2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 979-8-3503-2996-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASE56229.2023.00202

`PURLTL`: Mining LTL Specification from Imperfect Traces in Testing

1st Bo Peng
Sun Yat-Sen University
Guangzhou, China
pengb8@mail2.sysu.edu.cn

2nd Pingjia Liang
Sun Yat-Sen University
Guangzhou, China
liangpj3@mail2.sysu.edu.cn

3rd Tingchen Han
Sun Yat-Sen University
Guangzhou, China
hantch@mail2.sysu.edu.cn

4th Weilin Luo
Sun Yat-Sen University
Guangzhou, China
luowlin3@mail2.sysu.edu.cn

5th Jianfeng Du†
Guangdong University of Foreign Studies
Pazhou Lab
Guangzhou, China
jfdu@gdufs.edu.cn

6th Hai Wan†
Sun Yat-Sen University
Guangzhou, China
wanhai@mail.sysu.edu.cn

7th Rongzhen Ye
Sun Yat-Sen University
Guangzhou, China
yerzh@mail2.sysu.edu.cn

8th Yuhang Zheng
Sun Yat-Sen University
Guangzhou, China
zhengyh29@mail2.sysu.edu.cn

***Abstract*—*Formal specifications* are widely used in software testing approaches, while writing such specifications is a time-consuming job. Recently, a number of methods have been proposed to mine specifications from *execution traces*, typically in the form of *linear temporal logic (LTL)*. However, existing works have the following disadvantages: (1) ignoring the negative impact of imperfect traces, which come from partial profiling, missing context information, or buggy programs; (2) relying on templates, resulting in limited expressiveness; (3) requesting negative traces, which are usually unavailable in practice.**

In this paper, we propose `PURLTL`, which is able to mine arbitrary LTL specifications from imperfect traces. To alleviate the search space explosion and the wrong search bias, we propose a neural-based method to search LTL formulae, which, intuitively, simulates *LTL path checking* through differentiable parameter operations. To solve the problem of lacking negative traces, we transform the problem into learning from positive and unlabeled samples, by means of data augmentation and applying *positive and unlabeled learning* to the training process. Experiments show that our approach surpasses the previous start-of-the-art (SOTA) approach by a large margin. Besides, the results suggest that our approach is not only robust with imperfect traces, but also does not rely on formula templates.

***Index Terms*—specification mining, machine learning**

I. Introduction

Formal specifications are widely used in software testing approaches [1, 17]. But writing such specifications is a time-consuming job for software developers [9]. Therefore, mining specifications automatically from *execution traces* is an important task in software testing.

A number of approaches which intend to help developers draft formal specifications have been proposed in the past few years [2, 4, 8, 10, 11, 12, 13, 14, 15]. One kind of them mines *finite state machines (FSMs)* as specifications, however, which weakens the interpretability of the specifications [3]. Another kind of them mines *linear temporal logic (LTL)* formulae as the specifications since LTL formulae have much better interpretability [3], which is readability and unambiguousness.

†Corresponding authors.

Mining LTL specifications is a challenging task. First, the search space of LTL formulae is exponential [6]. Second, there are imperfect traces in industrial scenarios. Imperfect traces are the execution traces with some mutations, *e.g.*, missing events. These traces may come from partial profiling, missing context information, or buggy programs [18]. Imperfect traces lead to wrong search bias. Third, only execution (positive) traces are neither representative nor sufficiently diverse.

Previous works of mining LTL specifications have some disadvantages. (1) Some of them rely on a noise-free assumption, which barely holds in practice [13]. (2) Others rely on formulae templates, resulting in limited expressiveness [13, 18]. Recently, a neural-based approach [16] and a MaxSAT-based approach [7] cannot suffer from these disadvantages, but (3) training their models requires negative samples, which are hard to acquire in software testing.

