nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models

Matthias Cosler[2], Christopher Hahn[1](✉), Daniel Mendoza[1](✉), Frederik Schmitt[2], and Caroline Trippel[1]

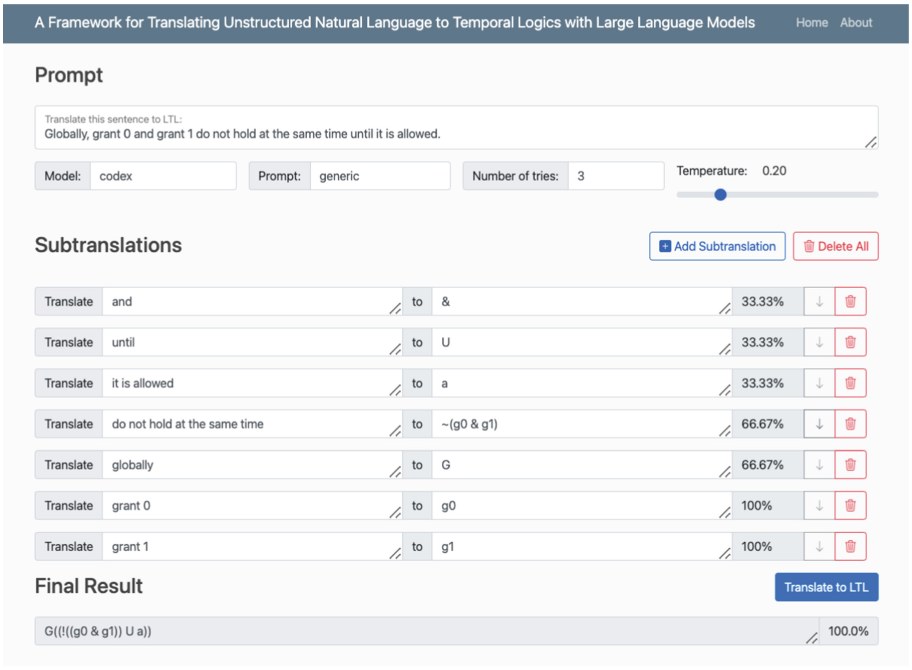
[1] Stanford University, Stanford, CA, USA
`hahn@cs.stanford.edu`, `{dmendo,trippel}@stanford.edu`
[2] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
`{matthias.cosler,frederik.schmitt}@cispa.de`

Abstract. A rigorous formalization of desired system requirements is indispensable when performing any verification task. This often limits the application of verification techniques, as writing formal specifications is an error-prone and time-consuming manual task. To facilitate this, we present `nl2spec`, a framework for applying Large Language Models (LLMs) to derive formal specifications (in temporal logics) from unstructured natural language. In particular, we introduce a new methodology to detect and resolve the inherent ambiguity of system requirements in natural language: we utilize LLMs to map subformulas of the formalization back to the corresponding natural language fragments of the input. Users iteratively add, delete, and edit these sub-translations to amend erroneous formalizations, which is easier than manually redrafting the entire formalization. The framework is agnostic to specific application domains and can be extended to similar specification languages and new neural models. We perform a user study to obtain a challenging dataset, which we use to run experiments on the quality of translations. We provide an open-source implementation, including a web-based frontend.

1 Introduction

A rigorous formalization of desired system requirements is indispensable when performing any verification-related task, such as model checking [7], synthesis [6], or runtime verification [20]. Writing formal specifications, however, is an error-prone and time-consuming manual task typically reserved for experts in the field. This paper presents `nl2spec`, a framework, accompanied by a web-based tool, to facilitate and automate writing formal specifications (in LTL [34] and similar temporal logics). The core contribution is a new methodology to decompose the natural language input into *sub-translations* by utilizing Large Language Models (LLMs). The `nl2spec` framework provides an interface to interactively

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13965, pp. 383–396, 2023.
https://doi.org/10.1007/978-3-031-37703-7_18


A Framework for Translating Unstructured Natural Language to Temporal Logics with Large Language Models

Prompt

Translate this sentence to LTL:
Globally, grant 0 and grant 1 do not hold at the same time until it is allowed.

