Mission-time Linear Temporal Logic Inference via Structured Grammar Evolution

Zili Wang

Iowa State University ziliw1@iastate.edu

Abstract. This report examines preliminary work in using Structured Grammar Evolution to tackle the problem of inferring Mission-time Linear Temporal Logic formulas from data.

1 Introduction

Learning intuitive high-level temporal properties from the the observed behaviors of complex systems is an important problem that arises in applications such as specification mining for formal verification, debugging systems, behavior classification, and explainable models.

We focus on the problem of learning, or inferring, Mission-time Linear Temporal Logic (MLTL) properties from observed traces of the system labeled as positive (desirable) or negative (undesirable). The goal is to infer interpretable MLTL classifier over finite traces that can separate the set of positive and negative traces.

We prove a theoretically interesting results for MLTL: that the class of languages recognized by MLTL classifiers is a strict subset of regular languages. Additionally, we propose a novel Grammatical Evolution(GE) algorithm, DSGE$^+$, an implementation of this algorithm, a set of MLTL benchmarks for future research, and experimental evaluation.

2 Related Work

There is a large corpus of work on learning regular languages, stemming from Angluin's $L*$ algorithm [1] that learns a minimal deterministic finite automata (DFA) from positive and negative examples.

Work in learning temporal logic formulae have also recently been explored, and two commonly defined settings are learning from only positive traces, or learning from both positive and negative traces. Roy et al.[13] proposes two approaches for learning LTL$_f$ from only positive traces. The first is a symbolic approach that encodes inference as a constraint satisfaction problem, and the second is a counter-example guided one that generates negative examples in order to guide the learning process. Camacho and McIlraith [2] proposed a symbolic approach to learn LTL$_f$ from positive and negative traces, by using alternating automata to search through a limited space of formula templates. Further assumption of noisy data motivated Luo et al.[9] to propose a deep learning approach to extract candidate formulas from graph neural networks trained on positive and negative traces. In 2018, Nenzi et al. applies evolutionary algorithms to learn Signal Temporal Logic (STL) formulas from (possibly noisy) positive and negative traces.

The problem of learning Mission-time Linear Temporal(MLTL) Logic has not been explored in the literature. Although MLTL is a more recently introduced fragment of Metric Temporal Logic (MTL), there also are no known related work on learning MTL formulae from example traces.

3 Mission-time Linear Temporal Logic

Mission-time linear temporal logic (MLTL)[7] is a finite variation of LTL over bounded, closed, discrete intervals of the form $[a, b]$ where $a, b \in \mathbb{N}$ and $0 \leqslant a \leqslant b$. The syntax of MLTL formulas, over a finite set of atomic propositions $\mathcal{AP}$, where $p \in \mathcal{AP}$ is a propositional variable, and φ and ψ are subformulas is as follows:

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

The symbols $\mathcal{F}, \mathcal{G}, \mathcal{U}, \mathcal{R}$ respectively denote the temporal operators Finally, Globally, Until, and Release. MLTL formulas are semantically defined over finite traces bounded by base-10 intervals. A trace π of length $|\pi| = m$ is a sequence $\{\pi[i]\}_{i=0}^{m-1}$ of sets of atomic propositions, $\pi[i] \subseteq \mathcal{AP}$. An atomic proposition p is true at time step i if and only if $p \in \pi[i]$, and the suffix of π starting at i (inclusive) is denoted by π_i, and $\pi_0 = \pi$. We define that a trace π satisfies MLTL formula α, denoted $\pi \models \alpha$, as follows:

$\pi \models p$ iff $p \in \pi[0]$ $\qquad$ $\pi \models \neg\alpha$ iff $\pi \not\models \alpha$

$\pi \models \alpha \wedge \beta$ iff $\pi \models \alpha$ and $\pi \models \beta$ $\qquad$ $\pi \models \alpha \vee \beta$ iff $\pi \models \alpha$ or $\pi \models \beta$

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

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

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

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

Two formulas φ and ψ are *semantically* equivalent, denoted $\varphi \equiv \psi$, when $\pi \models \varphi$ if and only if $\pi \models \psi$. These semantics preserve the usual operator equivalences in LTL, such as $\mathcal{F}_{[a,b]}\varphi \equiv \text{true}\,\mathcal{U}_{[a,b]}\varphi$, $\mathcal{G}_{[a,b]}\varphi \equiv \text{false}\,\mathcal{R}_{[a,b]}\varphi$, and $\varphi\mathcal{U}_{[a,b]}\psi \equiv \neg(\neg\varphi\mathcal{R}_{[a,b]}\neg\psi)$. Note that the neXt ($\mathcal{X}$) operator from LTL$_f$ is semantically equivalent $\mathcal{G}_{[1,1]}$.

The computation length of a formula φ, denoted cplen(φ), is defined in definition 5 (propagation delay) of [6], and can be thought of as the maximum length of a trace that φ can specify behavior over.

3.1 MLTL is Sub-Regular

A trace π with m time steps over n atomic propositions $\mathcal{AP} = \{p_0, ..., p_{n-1}\}$ can be represented as a binary string $w \in \{0, 1\}^*$ of length mn. Let $w[i]$ denote the i-th bit of w [1], and we may encode π by setting $w[n \cdot i + k] = 1$ if and only if $p_k \in \pi[i]$.

Example For $n = 2$ atomic propositions, the word $w = 110100$ specifies a trace over 3 time steps in which p_0 and p_1 are both true at time step 0, only p_1 is true at time step 1, and no atomic propositions are true at time step 2. Note that we will assume the

[1] We use 0-indexing in this paper.

string representation of computations for the remainder of this paper, and explicitly fix the value of n in all contexts.

Thus for a MLTL formula φ over n atomic propositions, we may define $\mathcal{L}(\varphi) = \{\pi \in \{0,1\}^* : |\pi| \equiv 0 (\bmod n), \pi \models \varphi\}$, and also define $L_{MLTL} = \{L \subseteq \{0,1\}^* : L = \mathcal{L}(\varphi) \text{ for some MLTL formula } \varphi\}$ as the class of languages recognized by MLTL formulas.

Theorem 1. *$L_{MLTL} \subsetneq REG$, where REG is the class of regular languages.*

Proof. We first prove that $L_{MLTL} \subseteq REG$. For any MLTL formula φ, [3] computes a regular expression reg(φ) that captures all traces π over comp_len(φ) time steps which satisfy φ. Note that this previous work assumes time steps are explicitly separated by commas, while we implicitly separate time steps by fixing n in all contexts. Then we have that $\mathcal{L}(\varphi) = \mathcal{L}\big(\text{reg}(\varphi)(S^n)^*\big)$, where S is shorthand for $(0|1)$, which is regular by definition.

