Improving Translation of English Instructions With Sequential Constraints to Linear Temporal Logic in Navigation Tasks via Prompt Engineering

Sam Liang

Advisor: Karthik Narasimhan

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in the Engineering

Department of Computer Science

Princeton University

May 2023

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

_____

Sam Liang

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Sam Liang

Abstract

Autonomous agents like self-driving cars, robots, and drones are becoming more relevant and prevalent in society. It is an immediate advancement of technology with many benefits to society because of how applicable it is. My goal is to enable autonomous agents to be more accessible and widely deployed across cities, focusing on robots executing navigation tasks in cities. I do this by researching methods to improve translation accuracy from English instructions with sequential constraints to Linear Temporal Logic (LTL) and improve generalizability in this translation domain. I take an LSTM-based approach and a large language model approach, focusing on prompt engineering in the latter.

Acknowledgements

I would like to thank my advisor Karthik Narasimhan, Xiaoyan Li, Alexander Wettig, Ziyi Yang, and the Princeton BSE Engineering Deparment for their support.

Contents

Chapter 1

Introduction

In navigation environments like cities, autonomous agents must understand natural language instructions and execute them correctly. The most prominent modern example is self-driving cars. The natural language instructions usually contain sequential requirements that the agent must meet in order to execute the instruction correctly. For example, the agent has to visit landmark H before visiting landmark A. Visiting them in the reverse order would be incorrect. In addition to correct execution of instructions, the agent must also be capable of operating in different cities. It must take what it's learned and generalize it across many different cities without needing to relearn how to operate in a new city. Not only does this mimic how humans learn which is a step towards general artificial intelligence, but it reduces the deployment cost and complexity of these agents.

In this paper, I present two approaches towards enabling agents to understand and execute natural language instructions. Both approaches utilize a formal language called linear temporal logic (LTL) [17] that an autonomous agent (a robotic system) can understand. It encodes the meaning and the sequential constraints of the instruction in its specification. The first approach uses a LSTM-based model to translate natural language instructions to linear temporal logic. The model was

trained via semi-supervised learning with a reinforcement learning objective. After this preliminary approach, I transitioned to a large language model framework, shown in Figure 1.1, to leverage the strong language capabilities and ability to generalize of large language models [3]. The framework consists of a referring expression recognition module, a proposition resolution module, and a translation module that allow for better performance and generalization across different cities. The green blocks are large language models and can be any large language model. I use OpenAI's 175-billion parameter GPT-3 model [3] and my research focuses entirely on tying to improve GPT-3's performance in the translation module by using its in-context learning abilities. In-context learning allows GPT-3 to achieve competitive performance on many downstream tasks without the need for finetuning on each downstream task after pretraining [3]. It only requires a prompt (usually a few pairs of task questions and answers) to do so. This saves computation time and cost and is especially useful in downstream tasks with little to no high-quality training data. I implement different methods of modifying the prompt and evaluate them by their affects on GPT-3's accuracy.

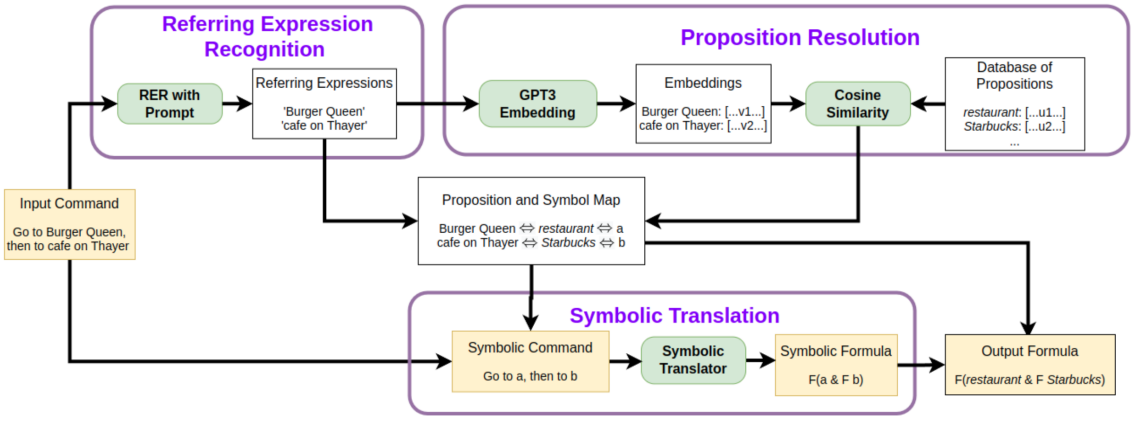

Figure 1.1: The complete framework for translating English instructions to LTL consisting of a referring expression recognition module, a proposition resolution module, and a translation module. The green blocks are the large language models.

Chapter 2

Related Work

Mapping Natural Language Sentences to Lambda-Calculus. Zettlemoyer and Collins [27] introduces an algorithm that maps natural language sentences to lambda-calculus through a grammar and probabilistic model. An example mapping would be:

Sentence: What states border Texas?

Logical form: $\lambda x.state(x) \wedge borders(x, texas)$

The algorithm learns from a training dataset of sentence-logical form pairs. However, they operate in a setting where there are not complete annotations of each sentence to its respective logical form. Therefore, their algorithm learns a probabilistic combinatory categorial grammar (PCCG) from this dataset and uses that to map test sentences to the most likely logical form. Because PCCGs only contain syntactic categories, they extend PCCGs to include semantic categories because the lambda-calculus form must encode the meaning of the sentence.

Generating Interfaces to Relational Databases. Tang and Mooney [20] pro-

poses a method to learn a natural language interface that combines both statistical and relational learning models to translate natural language queries into a first-order logic that can be used to query a database for the answer. The statistical model uses a probabilistic context-free grammar (PCFG) to parse natural language queries into a logical form. The PCFG is used to find the most likely logical form given a sentence and is learned through a training dataset. The relational learning model builds a structured representation of the database which is used by the statistical model to help it generate the logical form.

Natural Language to Linear Temporal Logic. Gopalan et al. [7] presents a similar approach of translating non-Markovian task specifications to action sequences that a robot executes via a sequence-to-sequence model. They use natural language to capture task specifications with temporal behaviors and use geometric linear temporal logic as an encoding for the action sequences. Geometric linear temporal logic is a probabilistic variant of linear temporal logic and can be passed into a MDP planner that outputs the action sequence corresponding to the input geometric linear temporal logic. They use a Gated Recurrent Unit encoder-decoder model trained on a dataset of natural language commands paired with the correct geometric linear temporal logic expression.

Wang et al. [24] develops a framework for parsing natural language instructions into any formal logic inspired by how children learn, but they focus on linear temporal logic. They use a parser consisting of an LSTM encoder and two LSTM decoders weakly supervised on a dataset to learn parameters. The dataset consists of natural language instructions paired with execution traces of the robot carrying out the command in the environment. The natural language instructions are not annotated directly with the correct linear temporal logic, so supervised learning cannot occur, but this is advantageous in real-world settings where such datasets rarely exist. The

encoder maps the input sentence into some representation which is passed to the parser. The parser generates a linear temporal logic formula which is then scored by the planner on how well it captures the intended behaviors and if there are more efficient executions. Finally, they have a generator model that attempts to reconstruct the input sentence from the encoder output to try to create representations that retain the meaning of the sentence.

Generalizing Natural Language to Linear Temporal Logic Translation. Previously described semantic parsers need to be trained on data collected from the exact environment (e.g. a city) that it will operate in for it to work. To operate in novel environments, it would need to be retrained on them. Berg et al. [1] uses CopyNet [9], a sequence-to-sequence model, that is trained to generate linear temporal logic like other seq-2-seq models but can also copy words from the input to the output. This generalizes the model because the landmarks are contained in the input sentence. The semantics of the sentence remains similar to those in the training data and only the landmarks differ. In addition, they have a landmark resolution model that maps the landmarks referred to by the user to real landmarks in the environment, modifying CopyNet's output with the real landmarks before being passed to the planner.

Chapter 3

Background

3.1 Linear Temporal Logic

Linear temporal logic (LTL) is used to capture the semantics of natural language instructions as well as the temporal constraints within them. It is similar to Boolean logic except the propositions must hold for a duration of time or at some specific time. An LTL formula ϕ is either true or false and has the following operators and syntax:

$$\phi \coloneqq \varphi|\neg\phi|\phi_1 \vee \phi_2|\phi_1 \wedge \phi_2|X\phi|\phi_2 U \phi_1|G\phi|F\phi$$

φ is an atomic proposition that evaluates to true or false. ϕ is any valid LTL formula. $\neg$ is the not operator, $\vee$ is the or operator, and $\wedge$ is the and operator. They have the same meaning and function as in Boolean logic. X is the next temporal operator. $X\phi$ is true if ϕ is true at the next timestep. U is the until temporal operator. $\phi_2\ U\ \phi_1$ is true only if ϕ_1 is true after or while ϕ_2 is true. G is the always temporal operator. $G\phi$ is true if ϕ is always true at all timesteps. F is the eventually temporal operator. $F\phi$ is true if ϕ is true at least once during some timestep in the future.

3.2 Sequence-to-Sequence Model

In a sequential modeling problem, the input comes in as a sequence with some time or positional component to it. Furthermore, the input at some time t depends on all the prior inputs. For example, predicting the next word in a sentence is a sequential modeling problem as each word has an order and the next word depends on the context or the previous words. To capture this relationship, early neural networks like recurrent neural networks (RNNs) used an architecture that stored history information. At each input timestep or position, the output is passed on to the next timestep or position and used for prediction. This is shown in Figure 3.1.

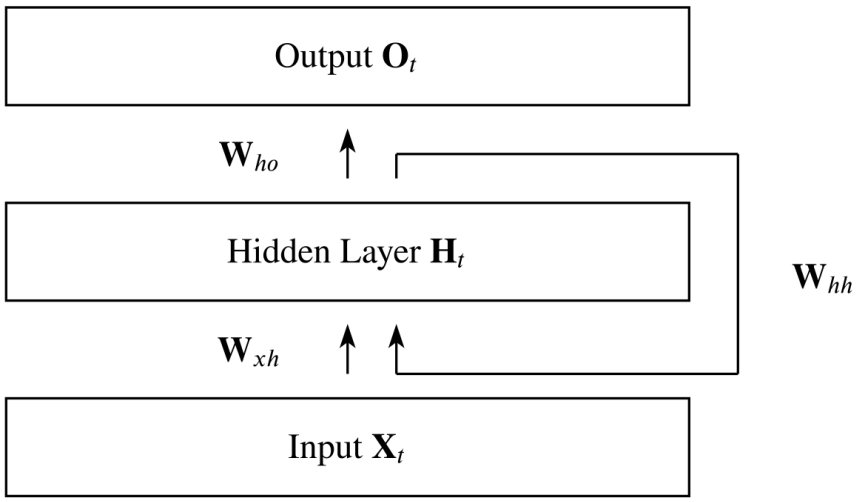

Figure 3.1: Recurrent Neural Network [18]

However, RNNs suffers from the vanishing gradient problem in which when we calculate the gradient with respect to some early hidden layer, because the gradient is backpropagated through a long sequence, it becomes extremely small when there are many small values along the way. The RNN parameters are then not learning

the relationship of early hidden states to the current hidden state [18]. LSTMs were proposed to solve the vanishing gradient problem. They store more history and memory in a cell state. Three gates control what is added or removed to the cell state: the output, hidden, and forget gates. See Figure 3.2 for a visual depiction.