In this paper, we proposed `PURLTL` to mine arbitrary LTL specification from imperfect execution traces. An LTL specification can be an *LTL path checking* function that examines a trace whether it satisfies the given LTL formula or not. Such functions are typically discrete and cannot be optimized by gradient-based approaches. We design a neural network which simulates a given path checking function under certain parameters. Then we transform the discrete path checking into its differentiable simulation. After training our model to classify the sampled traces, we can interpret the LTL formula from model parameters. Besides, thanks to the approximate inference of neural networks, our method is noise-tolerant.

The lack of negative samples leads to a severe label bias, which can result in overfitting the training set. To this end, we transform our problem into a *positive and unlabeled (PU)* learning problem, where the execution traces are positive samples and randomly generated traces are unlabeled samples.

Our key contributions involve the following aspects:

- To overcome the exponential search space and imperfect traces, we propose a neural-based approach to mine LTL

specifications from event logs. Our method does not rely on formula templates and is robust with noisy data.
- To overcome the lack of negative samples, we transform our problem into PU learning. With carefully designed loss functions, our model accepts positive and unlabeled samples and avoids the requirement of negative samples, which is hard to acquire in software testing.

We evaluate our approach with synthesis datasets according to 8 templates, which were widely used in previous works [10, 11, 12]. Experiments show that our approach surpasses the previous start-of-the-art (SOTA) approach by a large margin.

II. Our Approach

Due to the limited space, we recommend reading Luo et al.(2022)[16] for perliminary knowledge of LTL semantics.

Given a set of positive traces, we first sample a set of unlabeled traces. Then both sets of traces are fed into our model. The model simulates LTL path checking and outputs classification results for each trace. Then we update the model parameters according to the VPU[5] loss. This training process iterates several epochs until the loss converges. Finally, after the training process is completed, we interpret an LTL formula representing our mined specification from model parameters.

Since our approach mines LTL specifications on the PU setting with an RNN model, we name our approach `PURLTL`.

A. Model Structure and Parameter Definitions

In this subsection, we first show our approach to parameterizing an LTL formula. Then we give the expression that uses the parameters to calculate whether a trace satisfies the formula. In order to apply this to the problem of learning LTL formulae, we prove that under some conditions there exists a bijective function that maps parameters to LTL formulae. This shows that it is theoretically possible to extract LTL formulae from the parameters. Let ϕ be an LTL formula, its syntax tree $T(\phi)$ is defined as follows:

- if $\phi = p$, $T(\phi)$ is a tree with single node v_p
- if $\phi = \neg\phi_i$, $T(\phi)$'s root is $v_\neg$ and its left subtree is $T(\phi_i)$
- if $\phi = \mathsf{X}\phi_i$, $T(\phi)$'s root is v_X and its left subtree is $T(\phi_i)$
- if $\phi = \phi_i \wedge \phi_j$, the root node of $T(\phi)$ is $v_\wedge$ and its left subtree is $T(\phi_i)$ and its right subtree is $T(\phi_j)$
- if $\phi = \phi_i \mathsf{U} \phi_j$, the root node of $T(\phi)$ is v_U and its left subtree is $T(\phi_i)$ and its right subtree is $T(\phi_j)$

Let $\mathbb{P} = \{p_1, p_2, ..., p_{|\mathbb{P}|}\}$ be the set of atomic proposition, $\theta = \{\theta_\mathrm{r}, \theta_\mathrm{none}, \theta_\mathrm{a}, \theta_\neg, \theta_\wedge, \theta_\mathsf{X}, \theta_\mathsf{U}\}$ an LTL$_f$ encoding of tree size L where $\theta_\mathrm{r} \in \{0,1\}^{L\times L}$, $\theta_\mathrm{a} \in \{0,1\}^{L\times|P|}$, $\theta_\mathrm{none}, \theta_\neg, \theta_\wedge, \theta_\mathsf{X}, \theta_\mathsf{U} \in \{0,1\}^L$.

