ITG: Trace Generation via Iterative Interaction between LLM Query and Trace Checking

Weilin Luo[1], Weiyuan Fang [1*], Qiu Junming [1], Hai Wan [1,*], Yanan Liu [1], Rongzhen Ye [1]

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China

ICSE 2024

Content

Content

Relations of LTL SC&TG

Linear temporal logic (LTL) satisfiability checking (SC)

- Determines a LTL formula is satisfiable or unsatisfiable [8].
- Is a theoretical problem with PSPACE-Complete complexity [8].
- (SC with neural networks)
  Leaves hidden dangers that the result cannot be verified.

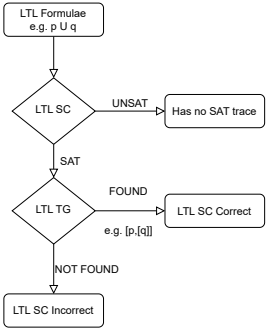

Figure 1: Relations of LTL SC&TG

Relations of LTL SC&TG

Linear temporal logic (LTL) satisfiability checking (SC)

- Determines a LTL formula is satisfiable or unsatisfiable [8].
- Is a theoretical problem with PSPACE-Complete complexity [8].
- (SC with neural networks)
  Leaves hidden dangers that the result cannot be verified.

Linear temporal logic (LTL) trace generation (TG)

- Gives a satisfiable trace to verify the LTLSC.
- Search space of satisfiable traces may explode exponentially.
- Existing heuristics for LTLTG
  is time-consuming and intelligence-required (manual)
  or high-training-overhead (automated).

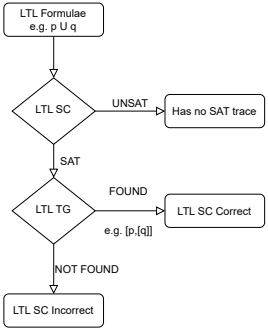

Figure 1: Relations of LTL SC&TG

Motivation

Related Work

- Hahn [2] demonstrated the potential of neural networks for trace generation by training the Transformer model.
- Prompt engineering research, such as the work by Wei [9], has explored how to guide LLMs for complex reasoning.
- Recent advancements in LLMs, represented by OpenAI's ChatGPT [4], have shown remarkable capabilities in tasks requiring reasoning, such as deduction and code generation.

Motivation

Why we choose Large Language Models (LLMs) to generate LTL trace?

- Neural networks for trace generation is limited by their size.
- LLMs have the ability to do logical reasoning.
- Users can spend less time training than using neural networks.

Insights

- LLMs have the ability to do logical reasoning.
- LLMs can **repair** their results by giving further information.
- Few-sample prompting is important for better performance.

Motivation

Why we choose Large Language Models (LLMs) to generate LTL trace?

- Neural networks for trace generation is limited by their size.
- LLMs have the ability to do logical reasoning.
- Users can spend less time training than using neural networks.

Insights

- LLMs have the ability to do logical reasoning.
- LLMs can **repair** their results by giving further information.
- Few-sample prompting is important for better performance.

Content

LLM Query

How to use LLMs to generate trace?

- We expect that LLMs can leverage
  rich LTL expertise to generate satisfiable traces heuristically.
- LLMs can be situated in
  a specific domain and elicited expertise in that domain by prompts.
- Although
  it is prevalent to control LLMs using prompts[1,5,6], our contribution
  is to design iterative interactive prompts for LTL trace generation.

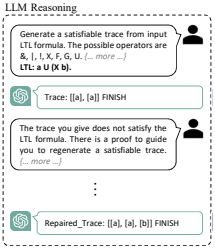

Figure 2: LLM Query

Trace Checking

How to check the trace is correct?

- Given an LTL formula and a trace, we can obtain the satisfiability proof in polynomial time by modifying the trace checking algorithm [3].
- We give the proof in text to guide the LLMs to repair the trace.

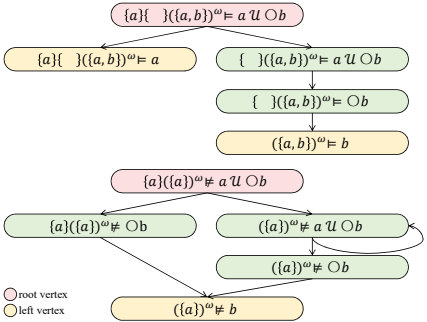

Figure 3: Example of Proof

Overview of ITG

We use a loop to build the ITG.

Algorithm 1: ITG

Input: an LTL formula ϕ.
Output: a trace π.