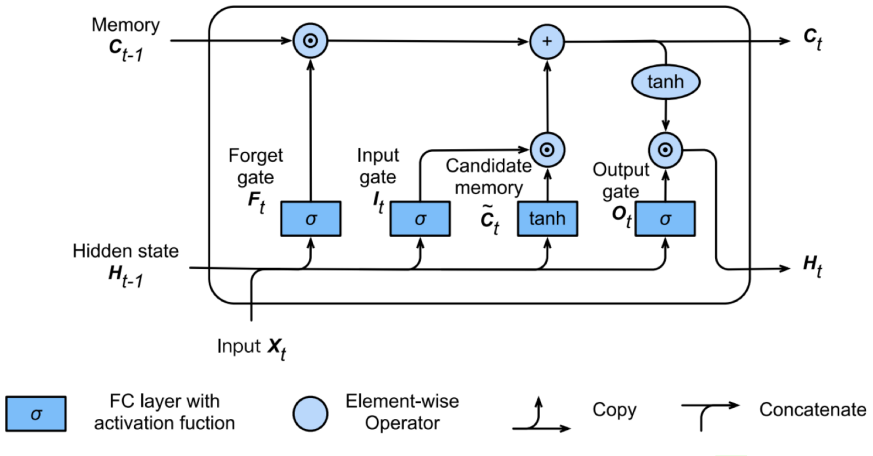

Figure 3.2: LSTM Cell [18]

In this paper, I code a LSTM-based sequence-to-sequence model to translate natural language instructions to LTL.

3.3 Large Language Models

Large language models are models with billions of parameters, or more, trained on an extremely large text corpus. They are generative models that output tokens (words) conditioned on previous tokens. Large language models are very successful because of their high accuracies on many downstream tasks when finetuned or given a prompt. They can achieve this performance because they exhibit the scaling law (more parameters leads to higher performance) and are trained on huge amounts of data. Higher

quality data also leads to better performance.

Large language models are scalable because they use the transformer architecture [23]. The transformer consists of multiple stacked encoder and decoder layers where the output of the last encoder layer is fed into every decoder layer. Each encoder layer contains a multi-head self attention sublayer and a feed forward sublayer. The multi-head self-attention layer is the key innovation of the transformer. It contains multiple self-attention heads. Each self-attention head acts as a way for the transformer to learn which parts of the sentence to pay attention to with respect to the current token. By having multiple of them, each self-attention layer can learn to focus on separate things in the sequence. For the ith token, it's respective output in the self-attention layer z_i is:

$$z_i = \sum_{j=1}^{n} \texttt{softmax}(\{\langle q_i, k_{j'} \rangle\}_{j'=1}^{n})_j \cdot v_j.$$

Figure 3.3: Self Attention Calculation [23]

Each decoder layer contains a multi-head self-attention sublayer, a feed forward sublayer, and a masked multi-head self-attention sublayer. The masked multi-head self attention sublayer is the same as the multi-head self attention sublayer, except it masks out the future tokens so that the model can perform autoregressive prediction because the model should not see what it is trying to predict. For a more detailed explanation, refer to the transformer paper [23].

After creating the LSTM-based sequence-to-sequence model, I transitioned to using OpenAI's 175-billion parameter GPT-3 [3] model. It is the current state-of-the-art

large language model that exhibits all the characteristics that I have described above. In addition, it also has in-context learning abilities. In-context learning allows GPT-3 to achieve competitive performance on many downstream tasks without the need for finetuning on each downstream task after pretraining. It only requires a prompt (usually a few pairs of task questions and answers) to do so. This saves computation time and cost and is especially useful in downstream tasks with little to no high-quality training data which is our setting. I focus on improving GPT-3's in-context learning performance through prompt engineering. I investigate different methods of modifying the prompt and seeing if that improves GPT-3's accuracy.

Chapter 4

Method

4.1 Problem Definition

I frame the problem in the setting of a user issuing an English instruction to an agent whose job is to execute this instruction correctly in a navigation environment. For example, the user might say "Go to Starbucks and then visit CVS pharmacy" and the agent must correctly complete this task. In particular, I focus on the subproblem of the agent understanding the natural language instruction by translating it to LTL. To tackle this translation problem, I propose two methods: using a LSTM-based sequence-to-sequence model and using prompting techniques with a large language model.

4.2 Dataset Creation

In the LSTM-based approach, a dataset of English instructions paired with trajectories was used. Trajectories are a sequence of landmarks in the instruction, appearing in the order that they need to be traversed. The reason this dataset was used was because initially, high-quality datasets of instructions paired with LTL did not exist and creating such a dataset is time-consuming. Therefore, we opted to use a dataset

of instructions paired with trajectories where trajectories could be easily acquired via Amazon Turks. The dataset was around 2500 datapoints.

In the large language model approach, a dataset of English instructions paired with the correct LTL's is required, either for training and/or performance evaluation. As mentioned, existing datasets do not suffice because they are low quality. They lack diversity in English instructions. Therefore, I, alongside others, created our own dataset. We created a dataset of 1000 datapoints covering fifteen types of LTL formula structures as defined by Menghi [16] with modifications. See Table 4.1 and Table D.1 for a detailed explanation of each LTL type and examples. We chose these LTL types because they fit to our navigational setting. For each LTL type, we each manually wrote at least one English instruction per number of landmarks to correspond to that LTL type. Each instruction contains 1, 2, 3, 4, or 5 landmarks. An LTL formula instance is defined by its type and its number of landmarks. There were 48 unique LTL formulas in our dataest. We ensured there was diversity in our language usage so that each LTL formula in the dataset has unique instructions corresponding to it. We also made sure our dataset was balanced. Each LTL formula has 24 unique instructions corresponding to it. Here is an example of a datapoint in our dataset:

```
English instruction: In some sequence, visit a, b, and c
                     in that order.

LTL: F & a F & b F c
```

where a, b, and c are landmarks and $\mathcal{F}$ is the eventually LTL operator. See Table 4.1 and Table D.1 for general examples of all the LTL formulas in the dataset. Note that compared to tables, in our dataset, we use & to represent $\wedge$ and ! to represent $\neg$. We also store our LTL formulas in prefix notation like the example above instead of the regular infix notation like in the tables because it reduces output lengths by

not needing any parentheses. However, I discovered that in the case of prompting, it reduces performance.

For my prompt engineering research, I only use the Core Movement Patterns in Table 4.1 because training feasibility due to available resources like GPUs and monetary restrictions due to cost of using the GPT-3 API. I also did not use fair visit data points because the instructions collected for it were mostly incorrect and of low quality. This resulted in a dataset with 550 datapoints with 22 unique LTL formulas.

4.3 Evaluation Method

To evaluate the performance of the symbolic translation component with each prompting technique, I create three holdout tests with each increasing in generalization difficulty and use accuracy as the evaluation metric. Accuracy is a fine metric here because the dataset is balanced as there is an even number of unique instructions per LTL formula.

- **Instruction Holdout**: Each LTL formula's corresponding instructions are randomly assigned to either a training set and a test set. The key idea here governing the split is that the training set will contain instructions such that all formula instances are in the training set. A formula instance is defined by its number of landmarks and LTL type. The model, either through the prompt or during the training phase, will have seen all formula instances, but it will not have seen the specific instructions utilized as input during the testing phase. Prior research on translating natural language instructions into LTL formulas for robot operations [8, 2] have concentrated on instruction holdout accuracy. Instruction holdout is generally the easiest test. I create two instruction holdout test sets using seeds 123 and 484. Each contained 475 datapoints. I provide the average instruction holdout accuracy on these test sets.

LTL Type	Definition	Instruction Example	LTL Formula (l_i is a landmark)
Visit	Visit a set of landmarks in any order.	Go to landmarks a, b, and c.	$\bigwedge_{i=1}^{n} \mathcal{F}(l_i)$
Sequenced Visit	Visit a set of landmarks in a sequence, one after the other.	Visit landmark a. Sometime afterwards, visit landmark b. Then, visit landmark c.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge ...\mathcal{F}(l_n)))$
Ordered Visit	Sequenced visit does not forbid visiting a later landmark in the sequence before visiting an earlier landmark in the sequence. Ordered visit is the same as sequenced visit but it forbids this behavior.	Do not visit landmark c until visiting landmark b. Do not visit landmark b until visiting landmark a.	$\left(\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \,\mathsf{U}\, l_i\right) \wedge \mathcal{F}(l_i)$
Strict Ordered Visit	Ordered visit allows any landmarks before the current landmark that is to be visited next to be visited multiple times. Strict ordered visit is the same as ordered visit but forbids this behavior.	Do not visit landmark c until you visit landmark b. Do no visit landmark b until you visit landmark a. Visit landmarks a and b only once.	$\left(\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \,\mathsf{U}\, l_i\right) \wedge \mathcal{F}(l_i)$ $\bigwedge_{i=1}^{n-1} (\neg l_i \,\mathsf{U}\, (l_i \,\mathsf{U}\, (\neg l_i \,\mathsf{U}\, l_{i+1})))$
Fair Visit	For any two landmarks in the set, the difference between the number of times the two landmarks are visited is at most one.	Visit landmarks a and b but the number of times you visit landmark a can never exceed the number of times you visit landmark b by more than one.	Omitted because fair visit is not used.
Patrolling	Continue indefinitely visiting a set of landmarks in no particular order	Visit landmarks a, b, and c forever	$\bigwedge_{i=1}^{n} \mathsf{G}\mathcal{F}(l_i)$

Table 4.1: This table shows the different LTL types falling under core movement patterns with an instruction example and definition. Each LTL type has a specific formula that differs only based on the number of propositions (landmarks) but has the same structure. $\mathcal{F}$ is the eventually LTL operator. G is the globally operator. U is the until operator. Adapted from [16].

- **Formula Holdout**: In formula holdout, none of the formula instances tested on have been exposed to the model during training or through the prompts. However, the model can encounter a formula instance not in the test set with the same LTL type but a different number of landmarks during training. I use K-fold cross-validation with seed 123 to generate 5 formula holdout test and training sets and make sure that all formula instances are in the test set at least once. Formula holdout is generally harder than instruction holdout because certain formula instances (and therefore certain English instruction structures) are not seen by the model during training. I provide the average formula holdout accuracy across the 5 folds.

- **Type Holdout**: With type holdout, no formulas of the withheld types are seen by the model during training. I then test on these withheld formula types. This is the hardest test because it requires the most generalization from the model by translating previously unobserved English semantics into a valid LTL formula and it has never seen the LTL type and its corresponding structure. Like formula holdout, I use K-fold cross-validation with seed 123 to generate type holdout test and training sets and make sure that all LTL types are in the test set at least once. I provide the average formula holdout accuracy across the 2 folds.