Model: codex
Prompt: generic
Number of tries: 3
Temperature: 0.20

Subtranslations

Translate	and	to	&	33.33%
Translate	until	to	U	33.33%
Translate	it is allowed	to	a	33.33%
Translate	do not hold at the same time	to	~(g0 & g1)	66.67%
Translate	globally	to	G	66.67%
Translate	grant 0	to	g0	100%
Translate	grant 1	to	g1	100%

Final Result

G((!((g0 & g1)) U a)) 100.0%

Fig. 1. A screenshot of the web-interface for `nl2spec`.

add, edit, and delete these *sub-translations* instead of attempting to grapple with the entire formalization at once (a feature that is sorely missing in similar work, e.g., [13,30]).

Figure 1 shows the web-based frontend of `nl2spec`. As an example, we consider the following system requirement given in natural language: "Globally, grant 0 and grant 1 do not hold at the same time until it is allowed". The tool automatically translates the natural language specification correctly into the LTL formula `G((!((g0 & g1)) U a))`. Additionally, the tool generates sub-translations, such as the pair ("do not hold at the same time", `!(g0 & g1)`), which help in verifying the correctness of the translation.

Consider, however, the following ambiguous example: "a holds until b holds or always a holds". Human supervision is needed to resolve the ambiguity on the operator precedence. This can be easily achieved with `nl2spec` by adding or editing a sub-translation using explicit parenthesis (see Sect. 4 for more details and examples). To capture such (and other types of) ambiguity in a benchmark data set, we conducted an expert user study specifically asking for challenging translations of natural language sentences to LTL formulas.

The key insight in the design of `nl2spec` is that the process of translation can be decomposed into many sub-translations automatically via LLMs, and the decomposition into sub-translations allows users to easily resolve ambiguous natural language and erroneous translations through interactively modifying sub-translations. The central goal of `nl2spec` is to keep the human supervision minimal and efficient. To this end, all translations are accompanied by a confidence score. Alternative suggestions for sub-translations can be chosen via a drop-down menu and misleading sub-translations can be deleted before the next loop of the translation. We evaluate the end-to-end translation accuracy of our proposed methodology on the benchmark data set obtained from our expert user study. Note that `nl2spec` can be applied to the user's respective application domain to increase the quality of translation. As proof of concept, we provide additional examples, including an example for STL [31] in the GitHub repository[1].

`nl2spec` is agnostic to machine learning models and specific application domains. We will discuss possible parameterizations and inputs of the tool in Sect. 3. We discuss our sub-translation methodology in more detail in Sect. 3.2 and introduce an interactive few-shot prompting scheme for LLMs to generate them. We evaluate the effectiveness of the tool to resolve erroneous formalizations in Sect. 4 on a data set obtained from conducting an expert user study. We discuss limitations of the framework and conclude in Sect. 5. For additional details, please refer to the complete version [8].

2 Background and Related Work

2.1 Natural Language to Linear-Time Temporal Logic

Linear-time Temporal Logic (LTL) [34] is a temporal logic that forms the basis of many practical specification languages, such as the IEEE property specification language (PSL) [22], Signal Temporal Logic (STL) [31], or System Verilog Assertions (SVA) [43]. By focusing on the prototype temporal logic LTL, we keep the `nl2spec` framework extendable to specification languages in specific application domains. LTL extends propositional logic with temporal modalities $\mathtt{U}$ (until) and $\mathtt{X}$ (next). There are several derived operators, such as $\mathtt{F}\varphi \equiv \mathit{true}\mathtt{U}\varphi$ and $\mathtt{G}\varphi \equiv \neg\mathtt{F}\neg\varphi$. $\mathtt{F}\varphi$ states that φ will *eventually* hold in the future and $\mathtt{G}\varphi$ states that φ holds *globally*. Operators can be nested: $\mathtt{GF}\varphi$, for example, states that φ has to occur infinitely often. LTL specifications describe a systems behavior and its interaction with an environment over time. For example given a process 0 and a process 1 and a shared resource, the formula $\mathtt{G}(r_0 \rightarrow \mathtt{F}g_0) \wedge \mathtt{G}(r_1 \rightarrow \mathtt{F}g_1) \wedge \mathtt{G}\neg(g_0 \wedge g_1)$ describes that whenever a process requests (r_i) access to a shared resource it will eventually be granted (g_i). The subformula $\mathtt{G}\neg(g_0 \wedge g_1)$ ensures that grants given are mutually exclusive.