$prompt_{init} \leftarrow$ generate the initialization prompt of ϕ
$ans \leftarrow \text{LLMQuery}(\phi, prompt_{init})$
while *not reach the maximum number of iterations* **do**
  $\pi \leftarrow$ extract a trace based on patterns from ans
  if $\textsc{TraceCheck}(\pi, \phi)$ *is false* **then**
    $\Pi \leftarrow \text{ProofGenerate}(\pi, \phi, \not\models)$
    $prompt_{repair} \leftarrow$ generate the repair prompt based on Π
    $ans \leftarrow \text{LLMQuery}(\phi, prompt_{repair})$
  else
    return π
return *UNKNOW*

Figure 4: Algorithm

Overview of ITG

The loop stops until a satisfying trace is generated as shown.

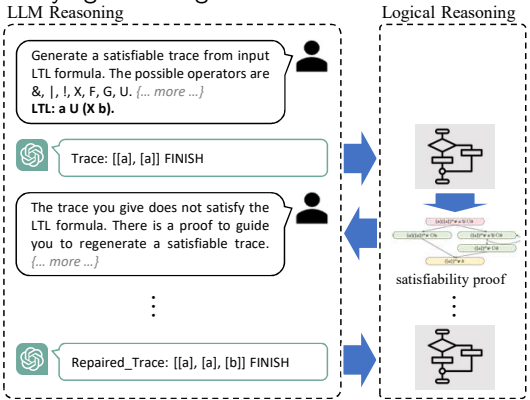

Figure 5: Overview of ITG

Examples

Example of Initially Generate Trace Prompt

```
1 Generate a satisfiable trace from input LTL formula. The possible atomic propositions
2 will be given. The possible operators are &, |, !, X, F, G, U. Trace should be less than
3 10 states. Each state should not contain dublicated atomic propositions. The output trace
4 should be a list of states. Do not use ... in output. For example: LTL: a U (X b) Trace:
5 [[a],[],[b]] FINISH LTL: F(a & X b) Trace: [[a],[b]] FINISH LTL: G(a | ! b) Trace:
6 [[],[a]] FINISH LTL: a U (X b) Trace: [[a],[a]] FINISH
```

Figure 6: Example of Initialization Prompt

Example of Repairing Trace Prompt

```
1 The trace you give does not satisfy the LTL formula. There is a proof to guide you to
2 regenerate a satisfiable trace. For example: LTL: a U (X b) Trace: [[a],[a]] Proof:{[[a]]
3 not satisfies b; [[a]] not satisfies X b; [[a]] not satisfies a U (X b); [[a],[a]] not satisfies
4 X b} Repaired_Trace: [[a], [a], [b]] FINISH
```

Figure 7: Example of Repairing Trace Prompt

Content

Dataset

We use the randltl tool of SPOT to generate random formulae. We generate satisfiable formulae into 5 sets according to atomic propositions size: [5-20),[20-40),[40-60),[60-80),[80-100). Figure 8 is an example of a single data.


```
{
  "inorder": "(X((p0) | (F((p0) | (F(X(X(p7)))))))) U (p6)",
  "preorder": "UX|p0F|p0FXXp7p6",
  "issat": true,
  "nuXmv_ic3_time": 0.06263470649719238,
  "nuXmv_ic3_trace": [
    [
      "p7"
    ],
    [],
    [
      "p6",
      "p7"
    ],
    []
  ],
  "nuXmv_ic3_loop_start": 1,
  "nuXmv_ic3_isvalid": true
},
```


Figure 8: Example of Dataset

Evaluation Metrics

We use semantic accuracy to evaluate the approaches. If the generated trace satisfies the formula, then it is semantically accurate.

Competitor, Dataset, and Setup

Competitor Models

- Random
- Transformer
- GPT-3.5-Turbo
- GPT-4

Competitor, Dataset, and Setup

Competitor Models

- Random
- Transformer
- GPT-3.5-Turbo
- GPT-4

Competitor Approaches

- Simple Input Output(Simple_IO)
- Chain of Thought by Node(CoT_Node)
- Chain of Thought by Tree(CoT_tree)
- ITG

Competitor, Dataset, and Setup

Competitor Models

- Random
- Transformer
- GPT-3.5-Turbo
- GPT-4

Competitor Approaches

- Simple Input Output(Simple_IO)
- Chain of Thought by Node(CoT_Node)
- Chain of Thought by Tree(CoT_tree)
- `ITG`

Setup

- Give a LTL formulae each time to the model.
- The model outputs a trace.
- Use model check tool (nuXmv e.g.) to check the satisfiability of the trace.

Are the SOTA neural network-based approach efficient?

`Transformer`[2] is the SOTA neural network-based approach in generating traces. We train and test `Transformer` on *SPOT* with different formula sizes. Besides, we randomly generate a trace in right syntactic for each formula, denoted by random.

Table 1: Semantic accuracy (%) of `Transformer` on datasets with various formula sizes.