As mentioned, I use GPT-3 as my large language model, specifically text-davinci-003 which I will refer to as GPT-3. On each of the holdout test sets, GPT-3 is fed every English instruction in the test sets and generates text output, hopefully LTL formulas. The correctness of these outputs is determined by using the Spot package [6]. Spot can check if two LTL formulas are equivalent, or specify the exact same things. It does this by converting the two LTL formulas, f and g, and their negations into their respective automata, and verifying that $\mathsf{A}_f \otimes \mathsf{A}_{\neg g}$ and $\mathsf{A}_g \otimes \mathsf{A}_{\neg f}$ are empty [6].

Evaluation for the LSTM model is described next.

4.4 LSTM Model

The LSTM model consists of an encoder-decoder architecture. The encoder is an LSTM and its purpose is to take in input and output hidden states. At each time step, the encoder outputs a hidden state which is a representation that encodes information from previous time steps. Its purpose is to encode context for our model to predict the next token based on the previous information. The final hidden state is then passed to a decoder. The decoder is a feed-forward neural network with dropout and RELU activation layers. The final hidden state is supposed to help the decoder output the correct tokens at each time step.

The model is trained on the dataset of instructions paired with trajectories. The reason this can work is because we can use the trajectories to verify if the model's LTL output is correct by running the trajectory on the LTL formula converted to a finite automaton and seeing if we end up at an accept state. To do this, we trained using a reinforcement learning and MML objective [10].

$$\sum_{(x,y)} \sum_{z} R(z) p_\theta(z|x) \tag{4.1}$$

$$p_\theta(y|x) = \sum_{z} p(y|z) p_\theta(z|x) \tag{4.2}$$

where $R(z) = 1$ only if the trajectory satisfies the model's output LTL, otherwise it is zero. It is supposed to learn from the correct programs and ignore the incorrect ones similar to reinforcement learning.

4.5 GPT-3 Prompting Techniques

Instead of finetuning GPT-3 on our dataset, I explore ways to improve off-the-shelf GPT-3 performance on our dataset via different prompting techniques. This bypasses the cost of finetuning GPT-3 on our downstream task which is expensive and allows it to be quickly deployed to other settings as well using the same prompting methods. I go into detail of each method in the following subsections.

The GPT-3 prompting methods for the translation task make use of the modular translation framework with large language models shown in Figure 1.1. I used this framework because the modular approach should allow GPT-3 to better output LTL formulas as it needs not identify explicit landmark names and can focus on formula structure generation. Landmarks are simply converted to placeholder symbols and in our case, they are a, b, c, d, and h as we only have instructions with up to 5 different landmarks. This smaller vocabulary should also help GPT-3's accuracy.

4.5.1 In-Context Learning versus Finetuning

GPT-3's key feature is its ability to understand the context of what is passed in (the prompt) and use it to generate appropriate responses. This is called in-context learning [3]. Similar to how humans can dynamically respond based on what was just said, GPT-3 can do something similar, but with text instead of speech. It can take in passed in information to generate a response that is relevant and makes sense in the context of what was passed in. This makes it a powerful model for performing a variety of tasks as it can adapt to a specific task once a prompt is fed in. We call this "prompting". By prompting GPT-3, it is able to generate coherent, context-sensitive text based on the input. An example of a prompt would be something like this:

```
# Prompt 1
```

```
Your task is to translate English sentences to French
   sentences.

Q: Hello, how are you doing?
A:
```

Listing 4.1: English-to-French Prompt 1

In this prompt, the sentence "Your task is to translate English sentences to French sentences." helps the model understand what it needs to do. In other words, we guide the model towards the task we want it to perform. Then, we feed in the question on the "Q:" line and we format it accordingly for it to generate its answer on the "A:" line. We could also have asked it "What is the French translation of Hello, how are you doing?". That would be an example of another prompt. With prompting, the pretrained langugae model is used off-the-shelf and no gradient updates are performed unlike finetuning.

In theory, this adaptability at the cost of almost nothing is amazing. In practice, however, there are performance tradeoffs and in some cases, those tradeoffs are quite drastic. Prior to in-context learning, the defacto approach towards using large language models in a specific domain or task would be to finetune the model on a task-specific dataset. In the translation task above, it would be a dataset consisting of English sentences as inputs paired with the correct translation as the targets. Finetuning means training the model on such a dataset and updating its pretrained weights via gradient updates. At the current moment, finetuning is still the best way to achieve the highest performance on a task. So, why do we care about prompting then? There are two big issues with finetuning. First, it is time-consuming and extremely costly to finetune models nowadays. Modern models have somewhere be-

tween billions to trillions of parameters, and the trend is still towards models with even more parameters. It requires expensive hardware and a lot of resouces to train this many parameters, even on smaller datasets. Second, for many tasks, like my task of translating English instructions to linear temporal logic, there are not enough datapoints in the dataset and it may be costly or infeasible to collect more data to train the model. There is also the problem of collecting high quality and diverse data which is one of the most important factors in training the model well. Therefore, finetuning may not be appropriate and prompting is the only solution.

Many researchers then started investigating different ways of improving prompting performance which we call prompt engineering. The most fundamental of these is the few-shot example prompting [3]. Taking our prompt from above, we modify it to look like this:

```
# Prompt 2
Your task is to translate English sentences to French
   sentences.

Q: Hello, how are you doing?
A: Bonjour comment allez-vous?

Q: What is your favorite food?
A: Quel est votre plat prefere?

Q: Do you like playing tennis?
A: Aimez-vous jouer au tennis?
```

```
Q: Do you like natural language processing?
A:
```

Listing 4.2: English-to-French Few-shot Prompt

In addition to the task description, we added a few examples of how to correctly perform the task. Compared to the prompt above, by seeing only a few examples, this confers significant gains to the model's performance and accuracy [3]. This is the most basic prompting technique and we start here in investigating other non-finetuning techniques to improve model performance.

4.5.2 Plain Prompting

For the task of translating English instructions to linear temporal logic, I followed the same template as few-shot prompting. The prompts are shown in Listing B.1 in the Appendix. I call this prompt method the "plain prompting".

Listing B.1 is only a part of the actual plain prompt for instruction holdout with an example or two each of the five LTL formula types that are in the dataset. The actual prompt passed into GPT-3 contains more examples. The question asks GPT-3 to translate an English sentence to LTL and the answer is the correct LTL formula. The last "Q:" line at the end is where each English sentence in the test dataset will be added to prompt GPT-3 to generate the answer.

For each holdout test, I follow the characteristics of each except that the training set is now the examples in the plain prompt. In instruction holdout, the plain prompt will contain instructions such that all formula instances are in the prompt, so GPT-3 will essentially have seen all formula instances before testing. In formula holdout, the prompt contains no examples with the formula instances in the test set. In type holdout, the prompt contains no formula types in the test set. Furthermore, for each

holdout test prompt, I only supply one example per unique formula instance in the prompt. As a reminder, a formula instance is defined by its number of landmarks and LTL type. The results of plain prompting on each of the three holdout tests can be found in Figure 5.1. Talk about how we did few-shot prompting on our specific task. results, what the prompts looked like, etc.

4.5.3 Chain of Thought

Chain of Thought prompting [25] is a prompting technique that builds on top of few-shot prompting. It is designed to elicit more coherent and logical reasoning from large language models by structuring prompts as a step-by-step thought process for arriving at the final answer. This is done by breaking the problem into intermediate steps and solving each of those steps first to get the final answer. It causes language models to explicitly build upon their previous solutions to subproblems in reaching the final answer and articulate a more sequential line of reasoning.

```
# CoT Prompt
Q: Roger has 5 tennis balls. He buys 2 more cans of
   tennis balls. Each can has 3 tennis balls. How many
    tennis balls does he have now?
A: A: Roger started with 5 balls. 2 cans of 3 tennis
   balls each is 6 tennis balls. 5 + 6 = 11. The
   answer is 11.

Q: Tracy used a piece of wire 4 feet long to support
   tomato plants in the garden. The wire was cut into
   pieces 6 inches long. How many pieces did she
   obtain?
```

```
A: The wire was 4 feet long. This means it was 4 * 12
   = 48 inches long. It was cut into pieces 6 inches
   long. This means she obtained 48 / 6 = 8 pieces.
   The answer is 8.

Q: Tom's ship can travel at 10 miles per hour. He is
   sailing from 1 to 4 PM. He then travels back at a
   rate of 6 mph. How long does it take him to get
   back?
A:
```

Listing 4.3: Chain of Thought Prompt

Listing 4.3 shows an example of how to perform Chain of Thought prompting. The only difference between this and plain prompting is the answer. In plain prompting, the answer portion only contains the answer. In Chain of Thought prompting, the answer contains all the intermediate steps used to reach the final answer. Upon seeing this, GPT-3 will follow this structure and sequentially reason through the task.

Empirical evidence show that large language models are better equipped to express their reasoning when using Chain of Thought prompting. This relatively simple technique has resulted in huge performance gains for large language models on complex reasoning tasks [25]. As such, I try to adapt Chain of Thought prompting for translating English to LTL. While not immediately evident, translating English to LTL can be formulated as a reasoning task with intermediate steps, albeit not perfectly.

Iteration 1

Listing B.2 in the Appendix shows a part of the first iteration of the chain of thought prompting. The idea is the same: I try to explain my reasoning for how to translate

English instructions into LTL by trying to outline intermediate steps. These steps mainly looked at breaking the instructions into different parts and evaluating them one by one. This included:

1. Identifying which notation to use
2. Identifying how many landmarks are in the instruction and what they are
3. Identifying key words that correspond to sequential constraints
4. Identify the order of the sequential constraints
5. Mapping those key words and the current sequential logic to LTL operators.
6. Map phrases in the instruction to subsets of LTL instances using the reasoned landmarks and LTL operators.
7. Combining the subanswers to each part into the final answer

Examining the prompt, the first part consists of the task description to guide GPT-3 towards what we want it to do. Then, for each chain of thought answer, we state the notation we're using, the number of landmarks, and the landmarks themselves. Then, we see if there are any words like "then" or "in any order" that indicate sequential constraints in the instruction. With that reasoned out, we try to explain the order in which we have to visit the landmarks and try to identify the key words we are going to use. Then, we use all that to map phrases in the instruction to LTL instances that we combine to form the final answer.

I tested this first iteration of chain of thought prompting on a small dataset of size 25 against plain prompting. The plain prompts had the same examples but without the chain of thought reasoning in the answer. Chain of thought had an 8 percent increase in accuracy over plain prompting. I tested the same prompts on a

larger dataset of size 50 and chain of thought again did 8 percent better. Gains were modest but promising.