Early work in translating natural language to temporal logics focused on grammar-based approaches that could handle structured natural language [17, 24]. A survey of earlier research before the advent of deep learning is provided in [4]. Other approaches include an interactive method using SMT solving and semantic parsing [15], or structured temporal aspects in grounded robotics [45] and planning [32]. Neural networks have only recently been used to translate

[1] The tool is available at GitHub: https://github.com/realChrisHahn2/nl2spec.

into temporal logics, e.g., by training a model for STL from scratch [21], fine-tuning language models [19], or an approach to apply GPT-3 [13,30] in a one-shot fashion, where [13] output a restricted set of declare templates [33] that can be translated to a fragment of LTLf [10]. Translating natural langauge to LTL has especially been of interest to the robotics community (see [16] for an overview), where datasets and application domains are, in contrast to our setting, based on structured natural language. Independent of relying on structured data, all previous tools lack a detection and interactive resolving of the inerherent ambiguity of natural language, which is the main contribution of our framework. Related to our approach is recent work [26], where generated code is iteratively refined to match desired outcomes based on human feedback.

2.2 Large Language Models

LLMs are large neural networks typically consisting of up to 176 billion parameters. They are pre-trained on massive amounts of data, such as "The Pile" [14]. Examples of LLMs include the GPT [36] and BERT [11] model families, open-source models, such as T5 [38] and Bloom [39], or commercial models, such as Codex [5]. LLMs are Transformers [42], which is the state of the art neural architecture for natural language proccessing. Additionally, Transformers have shown remarkable performance when being applied to classical problems in verification (e.g., [9,18,25,40]), reasoning (e.g., [28,50]), as well as the auto-formalization [35] of mathematics and formal specifications (e.g., [19,21,49]).

In language modelling, we model the probability of a sequence of tokens in a text [41]. The joint probability of tokens in a text is generally expressed as [39]:

$$p(x) = p(x_1, \dots, x_T) = \prod_{t=1}^{T} p(x_t | x_{<t}) \ ,$$

where x is the sequence of tokens, x_t represents the t-th token, and $x_{<t}$ is the sequence of tokens preceding x_t. We refer to this as an autoregressive language model that iteratively predicts the probability of the next token. Neural network approaches to language modelling have superseded classical approaches, such as n-grams [41]. Especially Transformers [42] were shown to be the most effective architecture at the time of writing [1,23,36].

While fine-tuning neural models on a specific translation task remains a valid approach showing also initial success in generalizing to unstructured natural language when translating to LTL [19], a common technique to obtain high performance with limited amount of labeled data is so-called "few-shot prompting" [3]. The language model is presented a natural language description of the task usually accompanied with a few examples that demonstrate the input-output behavior. The framework presented in this paper relies on this technique. We describe the proposed few-shot prompting scheme in detail in Sect. 3.2.

Currently implemented in the framework and used in the expert-user study are Codex and Bloom, which showed the best performance during testing.

Codex and GPT-3.5-turbo. Codex [5] is a GPT-3 variant that was initially of up to 12B parameters in size and fine-tuned on code. The initial version of GPT-3 itself was trained on variations of Common Crawl,[2] Webtext-2 [37], two internet-based book corpora and Wikipedia [3]. The fine-tuning dataset for the vanilla version Codex was collected in May 2020 from 54 million public software repositories hosted on GitHub, using 159GB of training data for fine-tuning. For our experiments, we used the commercial 2022 version of `code-davinci-002`, which is likely larger (in the 176B range[3]) than the vanilla codex models. GPT-3.5-turbo is the currently available follow-up model of GPT-3.

Bloom. Bloom [39] is an open-source LLM family available in different sizes of up to 176B parameters trained on 46 natural languages and 13 programming languages. It was trained on the `ROOTS` corpus [27], a collection of 498 huggingface [29,48] datasets consisting of 1.61 terabytes of text. For our experiments, we used the 176B version running on the huggingface inference API[4].

3 The `nl2spec` Framework

3.1 Overview

The framework follows a standard frontend-backend implementation. Figure 2 shows an overview of the implementation of `nl2spec`. Parts of the framework that can be extended for further research or usage in practice are highlighted. The framework is implemented in Python 3 and flask [44], a lightweight WSGI web application framework. For the experiments in this paper, we use the OpenAI library and huggingface (transformer) library [47]. We parse the LTL output formulas with a standard LTL parser [12]. The tool can either be run as a command line tool, or with the web-based frontend.