Next, we prove that for any fixed integer $n \geqslant 1$, no MLTL formula φ over n propositional variables exists such that $\mathcal{L}(\varphi) = L$, where $L = \mathcal{L}\big(S^*1\big)$. Suppose for the sake of contradiction that for some fixed integer $n \geqslant 1$, $\mathcal{L}(\varphi) = L$. By the previous argument, we have that $\mathcal{L}(\varphi) = \mathcal{L}\big(\text{reg}(\varphi)(S^n)^*\big) = \mathcal{L}(\text{reg}(\varphi)) \cdot \mathcal{L}(S^*)$, where $\cdot$ denotes the usual concatenation operator between languages. Observe that $\mathcal{L}(\text{reg}(\varphi))$ is a finite language containing only strings of length $n \times$ comp_len(φ). If $\mathcal{L}(\text{reg}(\varphi)) = S^{n \times \text{comp\_len}(\varphi)}$, then $\mathcal{L}(\varphi) = S^* \neq L = \mathcal{L}\big(S^*1\big)$. Otherwise, there exists $w \notin \mathcal{L}(\text{reg}(\varphi))$ such that $|w| = n \times$ comp_len(φ), but then $w \cdot 1 \notin \mathcal{L}(\varphi)$, while $w \cdot 1 \in L = \mathcal{L}\big(S^*1\big)$. Thus we conclude that no such φ may exist, and so $L_{MLTL} \subsetneq REG$.

3.2 MLTL Inference is Well-Defined

We formally define the MLTL inference problem and show that it is well-defined. Fix some integer $n \leqslant 0$, 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 1. *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 2. *For any instance of the MLTL inference problem, (Π^+, Π^-, n), there exists MLTL formula φ over n atomic propositions to separate $\Pi = \Pi^+ \cup \Pi^-$.*

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 only of theoretical interest. In practice, we should strive to learn formulas of size logarithmic 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 Grammar Evolution

Grammatical Evolution (GE) lie at the intersection of genetic algorithms and formal grammars. As a unique form of genetic programming, GE guides the evolutionary process by translating genetic strings, encoded as chromosomes, into syntactically correct programs or solutions based on a predefined context-free grammar.

In GE, individual candidate solutions are represented by a collection of genes, which make up the genetic chromosome of the individual. Chromosomes are commonly represented as lists of integers, with each integer representing a single gene. While genotype refers to the genetic makeup of an individual, an individual's phenotype refers to the expressed characteristics or the solution derived from the genotype. In traditional GE, the genotype to phenotype mapping builds a derivation tree by using an individual's genes to select rules from the grammar for expanding nonterminal symbols. One important consideration in this mapping process is some way of limiting the maximum depth of a derivation tree, in order to avoid uncontrolled blowup of solution sizes.

A group of such individuals forms a population in the evolutionary context. Within this population, each individual is evaluated to determine its fitness, which is a measure of how well the phenotype solves the problem at hand. This evaluation critically guides the selection process, during which individuals are chosen for reproduction based on their fitness levels. The selection process, often inspired by natural selection, favors individuals with higher fitness while allowing some degree of randomness to maintain population diversity.

Following selection, crossover and mutation operators generate a new generation of individuals. Crossover exchanges segments of chromosomes between pairs of individuals, resulting in offspring with genetic traits inherited from both parents. Mutation introduces random alterations to an individual's genes, which helps maintain genetic diversity within the population. These processes are crucially explore new regions of the solution space prevents the population from converging prematurely.

Lastly, the new generation of individuals then replaces some or all of the existing population in a step known as replacement. This could be done through various strategies, such as removing the least fit individuals or adopting an elitist approach where the best-performing individuals are retained. Through these iterative cycles of selection, crossover, mutation, and replacement, the population evolves over successive generations, ideally moving towards increasingly effective solutions.

GE is commonly criticized for its lack of locality and high redundancy. In [12], Rothlauf and Oetzel demonstrated that over 90% of changes to a particular gene in the genotype do not affect the phenotype. In the remaining 10% of cases, the change to the phenotype is often drastic, and two individuals with nearly identical genotypes may have very different phenotypes. These issues impede the search process, as the crossover and mutation operators essentially simulate a random walk in the solution space.

4.1 GE Variants

Structured Grammatical Evolution (SGE) [8] is a variant of GE that attempts to address the issues of locality and redundancy. In SGE, genotypes are a list of list of integers, where each nonterminal symbol in the grammar is associated with a list of integers. SGE limits the maximum depth of a derivation tree by introducing indexed copies of each recursive nonterminal, which can be expensive depending on grammar structure.

The genotype to phenotype mapping in SGE builds a derivation tree by using an individual's genes to select rules from the grammar for expanding nonterminal symbols. This extra structure imposed by SGE ensures that modification of a single gene affects only the corresponding nonterminal symbol in the phenotype.

Dynamic Structured Grammatical Evolution (DSGE) [8] introduced by Lourenco et al. addresses the issue of grammar preprocessing in SGE. The genotype to phenotype mapping takes a maximum depth parameter and upon reaching maximum depth, the `generate_terminal_expansion` function maps a recursive nonterminal to a non-recursive nonterminal or a terminal.

In DSGE, genotypes encode only genes that are expressed in the phenotype (the parse tree) of the individual. During evolution, changes to the genotype gives rise to instances where new integers may need to be added to a gene array that "runs out" of genes. As a result of the dynamic nature of the genotype, the initialization procedure requires first building a derivation tree from the grammar and then encoding the tree into a genotype. In contrast, GE and SGE initialize individuals by mapping randomly generated genotypes to phenotypes.

5 DSGE$^+$ for MLTL

We propose DSGE$^+$ as a variant of DSGE that addresses the unnecessary complexity of the initialization procedure as well as the phenomenon of individuals "running out" of genes during evolution. We proceed to describe the details of DSGE$^+$ in the context of a specific MLTL grammar.