From this iteration, there were issues with the chain of thought reasoning. The biggest issue is that the LTL formulas were in prefix notation. It made chain of thought much harder and hurt model performance. The reason prefix notation is difficult is because the mapping between the parts of the instruction to the LTL parts become less clear even for humans and it does not make sense where the LTL operators are placed, especially . Take for example this data point:

```
# Prefix Issues
Q: Translate the NLI "do not go to d until c is
   visited and do not go to c until b is visited and
   do not go to b until a is visited" to its LTL in
   prefix notation.
A: The answer must be in prefix notation. The
   instruction has four landmarks "a", "b", "c", and "
   d". Looking at parts of the instruction, the "until
   " phrases indicate there is a temporal constraint
   on visiting the landmarks. "do not go to d until c
   is visited" means we cannot go to "d" until we go
   to "c" which requires the "U" and "!" operator.
   There are landmarks to visit after "c" which
   requires "&". This is the prefix part "& U ! d c".
   "do not go to c until b is visited" and "do not go
   to b until a is visited" follow the same reasoning
   with different landmarks. This is the prefix part "
   & U ! c b" and "& U ! b a", respectively. We must
```

```
still visit "d" eventually which is the prefix part
 "F d". We combine the four parts while maintaining
 prefix notation. So the answer is "& U ! b a & U !
 c b & U ! d c F d".
```

Listing 4.4: Prefix Notation Issues

While we can map similar phrases to the same LTL formula, it does not make sense why U and ! and all appear in that order and before the landmarks, especially the when it has nothing to do with the phrase. However, omitting the would make it difficult for the model to place because it does not intuitively go between the LTL formulas. This also makes it difficult for the model to figure out how to combine the different LTL formulas for each phrase to get the final answer. And, we lose the readability of LTL. Looking at $U!dc$, we do not see that it means "do not go to c until b is visited". While we can explain these things to GPT-3, it would require an even longer answer. This is also an issue because GPT-3 has a context size of 4096 tokens which limits how long the prompt can be. Besides the notation issue, the answers contain statements that may not help the model like "which requires ''". It is crucial we do not have unnecessary information as we need to keep the prompt short because of the context size restriction. A longer prompt also means a higher cost per call to the GPT-3 API.

Iteration 2

Switching to infix notation for the LTL formulas addresses those issues. In iteration 2, our prompts look like this:

```
Q: What is "make sure a b c d and h are all visited"
   in LTL?
```

```
A: The instruction has 5 landmarks "a" and "b" and "c"
    and "d" and "h". "make sure...are all visited"
   specifies no time or sequential constraints.
   Therefore, we eventually go to "a" and "b" and "c"
   and "d" and "h". Eventually is the "F" LTL operator
   . "Eventually a" is "F a". "Eventually b" is "F b".
    "Eventually c" is "F c". "Eventually d" is "F d".
   "Eventually h" is "F h". We go to all 5 landmarks,
   so we join the 5 parts with "&". The answer is "F a
    & F b & F c & F d & F h".
```

Listing 4.5: Iteration 2 Chain of Thought Prompt

The structure is similar to iteration 1's prompt except we remove "The answer must be in prefix notation" as that was deemed unnecessary since the model should be able to easily infer the structure of the LTL formula from the answers and reasoning. Because the infix LTL formulas map really well to the English instructions and the patterns are more evident, we can reason much more easily about the intermediate steps and the phrases. We now rephrase the instruction to use verbs that map directly onto the LTL operators. In this example, we use "eventually" which is the F operator. Then we can concatenate the operators and the landmarks and directly map phrases like "eventually a" to Fa. With infix, we retain LTL's readability and its close associate with sequential instructions.

Iteration 3

Iteration 3 improves upon iteration 2 by continuing to refine and shorten the answers while retaining all important chain of thought reasoning. Specifically, I think of ways to keep the prompt size as close to constant as possible even as the number

of landmarks increases or the instructions get harder to parse. Listing B.3 in the Appendix shows one example each of all five LTL types.

We start off by making a general statement defining the LTL type of this instruction instead of defining the landmarks because that is redundant and scales with the number of landmarks. We then hone in on the relevant operators and identify the key words (then, after, forever, do not go back to previous) that hint towards the characteristics of the LTL type we are working with. According to the key words, we rephrase the sentence using the LTL operators corresponding verbs (eventually, until, only once, forever) paired with the landmark. We can remove the defining landmarks statement because we do that in this step. The bulk of the answer length comes from this part. We must explicitly write out each part for each landmark, otherwise we lose out on intermediate steps and may mislead the model. We can then break the rephrased sentence into parts that easily map to the corresponding LTL formula. When possible, we use statements like "Repeat for the remaining parts" to leverage the recursive nature of the LTL type which all of them exhibit and reduce answer length. We see that for types like visit and patrolling, it is easy to keep the answer at a near-constant length because we can be more concise during the rephrasing. However, for types like strict ordered visit, we must be precise because the LTL formula is more difficult to generate and we need to capture the full sequential constraints so that we can break it down and explain it clearly for the model.

Preliminary Testing

Due to the expensive cost of running chain of thought prompts on the entire dataset, for each holdout test, I sampled a smaller dataset of size 30 that contained at least one example each of the 22 unique formula instances. This ensures that we can maintain the characteristics of each holdout test. The same strategy is applied but on this smaller dataset to create these smaller holdout tests. I wanted to first see if

chain of thought would improve performance and use the results to refine my chain of thought prompts before committing to testing on the larger dataset. The results can be seen in Figure 4.1. For instruction hold, accuracy is on one instruction holdout set. For formula holdout, the accuracies are the average accuracies across five-fold cross validation. For type holdout, the average accuracies across two-fold cross validation.

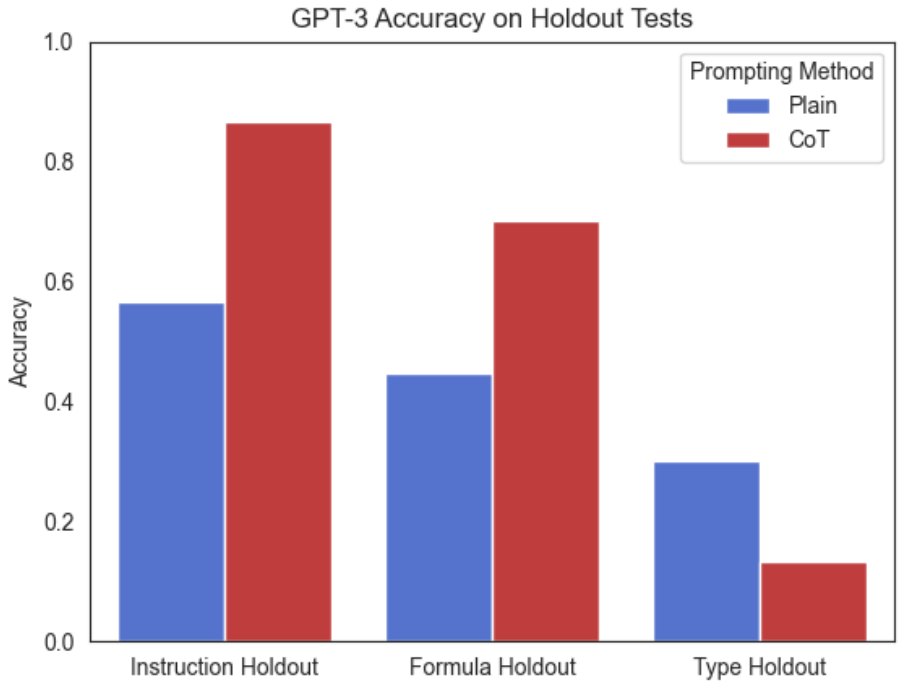

Figure 4.1: The average accuracies across the holdout tests of size 30.

We can see that the results are pretty good for chain of thought except for type holdout. The decrease in performance with type holdout is most likely due to the lessened generalizability of GPT-3 when it receives chain of thought prompts. I think the chain of thought prompts cause GPT-3 to rely more on the prompt examples for answers instead of its pretrained data which most likely contained the LTL types that are in the test set. Thus, we see a decrease there.

I examined the instructions that GPT-3 got wrong and many were all of the sequenced visit type and used the word “after”. An example is the instruction ”after

you have been to a make your way to b". What happened is that when GPT-3 sees "after", it thinks that the pattern type is strict ordered visit. It thinks this because in the prompt examples, there are strict ordered visit instructions like "make your way to c only after going to b before that make your way to b only after going to a do not go back to the previous location you have visited while heading to the next". During the chain of thought reasoning, I only highlighted the phrase "after" like this: "after" specifies we only visit one landmark after visiting another. GPT-3 learns from this and starts associating the phrase "after" with this definition and incorrectly reasons about the intermediate steps. I fixed this by specifying the phrase as "only after" instead of "after".

On the other hand, the remaining instructions that it got wrong highlighted the it also confuses strict ordered visit with sequenced visit the other way around. Take for example this instruction:

```
English instruction: visit a then b then c then d follow this
                     strict order and visit each location only
                     once.

Model Output: F(a & F(b & F(c & F(d & Fh)))) &
              (!a U (a U (!a U b))) &
              (!b U (b U (!b U c))) &
              (!c U (c U (!c U d))) &
              (!d U (d U (!d U h)))

Correct Output: (!b U a) & (!c U b) & (!d U c) & Fd &
                (!a U (a U (!a U b))) &
                (!b U (b U (!b U c))) &
                (!c U (c U (!c U d)))
```

I think this example perfectly highlights the biggest issue with instructions. They can be ambiguous. We can see that GPT-3 is decomposing the task according to our chain of thought reasoning. For strict ordered visit, it gets the visit once requirement, (!a U (a U (!a U b))) and so on, mostly correct except for the number of landmarks. However, it thinks the beginning part is sequenced visit because of the chain of "then" phrases in the instruction instead of correctly combining the meaning of the phrases "follow this strict order", "then" and "only once" to conclude that this part must use the U operator.

From these results, my takeaways to improve the chain of thought prompts were methods to reduce ambiguity in the English instruction and more precision on reasoning about strict ordered visit.

Final Iteration

Specifically, I modified my subsequent chain of thought prompts in the following ways:

1. The phrase "only after" specifies we only visit one landmark after visiting another instead of the phrase "after" to reduce ambiguity between sequenced visit and strict order visit

2. For strict order visits examples that can be misclassified as sequenced visits like the example above, I try to be even more specific in my reasoning. For example, here is how that is done for the above example: "then" specifies we visit landmarks in a specific order and "only visit...once" specifies that we visit a landmark only once. Therefore, combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks.

3. For strict order visits, GPT-3 got the number of landmarks incorrect for the visit once part, (!a U (a U (!a U b))). If we look at the old prompt, this

happens because we use the phrase “Repeat for the remaining parts” except we do not want to repeat this for the last landmark. In particular, looking at Listing B.3 under strict ordered visit, I changed ”Repeat for the remaining parts” to ”Repeat for the remaining landmarks except for the landmark d” and ”We visit each landmark only once” to ”We visit each landmark only once except for the last landmark, d”.

4. Because the model performs extremely well on the visit and patrolling formula types, I removed one visit example and two patrolling examples. These formula types, regardless of number of propositions, are really simple and each intermediate step is the same for each landmark and easily mapped to the appropriate LTL formula. This was done to reduce prompt length.

The full final chain of thought prompts for each holdout test are in the Appendix under Section C.2. The full plain prompts are in the Appendix under Section C.1. The evaluation of these prompts on their respective holdout tests are in Section 5.2.

4.5.4 Rewriting Instructions with ChatGPT

As we saw in chain of thought, a core issue in translation performance is the ambiguity in the instructions. Thus, I try to use ChatGPT, specifically, gpt-3.5-turbo, to rewrite instructions to be clearer. ChatGPT has an online interface that costs no money to use which makes testing free and its cost to produce 1000 tokens via the code API is a tenth of GPT-3’s cost. It is also supposed to be slightly better than GPT-3. These reasons led me to use ChatGPT as the language model for the rewriter module. This approach was also inspired by the idea of paraphrasing utterances into canonical utterances which is a user-defined language that can be easily mapped to the meaning representation (LTL in our case) [19]. They use a language model to do this paraphrasing via constrained decoding and then search over the space of outputs

to find candidates [19]. This approach can be computationally heavily and it also requires manual creation of a language with rules that make it easy to translate to LTL. Instead, I try to use ChatGPT to directly rewrite the instructions and map the instructions to a representation space that makes it easier for ChatGPT to parse to LTL. Since ChatGPT is closely related to GPT-3, this should also help GPT-3 in parsing the instruction. I initially kept GPT-3 as the translation module because I hypothesized that reusing the same model to rewrite and parse would not help performance, but I also tested this with ChatGPT as the translation module.

