Studying Formal Models Using ChatGPT

Peter Backeman[1 [0000−0001−7965−248X]], Rong Gu[1 [0000−0003−0570−6005]], and Florian Lorber[2 [0009−0006−2387−2778]]

[1] Mälardalen University, Sweden `first.last@mdu.se`
[2] Aalborg University, Denmark `florian.lorber@gmail.com`

Abstract. Since the release of Open AI's generative pre-trained transformer (GPT), researchers have been attempting to use such a powerful tool for text generation, such as natural language translation and code generation. In this paper, we explore an interesting usage of GPT in helping engineers understand and even detect flaws in formal models, which are well-known for being rigour but hard to understand by non-experts. We select several simple formal models in the fields of model checking and theorem proving. Through experiments, we aim to investigate three research questions that we compose to evaluate the powerfulness of GPT in interpreting existing formal models. As a preliminary report of our study, this paper only contains partial of the aim of our study. We outline in future work what a full article will contain.

Keywords: LLM · ChatGPT · Formal models.

1 Introduction

In the last year, large language models (LLM) have exploded in popularity, and have been applied to many different use cases. OpenAI's generative pre-trained transformer (GPT-3.5 & 4) have drawn wide attention from the mainstream media due to its remarkable performance of generating natural-language texts that are hard to distinguish from human-written words.

Therefore, plenty of studies have been carried out in the domains of language translation [16], text generation [20][21], model/code generation [17], and even human/robot motion generation [24][8]. Among these studies, there is a branch that focuses on using LLM in generating models that are hard to understand by non-experts, in particular, formal models such as temporal logic expressions [13][9] and automata [23].

Formal models are well-known for their precious mathematical foundation and ability to provide rigorous analysis of design artefacts. However, such models are hard to understand without detailed documentation and have a steep learning curve, so the aforementioned studies have been attempting to transform natural language (NL) descriptions into formal models. However, few researchers have considered a research question in the opposite direction, that is, *given an existing formal model, can we generate explanatory texts to help engineers understand the function and even detect flaws of the model?*

In this paper, we aim to explore the powers of GPT (in particular ChatGPT version 3.5) in helping engineers understand formal models, which lack comprehensive explanatory documentation.

The remainder of the paper is organized as follows. Section 2 introduces the background knowledge of this paper. Section 3 defines the research questions that we aim to study before we show the preliminary results of model explanation by ChatGPT in Section 4. Finally we conclude in Section. 5.

2 Background

In this paper, we work with ChatGPT version 3.5, a large language model by OpenAI. Using it requires writing *prompts*, which helps ChatGPT to generate responses that continue the conversation or expand on the given prompts. A prompt in this paper can be long, including formal models of hundreds of lines, and also takes into account previous prompts, i.e., it is possible to present a formal model and then make several prompts about it.

2.1 Formal Methods

In the field of formal methods, there are two main branches: model checking [10] and theorem proving [11]. Since there is a large number of formal models in each field, we have selected three topics to study: timed automata [1] (models described in UPPAAL [19]), satisfiability modulo theories (SMT) [3] and interactive theorem proving (proofs described in Coq [5]). These choices represent popular models and tools, but a more extensive study could include others as well.

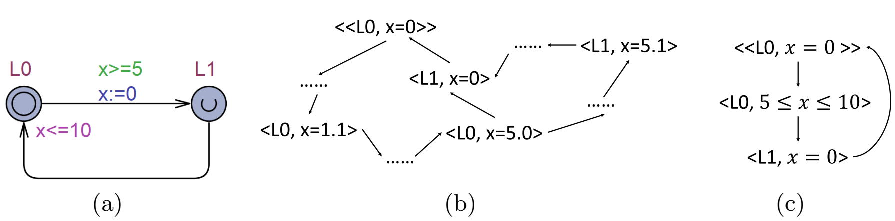

Fig. 1. An example of UPPAAL timed automata and different presentations of its state space. (a) The syntactic presentation of UTA. (b) One semantic presentation of the UTA: a timed transition system. (c) Another semantic presentation of the UTA: a symbolic timed transition system.

