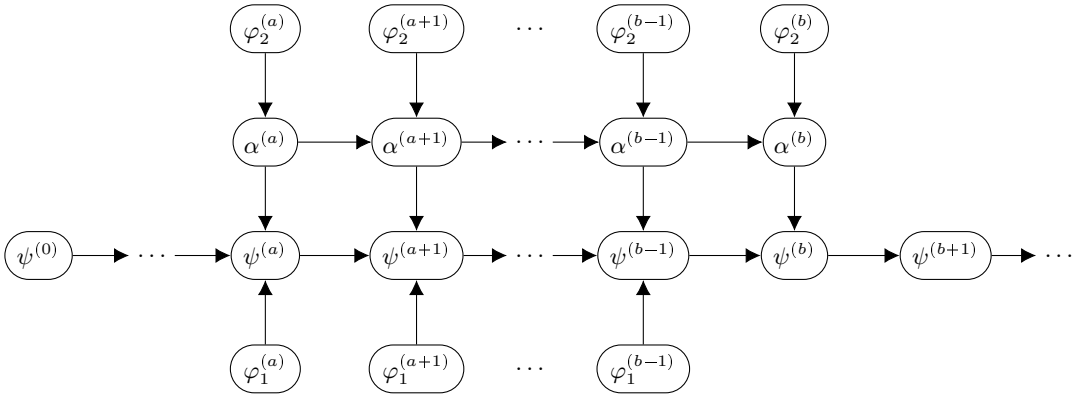


Fig. 8: Bayesian Network structure for $\psi = \varphi_1 \mathcal{R}_{[a,b]} \varphi_2$

5 Evaluation

We generate benchmarks by first choosing a target formula φ, and then generating a set of positive traces and negative traces $\Pi = \Pi^+ \cup \Pi^-$. Since the WEST tool [**?**] gives a regular expression for all traces satisfying φ, we can generate positive traces by randomly sampling the regular expressions. Likewise, we can sample negative traces by running WEST on $\neg\varphi$. However, WEST was not intended for the purpose of solving all-SAT for MLTL, and times out on some of the more difficult benchmarks. In these cases, we may default to randomly sampling traces, with each proposition being true with probability $\frac{1}{2}$, to populate Π^+ and Π^-. In total, we created 11 benchmarks, with 4 benchmarks that target the basic temporal operators, and 7 benchmarks that use MLTL formulae found in the literature. For each benchmark, we produced a total of 750 positive traces and 750 negative traces. 250 positive traces and 250 negative traces were used for training, and the remaining 1000 traces were held out for testing.

5.1 Datasets

We give a brief overview of the benchmarks used in this study. Four benchmarks target the basic temporal operators, and seven benchmarks use MLTL formulae found in the literature.

Basic Temporal Operators targets the basic temporal operators $\mathcal{F}, \mathcal{G}, \mathcal{U}, \mathcal{R}$. The target formulae are:

1. $\mathcal{F}_{[0,10]}(p_0 \vee p_1)$
2. $\mathcal{G}_{[0,10]}(p_0 \wedge (p_1 \wedge \neg p_2))$
3. $(p_0 \, \mathcal{R}_{[0,10]} \, p_1)$
4. $(p_0 \, \mathcal{U}_{[0,10]} \, p_1)$

DragonEye UAS was studied by Moosbrugger et al. [17] using the R2U2 tool. R2U2 is a framework for runtime monitoring of system health in Unmanned Aerial Systems (UAS). Several formulas we used from the case study are:

1. $\mathcal{G}_{[0,10]} \neg(\neg p_0 \wedge \neg(p_1 \, \mathcal{U}_{[0,10]} \, p_0))$
2. $\mathcal{G}_{[0,65525]} \neg(p_6 \wedge \neg(((\neg p_2 \wedge \neg p_3) \wedge \neg p_4) \, \mathcal{U}_{[0,65525]} \, p_5))$
3. $\mathcal{G}_{[0,300]}((p_4 \wedge p_5) \wedge p_6)$

Note that formulas 2 and 3 had unfeasibly long computation lengths, and so we scaled the computations lengths down to 20.