Iteration 1

Initial attempts were unimpressive as they were too simple and undetailed like "rewrite X" or "rephrase X". The first prompt that achieved success was this:

```
Prompt: I will be translating English instructions to
    linear temporal logic formulas. However, these
   instructions can be unclear and ambiguous. I want
   you to rewrite them so that they are clear and can
    be more easily translated to linear temporal
   logic formulas.

Issue: Unreliable responses.
```

Listing 4.6: ChatGPT Rewriting Prompt 1

On most instructions, ChatGPT do what it was asked and rewrite the sentences to be clearer. However, this prompt still proved unreliable as there were many instances where it would answer along the lines of "Sorry, this instruction is incomplete and unclear. Please provide more context or information". I needed to minimize these

occurrences as I want ChatGPT to rewrite all instructions and I would be doing the via the API and not the online interface.

Iteration 2

```
Prompt: I want you to rewrite English instructions
   that I provide such that they can be more easily
   mapped to linear temporal logic formulas.

Issue: Did not know "a", "b", and so on were landmarks
```

Listing 4.7: ChatGPT Rewriting Prompt 2

For this prompt, I tried rephrasing my prompt to be simpler to see if it would reduce the number of unreliable responses. It did not. ChatGPT kept responding with "specific location" which was convoluted, and sometimes, "I'm sorry, but "head to a" is not a complete instruction. Can you please provide me with a more detailed instruction that you want me to rewrite?". I initially thought the issue was again how I phrased my command. However, it turns out that it responded with "specific location" because it did not know that "a", "b", and so on were landmarks.

Iteration 3

```
Prompt: I want you to rewrite English instructions
   that I provide into clearer English instructions so
    that they can be more easily translated to linear
   temporal logic formulas. 'a', 'b', 'c', 'd', and 'h
   ' are all landmarks.
```

```
Issue: Returned outputs in a bad format and did things
    that I did not ask it to do.

Examples:
    1. Here's a possible rephrased version:
       "First, visit a. After visiting a, visit b at
          least once. Once you have visited both a and
           b, visit c at least once. After visiting a,
           b, and c, visit d at least once. Finally,
          after visiting a, b, c, and d, visit h at
          least once."
       Translated to LTL:
       F a & Fb & F(a & Fb) & Fc & F(a & Fb & Fc) & Fd
           & F(a & Fb & Fc & Fd) & Fh & F(a & Fb & Fc
          & Fd & Fh)
    2. Rewritten:
        * Fb after Fa
        * Fc after (Fa & Fb)
        * Fd after (Fa & Fb & Fc)
        * Fh after (Fa & Fb & Fc & Fd)
    3. The clearer instruction would be "Make a brief
       stop at a". The LTL formula for this
       instruction would be "F a".
```

Listing 4.8: ChatGPT Rewriting Prompt 3

In this iteration, I explicitly tell ChatGPT that the letters are landmarks, successfully removing the issue of it not knowing the landmarks completely. However, further

issues arose that required hammering out. As seen in the examples in Listing 4.8, it gave me answers in a non-formatted way. I needed the answers in a specific format so that I could extract the rewritten instruction from the response for all instructions. ChatGPT also misunderstood my instruction and thought I was asking it to give me the LTL formula as well.

Iteration 4

```
Prompt: I want you to rewrite English instructions
   that I provide into clearer English instructions so
    that they can be more easily translated to linear
   temporal logic formulas. 'a', 'b', 'c', 'd', and 'h
   ' are all landmarks. Write the response in the
   following format: Instruction: {response}. {
   response} should be replaced with the actual
   response you want to give.

Issue: Simple instructions became more complicated.
   Verbs became more complicated.

Examples:
    1. eventually you must go to a -> At some point in
        time, it is necessary to reach landmark 'a'.
    2. find -> identify
    3. keep visiting a and b -> Continuously visit
       landmarks a and b.
    4. please keep visiting a and b without stopping
       -> Continuously alternate between visiting
```

```
landmarks 'a' and 'b' without any breaks.
```

Listing 4.9: ChatGPT Rewriting Prompt 4

Prompt 4 solved the issue of the answers being in free-form. ChatGPT adheres to the specified template for answers really well. I think the "Instruction: response" part also implicitly cleared things up for it as it stopped giving me the LTL formulas. Everything looks good, but upon manual inspection, simple instructions and phrases became longer, more complicated, and used fancier verbs. ChatGPT's job is to make the instructions less ambiguous and simpler instructions do that better.

Iteration 5

```
Prompt: You will rewrite English instructions, if
   necessary, that I provide into clearer English
   instructions so that they can be more easily
   translated to linear temporal logic formulas. You
   must use simple verbs. For example, use "visit"
   instead of "go to" or "stop by".  You should use
   temporal connectives like "eventually", "then", "
   always", and "until" to specify temporal
   relationships between events. For example, use "
   eventually" instead of "at some point in time". "a"
   , "b", "c", "d", and "h" are all landmarks. Write
   the response in the following format: Instruction:
   {response}. {response} should be replaced with the
   actual response you want to give.
```

Issue: Changed the sequential constraints in the instructions.

Examples:

Visit

1. visit a -> always visit a (patrolling)
2. stop at a -> always stop at a (patrolling)
3. visit both a and b -> Visit a and then always visit b. (sequenced visit)
4. go to a and go to b -> Eventually visit a and then visit b. (sequenced visit)

Ordered Visit

1. reach c but not until you visit a and b and that specific order -> First visit a, then visit b, and only after that, visit c. (no mention of not visiting the subsequent landmarks until after the previous)
2. do not visit c until visiting b do not visit b until visiting a -> Visit a, then visit b, and only then visit c. (no mention of not visiting the subsequent landmarks until after the previous)

Strictly ordered

1. visit a only once then visit b but do not visit b before visiting a -> First, visit landmark a

```
   . Then, visit landmark b, but only after
   visiting landmark a. (loses the "visiting
   landmarks only once" part)
2. visit a once while staying away from b then
   visit b once while staying away from c finally
   visit c -> First, visit landmark a. Then, after
    leaving a, visit landmark b. Finally, after
   leaving b, visit landmark c. While visiting a,
   make sure to stay away from landmark b. While
   visiting b, make sure to stay away from
   landmark c. (loses the "once" part)

Patrolling
1. go to a c and b an infinite number of times ->
   Visit landmarks a and c, then visit landmark b
   an infinite number of times. (adds a "then")
2. continuously move to c b and a -> Continuously
   visit landmarks c, b, and a in that order. (
   adds an order to the visit)
3. travel to b c and a infinitely -> Travel to b,
   then c, and then a repeatedly. (adds an order)
```

Listing 4.10: ChatGPT Rewriting Prompt 5

With iteration 5, I changed three things. I told it to rephrase the instruction "if necessary" and to use "simple verbs" in order to address the issue of making simple instructions more complicated and using more difficult verbs for no reason. Simple verbs should help it understand the semantics better. I also added the component

on using temporal connectives like "eventually" because they correspond directly to LTL so that should really help the translation. While it did keep instructions simple, it started changing the sequential constraints of the instruction. For example, it changed the "visit a" to "always visit a", meaning a visit type changed to patrolling. It also removed or added sequential constraints. These are described under Examples in Listing 4.10.

Final Iteration

```
Prompt: You will rewrite English instructions that I
   provide into clearer English instructions, if
   necessary, so that they can be more easily
   translated to linear temporal logic formulas. You
   must use simple verbs. For example, use "visit"
   instead of "go to" or "stop by". When necessary,
   you can use temporal connectives like "eventually",
    "then", "always", and "until" to specify temporal
   relationships between events. For example, use "
   eventually" instead of "at some point in time". "a"
   , "b", "c", "d", and "h" are all landmarks. Write
   the response in the following format: Instruction:
   {response}. {response} should be replaced with the
   actual response you want to give.
```

Listing 4.11: ChatGPT Rewriting Prompt Final

To create the final prompt, I modified iteration 5 by changing "You should use temporal connectives like" to "When necessary, you can use temporal connectives like" because ChatGPT was taking that too literally and always changing and adding tem-

poral connectives unnecessarily. This fixed the issue. Manual inspection of the rewritten instructions produced simple, unambiguous instructions. Using this prompt, I tested the impact of adding a rewriter module to the pipeline and evaluate whether ChatGPT can effectively map instructions to a representation space that makes it easier for GPT-3 to translate. The results are in Section 5.3.

4.5.5 Learned Soft Prompts

Previous prompting methods focused on manually creating discrete prompts and its variations. However, manual prompt creation is difficult and prone to human errors. Moreover, it is hard to find the optimal prompt for a task to feed to the large language model. Thus, we also investigate continuous prompts and other finetuning alternatives. There are multiple methods to learn prompts and skip finetuning, including P-tuning [15], prefix-tuning [14], adapter-tuning [11], and prompt tuning [13]. These methods, like our methods above, try to reduce the number of parameters needed to be trained on the downstream task compared to finetuning. With our manual prompts, no parameter tuning was required. With these prompt learning methods and finetuning alternatives, we must learn parameters, but they all drastically reduce the number of parameters needed be learned compared to finetuning a model, usually less than 0.1 percent of the number of parameters in the large language models [13]. I opted for prompt tuning because it has competitive performance with the other methods but most importantly, it uses the least amount of parameters (less than 0.01 percent of task-specific parameters for models over a billion parameters) [13]. This is the biggest factor for me as I am trying to research ways to train as little parameters as possible and on a practical level, I do not have resources to train many parameters using these super large language models, so I want to train the least amount of parameters as possible.

Prompt tuning works by prepending prompt embeddings, which are sequences of

real vectors, to the input embeddings. We choose a prompt length which specifies how many tokens we want to prepend to the input tokens. We then use the prompt embeddings to get separate embeddings (vectors) for each token and prepend that to the input embeddings. This is then fed through the language model like normal to produce an output. The key idea is that we learn these prompt embeddings by training the model on a downstream training dataset: we perform this procedure on a training dataset, calculate losses based on the model output and the ground truth output, and then perform gradient updates using the loss function to update the prompt embeddings. The model parameters are frozen. By learning these prompt embeddings on a specific downstream, it should help the model adapt to the task like prompting without finetuning [13].

Unfortunately, we cannot do prompt tuning on GPT-3 because we do not have access to its word embeddings. Therefore, I chose Flan-T5 [4] as an open-source alternative to GPT-3. While this is not the best alternative, it is one of the few that is instruction fine-tuned like GPT-3 and offers good performance without being extremely large. The best open source alternatives to GPT-3 currently are LLaMA [22] and Alpaca [21] after instruction finetuning. Unfortunately, I cannot use these models as they are extremely large and I do not have the GPU resources to work with them. Even though we are freezing the model parameters, it still requires a lot of memory to store the model. Memory usage for inference and gradient calculations also increases with number of parameters in the model.