	$[5, 20)$	$[20, 40)$	$[40, 60)$	$[60, 80)$	$[80, 100)$
`Transformer`	93.30	76.21	58.66	58.32	56.35

Summary. Table 1 reports the evaluation results of `Transformer` on datasets with various formula sizes. It can be observed that `Transformer` exhibits performance degradation in tandem with the increase of formula sizes. This implies that the SOTA neural network-based approach fails to generalize in the reasoning scenarios with larger formula sizes.

What is the performance of ITG?

We compare ITG with Transformer and a variant of ITG (denoted by ITG-init). In ITG-init, we only use the initialization prompt to query LLMs, *i.e.*, there is no logical reasoning. We only train Transformer on *SPOT*-$[5, 20)$ and test all approaches on larger formulae to evaluate the generalization ability across formula sizes.

What is the performance of ITG?

Table 2: Semantic accuracy (%) of approaches on datasets with various formula sizes, where **boldface** numbers refer to the better results.

	$[5, 20)$	$[20, 40)$	$[40, 60)$	$[60, 80)$	$[80, 100)$
random	54.20	52.10	53.10	53.10	54.00
Transformer	**93.30**	71.30	60.30	54.20	51.60
ITG-init	69.90	61.50	59.60	57.60	57.20
ITG	91.20	**81.00**	**70.30**	**75.50**	**75.00**

Summary. Table 2 reports the evaluation results on datasets with various formula sizes.

- It can be seen that both ITG-init and ITG significantly outperform random on all datasets. These results reveal that LLMs are capable of providing heuristics for trace generation.
- It is evident that ITG outperforms Transformer by a significant margin on datasets with formula sizes ≥ 20, and the performance degradation of ITG is relatively gentle.It confirms the generalization ability of ITG.
- We can observe that ITG obtains significant performance gains over ITG-init on all datasets. This implies that the proposed iterative interaction is effective in improving the trace generation performance.

ITG & CoT

Table 3: Semantic accuracy (%) of approaches on datasets with various formula sizes, where **boldface** numbers refer to the better results.

	$[5, 20)$	$[20, 40)$	$[40, 60)$	$[60, 80)$	$[80, 100)$
CoT-node	77.00	74.50	69.30	64.70	59.80
CoT-tree	80.20	74.20	69.50	65.40	62.30
CoT-SC	88.00	80.40	76.50	73.60	72.60
ITG	**91.20**	**81.00**	**70.30**	**75.50**	**75.00**

Summary. Table 3 reports the evaluation results on datasets with various formula sizes. We also compare ITG with CoT methods. We can see that ITG outperform CoT methods on all dataset. It reveals that LLMs have the better performance to repair the results by Iteration.

Content

Conclusion and Future Work

Conclusion

1. Our experimental results provide preliminary evidence that ITG can provide heuristics for LTL trace generation.
2. A larger maximum number of iterations can trigger more updates to the generated trace and trial and error opportunities, which is beneficial to improving the accuracy of ITG.

[1] https://platform.openai.com/docs/models/gpt-4

Conclusion and Future Work

Conclusion

1. Our experimental results provide preliminary evidence that ITG can provide heuristics for LTL trace generation.
2. A larger maximum number of iterations can trigger more updates to the generated trace and trial and error opportunities, which is beneficial to improving the accuracy of ITG.

Future work

1. Token limit (within 8192[1]) and query history for querying using ChatGPT's API, results in ITG not support large-scale formulae and large number of iterations.
   - We will experiment with LLMs without token limit in future work to support larger scale formulae.
2. Since ChatGPT currently cannot be invoked locally, ChatGPT queries need to access the service through the Internet.
   - We can select to use LLMs that can be deployed locally, *e.g.*, GLM[10] and BLOOM[7], to mitigate this threat.

[1] https://platform.openai.com/docs/models/gpt-4

References I

[1] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. *CoRR*, abs/2304.07590, 2023.

[2] Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. In *ICLR*, 2021.

[3] Nicolas Markey and Philippe Schnoebelen. Model checking a path. In *CONCUR*, volume 2761, pages 248–262, 2003.

[4] OpenAI. Chatgpt, 2023.

[5] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[6] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[7] Teven Le Scao, Thomas Wang, Daniel Hesslow, Stas Bekman, M. Saiful Bari, Stella Biderman, Hady Elsahar, Niklas Muennighoff, Jason Phang, Ofir Press, Colin Raffel, Victor Sanh, Sheng Shen, Lintang Sutawika, Jaesung Tae, Zheng Xin Yong, Julien Launay, and Iz Beltagy. What language model to train if you have one million GPU hours? In *EMNLP*, pages 765–782, 2022.

[8] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

[10] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. GLM-130B: an open bilingual pre-trained model. In *ICLR*, 2023.

Thank you for your listening!