Timed Automata Formal models have rigorous definitions of syntax and semantics. For example, a well-known model checker UPPAAL [19] uses and extends timed automata [1] as the modelling language. We present an example

```xml
<template>
    <name x="5" y="5">Template</name>
    <declaration>clock x;</declaration>
    <location id="id0" x="127" y="0">
        <name x="117" y="-34">L1</name>
        <urgent/>
    </location>
    <location id="id1" x="0" y="0">
        <name x="-10" y="-34">L0</name>
        <label kind="invariant" x="8" y="8">x&lt;=10</label>
    </location>
    <init ref="id1"/>
    <transition>
        <source ref="id0"/>
        <target ref="id1"/>
        <nail x="127" y="51"/>
        <nail x="0" y="51"/>
    </transition>
```

Fig. 2. A part of the XML file of the example UPPAAL in Figure 1(a)

of UPPAAL timed automata (UTA) in Fig. 1(a) and different presentations of its semantics in Fig. 1(b) and Fig. 1(c). The syntactic presentation of UTA is static and stored as an XML file by UPPAAL. Fig. 2 partially shows the XML file of the example UTA in Fig. 1(a). All the syntactic notions of UTA, such as locations, are clearly declared in this file. Moreover, comments are also included in this XML file, such as explanations of the codes. Intuitively, these comments should be helpful for ChatGPT to interpret the model.

Although the syntactic presentation of a formal model is often explicit, the semantics of a formal model are underneath its syntactic presentation, which is not explicitly shown. For example, by looking at Fig. 1(a) or reading the XML file in Fig. 2, it is difficult for one who does not have the expertise of UPPAAL and timed automata to interpret model's semantics. Additionally, the original definition of the semantics of timed automata is timed transition systems [1] (see Fig. 1(b) as an example). UPPAAL uses Difference Bound Matrices (DBM) [4] to represent the semantic model of UTA, which is needed for symbolic verification.

To manually obtain the semantic models of large and complex UTA is extremely difficult, if not entirely impossible. Hence, model checkers, such as UPPAAL, are designed for automatically executing the formal models and generating their semantic models (aka, state spaces). For the sake of saving memory space, these tools often employ the so-called on-the-fly exploration of the state spaces when verifying the models. In summary, the semantic presentation of a formal model is extremely difficult to obtain by only reading its syntactic presentation without executing the model. We believe ChatGPT 3.5 may be able to interpret formal models at the syntactic level, but cannot fully understand their semantics. However, we wish to investigate to what extent it can provide an intuition.

Satisfiabilty Modulo Theories Satisfiability Modulo Theories (SMT) is the problem of determining if a formula is satisfiable or note, i.e., if there is a model satisfying formula. We only give a brief overview of SMT and do not consider the details, but in principle a problem consists of a set of variables (of different types) and a set of constraint over them (usually with a Boolean structure). A SMT solver then tries to find assignment to variables such that all constraints are satisfied. This process involves using so called theory-solvers, each design to handle a specific background theory (e.g., linear integer arithmetic).

An SMT instance is often written in a standard format, calletd SMT-LIB[2]. In Figure 3 an example SMT instance is shown. It begins by stating that the logic involved is Quantifier-Free Linear Integer-Arithmetic (QF_LIA), followed by a declaration of two integer variables (called constants) x, y, a constraint that $x > y$, and finally a command checking if the problem is satisfiable or not.

```
(set-logic QF_LIA)
(declare-const x Int)
(declare-const y Int)
(assert (> x y))
(check-sat)
```

Fig. 3. Example SMT instance.

Comments can also be included (preceded by semi-colon), and instances are often passed to an SMT solver, e.g., Z3 [12], which provides an answer if it is satisfiable or not. In the former case, it will also provide a model, e.g., assignments to all the variables of the problem. In this instance, a solution could be $x = 1, y = 0$.

Interactive Theorem Proving Achieving fully automated generation of proofs of formulas is a very hard task. An interactive theorem prover relies on interaction between the user and a proof-assistant. The resulting product becomes a proof which has been checked and is guaranteed to be correct, thus implying the correctness of the result of the proof. The syntax of such proofs can be tricky to understand even when the formula is simple, especially for non-experts. Consider, for example, the proof in Figure 4[3]. It is a simple proof the applying negation twice cancels out, but it is not obvious at a glance.

3 Problem Description

In this section, we introduce the research questions that this paper answers and define the research problem of this study.