I choose Flan-T5-base as my model of choice because the Flan-T5-XXL models are also a bit too big. I was only barely able to train Flan-T5-base through using methods like smaller batch sizes, gradient accumulation, gradient checkpointing, and mixed precision training. These methods reduce GPU memory usage but increase training time. Implementation of prompt tuning and training is done using HuggingFace [26]. I trained the model for 45000 steps using an initial learning rate of 0.3; weight decay

of $1e^{-5}$; and the Adafactor optimizer with a decay rate of 0.8 and scale parameter off. For the dataset, I first permuted each instruction by changing the order of the placeholder landmarks to create more unique instructions and increase the dataset size from 550 to 36000. I only trained the model on the instruction holdout training set which was of size 26000. The evaluation set is of size 10000. Unfortunately, I could not evaluate my model on the test dataset because I could not get the GPU to stop running out of memory during evaluation. I report the training loss in Section 5.4.

4.5.6 Black-Box Prompt Learning

Black-box prompt learning tries to address the problem of not being able to utilize prompt learning methods with black-box models like GPT-3. Black-box prompt learning tries to learn optimal prompts without access to the model's parameters by using a gradient-free optimization method. They use a policy gradient inspired algorithm called variance-reduced policy gradient estimator to learn the optimal prompts [5]. Their evaluation on various natural language benchmarks show competitive results. Given that I initially tried to prompt learn with GPT-3, I adapted this method to the translation task of English instructions to LTL. Unfortunately, I could not train use black-box prompt learning with GPT-3 because even a few steps of training required $60-70 in API calls and the number of steps needed for convergence is unknown and most likely relative large.

Chapter 5

Evaluation

5.1 LSTM Results

After training the LSTM on the training dataset, we ran it on a test dataset of instructions paired with trajectories. The model performed extremely pooring, getting an accuracy close to 0. The reason for this is the approach was flawed. Trying to train a language model from scratch using reinforcement learning objectives is not feasible and $R(z)$ was not a good reward function. There was no learning from mistakes or any sense of weighted learning towards certain formulas over others. Moreover, there was the issue of spurious LTL formulas which were formulas that the trajectory would satisfy, so we deem them as correct, but in reality, they are incorrect LTL formulas. The model tended to learn spurious formulas to achieve the reward of 1. Trying to learn a language model without the ground truths and using an RL objective proved fatal.

5.2 Chain of Thought Prompting

Across the three holdout test sets, we see that chain of thought has mixed results. On instruction holdout, chain of thought confers a 10 percent boost in accuracy but does

5 percent worse on formula holdout and slightly worse on type holdout. These results highlight the nuanced nature of chain of thought prompting in LTL. Specifically, GPT-3 has limited knowledge of LTL and translating to it from English. To confirm this, I used a zero-shot method developed by Kojima [12] to prompt GPT-3 and probe it for LTL knowledge. The method is really simple: we provide GPT-3 the question and then prompt the method with "Let's think step by step". No examples are given. GPT-3 outputs:

```
GPT-3 Output: First, we need to express the idea of visiting a
              and then b. This can be expressed as "visit a and
              then visit b". In LTL, this can be written as:
```

We subsequently add on "Therefore, the answer in LTL is" to GPT-3's output and feed everything, including the initial question, back into the model. For any question, GPT-3 responds with ":" and never gives an answer. This supports the idea that it does not know LTL well nor how to translate from English to LTL.

Therefore, it must rely heavily on the prompt to perform the task well as seen in the performance increases with chain of thought on instruction holdout. However, chain of thought causes the model to rely too much on the prompt and only use the same reasoning as outlined there. This makes it more difficult for the model to generalize which is what formula and type holdout is testing. Thus, plain prompts do slightly better there. Interestingly, plain prompts on formula holdout do better than plain prompts on instruction holdout. A possible explanation is that on the folds where the formulas being tested are much simpler (e.g. less landmarks) than the examples in the prompt, the model can generlize decently well since LTL has a similar structure and close mapping to the phrases with sequential constraints. It can also output this simpler formula better because the prompt contains the harder formulas and is shorter than in instruction holdout so GPT-3 has to focus on less

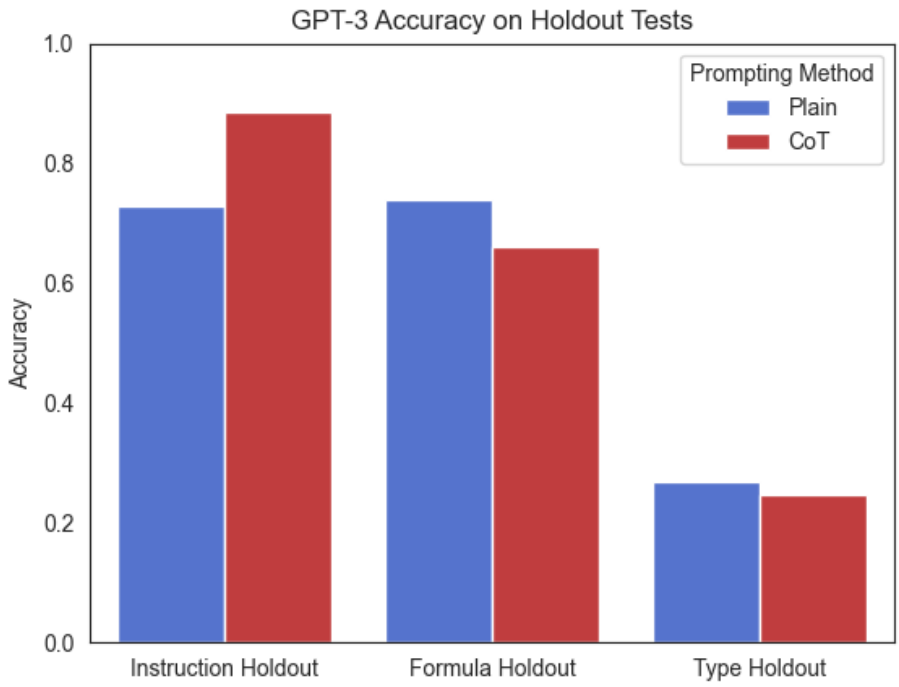

Figure 5.1: The average accuracies across the holdout tests

information and better find the relevant context.

5.3 ChatGPT Rewriter Module

The empirical results of Figure 5.2 suggests that rewriting instructions with ChatGPT does not help and in fact hurts performance across all holdout tests irrespective of the prompting method. This could be due to a number of reasons. It could be that the rewritten sentences, even though simpler, do not help because they deviate too much from the instructions in the prompt examples. It could also be that whatever output space that ChatGPT maps these instructions to is ill-suited for GPT-3 but could be better suited for ChatGPT itself. Or, it could be that the rewritten sentences are just bad quality which I think is unlikely.

To test the first two hypotheses, I ran two more experiments. In the first ex-

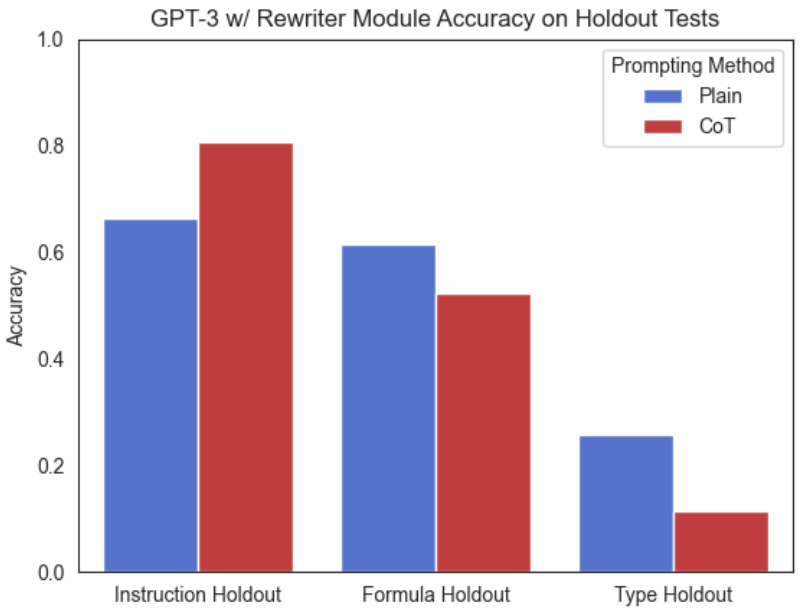

Figure 5.2: The average GPT-3 accuracies across the holdout tests with the rewriter module.

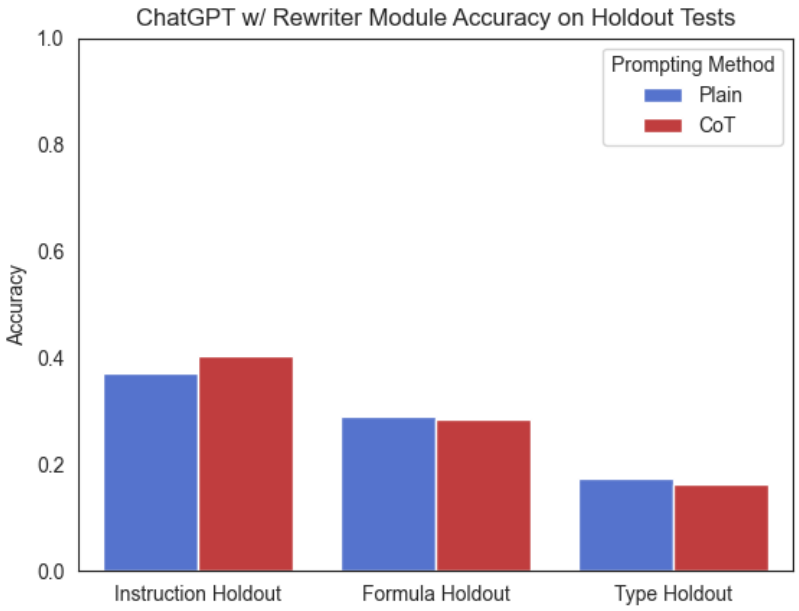

Figure 5.3: The average accuracies across the holdout tests with the rewriter module and ChatGPT as the translation module.

periment, I used ChatGPT as the translation module. I ran ChatGPT without the rewriter module on only the instruction holdout test and ran it with the rewriter module across all three holdout tests. Results are in Figures 5.4 and 5.3 In the second experiment, I used ChatGPT to rewrite the instructions in the final plain prompts. Results are in 5.5.