NASA's NextGen Air Traffic Control is a air traffic management system that was evaluated by Gario et al. [7]. They came up with 1620 most relevant designs, and we used 2 MLTL formulas from their case study as benchmarks. The target formulae are:

1. $\neg((p_1 \wedge \neg(\mathcal{G}_{[505,713]}((\mathcal{G}_{[100,100]}(\text{true} \, \mathcal{U}_{[201,979]} \, p_0)) \wedge (\mathcal{G}_{[0,287]}(\text{true} \, \mathcal{U}_{[431,483]} \neg p_0)))) \wedge \text{true})$
2. $(\mathcal{G}_{[369,504]}((\mathcal{G}_{[0,34]}(\neg(p_0 \wedge \neg(\mathcal{G}_{[605,1000]} p_0)))) \wedge (\mathcal{G}_{[496,496]}(\neg(p_0 \wedge \neg(\neg(\text{true} \, \mathcal{U}_{[887,1000]} \neg p_0)))))) \wedge \text{true})$

Again, we scaled the computation lengths down to 20 for computational feasibility.

Dataset	Generating Formula	Formula Size	Num. Variables	Avg. Trace Length
basic_future	$\mathcal{F}_{[0,10]}(p_0 \vee p_1)$	4	2	14.9
basic_global	$\mathcal{G}_{[0,10]}(p_0 \wedge p_1 \wedge \neg p_2)$	7	3	14.6
basic_release	$p_0 \mathcal{R}_{[0,10]} \, p_2$	3	3	14.0
basic_until	$p_1 \mathcal{U}_{[0,10]} \, p_2$	3	3	14.0
fmsd17_formula1	$\mathcal{G}_{[0,10]} \neg(\neg p_0 \wedge \neg(p_1 \mathcal{U}_{[0,10]} \, p_0))$	9	2	24.0
fmsd17_formula2	$\mathcal{G}_{[0,99]} \neg(p_6 \wedge \neg((\neg p_2 \wedge \neg p_3 \wedge \neg p_4) \mathcal{U}_{[0,99]} \, p_5))$	15	7	23.9
fmsd17_formula3	$\mathcal{G}_{[0,99]}(p_4 \wedge p_5 \wedge p_6)$	6	7	14.0
nasa-atc_formula1	$\mathcal{G}_{[0,10]} \neg(\neg p_0 \wedge \neg(p_1 \mathcal{U}_{[0,10]} \, p_0))$	9	2	23.9
nasa-atc_formula2	$\mathcal{G}_{[0,99]} \neg(p_6 \wedge \neg((\neg p_2 \wedge \neg p_3 \wedge \neg p_4) \mathcal{U}_{[0,99]} \, p_5))$	15	7	23.9
rv14_formula1	$\mathcal{G}_{[0,99]} \neg(p_1 \wedge (\mathtt{tt} \mathcal{U}_{[0,30]} \, p_0) \wedge (\mathtt{tt} \mathcal{U}_{[0,99]} \, p_2))$	11	3	14.0
rv14_formula2	$\mathcal{G}_{[0,99]} \neg(\neg p_0 \wedge \neg(p_1 \mathcal{U}_{[0,30]} \, p_0))$	9	2	14.1

Fig. 9: Dataset statistics

5.2 Results

Since informed beam search and template driven search were the only algorithms that succesfully yielded results, we give an individual summary of the results for these two algorithms in table 10 and table 11 respectively.