5.1 MLTL Grammar

A common challenge in all applications of GE is a good grammar representation of the solution space. Such a grammar should capture certain structural properties of the problem domain, and we use a grammar that is defined dynamically based on the example traces. Let (Π^+, Π^-, n), where $\Pi = \Pi^+ \cup \Pi^-$ be an instance of the MLTL inference problem, and we define the length of the longest trace as $m = \max_{\pi \in \Pi} |\pi|$. Then we only need to consider variables $p_0, ..., p_{n-1}$ and intervals of the form $[a, b]$ for $0 \leqslant a \leqslant b \leqslant m$. We define the MLTL grammar as follows:

$$\begin{aligned}
\langle \text{wff} \rangle ::= {} & \langle \text{prop\_cons} \rangle \\
& |\langle \text{prop\_var} \rangle \\
& |\neg\langle \text{wff} \rangle \\
& |(\langle \text{wff} \rangle \langle \text{binary\_prop\_conn} \rangle \langle \text{wff} \rangle) \\
& |\langle \text{unary\_temp\_conn} \rangle \langle \text{interval} \rangle \langle \text{wff} \rangle \\
& |(\langle \text{wff} \rangle \langle \text{binary\_temp\_conn} \rangle \langle \text{interval} \rangle \langle \text{wff} \rangle) \\
\langle \text{binary\_prop\_conn} \rangle ::= {} & \wedge \mid \vee \mid \rightarrow \\
\langle \text{unary\_temp\_conn} \rangle ::= {} & F \mid G
\end{aligned}$$

$$
\begin{aligned}
\langle \text{binary\_temp\_conn} \rangle &::= U \mid R \\
\langle \text{interval} \rangle &::= [a, b] \quad \text{for } 0 \leqslant a \leqslant b \leqslant m \\
\langle \text{prop\_cons} \rangle &::= \text{true} \mid \text{false} \\
\langle \text{prop\_var} \rangle &::= p_i \quad \text{for } 0 \leqslant i < n
\end{aligned}
$$

This grammar has 7 nonterminals symbols, denoted by $\langle \cdot \rangle$. Notably, $\langle \text{wff} \rangle$ captures the previously defined syntax of MLTL, and is the only recursive nonterminal. The overall structure of this grammar would be very similar to that of other popular temporal logics such as Linear Temporal Logic over finite traces, Metric Temporal Logic, Signal Temporal Logic, Computation Tree Logic, etc.

5.2 Genotype to Phenotype Mapping

We define the branching factor of a grammar as the maximum number of recursive nonterminal symbols in any right-hand-side term of a rule of the form $lhs \rightarrow rhs$. In the MLTL grammar, the branching factor is 2 because of the two rules for binary temporal connectives and binary propositional connectives.

For a fixed branching factor b and maximum recursion depth d, we may calculate that at most $\frac{b^d-1}{b-1}$ genes will be expressed from each nonterminal genome. For $b = 2$, we bound the number of genes expressed from each nonterminal genome by $2^d - 1$.

Then the genotype to phenotype mapping is described by algorithm 0. A direct consequence of this mapping is that the list of popped genes for each nonterminal keeps track of the number of genes that have been expressed. This approach enables us to benefit from the advantages of DSGE over GE, specifically in terms of locality and redundancy, as well as DSGE's advantages over SGE in bypassing grammar preprocessing. Additionally, it prevents the issue of individuals "running out" of genes during the evolution process and avoids the complexity of the initialization procedure. Specifically, an individual x may be initialized with a random integer array of length $2^d - 1$ for each nonterminal in the grammar.

Algorithm 1 Initialize Parse Tree from Genotype

```
1: procedure InitializeParseTree(genotype, grammar)
2:     $tree$ ← CreateEmptyTree()
3:     $root$ ← CreateNode("<wff>")
4:     $tree$.AddRoot($root$)
5:     $stack$ ← new Stack()
6:     $stack$.push(($root$, 0))                                  ▻ Stack of (node, depth)
7:     $poppedGenes$ ← $\varnothing$
8:     while ¬$stack$.isEmpty() do
9:         ($node$, $depth$) ← $stack$.pop()
10:        if $depth \geqslant$ MaxTreeDepth and IsNonTerminal($node$) then
11:            ExpandTerminal($node$, $genotype$, $poppedGenes$)
12:        else if IsNonTerminal($node$) then
13:            ExpandNonTerminal($node$, $genotype$, $grammar$, $stack$, $depth$, $poppedGenes$)
14:    UpdateGenotype($genotype$, $poppedGenes$)
15:    return $tree$
```

5.3 Genetic Algorithm

We now proceed to describe the genetic algorithm used to evolve the population of individuals. Let $\mathit{\Pi} = \mathit{\Pi}^+ \cup \mathit{\Pi}^-$ be the set of example traces, $m = \max_{\pi \in \mathit{\Pi}} |\pi|$ be the length of the longest trace, and fix $n \leqslant 1$ to be the number of atomic propositions.

Fitness of an individual with phenotype φ is defined as:

$$\text{fitness}(\varphi) = w_1 \cdot \text{acc}(\varphi, \mathit{\Pi}) + w_2 \cdot \frac{|\text{comp\_len}(\varphi) - m|}{m} + w_3 \cdot \frac{1}{|\varphi|} + w_4 \cdot \frac{1}{\text{depth}(\varphi)}$$

where $\text{acc}(\varphi, \mathit{\Pi})$ is the accuracy of φ over $\mathit{\Pi}$, $\text{comp\_len}(\varphi)$ is the computation length of φ, $\text{depth}(\varphi)$ is the depth of the parse tree of φ, and w_1, w_2, w_3, w_4 are hyperparameters that we may tune to emphasize certain properties of the formula.

Achieving high accuracy is the primary goal of the MLTL inference problem, and so we set $w_1 = 0.8$. The second term of the fitness function measures the absolute deviation between the computation length of φ and the length of the longest trace in $\mathit{\Pi}$. This term encourages the formula to encode a specification appropriate to observed traces in $\mathit{\Pi}$, and we set $w_2 = 0.1$. The third term of the fitness function measures the inverse of the length of the formula, defined simply as the number of characters in the formula. This encourages the formula to be concise and therefore interpretable, and we set $w_3 = 0.04$. The fourth term of the fitness function measures the inverse of the depth of the parse tree of φ. Again, this encourages the formula to be concise and interpretable, and we set $w_4 = 0.01$. Each of the four terms of the fitness function are normalized to be in the range $[0, 1]$, and so the overall fitness is also in the range $[0, 1]$.

Selection is currently done at each iteration by simply selecting the top 20% of the population based on fitness.