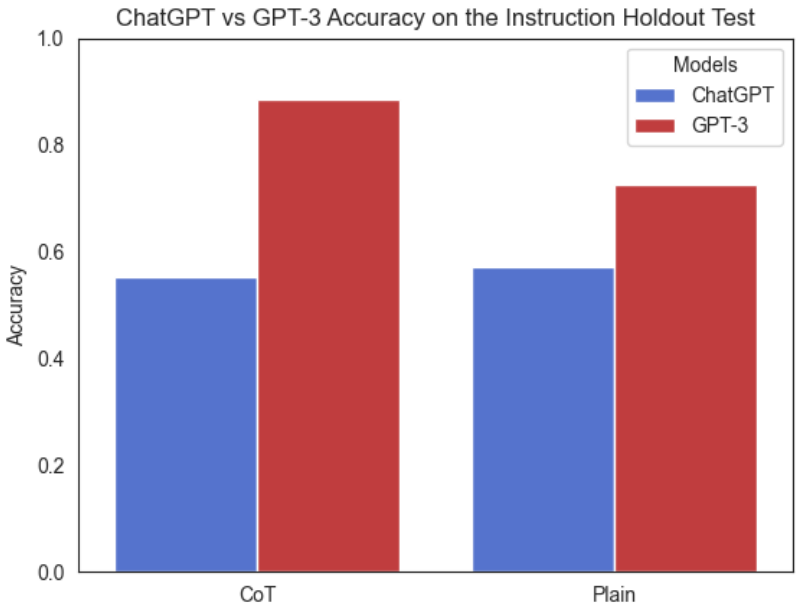

Figure 5.4: The average accuracy on the instruction holdout test with no rewriter module and ChatGPT as the translation module versus the average accuracy on the instruction holdout test with no rewriter module and GPT-3 as the translation module.

Results from Figure 5.4 indicate that ChatGPT with no rewriter module does worse with any prompting method compared to GPT-3 with no rewriter module. This could be due to the fact that ChatGPT is optimized for a chatbot-like experience and chain of thought/few-shot prompting are mainly methods for GPT-3 which follows the more traditional input model. Figure 5.3 demonstrates that ChatGPT with the rewriter module does much worse across all holdout tests compared to GPT-3 with or without the rewriter module. It seems that the output space of the rewritten instructions does not benefit ChatGPT at all. In fact, like GPT-3, it harms ChatGPT's

performance. Figure 5.5 shows that compared to not using the rewriter module to rewrite the instructions in the prompt examples, doing so hurts performance across the board for GPT-3. Overall, the rewriter module hurts performance and was an unsuccessful addition.

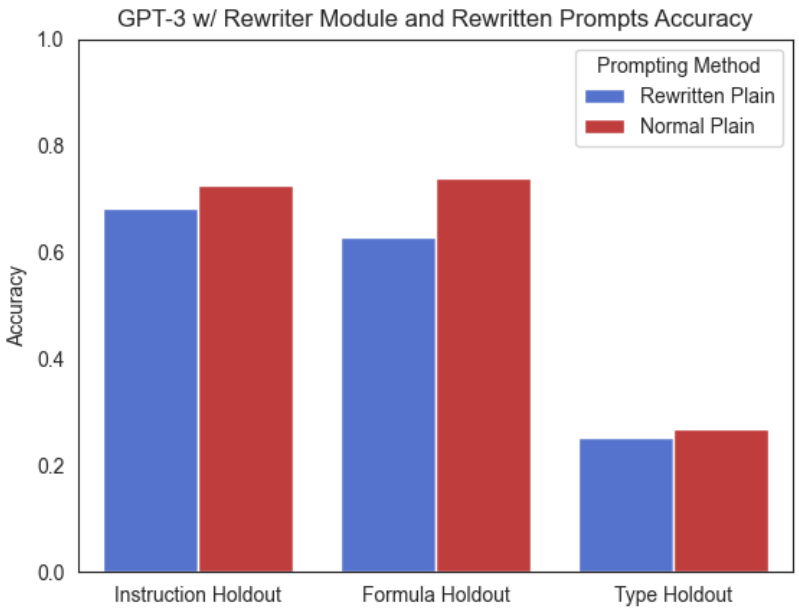

Figure 5.5: The average GPT-3 accuracies across the holdout tests with the rewriter module and rewritten prompts compared to the average GPT-3 accuracies across the holdout tests with just the rewriter module and non-rewritten prompts.

5.4 Prompt Tuning Training Results

I report the training loss for training Flan-T5-Base with prompt tuning. Training was done on an RTX3080. Unfortunately, I could not evaluate on the test dataset because the GPU kept running out of memory. I hypothesize that with access to better GPUs, specifically ones with more memory, then I would be able to evaluate the model. However, looking at the training curve, I expect the model to perform really well, if not the best, on the test set. I hope to be able to evaluate this approach

in the future.

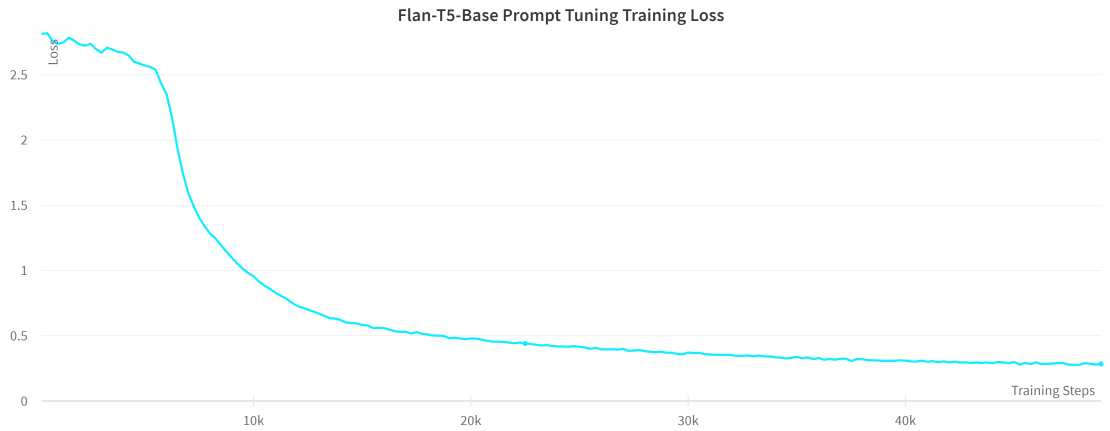

Figure 5.6: Prompt Tuning Training Loss with Flan-T5-Base

Chapter 6

Conclusion

In this paper, I showed different approaches to translating English instructions to LTL and reseached ways to improve the translation accuracy. An LSTM-based approached trained using an reinforcement learning objective proved ineffective. I transitioned to a large language model approach which performed decently without finetuning. I researched prompting methods like plain prompting (few-shot), zero-shot prompting, chain of thought prompting, rewriter module + prompting, prompt tuning, and black-box prompt learning. These were all methods that required minimal parameters to be trained, and thus also allowed for more generalizability. Results were mixed with some performing better than others in certain situations and some methods were unable to be evaluated. These are only some ways to try to improve LLM LTL translation accuracy, and with the current pace of AI innovation, many more methods will appear. Given the robustness of LTL to many applications, this research area can prove important and vital to focus on.

Appendix A

Code

Code for the models and experiments can be found at
https://github.com/iamsamliang/LTL_prompt_eng.

Appendix B

Iterations of Plain and Chain of Thought Prompts

```
# Plain LTL Prompts
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: Fa

Q: What is "move to a b c and d in any order" in LTL?
A: Fa & Fb & Fc & Fd

Q: What is "visit a and then b" in LTL?
A: F(a & F(a & Fb))

Q: What is "go to a then go to b then go to c then go
   to d finally go to h" in LTL?
```

```
A: F(a & F(b & F(c & F(d & Fh))))

Q: What is "do not go to h until d is visited and do
   not go to d until c is visited and do not go to c
   until b is visited and do not go to b until a is
   visited along the path never return to previous
   location you have been to" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!
   a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U
    (c U (!c U d))) & (!d U (d U (!d U h)))

Q: What is "visit a as many times as you want while
   avoiding b c and d then visit b as many times as
   you want while avoiding c and d then visit c as
   many times as you want while avoiding d finally
   visit d" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & Fd

Q: What is "keep visiting a b and c in any order" in
   LTL?
A: GFa & GFb & GFc

Q:
```

Listing B.1: Part of the Plain LTL Prompt for Instruction Holdout

```
# CoT LTL Prompt Iteration 1
```

```
Your task is to translate English natural language
   instructions (NLI) into linear temporal logic (LTL)
   .

Q: Translate the NLI "visit a and b in any order" to
   its LTL in prefix notation.
A: The answer must be in prefix notation. The
   instruction has two landmarks "a" and "b". It
   contains no temporal constraint on visiting "a" or
   "b". Looking at parts of the instruction, "in any
   order" implies that we can visit "a" and "b" in no
   particular order but that we eventually must visit
   both which requires the "F" operator. So the answer
    is "& F a F b".

Q: Translate the NLI "go to a then go to b then go to
   c" to its LTL in prefix notation.
A: The answer must be in prefix notation. The
   instruction has three landmarks "a", "b", and "c".
   Looking at parts of the instruction, the "then"
   phrases indicate there is a temporal constraint. We
    first visit "a", then "b", and then "c". We must "
   go to a" at some point in time. This requires the "
   F" operator. We must visit other landmarks after "a
   " which requires the "&" operator. This corresponds
    to the prefix part "F & a". Similarly, "then go to
    b" tells us we must visit "b" some time after
```

```
   visiting "a" which is the prefix part "F & b".
   Lastly, "then go to c" tells us we must visit "c"
   some time after visiting "a" and "b" which is the
   prefix part "F c". We combine the three parts while
    maintaining prefix notation. So the answer is "F &
    a F & b F c".

Q: Translate the NLI "visit c but only after visiting
   b and visit b only after visiting a" to its LTL in
   prefix notation.
A:
```

Listing B.2: Part of Chain of Thought LTL Prompt First Iteration

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

# Visit
Q: What is "move to a b c and d in any order" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "move to
   " specifies we must "eventually visit a, b, c, and
   d". Examining each part, "eventually visit a" is "
   Fa". "eventually visit b" is "Fb". Repeat for the
   remaining parts. The answer is "Fa & Fb & Fc & Fd".

# Sequenced Visit
```

Q: What is "head to a and then to b after a is visited and then to c after b is visited and then to d after c is visited" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and then visit c and then visit d, eventually". Working backwards, "eventually, visit d" is "Fd". " eventually, visit c and then visit d" is "F(c & Fd) ". "eventually, visit b and then c and then visit d " is "F(b & F(c & Fd))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & Fd)))".

 # Strict Ordered Visit

Q: What is "make your way to d only after going to c before that make your way to c only after going to b before that make your way to b only after going to a do not go back to the previous location you have visited while heading to the next" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "only after" specifies we only visit one landmark after visiting another. "do not go back to previous" specifies that we visit a landmark only

once. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "eventually visit d" is "Fd". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining parts. The answer is "(!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))".

 # Ordered Visit

Q: What is "do not go to h until d is visited and do not go to d until c is visited and do not go to c until b is visited and do not go to b until a is visited" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each

```
   part, "visit b but not until visit a" is "(!b U a)"
   . "visit c but not until visit b" is "(!c U b)".
   Repeat for the remaining parts. "eventually visit h
   " is "Fh". The answer is "(!b U a) & (!c U b) & (!d
    U c) & (!h U d) & Fh".