Dataset	Best Formula(s) Found	Formula Size	Compute Time (s)	Formula Accuracy
basic_future	$\mathcal{F}_{[0,10]}(p_1 \vee p_0)$	4	0.1	100%
basic_global	$\mathcal{G}_{[0,10]}(p_0 \wedge p_1 \wedge \neg p_2)$	7	0.2	100%
basic_release	$p_0 \mathcal{R}_{[0,10]}\ p_2$	3	0.2	100%
basic_until	$p_1 \mathcal{U}_{[0,10]}\ p_2$	3	0.2	100%
fmsd17_formula1	$\mathcal{G}_{[0,10]}(p_1 \mathcal{U}_{[0,15]}\ p_0)$	4	1.4	100%
fmsd17_formula2	$\mathcal{G}_{[0,10]}(((\neg p_6 \vee p_5)\mathcal{U}_{[0,15]}\ (\neg p_2 \wedge \neg p_3)) \vee \neg p_6 \vee p_5)$	16	13.3	98.2%
	$\mathcal{G}_{[0,10]}((\neg p_6 \mathcal{U}_{[0,5]}\ p_5) \vee \neg p_6 \vee \neg p_4)$ $\mathcal{R}_{[0,10]}\ \mathcal{G}_{[0,10]}(((\neg p_6 \vee p_5)\mathcal{U}_{[0,5]}\ (\neg p_2 \wedge \neg p_3)) \vee \neg p_6 \vee p_5)$	28	138.9	99.4%
fmsd17_formula3	$\mathcal{G}_{[0,10]}(p_4 \wedge p_5 \wedge p_6)$	6	19.4	100%
nasa-atc_formula1	$\mathcal{G}_{[0,10]}(p_1 \mathcal{U}_{[0,10]}\ p_0)$	4	2.3	100%
nasa-atc_formula2	$\mathcal{G}_{[0,10]}(((\neg p_6 \vee p_5)\mathcal{U}_{[0,10]}\ (\neg p_2 \wedge \neg p_3)) \vee \neg p_6 \vee p_5)$	16	21.8	97.6%
	$\mathcal{G}_{[0,10]}(((\neg p_6 \vee \neg p_4)\mathcal{U}_{[0,5]}\ p_5) \vee \neg p_6)$ $\mathcal{R}_{[0,10]}\ \mathcal{G}_{[0,10]}((\neg p_2 \wedge \vee p_3)\mathcal{U}_{[0,5]}(\neg p_6 \vee p_5))$	23	4022.4	99.2%
rv14_formula1	$\mathcal{G}_{[0,10]}((p_1 \mathcal{R}_{[0,5]}\ (\neg p_2 \vee \neg p_0)) \vee \neg p_1)$	11	2.3	100%
	$\mathcal{G}_{[0,10]}(\neg p_2 \vee \neg p_1 \vee \neg p_0)$	9	3.3	100%
rv14_formula2	$\mathcal{G}_{[0,10]}((p_0 \wedge p_1)\mathcal{U}_{[0,5]}\ p_0)$	6	0.8	100%

Fig. 10: Beam Search results

Dataset	Best Formula(s) Found	Formula Size	Compute Time (s)	Formula Accuracy
basic_future	$\mathcal{F}_{[0,10]}(p_1 \vee p_0)$	4	1.0	100%
basic_global	$\mathcal{G}_{[0,10]}(p_0 \wedge p_1 \wedge \neg p_2)$	7	7.9	100%
basic_release	$p_0 \mathcal{R}_{[0,10]}\ p_2$	3	1.0	100%
basic_until	$p_1 \mathcal{U}_{[0,10]}\ p_2$	3	1.8	100%

Fig. 11: Template Driven Search results

Additionally, the development of `libmltl` also retroactively improved the performance of the previous approach using Genetic Programming. This is because the bottle neck of the Genetic Programming approach was the evaluation of the fitness function, which required the evaluation of a candidate formula on all traces in the dataset. A comparison of the performance of all three approaches is given in figure 12.

As can be seen from the results, the template driven search approach was able to find the correct formula for the four basic temporal operators benchmarks. However, it was unable to terminate within a reasonable amount of time for the more complex benchmarks. The previous approach using Genetic Programming is able to learn formulas in a reasonable amount of time for the more complex benchmarks, but the accuracy of the learned formulas drops on the more complex benchmarks. Beam search achieved the best results, finding formulas for all benchmarks, including the more complex benchmarks, with high accuracy and within a reasonable amount of time.

6 Conclusion

To conclude this project, we have explored the problem of MLTL inference with four new approaches: informed beam search, template driven search, sequence to sequence

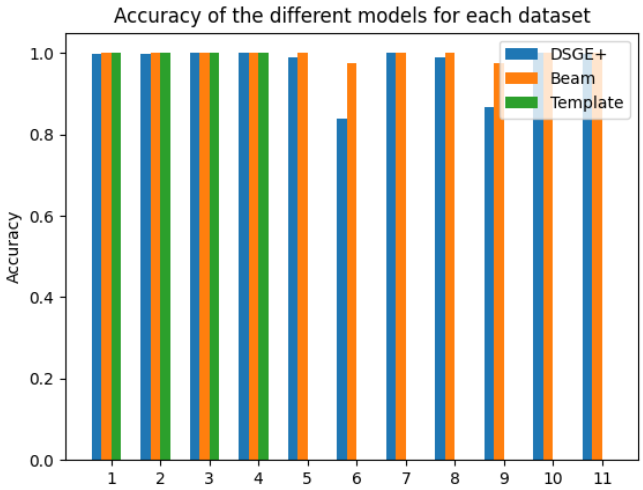


Fig. 12: Comparison of performance of Genetic Programming, Beam Search and Template Driven Search