[3] From https://gist.github.com/alpaylan/54cd1482e242e15283bf1c351cbe8cdc

```
Inductive bool: Type :=
  | true
  | false.

 Definition negb (b:bool) : bool :=
  match b with
  | true -> false
  | false -> true
  end.

Theorem negb_involutive : A b : bool,
  negb (negb b) = b.
Proof.
  intros b. destruct b eqn:E.
  - reflexivity.
  - reflexivity.
Qed.
```

Fig. 4. Example Coq proof. Syntax slightly modified to work in UTF-8 encoding.

3.1 Research Questions

Formal models are well-known for their precise mathematical foundation, which supports rigorous analysis and verification. However, the mathematical notions and abstract presentations in such models often hinder engineers from using them in their daily work. As our world is becoming more and more digitalized and emerging complex safety-critical systems such as autonomous vehicles are approaching, traditional methods for system development and verification are not sufficient to guarantee the correctness of such systems.

Many fields combine machine learning and formal methods [18], such as autonomous systems [6][15], cyber-physical systems [22], and multi-agent systems [14]. One interesting phenomenon in this research direction is that most studies focus on either the correctness guarantee of machine learning or model/code generation by using machine learning. However, few studies have paid attention to the knowledge gap between practitioners and researchers, that is, formal models are hard to learn and understand! Concretely, we identify several scenarios where engineers need help in understanding models of their systems.

1. When facing a historical system that does not have sufficient documentation, engineers need help in understanding what the system is designed for and even a detailed explanation of the function of a specific component.
2. When new employees join a team, they need to understand the existing system with possibly limited help from their colleagues.
3. In agile development, engineers need to quickly and automatically generate documents of their design artefacts.

4. When a third party needs to test a system and the design artefacts are not self-explanatory, they need help in understanding the diagrams in the documentation.

In all these scenarios, engineers need an interpreter to help them understand the design artefacts of the system. However, due to the limited resources that they have and the restricted time for the projects, the help that they can get from the organisation is often very limited. In this paper, we study how ChatGPT can be leveraged to help us explain system artefacts, in particular formal models. In summary, we define the research question that this paper aims to answer.

Research Question 1 (Interpretation and Explanation) *To what extent do large language models have the potential to interpret and explain formal models used by different verification techniques in natural language?*

Research Question 2 (Assistance for Modeling) *Can a large language model assist in development of a formal model, by providing help on modification of existing or introduction of new parts?*

Research Question 3 (Generalization) *How is the interpretation of ChatGPT affected when studying models that are exposed on the Internet before the day they finish training better than those that are not.*

ChatGPT is a pre-trained LLM, which means when we use them to interpret models, they have already been trained by using the online datasets [7]. Table 1 shows the mixture of datasets used for training GPT-3.5. We wish to investigate how much of the capability of ChatGPT can be generalized to arbitrary models. For example, how is the explanation of a model affected if we rename variables, or move sections of the model around, making it different syntactically while retaining the exact semantics.

Table 1. Datasets used to train GPT-3 [7].

Dataset	Quantity (tokens)	Weight in training mix
Common Crawl (filtered)	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

We would also like to study the differences in model interpretation between the models that were published before GPT-3 and the ones that were published later or not unpublished at all.

4 Interpretating and Explaning Formal Models

In this section, we discuss how LLMs can be used how to explain and interpret the selected formal models. We present our current findings as a set of use cases.

4.1 Use Case (UPPAAL) Train Gate

We begin by studying the *train* model, which is provided by deafult with UPPAAL 5.0. We input the entire model ($\sim$ 400 lines) and ask ChatGPT: *"What does this model do?"*. After the reply we asks for clarification about a specific query: *"Could you explain this query in more depth Verifying that when a train approaches, it eventually crosses the bridge."*. We do not present the full replies here but highlight interesting parts.

Liveness Properties In the reply, ChatGPT correctly identifies that the UPPAAL query is a liveness property:

> The query "Verifying that when a train approaches, it eventually crosses the bridge" is a liveness property

This is followed by an explanation of liveness properties in general (excluded here). What is interesting is the description of this particular query's meaning:

> This means that every time a train initiates the "approach" action (Train(i).Appr), it should lead to the "crossing" state (Train(i).Cross) for that specific train.

which is followed by an interpretation of the query:

> In practical terms, this property ensures that no train remains stuck indefinitely in the "approach" state after initiating the approach action. It guarantees that each train will successfully complete its crossing of the bridge if it begins the approach process.