The frontend handles the interaction with a human-in-the-loop. The interface is structured in three views: the "Prompt", "Sub-translations", and "Final Result" view (see Fig. 1). The tool takes a natural language sentence, optional sub-translations, the model temperature, and number of runs as input. It provides sub-translations, a confidence score, alternative sub-translations and the final formalization as output. The frontend then allows for interactively selecting, editing, deleting, or adding sub-translations. The backend implements the handling of the underlying neural models, the generation of the prompt, and the ambiguity resolving, i.e., computing the confidence score including alternative sub-translations and the interactive few-shot prompting algorithm (cf. Sect. 3.2). The framework is designed to have an easy interface to implement new models and write domain-specific prompts. The prompt is a .txt file that can be adjusted to specific domains to increase the quality of translations. To apply the sub-translation refinement methodology, however, the prompt needs to follow our interactive prompting scheme, which we introduce in the next section.

[2] https://commoncrawl.org/.
[3] https://blog.eleuther.ai/gpt3-model-sizes/.
[4] https://huggingface.co/inference-api.

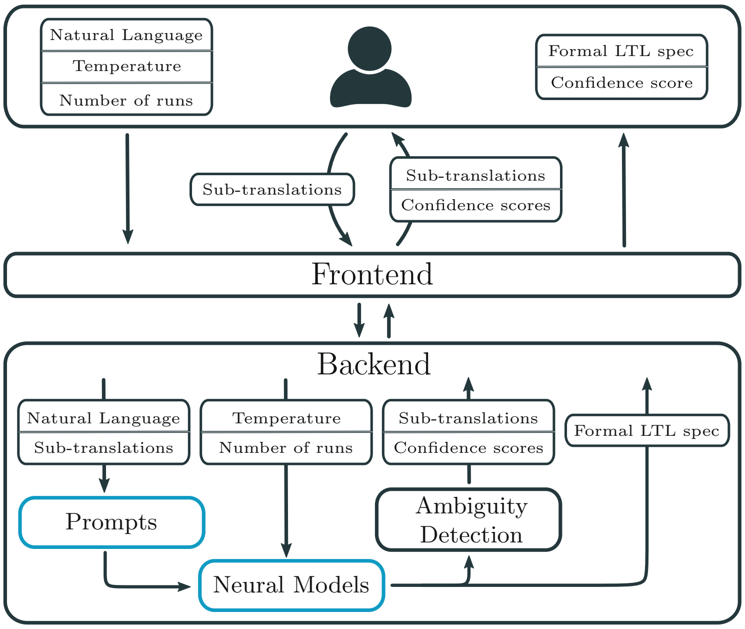


Fig. 2. Overview of the `nl2spec` framework with a human-in-the-loop: highlighted areas indicate parts of the framework that are effortlessly extendable.

3.2 Interactive Few-Shot Prompting

The core of the methodology is the decomposition of the natural language input into sub-translations. We introduce an interactive prompting scheme that generates sub-translations using the underlying neural model and leverages the sub-translations to produce the final translation. Algorithm 1 depicts a high-level overview of the interactive loop. The main idea is to give a human-in-the-loop the options to add, edit, or delete sub-translations and feed them back into the language models as "Given translations" in the prompt (see Fig. 3). After querying a language model M with this prompt F, model specific parameters P and the interactive prompt that is computed in the loop, the model generates a natural language explanation, a dictionary of sub-translations, and the final translation. Notably, the model M can be queried multiple times as specified by the number of runs r, thereby generating multiple possible sub-translations. The confidence score of each sub-translation is computed as votes over multiple queries and by default the sub-translation with the highest confidence score is selected to be used as a given sub-translation in the next iteration. In the frontend, the user may view and select alternative generated sub-translations for each sub-translation via a drop-down menu (see Fig. 1).

Figure 3 shows a generic prompt, that illustrates our methodology. The prompting scheme consists of three parts. The specification language specific part (lines 1–4), the fewshot examples (lines 5–19), and the interactive prompt minimal.txt

```
1  Translate the following natural language sentences into an LTL formula and explain your
2  translation step by step. Remember that X means "next", U means "until", G means
3  "globally", F means "finally", which means GF means "infinitely often". The formula
4  should only contain atomic propositions or operators  &, ~, ->, <->, X, U, G, F.
5  Natural Language: Globally if a holds then c is true until b. Given translations: {}
6  Explanation: "a holds" from the input translates to the atomic proposition a.
7  "c is true until b" from the input translates to the subformula c U b. "if x then y"
8  translates to an implication x -> y, so "if a holds then c is true until b" translates
9  to an implication a -> c U b. "Globally" from the input translates to the temporal
10 operator G. Explanation dictionary: {"a holds" : "a", "c is true until b" : "c U b",
11 "if a holds then c is true until b" : "a -> c U b", "Globally" : "G"} So the final
12 LTL translation is G a -> c U b.FINISH Natural Language: Every request r is
13 eventually followed by a grant g. Given translations: {} Explanation: "Request r"
14 from the input translates to the atomic proposition r and "grant g" translates to the
15 atomic proposition g. "every" means at every point in time, i.e., globally, "never"
16 means at no point in time, and "eventually" translates to the temporal operator F.
17 "followed by" is the natural language representation of an implication. Explanation
18 dictionary: {"Request r" : "r", "grant g" : "g", "every" : "G", "eventually": "F",
19 "followed by" : "->"} So the final LTL translation is G r -> F g.FINISH
```