neural networks, and Bayesian networks. This project has been immensely succesful in exploring the feasibility of these approaches, and beam search in particular has yielded promising results. Template driven search leverages more domain specific knowledge, and further work is needed to improve the scalability of this approach. Although the sequence to sequence neural network approach was not able to learn MLTL formulas, it was still a novel application of sequence to sequence neural networks to the problem of MLTL inference. Future work is needed to better frame the MLTL inference problem as a supervised learning problem to leverage the power of neural networks. Lastly, although the translation of MLTL formulas to Bayesian networks did not give an immediate solution to the MLTL inference problem, it is a novel way of modeling temporal logic that has applications beyond MLTL inference. Additionally, we still need to explore the feasibility of using algorithms with latent variable support to learn the structure of the Bayesian network from data, and infering a MLTL formula from such a learned structure.

7 Statement of Contributions

Throughout this project, all team members have consistently collaborated, from the initial brainstorming of approaches to discussions of challenges encountered and potential solutions. The project's workload was divided in a manner that allowed each team member to leverage their expertise and interests to explore the feasibility of a different approach. The notable contributions of each team member are outlined below.

Zili Wang – Lead the project, provided support with connect previously written code to the new approaches, and developed the MLTL to Bayesian network translation. Generated benchmarks for evaluation, and wrote remaining portions of the report beyond the individual contributions.

Luke Marzen – Contributed the beam search based solution for the MLTL-inference problem. Wrote a working multi-threaded beam search implementation in C++ and evaluated its performance. Implemented the Quine-McCluskey algorithm to generate simplified DNF boolean functions.

Nhan Tran – Contributed the Sequence to Sequence Neural Network model. Wrote an encoding scheme for the traces and formulas. Identified problems and possible solutions with Neural Network approach.

Swaminathan Jayaraman – Contributed the template drive search approach for the MLTL formula synthesis. Explored and implemented a form of Counterexample Guided Inductive Synthesis (CEGIS) logic in python through a combination of inductive synthesis, heuristic scoring, iterative deepening search and abstraction refinement.

References

1. Abate, A., David, C., Kesseli, P., Kroening, D., Polgreen, E.: Counterexample guided inductive synthesis modulo theories. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification. pp. 270–288. Springer International Publishing, Cham (2018)
2. Aurandt, A., Jones, P.H., Rozier, K.Y.: Runtime Verification Triggers Real-Time, Autonomous Fault Recovery on the CySat-I. In: Proceedings of the 13th NASA Formal Methods Symposium (NFM). pp. 816–825. LNCS, Springer International Publishing (2022). https://doi.org/10.1007/978-3-031-06773-0_45
3. Camacho, A., McIlraith, S.A.: Learning interpretable models expressed in linear temporal logic. Proceedings of the International Conference on Automated Planning and Scheduling **29**(1), 621–630 (May 2021). https://doi.org/10.1609/icaps.v29i1.3529, https://ojs.aaai.org/index.php/ICAPS/article/view/3529
4. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) Computer Aided Verification. pp. 154–169. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
5. Dabney, J.B., Badger, J.M., Rajagopal, P.: Adding a verification view for an autonomous real-time system architecture. In: AIAA Scitech 2021 Forum. p. 0566 (2021)
6. Elwing, J., Gamboa-Guzman, L., Sorkin, J., Travesset, C., Wang, Z., Rozier, K.Y.: Mission-time LTL (MLTL) formula validation via regular expressions. In: Herber, P., Wijs, A. (eds.) iFM 2023. pp. 279–301. Springer Nature Switzerland, Cham (2024)
7. Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., Rozier, K.Y.: Model checking at scale: Automated air traffic control design space exploration. Computer Aided Verification (2016). https://doi.org/10.1007/978-3-319-41540-6_1, https://doi.org/10.1007/978-3-319-41540-6_1
8. Hammer, A., Cauwels, M., Hertz, B., Jones, P., Rozier, K.Y.: Integrating Runtime Verification into an Automated UAS Traffic Management System. Innovations in Systems and Software Engineering: A NASA Journal (July 2021). https://doi.org/10.1007/s11334-021-00407-5
9. Hertz, B., Luppen, Z., Rozier, K.Y.: Integrating runtime verification into a sounding rocket control system. In: Proceedings of the 13th NASA Formal Methods Symposium (NFM 2021). pp. 151–159. LNCS, Springer International Publishing (May 2021). https://doi.org/10.1007/978-3-030-76384-8_10
10. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2. In: Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS). Lecture Notes in Computer Science (LNCS), vol. 12288, pp. 196–214. Springer, Vienna, Austria (September 2020). https://doi.org/10.1007/978-3-030-57628-8_12
11. Lazic, N., Bishop, C., Winn, J.: Structural expectation propagation (sep): Bayesian structure learning for networks with latent variables. In: Carvalho, C.M., Ravikumar, P. (eds.) Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 31, pp. 379–387. PMLR, Scottsdale, Arizona, USA (29 Apr–01 May 2013), https://proceedings.mlr.press/v31/lazic13a.html
12. Li, D., Cai, M., Vasile, C.I., Tron, R.: Learning signal temporal logic through neural network for interpretable classification. In: 2023 American Control Conference (ACC). pp. 1907–1914 (2023). https://doi.org/10.23919/ACC55779.2023.10156357
13. Luo, W., Liang, P., Du, J., Wan, H., Peng, B., Zhang, D.: Bridging ltlf inference to gnn inference for learning ltlf formulae. Proceedings of the AAAI Conference on Artificial Intelligence **36**(9), 9849–9857 (Jun 2022). https://doi.org/10.1609/aaai.v36i9.21221, https://ojs.aaai.org/index.php/AAAI/article/view/21221