This is perhaps the most interesting result in this use case, as it is a very good explanation of the intuition behind the query.

4.2 Use Case (SMT): Pigeonhole

In the SMT-LIB, a set of benchmarks for SMT-solvers [2], there is formalization of a variant of the pigeonhole principle. The principle states that "if n items are placed in m containers and $n > m$, then there must be at least two items in one container". The benchmark[4] formulates the question "in a grid with 11 rows and 10 columns, if there is at least one pigeon in each row, can there be at most one pigeon in each column" (which is impossible). We ask ChatGPT to explain the intuition of the formula: *Can you explain the intuition behind the following SMT formula:* , followed by the SMT instance ($\sim$ 350 lines):

[4] `QF_LIA/pidgeons/pigeon-hole-10.smt2`

```
In summary, this SMT formula appears to describe a grid of
binary variables with constraints that require at least one
variable to be set to 1 in each row and at most one variable
to be set to 1 in each column. The goal is to determine if
there exists a valid assignment of these binary variables that
 satisfies these conditions.
```

Interestingly, when querying about the satisfiability (*Do you think it is satisfiable?*), ChatGPT provides a correct reasoning, with only the grid size being wrong (unfortunately, that mistake makes the whole reasoning confusing and faulty):

```
Based on the constraints provided in the SMT formula, it
appears that the formula is unsatisfiable. The reason is that
the formula imposes two conflicting constraints on each row
and column:

1. In each row, it requires that the sum of the values of the
variables in that row must be greater than or equal to 1,
which means at least one variable in each row must be set to
1.

2. In each column, it requires that the sum of the values of
the variables in that column must be less than or equal to 1,
which means at most one variable in each column can be set to
1.

These two sets of constraints conflict with each other. It is
not possible to satisfy both conditions simultaneously for all
 rows and columns in an 11x11 grid because if you set at least
 one variable to 1 in each row, it will necessarily set more
than one variable to 1 in some columns (violating the column
constraints), and if you set at most one variable to 1 in each
 column, it will result in some rows having all variables set
to 0 (violating the row constraints).

Therefore, the SMT formula is expected to return unsat,
indicating that there is no valid assignment of values to the
variables that simultaneously satisfies all the constraints.
```

4.3 Use Case (Coq): Direct Proof

Finally, in our study of explanation of formal models, we look at ChatGPTs capability of explaining the intuition behind a proof. Coq [?] is an interactive theorem prover, where the user can write proofs which are checked by Coq to ensure their correctness. We give a quite simple proof, showing that negation is involutive: `Lemma negb_involutive : forall b:bool, negb (negb b) = b.`[5]. While the proof is simple, for a person with no experience with Coq notation can be a bit bothersome to read. We ask ChatGPT to provide an explanation (*Can you explain the following proof using standard mathematical notation:*), which after a follow-up prompt *A bit shorter please.* yields:

```
Theorem: For all Boolean values b, negb(negb(b)) = b.

Proof:

    Introduction (intros): Assume an arbitrary Boolean value b.

    Case Analysis (destruct):
        When b is true, we have negb(negb(true)) = true.
        When b is false, we have negb(negb(false)) = false.

    In both cases, the equation holds, demonstrating that negb(
negb(b)) is equal to b. Therefore, the theorem is proven.
```

And this final output provides something which is quite easy to understand quickly, without any of the syntax remaining.

5 Conclusions

We have proposed three research questions regarding investigation of the capabilities of ChatGPT to explain formal models. In three use case studies we have begun identifying some interesting directions for further study. The results are very preliminary but promising, showing that it can indeed be the case that ChatGPT is useful for non-experts to interpret formal models.