Intuitively these parameters describe the structure of an LTL syntax tree. The operator or the atomic proposition that the node v_i represents is related to $\theta_\mathrm{a}, \theta_\neg, \theta_\wedge, \theta_\mathsf{X}, \theta_\mathsf{U}$. For example, if $(\theta_\neg)_i = 1$ then $v_i = v_\neg$. If the node v_i is not used in the syntax tree, then $(\theta_\mathrm{none})_i$ should be set to 1. The right son of v_i is related to $(\theta_\mathrm{r})_i$. If the right son of v_i is v_j then $(\theta_\mathrm{r})_{i,j}$ should be set to 1. We assume the left son of v_i is v_{i+1} unless v_{i+1} is a right son of a node.

Now formally define of the encoding of ϕ. Let the preorder traversal of $T(\phi)$ be the node sequence $\langle v_1, v_2, \ldots, v_L\rangle$ Then the encoding of ϕ of size $L' >= L$, θ_ϕ is defined as follows:

- $(\theta_\mathrm{r})_{i,j} = 1$ *iff.* v_j is v_i's right son, otherwise $(\theta_\mathrm{r})_{i,j} = 0$.
- $(\theta_\mathrm{a})_{i,j} = 1$ *iff.* $v_i = v_{p_j}$, otherwise $(\theta_\mathrm{a})_{i,j} = 0$.
- $(\theta_\neg)_i = 1$ *iff.* $v_i = v_\neg$, otherwise $(\theta_\neg)_i = 0$.
- $(\theta_\wedge)_i = 1$ *iff.* $v_i = v_\wedge$, otherwise $(\theta_\wedge)_i = 0$.
- $(\theta_\mathsf{X})_i = 1$ *iff.* $v_i = v_\mathsf{X}$, otherwise $(\theta_\mathsf{X})_i = 0$.
- $(\theta_\mathsf{U})_i = 1$ *iff.* $v_i = v_\mathsf{U}$, otherwise $(\theta_\mathsf{U})_i = 0$.
- $(\theta_\mathrm{none})_i = 1$ *iff.* $i > L$, otherwise $(\theta_\mathrm{none})_i = 0$.

In order to use the parameters to calculate satisfaction relation, we give a straightforward approach to transforming a trace. Given a trace $\pi = \langle s_0, s_1, \ldots, s_{n-1}\rangle$ where $s_i \in \mathbb{P}$, the encoding of s_i is $\mathbf{h}_{s_i} = \mathrm{OneHot}(s_i)$ where $\mathrm{OneHot}(s_i)_j = \mathbb{I}(s_i = p_j)$, where $\mathbb{I}$ is the indicator function. The encoding of π is $C(\pi) = \langle \mathbf{h}_{s_0}, \mathbf{h}_{s_1}, \ldots, \mathbf{h}_{s_{n-1}}\rangle$.

Let θ be an encoding of LTL formula ϕ with tree size L, $\pi = \langle s_0, s_1, \ldots, s_{n-1}\rangle$ be a trace of events and $C(\pi)$ be its encoding, then the satisfaction vector $\mathbf{x}_{s_i} \in \{0,1\}^L$ represents the satisfaction of the subtrace starting from s_i, *i.e.* $\langle s_i, s_{i+1}, \ldots, s_{n-1}\rangle$ according to each subtree of ϕ. $(\mathbf{x}_{s_i})_j = 1$ indicates that the subtrace satisfies the j-th subformula. $(\mathbf{x}_{s_i})_j = 0$ indicates that the subtrace does not satisfy the j-th subformula. If $i = n-1$, *i.e.* s_i is the last event in the trace, $\mathbf{x}_{s_i}$ could be calculated with Equation (1). Otherwise it could be calculated with Equation (2).