However, I wish to explore the effect of the a genetic difference operator that measures the genetic difference between two individuals x and y. Suppose that A is some nonterminal in the grammar, and x_A and y_A are the expressed genomes, represented as a list of integers, of x and y for nonterminal A. Then define the genetic difference [2] between x_A and y_A as:

$$\text{diff}(x_A, y_A) = \sum_{j=0}^{\max(|x_A|,|x_B|)-1} 1_{\{x_A[j] \neq y_A[j]\}} \cdot r^j$$

and the genetic difference between x and y as:

$$\text{diff}(x, y) = \sum_{A \in \text{nonterminals}} w_A \cdot \text{diff}(x_A, y_A)$$

where $1_{\text{condition}}$ is the indicator function that evaluates to 1 if the condition is true, and 0 otherwise, and w_A is a tunable hyperparameter that we use to emphasize the importance of A in the grammar.

[2] It would be cool to prove that this is actually a well-defined distance function satisfying the triangle inequality. At which point the space of solutions forms a metric space and we can look to formalize some aspects of GE with real analysis and topology.

To ensure normalization of the genetic difference in the interval $[0, 1]$, we can find the condition on r such that diff$(x, y) \leqslant 1$ for all x, y:

$$\begin{aligned}
\text{diff}(x, y) &= \sum_{A \in \text{nonterminals}} w_A \cdot \sum_{j=0}^{\max(|x_A|,|x_B|)-1} 1_{\{x_A[j] \neq y_A[j]\}} \cdot r^j \\
&\leqslant \sum_{A \in \text{nonterminals}} w_A \cdot \sum_{j=0}^{\infty} r^j \\
&= \sum_{A \in \text{nonterminals}} w_A \cdot \frac{r}{r-1} \\
&\leqslant 1, \text{ implying that } r \geqslant \frac{1}{1 + \sum_{A \in \text{nonterminals}} w_A}.
\end{aligned}$$

The motivation for the exponential factor r^j is to emphasize the importance of the first few genes in the genome, and to de-emphasize the importance of the later genes. For instance, in the context of our MLTL grammar, the first few genes of the $\langle$wff$\rangle$ genome are more important than the later genes because the first few genes determine overall structure of the formula. Having tunable weights w_A allows us to emphasize certain importance of certain nonterminals determining the structure of the solution. We set $w_A = 1$ for all $A \in$ nonterminals for now, but we may wish to emphasize the importance of $\langle$wff$\rangle$ in the future.

This genetic difference operator can be used to perform k-clustering on the population, where k is the number of groups we wish to cluster the population into. We can then perform selection by selecting the top individuals from each cluster, thereby maintaining population diversity without sacrificing the quality of the population.

Crossover is done with the standard one-point crossover operation. Crossover between two genomes x_A and y_A is done by selecting a random index i and swapping the suffixes of x_A and y_A starting at index i. i is chosen so that it is always at index before the last expressed gene of x_A and y_A, and so that the resulting genome always guarantees an individual inherits phenotype attributes from both parents.

Mutation is done with the standard point mutation operation. Mutation of a genome x_A simply mutates each gene of x_A with probability p. At each iteration, we apply an exponential decay factor of 0.95 to p to encourage early exploration of the solution space and later exploitation of the solution space.

6 Experimental Evaluation

Due to the lack of existing MLTL inference benchmarks, we propose a set of benchmarks for future research. We report and analyze the performance of DSGE$^+$ on these benchmarks. For relatively easy benchmarks, we use a population size of 100 and run the genetic algorithm for 30 generations, while for more difficult benchmarks, we use a population size of 300 to 1000 and run the genetic algorithm for 50 generations.

One achievement to highlight is the improvement of the efficiency of computing an individual's accuracy score over a set of traces. This single computation is the bottleneck of the entire algorithm, and significant effort was spent on optimizing this computation. Originally, the WEST tool [3] provides a Python implementation of the MLTL semantics, that on inputs MLTL formula φ and trace π, returns whether $\pi \models \varphi$. I reimplemented this functionality in C++ and achieved a significant (maybe I should quantify this) speedup. Further optimizations on batched computation over a set of traces, rather than computing the accuracy score of an individual over each trace individually, resulted in a 20 times speedup. Lastly, parallelizing the computation for each population member over multiple threads resulted in another 3 times speedup. In total, these optimizations resulted in a 60 times speedup from initial implementation to final implementation.

6.1 Benchmarks

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 [3] 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 to populate Π^+ and Π^-.

In total, we created 9 benchmarks, with 4 benchmarks that target the basic temporal operators, and 5 benchmarks that use MLTL formulae found in the literature. For each benchmark, we produced a total of 500 positive traces and 500 negative traces. 100 positive traces and 100 negative traces were used for training, and the remaining 400 traces were held out for testing.

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. [10] 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 for the purposes of a genetic algorithm, 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. [4]. 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 the purposes of a genetic algorithm.

6.2 Results

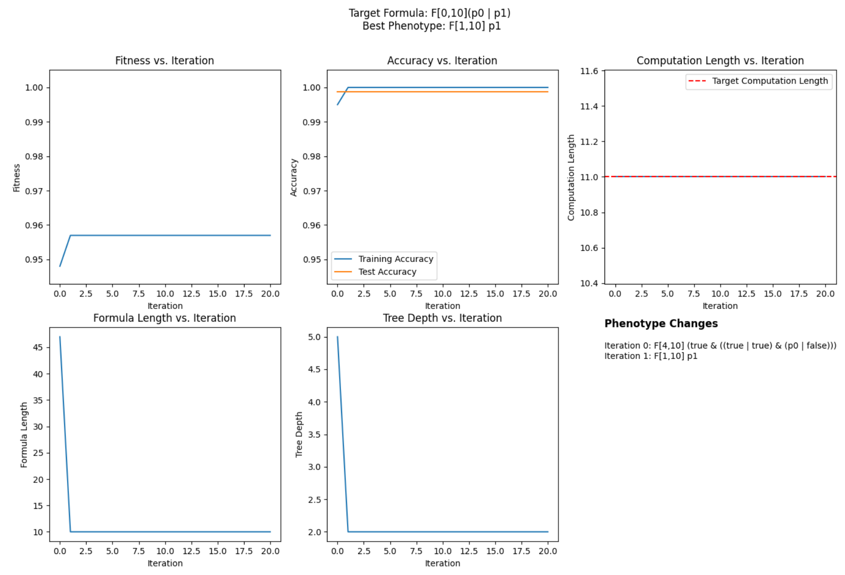
We now report and analyze the performance of DSGE$^+$ on each of the benchmarks. For each benchmark, we track the fitness score, accuracy, computation length, formula length, parse tree depth, and phenotype of the best individual in the population at each generation.