Fig. 3. Prompt with minimal domain knowledge of LTL.

including the natural language and sub-translation inputs (not displayed, given as input). The specification language specific part leverages "chain-of-thought" prompt-engineering to elicit reasoning from large language models [46]. The key of `nl2spec`, however, is the setup of the few-shot examples. This minimal prompt consists of two few-shot examples (lines 5–12 and 12–19). The end of an example is indicated by the "FINISH" token, which is the stop token for the machine learning models. A few-shot example in `nl2spec` consists of the natural language input (line 5), a dictionary of given translations, i.e., the sub-translations (line 5), an explanation of the translation in natural language (line 6–10), an explanation dictionary, summarizing the sub-translations, and finally, the final LTL formula.

This prompting scheme elicits sub-translations from the model, which serve as a fine-grained explanation of the formalization. Note that sub-translations provided in the prompt are neither unique nor exhaustive, but provide the context for the language model to generate the correct formalization.

4 Evaluation

In this section, we evaluate our framework and prompting methodology on a data set obtained by conducting an expert user study. To show the general applicability of this framework, we use the `minimal` prompt that includes only minimal domain knowledge of the specification language (see Fig. 3). This prompt has intentionally been written *before* conducting the expert user study. We limited the few-shot examples to two and even provided no few-shot example that includes "given translations". We use the minimal prompt to focus the evaluation on the effectiveness of our interactive sub-translation refinement methodology in

Algorithm 1: Interactive Few-shot Prompting Algorithm

```
1 Input: Natural language $S$, Few-shot prompt $F$, set of given sub-translations
  $(s, \varphi)$, and language model $M$
2 Interactions: set of sub-translations $(s, \varphi)$, confidence scores $C$
3 Set of Model specific parameter $P$: e.g., model-temperature $t$, number of
  runs $r$
4 Output: LTL formula $\psi$ that formalizes $S$
   1: $\psi$, $(s, \varphi)$ , $C$ = empty
   2: while user not approves LTL formula $\psi$ do
   3:     interactive_prompt = compute_prompt(S, F, $(s, \varphi)$)
   4:     $\psi$, $(s, \varphi)$ , $C$ = query(M, P, interactive_prompt)
   5:     $(s, \varphi)$ = user_interaction($(s, \varphi)$ , $C$)
   6: end while
   7: return $\psi$
```

resolving ambiguity and fixing erroneous translations. In practice, one would like to replace this minimal prompt with domain-specific examples that capture the underlying distribution as closely as possible. As a proof of concept, we elaborate on this in the full version [8].

4.1 Study Setup

To obtain a benchmark dataset of *unstructured* natural language and their formalizations into LTL, we asked five experts in the field to provide examples that the experts thought are challenging for a neural translation approach. Unlike existing datasets that follow strict grammatical and syntatical structure, we posed no such restrictions on the study participants. Each natural language specification was restricted to one sentence and to five atomic propositions a, b, c, d, e. Note that `nl2spec` is not restricted to a specific set of atomic propositions (cf. Fig. 1). Which variable scheme to use can be specified as an initial sub-translation. We elaborate on this in the full version [8]. To ensure unique instances, the experts worked in a shared document, resulting in 36 benchmark instances. We provide three randomly drawn examples for the interested reader:

natural language S	LTL specification ψ
If b holds then, in the next step, c holds until a holds or always c holds	`b -> X ((c U a) || G c)`
If b holds at some point, a has to hold somewhere beforehand	`(F b) -> (!b U (a & !b))`
One of the following aps will hold at all instances: a,b,c	`G( a | b | c)`

The poor performance of existing methods (cf. Table 1) exemplify the difficulty of this data set.