$$\begin{aligned}(\mathbf{x}_{s_{n-1}})_j = \sigma((l_{s_{n-1}})_j &+ (\theta_\neg)_j\sigma(1-(\mathbf{x}_{s_{n-1}})_{j+1})\\ &+(\theta_\wedge)_j\sigma((\mathbf{x}_{s_{n-1}})_{j+1}+(r_{s_{n-1}})_j-1)\\ &+(\theta_\mathsf{U})_j(r_{s_{n-1}})_j)\end{aligned} \tag{1}$$

$$\begin{aligned}(\mathbf{x}_{s_i})_j = \sigma((l_{s_i})_j &+ (\theta_\neg)_j\sigma(1-(\mathbf{x}_{s_{n-1}})_{j+1})\\ &+(\theta_\wedge)_j\sigma((\mathbf{x}_{s_{n-1}})_{j+1}+(r_{s_{n-1}})_j-1)\\ &+(\theta_\mathsf{X})_j(x_{s_{i+1}})_{j+1}\\ &+(\theta_\mathsf{U})_j((r_{s_i})_j+\sigma((\mathbf{x}_{s_i})_{j+1}+(x_{s_{i+1}})_j-1)))\end{aligned} \tag{2}$$

where

$$(r_{s_i})_j = \sum_{k=L}^{j+2}(\theta_\mathrm{r})_{j,k}(\mathbf{x}_{s_i})_k, (l_{s_i})_j = \sum_{k=1}^{|\mathbb{P}|}(\theta_\mathrm{a})_{j,k}(P_{s_i})_k,$$

$$\sigma(x) = \begin{cases} \alpha x, & x < 0 \\ x, & 0 \le x < 1 \\ 1+\alpha(x-1), & 1 \le x \end{cases}$$

and α is a hyper parameter.

Let θ be the LTL encoding parameters of ϕ. Then for any trace $\pi = \langle s_0, s_1, \ldots, s_{n-1}\rangle$, it holds that the satisfaction vector $(\mathbf{x}_{s_0})_1 = 1 \Leftrightarrow \pi \models \phi$ and $(\mathbf{x}_{s_0})_1 = 0 \Leftrightarrow \pi \not\models \phi$.

Intuitively the satisfaction vector represents the satisfaction relation. If $(\mathbf{x}_{s_i})_j = 1$, then the j-th subformula is satisfied on π_i. This is obvious because the calculation corresponds to the semantics of LTL. For example, let $\phi_1 = \phi_2\mathsf{U}\phi_3$ then $\pi_0 \models \phi_1 \Leftrightarrow \pi_0 \models \phi_3 \vee (\pi_0 \models \phi_2 \wedge \pi_1 \models \phi_1)$. And we can know that $(x_{s_0})_1 = \sigma((x_{s_0})_3 + \sigma((x_{s_0})_2 + (x_{s_1})_1 - 1))$, which corresponds to the semantics of operator U.

Therefore, given a trace of event π, the output of our model with parameters θ is $\mathrm{model}_\theta(C(\pi)) = (\mathbf{x}_{s_0})_1$.