Across all benchmarks, we observe that the fitness score of the best individual in the population increases monotonically over time, by design of the genetic algorithm. On all of the basic temporal operator benchmarks and formula 3 of the DragonEye UAS benchmark, a test accuracy of 100% is achieved. It's interesting to note that a majority of the time, test accuracy is strictly lower than training accuracy, which is likely due to overfitting. Additionally, if the training set does not contain an informative trace that the test set contains, then no algorithm can guarantee 100% accuracy on the test set. For intance in the Until benchmark, the language of the best individual is $\mathcal{L}(\mathcal{F}_{[1,10]}p_1)$, which is a strict subset of the language of the target formula $\mathcal{L}(\mathcal{F}_{[0,10]}(p_0 \vee p_1))$. This is because the training set does not contain an informative trace in the difference of these languages. Such a trace would require p_1 to hold at some time step, and then p_0 to never hold for the next 10 time steps, which is a very specific trace that is unlikely to be sampled randomly.

For the more difficult benchmarks, formulae 1 and 2 of the DragonEye UAS benchmark, and formulae 1 and 2 of the NASA's NextGen Air Traffic Control benchmark, we observe that formula length and parse tree depth first increase, then decrease. This trend is due to the high weight given to the formula accuracy, and thus the genetic algorithm prioritizes improving the accuracy first, and then improving the formula length and parse tree depth after the accuracy can not be easily improved.

Another notable trend is that the computation length of the best individual often hovers around m, the target computation length. This behavior is expected and desirable, as the genetic algorithm is incentivized to produce a formula that is appropriate for the observed traces in Π.

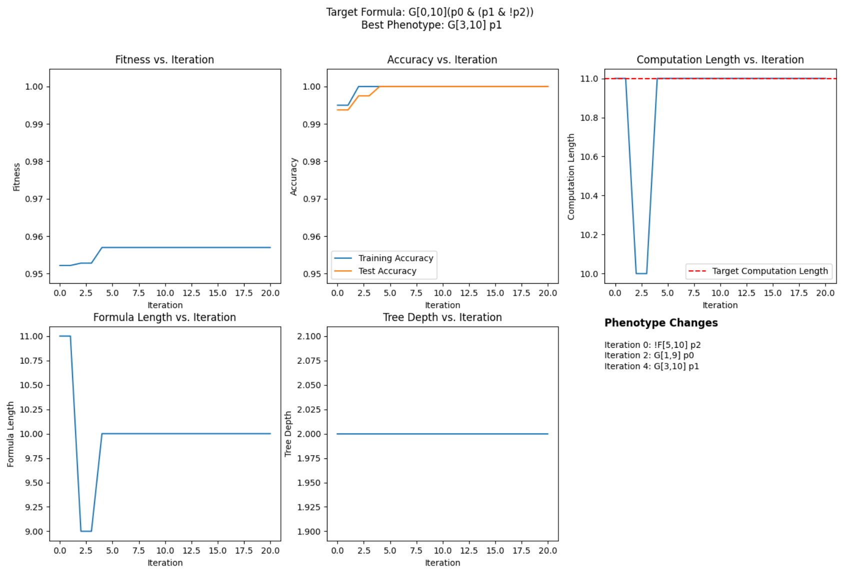
Lastly, we observe that the genetic algorithm is able to learn formulae that are more concise than the target formulae. For instance, the learned formula in formula 2 of the DragonEye UAS benchmark, and formulae 1 and 2 of the NASA's NextGen Air Traffic Control benchmark, are all more concise than the target formulae. However, the accuracy of these learned formulae are not 100%. This suggests that the learned formulae may be a good approximation of the target formulae, and could be a good starting point for rewriting the target formulae into a more concise formula.



Phenotype Changes

Iteration 0: F[4,10] (true & ((true | true) & (p0 | false)))
Iteration 1: F[1,10] p1

Fig. 1: Training plots of $DSGE^+$ on the future time operator benchmark.



Phenotype Changes

Iteration 0: !F[5,10] p2
Iteration 2: G[1,9] p0
Iteration 4: G[3,10] p1

Fig. 2: Training plots of $DSGE^+$ on the globally operator benchmark.

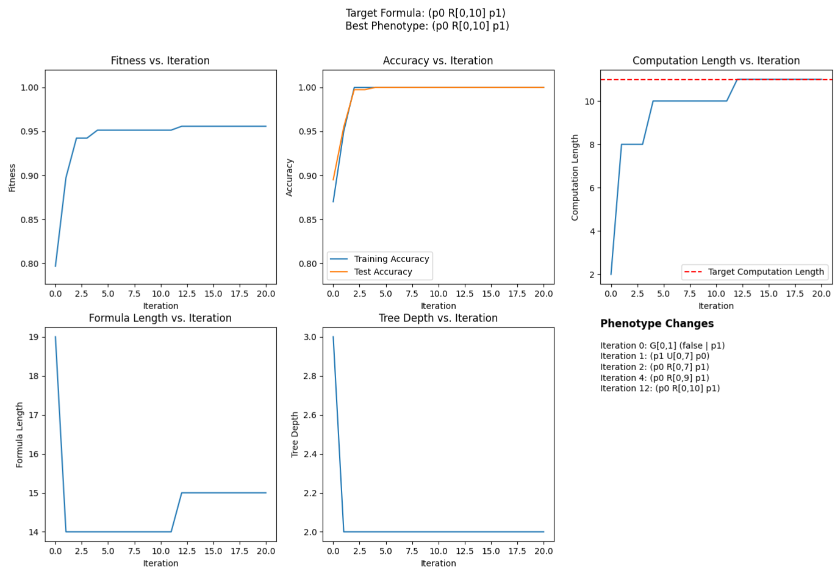


Fig. 3: Training plots of $DSGE^+$ on the release operator benchmark.