14. Luppen, Z., Jacks, M., Baughman, N., Hertz, B., Cutler, J., Lee, D.Y., Rozier, K.Y.: Elucidation and Analysis of Specification Patterns in Aerospace System Telemetry. In: Proceedings of the 14th NASA Formal Methods Symposium (NFM 2022). Lecture Notes in Computer Science (LNCS), vol. 13260, pp. 527–537. Springer, Cham, Caltech, California, USA (May 2022). https://doi.org/10.1007/978-3-031-06773-0_28
15. Luppen, Z.A., Lee, D.Y., Rozier, K.Y.: Correction: A case study in formal specification and runtime verification of a cubesat communications system. In: AIAA Scitech 2021 Forum. p. 0997. American Institute of Aeronautics and Astronautics, Nashville, TN, USA (January 2021). https://doi.org/10.2514/6.2021-0997.c1
16. McCluskey, E.J.: Minimization of boolean functions. The Bell System Technical Journal **35**(6), 1417–1444 (1956). https://doi.org/10.1002/j.1538-7305.1956.tb03835.x
17. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2u2: monitoring and diagnosis of security threats for unmanned aerial systems. Formal Methods in System Design **51**(1), 31–61 (Aug 2017). https://doi.org/10.1007/s10703-017-0275-x, https://doi.org/10.1007/s10703-017-0275-x
18. Okubo, N.: Using R2U2 in JAXA program. Electronic correspondence (November–December 2020), series of emails and zoom call from JAXA with technical questions about embedding MLTL formula monitoring into an autonomous satellite mission with a provable memory bound of 200KB
19. Pnueli, A.: The Temporal Logic of Programs. In: Proceedings of the 18th IEEE Annual Symposium on the Foundations of Computer Science. pp. 46–57. IEEE Computer Society Press, Providence (1977)
20. Quine, W.V.: The problem of simplifying truth functions. The American Mathematical Monthly **59**(8), 521–531 (1952). https://doi.org/10.1080/00029890.1952.11988183, https://doi.org/10.1080/00029890.1952.11988183
21. Quine, W.V.: A way to simplify truth functions. The American Mathematical Monthly **62**(9), 627–631 (1955). https://doi.org/10.1080/00029890.1955.11988710, https://doi.org/10.1080/00029890.1955.11988710
22. Roy, R., Gaglione, J.R., Baharisangari, N., Neider, D., Xu, Z., Topcu, U.: Learning interpretable temporal properties from positive examples only. In: Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence. AAAI'23/IAAI'23/EAAI'23, AAAI Press (2023). https://doi.org/10.1609/aaai.v37i5.25800, https://doi.org/10.1609/aaai.v37i5.25800
23. Scanagatta, M., Salmerón, A., Stella, F.: A survey on bayesian network structure learning from data. Progress in Artificial Intelligence **8** (05 2019). https://doi.org/10.1007/s13748-019-00194-y
24. Wang, Z., Gamboa Guzman, L., Rozier, K.Y.: West: Interactive validation of mission-time linear temporal logic (mltl), invited submission to Research Software from the integrated Formal Methods (iFM) conference 2023