B. VPU Loss

```
Algorithm 1: Calculating $\mathcal{L}_{\mathrm{VPU}}$
Input  : Model $\mathrm{model}_\theta$, positive and unlabeled sample batches
         $\Pi_P = \{\pi_0^P, \pi_1^P, \ldots, \pi_{b-1}^P\}, \Pi_U = \{\pi_0^U, \pi_1^U, \ldots, \pi_{b-1}^U\}$, hyperparameters $\alpha, \lambda$
Output : Loss $\mathcal{L}_{\mathrm{VPU}}$
1 $b \leftarrow |\Pi_P|$;
2 $\mathcal{L}_{\mathrm{var}} \leftarrow \log \frac{1}{b} \sum_{\pi_i^U \in \Pi_U} \mathrm{model}_\theta(C(\pi_i^U)) - \frac{1}{b} \sum_{\pi_i^P \in \Pi_P} \log \mathrm{model}_\theta(C(\pi_i^P))$;
3 Sample $\gamma \sim \mathrm{Beta}(\alpha, \alpha)$;
4 for $0 \le i < b$ do
5     $\widetilde{\mathbf{x}}_\mathbf{i} \leftarrow \gamma C(\pi_i^P) + (1-\gamma) C(\pi_i^U)$;
6     $\widetilde{y}_i \leftarrow \gamma + (1-\gamma)\mathrm{model}_\theta(C(\pi_i^U))$;
7 $\mathcal{L}_{\mathrm{reg}} \leftarrow \frac{1}{b} \sum_{i=1}^{b} (\log \mathrm{model}_\theta(\widetilde{\mathbf{x}}_\mathbf{i}) - \log \widetilde{y}_i)^2$;
8 $\mathcal{L}_{\mathrm{VPU}} \leftarrow \mathcal{L}_{\mathrm{var}} + \lambda \mathcal{L}_{\mathrm{reg}}$;
9 return $\mathcal{L}_{\mathrm{VPU}}$
```

Negative samples are usually unavailable in the scenario of specification mining. We usually have positive samples from event logs, and we can generate unlabeled samples randomly. Therefore we apply *variational PU (VPU)* loss proposed by Chen et al.(2020)[5], as shown in Algorithm 1.

C. Applying Formula Template

It is possible to apply a template to our model by setting and freezing part of the parameters. We traversal the tree-structured formula template by preorder and set the parameters. These parameters cannot be updated during the training process.

D. Interpreting LTL Formula

Finally, we interpret the LTL formula from model parameters through the algorithm described in Algorithm 2.

This algorithm interpret formulae from bottom to top. In $f_i = \{(\phi_{i,j}, s_{i,j})\}$, We consider the score $s_{i,j}$ as the probability of $\phi_{i,j}$ at v_i, *i.e.* $s_{i,j} = P\{\phi_{i,j} | v_i\}$. For example, $(\theta_\mathrm{r})_{i,k}$ is the probability of the root of v_i's right subtree being v_k. $(\theta_\wedge)_i$ is considered as the probability of v_i being a node representing $\wedge$. Thus the score of $\phi_{i,j} = \phi_{i+1,p} \wedge \phi_{k,q}$ is

$$
\begin{aligned}
s_{i,j} =& P\{\phi_{i_j} = \phi_{i+1,p} \wedge \phi_{k,q} | v_i\} \\
=& P\{\wedge | v_i\} P\{\phi_{i+1,p} | v_{i+1}\} P\{\phi_{k,q} | v_k\} \\
& P\{\mathrm{rightSubtree}(v_k, v_i)\} \\
=& (\theta_\wedge)_i s_{i+1,p} s_{k,q} (\theta_\mathrm{r})_{i,k}
\end{aligned}
\tag{3}
$$

Clearly, short formulae would take advantage of the scoring method, therefore our approach prefers short formulae rather than long formulae.

```
Algorithm 2: Interpreting LTL Formula
Input  : Model parameters $\theta$, width of beam search $k$, max tree size $L$, the training set $\Pi$
Output : An LTL formula $\phi$ interpreted from $\theta$
1 $i \leftarrow L$;
2 while $i \ge 1$ do
3     $f_i \leftarrow \emptyset$;
4     for $p_k \in \mathbb{P}$ do
5         $f_i \leftarrow f_i \cup \{(p_k, (\theta_\mathrm{a})_{i,k})\}$;
6     for $(\phi_k, s_k) \in f_{i+1}$ do
7         $f_i \leftarrow f_i \cup \{((\neg, \phi_k), (\theta_\neg)_i s_k), ((\mathsf{X}, \phi_k), (\theta_\mathsf{X})_i s_k)\}$;
8     for $i + 2 \le j \le L$ do
9         for $((\phi_l, s_l), (\phi_r, s_r)) \in f_{i+1} \times f_j$ do
10            $f_i \leftarrow f_i \cup \{((\wedge, \phi_l, \phi_r), (\theta_\wedge)_i (\theta_\mathrm{r})_{i,r} s_l s_r)\}$;
11            $f_i \leftarrow f_i \cup \{((\mathsf{U}, \phi_l, \phi_r), (\theta_\mathsf{U})_i (\theta_\mathrm{r})_{i,r} s_l s_r)\}$;
12    Sort $f_i = \{(\phi_j, s_j)\}$ according to $s_j$, keep top $k$ elements and remove the others;
13    $i \leftarrow i - 1$;
14 $\phi \leftarrow$ the best formula from $f_1$ according to classification accuracy on $\Pi$;
15 return $\phi$;
```