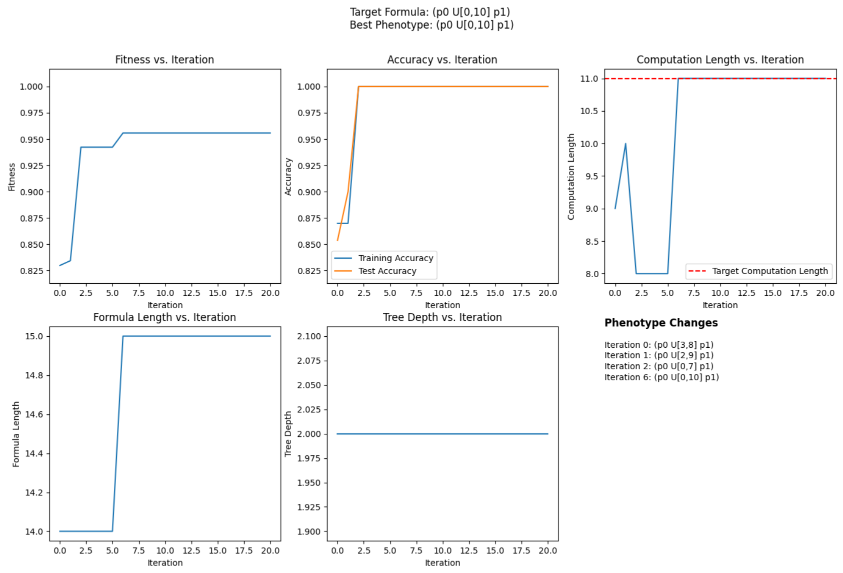
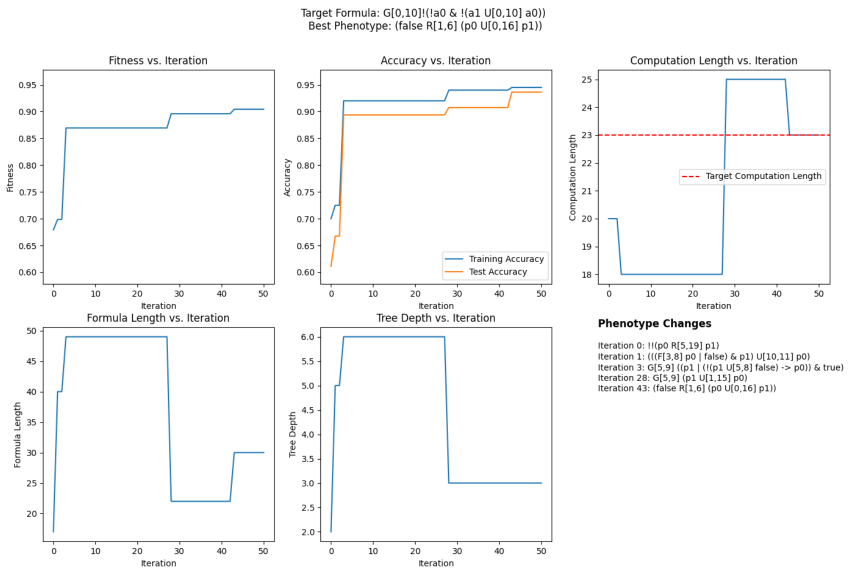


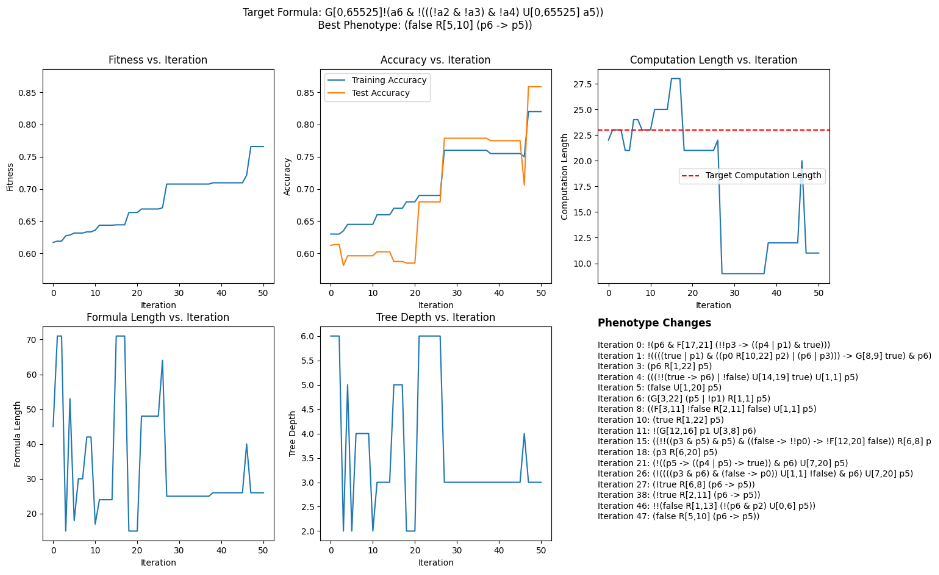
Fig. 4: Training plots of $DSGE^+$ on the until operator benchmark.



Phenotype Changes

```
Iteration 0: !!(p0 R[5,19] p1)
Iteration 1: (((F[3,8] p0 | false) & p1) U[10,11] p0)
Iteration 3: G[5,9] ((p1 | (!(p1 U[5,8] false) -> p0)) & true)
Iteration 28: G[5,9] (p1 U[1,15] p0)
Iteration 43: (false R[1,6] (p0 U[0,16] p1))
```

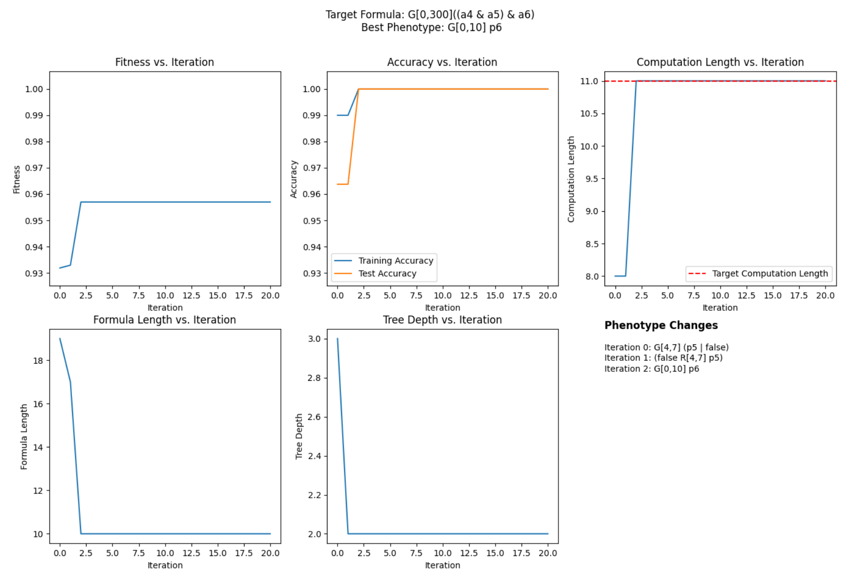
Fig. 5: Training plots of $DSGE^{+}$ formula 1 of the DragonEye UAS benchmark.