4.2 Results

We evaluated our approach using the `minimal` prompt (if not otherwise stated), with number of runs set to three and with a temperature of 0.2.

Quality of Initial Translation. We analyze the quality of *initial* translations, i.e., translations obtained *before* any human interaction. This experiment demonstrates that the initial translations are of high quality, which is important to ensure an efficient workflow. We compared our approach to fine-tuning language models on structured data [19] and to an approach using GPT-3 or Rasa [2] to translate natural language into a restricted set of declare patterns [13] (which could not handle most of the instances in the benchmark data set, even when replacing the atomic propositions with their used entities). The results of evaluating the accuracy of the initial translations on our benchmark expert set is shown in Table 1.

At the time of writing, using Codex in the backend outperforms GPT-3.5-turbo and Bloom on this task, by correctly translating 44.4% of the instances using the minimal prompt. We only count an instance as correctly translated if it matches the intended meaning of the expert, no alternative translation to ambiguous input was accepted. Additionally to the experiments using the minimal prompt, we conducted experiments on an augmented prompt with in-distribution examples after the user study was conducted by randomly drawing four examples from the expert data set (3 of the examples haven't been solved before, see the GitHub repository or full version for more details). With this in-distribution prompt (ID), the tool translates 21 instances (with the four drawn examples remaining in the set), i.e., 58.3% correctly.

This experiment shows 1) that the initial translation quality is high and can handle unstructured natural language better than previous approaches and 2) that drawing the few-shot examples in distribution only slightly increased translation quality for this data set; making the key contributions of `nl2spec`, i.e., ambiguity detection and effortless debugging of erroneous formalizations, valuable. Since `nl2spec` is agnostic to the underlying machine learning models, we expect an even better performance in the future with more fine-tuned models.

Teacher-Student Experiment. In this experiment, we generate an initial set of sub-translations with Codex as the underlying neural model. We then ran the tool with Bloom as a backend, taking these sub-translations as input. There were 11 instances that Codex could solve initially that Bloom was unable to solve. On these instances, Bloom was able to solve 4 more instances, i.e., 36.4% with sub-translations provided by Codex. The four instances that Bloom was able to solve

Table 1. Translation accuracy on the benchmark data set, where B stands for Bloom and C stands for Codex and G for GPT-3.5-Turbo.

nl2ltl [13] `rasa`	T-5 [19] `fine-tuned`	`nl2spec`+B `initial`	`nl2spec`+C `initial`	`nl2spec`+C `initial+ID`	`nl2spec`+C `interactive`
1/36 (2.7%)	2/36 (5.5%)	5/36 (13.8%)	16/36 (44.4%)	21/36 (58.3%)	31/36 (86.1%)
–	–	–	`nl2spec`+G `initial`	`nl2spec`+G `initial+ID`	`nl2spec`+G `interactive`
–	–	–	12/36 (33.3%)	17/36 (47.2%)	21/36 (58.3%)

with the help of Codex were: “It is never the case that a and b hold at the same time.”, “Whenever a is enabled, b is enabled three steps later.”, “If it is the case that every a is eventually followed by a b, then c needs to holds infinitely often.”, and “One of the following aps will hold at all instances: a,b,c”. This demonstrates that our sub-translation methodology is a valid appraoch: improving the quality of the sub-translations indeed has a positive effect on the quality of the final formalization. This even holds true when using underperforming neural network models. Note that no supervision by a human was needed in this experiment to improve the formalization quality.

Ambiguity Detection. Out of the 36 instances in the benchmark set, at least 9 of the instances contain ambiguous natural language. We especially observed two classes of ambiguity: 1) ambiguity due to the limits of natural language, e.g., operator precedence, and 2) ambiguity in the semantics of natural language; `nl2spec` can help in resolving both types of ambiguity. Details for the following examples can be found in the full version [8].

An example for the first type of ambiguity from our dataset is the example mentioned in the introduction: “a holds until b holds or always a holds”, which the expert translated into `(a U b) | G a`. Running the tool, however, translated this example into `(a U (b | G(a)))`. By editting the sub-translation of “a holds until b holds” to `(a U b)` through adding explicit parenthesis, the tool translates as intended. An example for the second type of ambiguity is the following instance from our data set: “Whenever a holds, b must hold in the next two steps.” The intended meaning of the expert was `G (a -> (b | X b))`, whereas the tool translated this sentence into `G((a -> X(X(b))))`. After changing the sub-translation of “b must hold in the next two steps” to `b | X b`, the tool translates the input as intended.