III. Empirical Evaluation

A. Dataset and Competitors

Our dataset contains 8 benchmarks, each benchmark corresponds to a template in Table I.

TABLE I
LTL TEMPLATES WE APPLIED IN OUR DATASET.

Name	LTL
Response	$\mathsf{G}(x \to \mathsf{XF}y)$
Alternating	$(\neg y \mathsf{W} x) \wedge \mathsf{G}((x \to \mathsf{X}(\neg x \mathsf{U} y) \wedge (y \to \mathsf{X}(\neg y \mathsf{W} x)))$
MultiEffect	$(\neg y \mathsf{W} x) \wedge \mathsf{G}(x \to \mathsf{X}(\neg x \mathsf{U} y))$
MultiCause	$(\neg y \mathsf{W} x) \wedge \mathsf{G}(x \to \mathsf{XF} y) \wedge \mathsf{G}(y \to \mathsf{X}(\neg y \mathsf{W} x)))$
EffectFirst	$\mathsf{G}((x \to \mathsf{X}(\neg x \mathsf{U} y)) \wedge (y \to \mathsf{X}(\neg y \mathsf{W} x)))$
CauseFirst	$(\neg y \mathsf{W} x) \wedge \mathsf{G}(x \to \mathsf{XF} y)$
OneCause	$\mathsf{G}(x \to \mathsf{X}(\neg x \mathsf{U} y))$
OneEffect	$\mathsf{G}(x \to \mathsf{XF} y) \wedge \mathsf{G}(y \to \mathsf{X}(\neg y \mathsf{W} x))$

For each benchmark, there is a test set containing 200 positive traces and 200 negative traces. Traces in test sets are generated randomly and verified by a LTL solver.

For each benchmark, there are 6 training sets marked as PUx for $x \in \{0, 1, 2, 3, 4, 5\}$. Each of them contains 200 positive traces and 200 *unlabeled* traces, with a noise rate of $\frac{x}{10}$.

We compare our approach with `Texada` [13], which is the SOTA search-based LTL specification mining approach and is widely used in various downstream models such as DICE [8]. Another competitor is `GLTLf` [16], the SOTA neural-based LTL formula learning approach.

B. Research Questions

RQ 1: How effective is `PURLTL` under noise-free inputs?

TABLE II
F_1 SCORES (%) OF Texada, GLTLf AND PURLTL ON PU0. THE BEST RESULTS ARE MARKED WITH UNDERLINE (SAME ON OTHER TABLES).

	Texada	GLTLf	PURLTL
Response	100	49.41	100
Alternating	100	81.97	89.49
MultiEffect	100	80	99.5
MultiCause	100	86.58	100
EffectFirst	100	58.61	99.5
CauseFirst	100	85.11	100
OneCause	100	66.52	100
OneEffect	100	61.98	100

To answer this question, we trained GLTLf and PURLTL on PU0. We also ran Texada using the positive traces in PU0, for Texada can not make use of unlabeled data.

As shown in Table II, Texada successfully mines all formulae correctly with the noise-free assumption, which barely holds in practice. On the other hand, PURLTL falls behind with a small gap, this result shows that PURLTL gets a competitive performance, while having other advantages such as robustness under imperfect traces and not relying on templates.