Phenotype Changes

```
Iteration 0: !(p6 & F[17,21] (!!p3 -> ((p4 | p1) & true)))
Iteration 1: !((((true | p1) & ((p0 R[10,22] p2) | (p6 | p3))) -> G[8,9] true) & p6)
Iteration 3: (p6 R[1,22] p5)
Iteration 4: (((!!(true -> p6) | !false) U[14,19] true) U[1,1] p5)
Iteration 5: (false U[1,20] p5)
Iteration 6: (G[3,22] (p5 | !p1) R[1,1] p5)
Iteration 8: ((F[3,11] !false R[2,11] false) U[1,1] p5)
Iteration 10: (true R[1,22] p5)
Iteration 11: !(G[12,16] p1 U[3,8] p6)
Iteration 15: ((!!((p3 & p5) & p5) & ((false -> !!p0) -> !F[12,20] false)) R[6,8] p
Iteration 18: (p3 R[6,20] p5)
Iteration 21: (!((p5 -> ((p4 | p5) -> true)) & p6) U[7,20] p5)
Iteration 26: (!((((p3 & p6) & (false -> p0)) U[1,1] !false) & p6) U[7,20] p5)
Iteration 27: (!true R[6,8] (p6 -> p5))
Iteration 38: (!true R[2,11] (p6 -> p5))
Iteration 46: !!(false R[1,13] (!(p6 & p2) U[0,6] p5))
Iteration 47: (false R[5,10] (p6 -> p5))
```

Fig. 6: Training plots of $DSGE^{+}$ formula 2 of the DragonEye UAS benchmark.

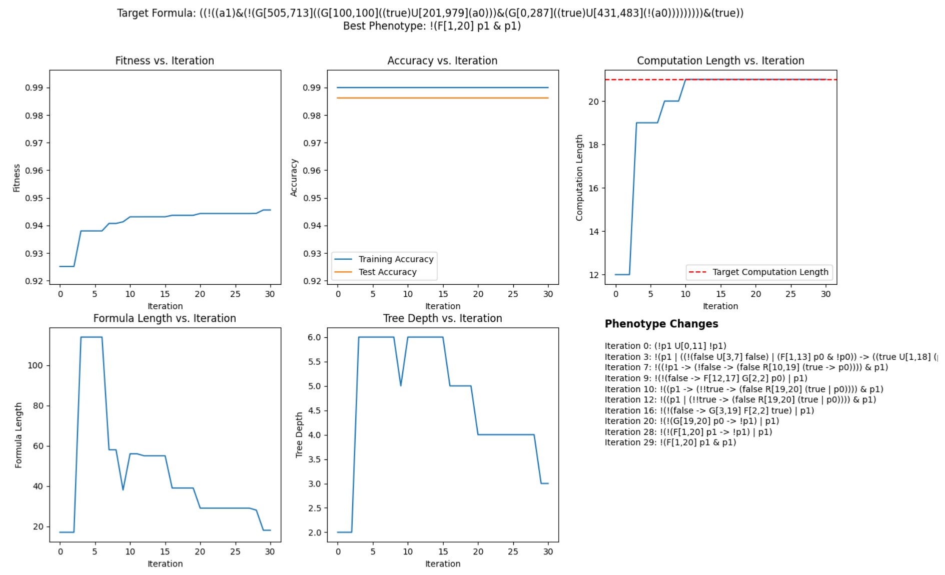


Target Formula: G[0,300]((a4 & a5) & a6)
Best Phenotype: G[0,10] p6

Phenotype Changes

```
Iteration 0: G[4,7] (p5 | false)
Iteration 1: (false R[4,7] p5)
Iteration 2: G[0,10] p6
```

Fig. 7: Training plots of $DSGE^+$ formula 3 of the DragonEye UAS benchmark.

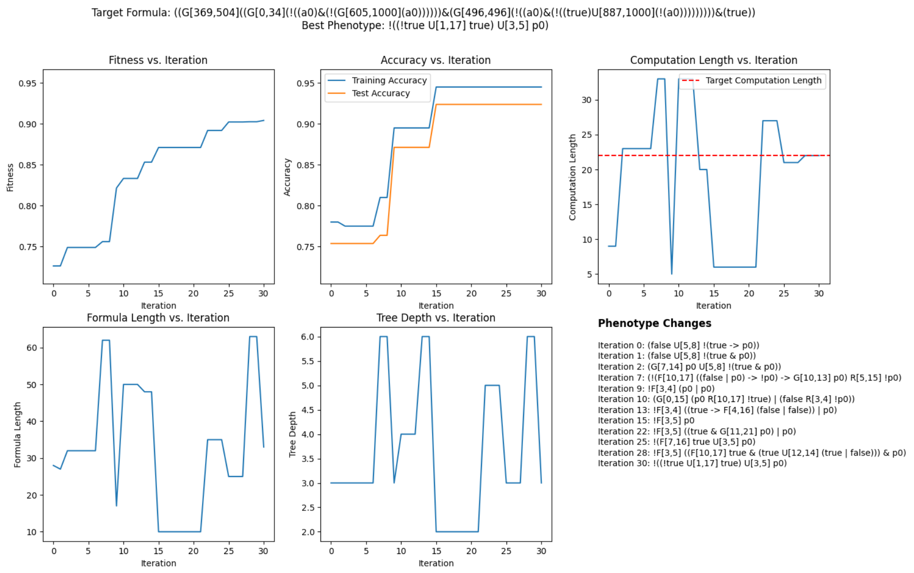


Target Formula: ((!((a1)&(!(G[505,713]((G[100,100]((true)U[201,979](a0)))&(G[0,287]((true)U[431,483](!(a0)))))))))&(true))
Best Phenotype: !(F[1,20] p1 & p1)