5.1 Future Work

The next steps are both in investigating the first research question more thoroughly and in a more systematic manner. This will be done by investigating multiple models and compare results and try to identify strength and weaknesses of ChatGPT. The second research questions, concerned how ChatGPT can assists in development of models, will be studied by asking for extensions and modifications of the different models. Finally, the third research question will be investigated by defining syntactical transformations (which are semantical equivalent) and see how the explanations differ. A tough part component of all these questions is to handle the non-determinism of ChatGPT in a suitable

[5] This is the description of the Lemma, the proof can be found as an standard example of Coq

way, as well as the learning component of the LLM (i.e., if we ask about the same model multiple times, ChatGPT could learn from this interaction thus becoming more apt in understanding the model).

5.2 Acknowledgements

We acknowledge the support of the Swedish Knowledge Foundation via the synergy ACICS – Assured Cloud Platforms for Industrial Cyber-Physical Systems, grant nr. 20190038.

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science **126**, 183–235 (1994)
2. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
3. Barrett, C., Tinelli, C.: Satisfiability modulo theories. Springer (2018)
4. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Advanced Course on Petri Nets. pp. 87–124. Springer (2003)
5. Bertot, Y., Castéran, P.: Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions. Springer Science & Business Media (2013)
6. Bouton, M., Karlsson, J., Nakhaei, A., Fujimura, K., Kochenderfer, M.J., Tumova, J.: Reinforcement learning with probabilistic guarantees for autonomous driving. arXiv preprint arXiv:1904.07189 (2019)
7. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
8. Chen, Y., Arkin, J., Zhang, Y., Roy, N., Fan, C.: Autotamp: Autoregressive task and motion planning with llms as translators and checkers. arXiv preprint arXiv:2306.06531 (2023)
9. Chen, Y., Gandhi, R., Zhang, Y., Fan, C.: Nl2tl: Transforming natural languages to temporal logics using large language models. arXiv preprint arXiv:2305.07766 (2023)
10. Clarke, E.M.: Model checking. In: Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17. pp. 54–56. Springer (1997)
11. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**(7), 394–397 (1962)
12. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
13. Fuggitti, F., Chakraborti, T.: Nl2ltl–a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas. In: AAAI Conference on Artificial Intelligence (2023)
14. Gross, D., Schmidl, C., Jansen, N., Pérez, G.A.: Model checking for adversarial multi-agent reinforcement learning with reactive defense methods. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 33, pp. 162–170 (2023)

15. Gu, R., Enoiu, E., Seceleanu, C., Lundqvist, K.: Verifiable and scalable mission-plan synthesis for autonomous agents. In: Formal Methods for Industrial Critical Systems: 25th International Conference, FMICS 2020, Vienna, Austria, September 2–3, 2020, Proceedings 25. pp. 73–92. Springer (2020)
16. Jiao, W., Wang, W., Huang, J., Wang, X., Tu, Z.: Is chatgpt a good translator? yes with gpt-4 as the engine. arXiv preprint arXiv:2301.08745 (2023)
17. Khan, J.Y., Uddin, G.: Automatic code documentation generation using gpt-3. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. pp. 1–6 (2022)
18. Larsen, K., Legay, A., Nolte, G., Schlüter, M., Stoelinga, M., Steffen, B.: Formal methods meet machine learning (f3ml). In: International Symposium on Leveraging Applications of Formal Methods. pp. 393–405. Springer (2022)
19. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International journal on software tools for technology transfer **1**(1-2), 134–152 (1997)
20. Liao, Y., Wang, Y., Liu, Q., Jiang, X.: Gpt-based generation for classical chinese poetry. arXiv preprint arXiv:1907.00151 (2019)
21. Mager, M., Astudillo, R.F., Naseem, T., Sultan, M.A., Lee, Y.S., Florian, R., Roukos, S.: Gpt-too: A language-model-first approach for amr-to-text generation. arXiv preprint arXiv:2005.09123 (2020)
22. Meng, Y., Fan, C.: Hybrid systems neural control with region-of-attraction planner. In: Learning for Dynamics and Control Conference. pp. 1400–1415. PMLR (2023)
23. Yang, Y., Gaglione, J.R., Topcu, U.: Learning automata-based task knowledge representation from large-scale generative language models. arXiv preprint arXiv:2212.01944 (2022)
24. Zhang, J., Zhang, Y., Cun, X., Huang, S., Zhang, Y., Zhao, H., Lu, H., Shen, X.: T2m-gpt: Generating human motion from textual descriptions with discrete representations. arXiv preprint arXiv:2301.06052 (2023)