TABLE III
F_1 SCORES (%) OF GLTLf AND PURLTL WITH DIFFERENT NOISE RATES.

		PU0	PU1	PU3	PU5
Response	GLTLf	49.41	49.25	49.25	49.25
	PURLTL	100	100	100	100
Alternating	GLTLf	81.97	81.97	80	80
	PURLTL	89.49	89.49	89.49	89.49
MultiEffect	GLTLf	80	80	80	80
	PURLTL	99.5	99.25	97.3	97.3
MultiCause	GLTLf	86.58	86.58	86.58	86.58
	PURLTL	100	100	100	100
EffectFirst	GLTLf	58.61	58.61	58.61	58.61
	PURLTL	99.5	99.5	96.37	96.37
CauseFirst	GLTLf	85.11	85.11	85.11	85.11
	PURLTL	100	100	100	100
OneCause	GLTLf	66.52	66.52	66.52	66.52
	PURLTL	100	100	100	100
OneEffect	GLTLf	61.98	61.98	61.98	61.98
	PURLTL	100	92.35	92.35	95.69

RQ 2: How robustness is PURLTL under imperfect traces? We trained GLTLf and PURLTL on PUx training sets and compare their results. According to Table III we can find that PURLTL surpassed GLTLf on all benchmarks by large margins and shows great robustness under imperfect traces.

RQ 3: What does the PU setting contribute?
We trained our model on PU0 training sets with MSE and VPU loss functions separately. To better see the performance gap, templates were disabled in this experiment. According to Table IV, VPU loss function reaches better performance compared to traditional MSE loss function in most benchmarks.

TABLE IV
F_1 SCORES (%) OF PURLTL WITH DIFFERENT LOSS FUNCTIONS.

	Response	Alternating	MultiEffect	MultiCause
MSE	49.41	88.89	88.79	86.58
VPU	66.42	98.04	96.11	86.58
	EffectFirst	**CauseFirst**	**OneCause**	**OneEffect**
MSE	75.14	85.11	61.3	72.49
VPU	61.21	85.11	83.68	72.49

RQ 4: Is PURLTL able to work without templates?

TABLE V
F_1 SCORES (%) ON PU0 W/ AND W/O TEMPLATES.

	Response	Alternating	MultiEffect	MultiCause
w/ t.	100	100	99.5	100
w/o t.	66.42	98.04	96.11	86.58
	EffectFirst	**CauseFirst**	**OneCause**	**OneEffect**
w/ t.	99.5	100	100	100
w/o t.	61.21	85.11	83.68	72.49

We trained our model on PU0 training sets with and without templates. As shown in Table V, our approach is able to mine LTL formula without templates with decent performance.

IV. Conclusion and Future Work

Specification mining is an important and challenging problem in software engineering. In this paper, we transform LTL specification mining to formula learning with positive and unlabeled traces. Our main contribution is to design a continuous approximation of path checking based on neural networks, which allows us to model formula learning as a neural network parameter search. Besides, we minimize the label bias from the lack of negative traces via VPU loss. Our approach is equipped with the following advantages: (1) it is robust with imperfect traces, (2) formula templates are optional to our approach, and (3) we minimized the label bias from the lack of negative traces. Experiments confirm that these advantages bring significant performance improvement.

Future work will extend our method to learn a finite state machine that is richer expressive than LTL.

Acknowledgment

This paper was supported by the National Natural Science Foundation of China (No. 62276284, 61976232, 61876204), the National Key Research and Development Program of China (No. 2021YFA1000504), the Guangdong Basic and Applied Basic Research Foundation (No. 2023A1515011470, 2022A1515011355, 2020A1515010642), the Guangzhou Science and Technology Project (No. 202201011699), the Guizhou Provincial Science and Technology Projects (No. 2022-259), the Humanities and Social Science Research Project of Ministry of Education (No. 18YJCZH006), as well as the Fundamental Research Funds for the Central Universities, Sun Yat-sen University (No. 23ptpy31).