Phenotype Changes

```
Iteration 0: (!p1 U[0,11] !p1)
Iteration 3: !(p1 | ((!(false U[3,7] false) | (F[1,13] p0 & !p0)) -> ((true U[1,18] (
Iteration 7: !((!p1 -> (!false -> (false R[10,19] (true -> p0)))) & p1)
Iteration 9: !(!(false -> F[12,17] G[2,2] p0) | p1)
Iteration 10: !((p1 -> (!!true -> (false R[19,20] (true | p0)))) & p1)
Iteration 12: !((p1 | (!!true -> (false R[19,20] (true | p0)))) & p1)
Iteration 16: !(!(false -> G[3,19] F[2,2] true) | p1)
Iteration 20: !(!(G[19,20] p0 -> !p1) | p1)
Iteration 28: !(!(F[1,20] p1 -> !p1) | p1)
Iteration 29: !(F[1,20] p1 & p1)
```

Fig. 8: Training plots of $DSGE^+$ formula 1 of the NASA's NextGen Air Traffic Control benchmark.



```
Target Formula: ((G[369,504]((G[0,34](!((a0)&(!(G[605,1000](a0))))))&(G[496,496](!((a0)&(!((true)U[887,1000](!(a0)))))))))&(true))
Best Phenotype: !((!true U[1,17] true) U[3,5] p0)
```

Phenotype Changes

```
Iteration 0: (false U[5,8] !(true -> p0))
Iteration 1: (false U[5,8] !(true & p0))
Iteration 2: (G[7,14] p0 U[5,8] !(true & p0))
Iteration 7: (!(F[10,17] ((false | p0) -> !p0) -> G[10,13] p0) R[5,15] !p0)
Iteration 9: !F[3,4] (p0 | p0)
Iteration 10: (G[0,15] (p0 R[10,17] !true) | (false R[3,4] !p0))
Iteration 13: !F[3,4] ((true -> F[4,16] (false | false)) | p0)
Iteration 15: !F[3,5] p0
Iteration 22: !F[3,5] ((true & G[11,21] p0) | p0)
Iteration 25: !(F[7,16] true U[3,5] p0)
Iteration 28: !F[3,5] ((F[10,17] true & (true U[12,14] (true | false))) & p0)
Iteration 30: !((!true U[1,17] true) U[3,5] p0)
```

Fig. 9: Training plots of $DSGE^{+}$ formula 2 of the NASA's NextGen Air Traffic Control benchmark.

7 Conclusion and Future Work

We have shown that MLTL is a strict subset of regular languages, and that the MLTL inference problem is well-defined. We proposed a novel GE algorithm, DSGE^{+}, that combines the advantages of DSGE and SGE while addressing the initialization complexity and insufficient genotype length issues of DSGE. We also proposed a set of MLTL benchmarks for future research, and reported and analyzed the performance of DSGE^{+} on these benchmarks.

Future work includes proving theoretical learnability of MLTL from example traces. Relatively simply formulae, such as $p\mathcal{U}_[0, 10]q$, characterizes an exponential number of traces. Thus we hope to prove that some $(1 + \epsilon)$ approximation of a set of traces Π can be achieved by a formula of length logarithmic in $|\Pi|$. Possible works we can explore include the general Probably Approximately Correct (PAC) learning framework introduced by Valiant [14] in 1984, and the recent work by Perez et al. [11] on a PAC algorithm for learning LTL.

As discussed in our experimental evaluation section, the quality of the example traces Π is crucial to GE. Thus we wish to explore the effects of different sampling strategies for generating Π that gives better edge cases to distinguish between formulae such as $\mathcal{G}_[a, b](p_0 \wedge p_1)$ and $\mathcal{G}_[a, b]p_0$. One possible approach is to sample traces where each bit is 1 with probability p and 0 with probability $1 - p$, and ranging p uniformly from 0 to 1. Another approach is to explore notions of formula branch coverage, posed by work stemming from the FPROGG tool (backslash cite Alek).

As seen in our experimental evaluation, many formulae were learned throughout DSGE^{+} that could be simplified to a more concise formula. Thus we wish to explore the effects of applying a formula simplification operator to the population at each iteration. For instance, Johannsen et al. [5] integrates MLTL formula rewriting rules into the recently introduced C2PO extension for R2U2.

Additionally we wish to explore the effects of the genetic difference operator on the performance of DSGE$^+$. Specifically, we conjecture that applying a k-clustering selection strategy will improve the performance of DSGE$^+$ by maintaining population diversity while still selecting for high fitness individuals.

Lastly, we wish to perform a more thorough experimental evaluation of DSGE$^+$. One approach is to compare the performance of DSGE$^+$ against GE, SGE, and DSGE on the same set of benchmarks. A more interesting approach would be to conduct a case study on a real-world application of MLTL inference, such as specification mining for formal verification, debugging systems, behavior classification, and formula reparation. One of the original motivations for applying GE to MLTL inference was to generate a population of interpretable MLTL formulae that could help a user repair an incorrectly specified MLTL formula.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2), 87–106 (Nov 1987). https://doi.org/10.1016/0890-5401(87)90052-6, https://www.sciencedirect.com/science/article/pii/0890540187900526
2. 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
3. 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. p. 279–301. Lecture Notes in Computer Science, Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-47705-8_15
4. 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
5. Johannsen, C., Jones, P., Kempa, B., Rozier, K.Y., Zhang, P.: R2u2 version 3.0: Re-imagining a toolchain for specification, resource estimation, and optimized observer generation for runtime verification in hardware and software. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. p. 483–497. Lecture Notes in Computer Science, Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-37709-9_23
6. 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, http://research.temporallogic.org/papers/KZJZR20.pdf
7. Li, J., Vardi, M.Y., Rozier, K.Y.: Satisfiability checking for mission-time ltl (mltl). Information and Computation **289**, 104923 (Nov 2022). https://doi.org/10.1016/j.ic.2022.104923, https://www.sciencedirect.com/science/article/pii/S0890540122000761
8. Lourenço, N., Pereira, F.B., Costa, E.: SGE: A Structured Representation for Grammatical Evolution. In: Bonnevay, S., Legrand, P., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) Artificial Evolution. pp. 136–148. Lecture Notes in Computer Science, Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-31471-6_11
9. 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
10. 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
11. Perez, M., Somenzi, F., Trivedi, A.: A pac learning algorithm for ltl and omega-regular objectives in mdps (arXiv:2310.12248) (Oct 2023). https://doi.org/10.48550/arXiv.2310.12248, http://arxiv.org/abs/2310.12248, arXiv:2310.12248 [cs]
12. Rothlauf, F., Oetzel, M.: On the locality of grammatical evolution. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) Genetic Programming. p. 320–330. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11729976_29

13. 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
14. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (nov 1984). https://doi.org/10.1145/1968.1972, https://doi.org/10.1145/1968.1972