Fixing Erroneous Translation. With the inherent ambiguity of natural language and the unstructured nature of the input, the tool’s translation cannot be expected to be always correct in the first try. Verifying and debugging sub-translations, however, is significantly easier than redrafting the complete formula from scratch. Twenty instances of the data set were not correctly translated in an initial attempt using Codex and the minimal prompt in the backend (see Table 1). We were able to extract correct translations for 15 instances by performing at most three translation loops (i.e., adding, editing, and removing sub-translations), We were able to get correct results by performing 1.86 translation loops on average. For example, consider the instance, “whenever a holds, b holds as well”, which the tool mistakenly translated to `G(a & b)`. By fixing the sub-translation “b holds as well” to the formula fragment `-> b`, the sentence is translated as intended. Only the remaining five instances that contain highly complex natural language requirements, such as, “once a happened, b won’t happen again” were need to be translated by hand.

In total, we correctly translated 31 out of 36 instances, i.e., 86.11% using the `nl2spec` sub-translation methodology by performing only 1.4 translation loops on average (see Table 1).

5 Conclusion

We presented nl2spec, a framework for translating unstructured natural language to temporal logics. A limitation of this approach is its reliance on computational resources at inference time. This is a general limitation when applying deep learning techniques. Both, commercial and open-source models, however, provide easily accessible APIs to their models. Additionally, the quality of initial translations might be influenced by the amount of training data on logics, code, or math that the underlying neural models have seen during pre-training.

At the core of nl2spec lies a methodology to decompose the natural language input into sub-translations, which are mappings of formula fragments to relevant parts of the natural language input. We introduced an interactive prompting scheme that queries LLMs for sub-translations, and implemented an interface for users to interactively add, edit, and delete the sub-translations, which avoids users from manually redrafting the entire formalization to fix erroneous translations. We conducted a user study, showing that nl2spec can be efficiently used to interactively formalize unstructured and ambigous natural language.

Acknowledgements. We thank OpenAI for providing academic access to Codex and Clark Barrett for helpful feedback on an earlier version of the tool.

References

1. Al-Rfou, R., Choe, D., Constant, N., Guo, M., Jones, L.: Character-level language modeling with deeper self-attention. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 3159–3166 (2019)
2. Bocklisch, T., Faulkner, J., Pawlowski, N., Nichol, A.: Rasa: open source language understanding and dialogue management. arXiv preprint arXiv:1712.05181 (2017)
3. Brown, T., et al.: Language models are few-shot learners. Adv. Neural Inf. Process. Syst. **33**, 1877–1901 (2020)
4. Brunello, A., Montanari, A., Reynolds, M.: Synthesis of ltl formulas from natural language texts: state of the art and research directions. In: 26th International Symposium on Temporal Representation and Reasoning (TIME 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019)
5. Chen, M., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
6. Church, A.: Application of recursive arithmetic to the problem of circuit synthesis. J. Symb. Logic **28**(4) (1963)
7. Clarke, E.M.: Model checking. In: Ramesh, S., Sivakumar, G. (eds.) FSTTCS 1997. LNCS, vol. 1346, pp. 54–56. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0058022
8. Cosler, M., Hahn, C., Mendoza, D., Schmitt, F., Trippel, C.: nl2spec: interactively translating unstructured natural language to temporal logics with large language models. arXiv preprint arXiv:2303.04864 (2023)
9. Cosler, M., Schmitt, F., Hahn, C., Finkbeiner, B.: Iterative circuit repair against formal specifications. In: International Conference on Learning Representations (to appear) (2023)

10. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI 2013 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, pp. 854–860. Association for Computing Machinery (2013)
11. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
12. Fuggitti, F.: LTLf2DFA. Zenodo (2019). https://doi.org/10.5281/ZENODO.3888410, https://zenodo.org/record/3888410
13. Fuggitti, F., Chakraborti, T.: Nl2ltl-a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas (2023)
14. Gao, L., et al.: The pile: an 800 gb dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027 (2020)
15. Gavran, I., Darulova, E., Majumdar, R.: Interactive synthesis of temporal specifications from examples and natural language. Proc. ACM Program. Lang. **4**(OOPSLA), 1–26 (2020)
16. Gopalan, N., Arumugam, D., Wong, L.L., Tellex, S.: Sequence-to-sequence language grounding of non-markovian task specifications. In: Robotics: Science and Systems, vol. 2018 (2018)
17. Grunske, L.: Specification patterns for probabilistic quality properties. In: 2008 ACM/IEEE 30th International Conference on Software Engineering, pp. 31–40. IEEE (2008)
18. Hahn, C., Schmitt, F., Kreber, J.U., Rabe, M.N., Finkbeiner, B.: Teaching temporal logics to neural networks. In: International Conference on Learning Representations (2021)
19. Hahn, C., Schmitt, F., Tillman, J.J., Metzger, N., Siber, J., Finkbeiner, B.: Formal specifications from natural language. arXiv preprint arXiv:2206.01962 (2022)
20. Havelund, K., Roşu, G.: Monitoring java programs with java pathexplorer. Electron. Notes Theor. Comput. Sci. **55**(2), 200–217 (2001)
21. He, J., Bartocci, E., Ničković, D., Isakovic, H., Grosu, R.: Deepstl: from english requirements to signal temporal logic. In: Proceedings of the 44th International Conference on Software Engineering, pp. 610–622 (2022)
22. IEEE-Commission, et al.: IEEE standard for property specification language (PSL). IEEE Std 1850–2005 (2005)
23. Kaplan, J., et al.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020)
24. Konrad, S., Cheng, B.H.: Real-time specification patterns. In: Proceedings of the 27th International Conference on Software Engineering, pp. 372–381 (2005)
25. Kreber, J.U., Hahn, C.: Generating symbolic reasoning problems with transformer gans. arXiv preprint arXiv:2110.10054 (2021)
26. Lahiri, S.K., et al.: Interactive code generation via test-driven user-intent formalization. arXiv preprint arXiv:2208.05950 (2022)
27. Laurençon, H., et al.: The bigscience roots corpus: a 1.6 tb composite multilingual dataset. In: Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2022)
28. Lewkowycz, A., et al.: Solving quantitative reasoning problems with language models. arXiv preprint arXiv:2206.14858 (2022)
29. Lhoest, Q., et al.: Datasets: a community library for natural language processing. arXiv preprint arXiv:2109.02846 (2021)

30. Liu, J.X., et al.: Lang2ltl: translating natural language commands to temporal specification with large language models. In: Workshop on Language and Robotics at CoRL 2022
31. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
32. Patel, R., Pavlick, R., Tellex, S.: Learning to ground language to temporal logical form. In: NAACL (2019)
33. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006). https://doi.org/10.1007/11837862_18
34. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), pp. 46–57. IEEE (1977)
35. Rabe, M.N., Szegedy, C.: Towards the automatic mathematician. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 25–37. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_2
36. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
37. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
38. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. **21**(140), 1–67 (2020). http://jmlr.org/papers/v21/20-074.html
39. Scao, T.L., et al.: Bloom: a 176b-parameter open-access multilingual language model. arXiv preprint arXiv:2211.05100 (2022)
40. Schmitt, F., Hahn, C., Rabe, M.N., Finkbeiner, B.: Neural circuit synthesis from specification patterns. Adv. Neural Inf. Process. Syst. **34**, 15408–15420 (2021)
41. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948)
42. Vaswani, A., et al.: Attention is all you need. Adv. Neural Inf. Process. Syst. **30** (2017)
43. Vijayaraghavan, S., Ramanathan, M.: A Practical Guide for SystemVerilog Assertions. Springer, Heidelberg (2005). https://doi.org/10.1007/b137011
44. Vyshnavi, V.R., Malik, A.: Efficient way of web development using python and flask. Int. J. Recent Res. Asp **6**(2), 16–19 (2019)
45. Wang, C., Ross, C., Kuo, Y.L., Katz, B., Barbu, A.: Learning a natural-language to ltl executable semantic parser for grounded robotics. arXiv preprint arXiv:2008.03277 (2020)
46. Wei, J., et al.: Chain of thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903 (2022)
47. Wolf, T., et al.: Huggingface's transformers: state-of-the-art natural language processing. arXiv preprint arXiv:1910.03771 (2019)
48. Wolf, T., et al.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45 (2020)
49. Wu, Y., et al.: Autoformalization with large language models. arXiv preprint arXiv:2205.12615 (2022)
50. Zelikman, E., Wu, Y., Goodman, N.D.: Star: bootstrapping reasoning with reasoning. arXiv preprint arXiv:2203.14465 (2022)