References

[1] Glenn Ammons, Rastislav Bodík, and James R. Larus. Mining specifications. In *POPL*, pages 4–16, 2002. doi: 10.1145/503272.503275.

[2] Ivan Beschastnikh, Yuriy Brun, Sigurd Schneider, Michael Sloan, and Michael D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *SIGSOFT FSE*, pages 267–277, 2011. doi: 10.1145/2025113.2025151.

[3] Alberto Camacho and Sheila A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *ICAPS*, pages 621–630, 2019.

[4] Zhi Cao and Nan Zhang. Deep specification mining with attention. In *COCOON*, volume 12273 of *Lecture Notes in Computer Science*, pages 186–197, 2020. doi: 10.1007/978-3-030-58150-3_15.

[5] Hui Chen, Fangqing Liu, Yin Wang, Liyue Zhao, and Hao Wu. A variational approach for learning from positive and unlabeled data. In *NeurlPS*, 2020.

[6] Nathanaël Fijalkow and Guillaume Lagarde. The complexity of learning linear temporal formulas from examples. In *ICGI*, volume 153 of *Proceedings of Machine Learning Research*, pages 237–250. PMLR, 2021. URL https://proceedings.mlr.press/v153/fijalkow21a.html.

[7] Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. Learning linear temporal properties from noisy data: A maxsat approach. *CoRR*, abs/2104.15083, 2021.

[8] Hong Jin Kang and David Lo. Adversarial specification mining. *ACM Trans. Softw. Eng. Methodol.*, 30(2):16:1–16:40, 2021. doi: 10.1145/3424307.

[9] John C Knight, Colleen L DeJong, Matthew S Gibble, and Luis G Nakano. Why are formal methods not used more widely? In *Fourth NASA formal methods workshop*, 1997.

[10] Ivo Krka, Yuriy Brun, and Nenad Medvidovic. Automatic mining of specifications from invocation traces and method invariants. In *SIGSOFT FSE*, pages 178–189, 2014. doi: 10.1145/2635868.2635890.

[11] Tien-Duy B. Le and David Lo. Deep specification mining. In *ISSTA*, pages 106–117, 2018. doi: 10.1145/3213846.3213876.

[12] Tien-Duy B. Le, Xuan-Bach Dinh Le, David Lo, and Ivan Beschastnikh. Synergizing specification miners through model fissions and fusions (T). In *ASE*, pages 115–125, 2015. doi: 10.1109/ASE.2015.83.

[13] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL specification mining (T). In *ASE*, pages 81–92, 2015. doi: 10.1109/ASE.2015.71.

[14] David Lo and Siau-Cheng Khoo. Smartic: towards building an accurate, robust and scalable specification miner. In *SIGSOFT FSE*, pages 265–275, 2006. doi: 10.1145/1181775.1181808.

[15] David Lo, Leonardo Mariani, and Mauro Pezzè. Automatic steering of behavioral model inference. In *ESEC/SIGSOFT FSE*, pages 345–354, 2009. doi: 10.1145/1595696.1595761.

[16] Weilin Luo, Pingjia Liang, Jianfeng Du, Hai Wan, Bo Peng, and Delong Zhang. Bridging ltlf inference to gnn inference for learning ltlf formulae. In *AAAI*, 2022.

[17] Weikai Miao and Shaoying Liu. A formal specification-based integration testing approach. In *SOFL*, volume 7787 of *Lecture Notes in Computer Science*, pages 26–43, 2012. doi: 10.1007/978-3-642-39277-1_3.

[18] Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. Perracotta: mining temporal API rules from imperfect traces. In *ICSE*, pages 282–291, 2006. doi: 10.1145/1134285.1134325.