# Patrolling
Q: What is "visit a b h d and c with no limits" in LTL
   ?
A: We must eventually visit some landmarks in any
   order and do so forever. Eventually is the "F" LTL
   operator. Forever is the "G" LTL operator. "no
   limits" specifies we must "eventually visit a, b, c
   , d, and h forever". Examining each part, "
   eventually visit a forever" is "GFa". "eventually
   visit b forever" is "GFb". Repeat for the remaining
    parts. The answer is "GFa & GFb & GFc & GFd & GFh"
   .
```

Listing B.3: Iteration 3 Chain of Thought Prompt. Comments (#) are not part of the actual prompt

Appendix C

Final Prompts

C.1 Final Plain Prompt

C.1.1 Instruction Holdout Plain Prompt

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: Fa

Q: What is "visit c a and b in no specific order" in
   LTL?
A: Fa & Fb & Fc

Q: What is "move to a b c and d in any order" in LTL?
A: Fa & Fb & Fc & Fd
```

```
Q: What is "make sure a b c d and h are all visited"
   in LTL?
A: Fa & Fb & Fc & Fd & Fh

Q: What is "visit a and then b" in LTL?
A: F(a & F(a & Fb))

Q: What is "in some sequence visit a b and c in that
   order" in LTL?
A: F(a & F(b & Fc))

Q: What is "head to a and then to b after a is visited
    and then to c after b is visited and then to d
   after c is visited" in LTL?
A: F(a & F(b & F(c & Fd)))

Q: What is "go to a then go to b then go to c then go
   to d finally go to h" in LTL?
A: F(a & F(b & F(c & F(d & Fh))))

Q: What is "do not go to c until you visit b and do
   not go to b until you visit a and make sure you
   visit c" in LTL?
A: (!b U a) & (!c U b) & Fc

Q: What is "go to a only once but keep away from b and
    then visit b but keep away from a" in LTL?
```

A: (!b U a) & Fb & (!a U (a U (!a U b)))

Q: What is "make your way to c only after going to b before that make your way to b only after going to a do not go back to the previous location you have visited while heading to the next" in LTL?
A: (!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))

Q: What is "make your way to d only after going to c before that make your way to c only after going to b before that make your way to b only after going to a do not go back to the previous location you have visited while heading to the next" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))

Q: What is "do not go to h until d is visited and do not go to d until c is visited and do not go to c until b is visited and do not go to b until a is visited along the path never return to previous location you have been to" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))

Q: What is "find b but not until you find a first" in LTL?

A: (!b U a) & Fb

Q: What is "make sure at any point in time you will visit a at least once in the future" in LTL?

A: GFa

Q: What is "visit a as many times as you want while avoiding b c and d then visit b as many times as you want while avoiding c and d then visit c as many times as you want while avoiding d finally visit d" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & Fd

Q: What is "do not go to h until d is visited and do not go to d until c is visited and do not go to c until b is visited and do not go to b until a is visited" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh

Q: What is "keep visiting a b and c in any order" in LTL?

A: GFa & GFb & GFc

Q: What is "visit a b h d and c with no limits" in LTL ?

```
A: GFa & GFb & GFc & GFd & GFh


Q:
```

C.1.2 Formula Holdout Plain Prompt Fold 0

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: Fa

Q: What is "visit b and a" in LTL?
A: Fa & Fb

Q: What is "visit c a and b in no specific order" in
   LTL?
A: Fa & Fb & Fc

Q: What is "in some sequence visit a b and c in that
   order" in LTL?
A: F(a & F(b & Fc))

Q: What is "visit a b c d and h sequentially" in LTL?
A: F(a & F(b & F(c & F(d & Fh))))
```

```
Q: What is "in strictly this order visit a then
   eventually visit b and finally eventually c" in LTL
   ?
A: (!b U a) & (!c U b) & Fc

Q: What is "go to a exactly once while avoiding b then
    go to b" in LTL?
A: (!b U a) & Fb & (!a U (a U (!a U b)))

Q: What is "visit a exactly once while avoiding b and
   c then visit b exactly once while avoiding c
   finally visit c" in LTL?
A: (!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) &
   (!b U (b U (!b U c)))

Q: What is "visit a then b then c and then d you can
   only visit each landmark once" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!
   a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U
    d)))

Q: What is "visit a then b then c then d and then h
   visit each landmark only once" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!
   a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U
    (c U (!c U d))) & (!d U (d U (!d U h)))
```

Q: What is "find b but not until you find a first" in LTL?

A: (!b U a) & Fb

Q: What is "go to a an infinite number of times" in LTL?

A: GFa

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & Fd

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: GFa & GFb & GFc

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

A: GFa & GFb & GFc & GFd & GFh

```
Q: What is "go to a and b infinitely often" in LTL?
A: GFa & GFb

Q:
```

C.1.3 Formula Holdout Plain Prompt Fold 1

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: Fa

Q: What is "visit b and a" in LTL?
A: Fa & Fb

Q: What is "visit c a and b in no specific order" in
   LTL?
A: Fa & Fb & Fc

Q: What is "go to four locations a b d and c" in LTL?
A: Fa & Fb & Fc & Fd

Q: What is "sometime visit a c b h and d" in LTL?
A: Fa & Fb & Fc & Fd & Fh

Q: What is "visit a and then b" in LTL?
```

A: F(a & F(a & Fb))

Q: What is "in some sequence visit a b and c in that order" in LTL?
A: F(a & F(b & Fc))

Q: What is "visit a then visit b then visit c then visit d" in LTL?
A: F(a & F(b & F(c & Fd)))

Q: What is "in strictly this order visit a then eventually visit b and finally eventually c" in LTL ?
A: (!b U a) & (!c U b) & Fc

Q: What is "go to a exactly once while avoiding b then go to b" in LTL?
A: (!b U a) & Fb & (!a U (a U (!a U b)))

Q: What is "visit a exactly once while avoiding b and c then visit b exactly once while avoiding c finally visit c" in LTL?
A: (!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))

Q: What is "visit a then b then c and then d you can only visit each landmark once" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))

Q: What is "visit a then b then c then d and then h visit each landmark only once" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: GFa & GFb & GFc

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: GFa & GFb & GFc & GFd

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

```
A: GFa & GFb & GFc & GFd & GFh


Q:
```

C.1.4 Formula Holdout Plain Prompt Fold 2

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "visit c a and b in no specific order" in
   LTL?
A: Fa & Fb & Fc

Q: What is "go to four locations a b d and c" in LTL?
A: Fa & Fb & Fc & Fd

Q: What is "sometime visit a c b h and d" in LTL?
A: Fa & Fb & Fc & Fd & Fh

Q: What is "visit a and then b" in LTL?
A: F(a & F(a & Fb))

Q: What is "in some sequence visit a b and c in that
   order" in LTL?
A: F(a & F(b & Fc))
```

```
Q: What is "visit a then visit b then visit c then
   visit d" in LTL?
A: F(a & F(b & F(c & Fd)))

Q: What is "visit a b c d and h sequentially" in LTL?
A: F(a & F(b & F(c & F(d & Fh))))

Q: What is "in strictly this order visit a then
   eventually visit b and finally eventually c" in LTL
   ?
A: (!b U a) & (!c U b) & Fc

Q: What is "go to a exactly once while avoiding b then
    go to b" in LTL?
A: (!b U a) & Fb & (!a U (a U (!a U b)))

Q: What is "visit a exactly once while avoiding b and
   c then visit b exactly once while avoiding c
   finally visit c" in LTL?
A: (!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) &
   (!b U (b U (!b U c)))

Q: What is "visit a then b then c then d and then h
   visit each landmark only once" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!
   a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U
    (c U (!c U d))) & (!d U (d U (!d U h)))
```

Q: What is "find b but not until you find a first" in LTL?

A: (!b U a) & Fb

Q: What is "go to a an infinite number of times" in LTL?

A: GFa

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & Fd

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: GFa & GFb & GFc & GFd

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

A: GFa & GFb & GFc & GFd & GFh

```
Q: What is "go to a and b infinitely often" in LTL?
A: GFa & GFb

Q:
```

C.1.5 Formula Holdout Plain Prompt Fold 3

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: Fa

Q: What is "visit b and a" in LTL?
A: Fa & Fb

Q: What is "visit c a and b in no specific order" in
   LTL?
A: Fa & Fb & Fc

Q: What is "go to four locations a b d and c" in LTL?
A: Fa & Fb & Fc & Fd

Q: What is "sometime visit a c b h and d" in LTL?
A: Fa & Fb & Fc & Fd & Fh
```

```
Q: What is "visit a and then b" in LTL?
A: F(a & F(a & Fb))

Q: What is "in some sequence visit a b and c in that
   order" in LTL?
A: F(a & F(b & Fc))

Q: What is "visit a then visit b then visit c then
   visit d" in LTL?
A: F(a & F(b & F(c & Fd)))

Q: What is "visit a b c d and h sequentially" in LTL?
A: F(a & F(b & F(c & F(d & Fh))))

Q: What is "visit a then b then c and then d you can
   only visit each landmark once" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!
   a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U
    d)))

Q: What is "visit a then b then c then d and then h
   visit each landmark only once" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!
   a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U
    (c U (!c U d))) & (!d U (d U (!d U h)))
```

Q: What is "find b but not until you find a first" in LTL?

A: (!b U a) & Fb

Q: What is "go to a an infinite number of times" in LTL?

A: GFa

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & Fd

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: GFa & GFb & GFc

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: GFa & GFb & GFc & GFd

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

A: GFa & GFb & GFc & GFd & GFh

Q: What is "go to a and b infinitely often" in LTL?

A: GFa & GFb

```
Q:
```

C.1.6 Formula Holdout Plain Prompt Fold 4

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: Fa

Q: What is "visit b and a" in LTL?
A: Fa & Fb

Q: What is "go to four locations a b d and c" in LTL?
A: Fa & Fb & Fc & Fd

Q: What is "sometime visit a c b h and d" in LTL?
A: Fa & Fb & Fc & Fd & Fh

Q: What is "visit a and then b" in LTL?
A: F(a & F(a & Fb))

Q: What is "visit a then visit b then visit c then
   visit d" in LTL?
A: F(a & F(b & F(c & Fd)))

Q: What is "visit a b c d and h sequentially" in LTL?
```

```
A: F(a & F(b & F(c & F(d & Fh))))

Q: What is "in strictly this order visit a then
   eventually visit b and finally eventually c" in LTL
   ?
A: (!b U a) & (!c U b) & Fc

Q: What is "go to a exactly once while avoiding b then
    go to b" in LTL?
A: (!b U a) & Fb & (!a U (a U (!a U b)))

Q: What is "visit a exactly once while avoiding b and
   c then visit b exactly once while avoiding c
   finally visit c" in LTL?
A: (!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) &
   (!b U (b U (!b U c)))

Q: What is "visit a then b then c and then d you can
   only visit each landmark once" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!
   a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U
    d)))

Q: What is "find b but not until you find a first" in
   LTL?
A: (!b U a) & Fb
```

Q: What is "go to a an infinite number of times" in LTL?

A: GFa

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & Fd

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: GFa & GFb & GFc

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: GFa & GFb & GFc & GFd

Q: What is "go to a and b infinitely often" in LTL?

A: GFa & GFb

Q:

C.1.7 Type Holdout Plain Prompt Fold 0

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: Fa

Q: What is "visit b and a" in LTL?
A: Fa & Fb

Q: What is "visit c a and b in no specific order" in
   LTL?
A: Fa & Fb & Fc

Q: What is "go to four locations a b d and c" in LTL?
A: Fa & Fb & Fc & Fd

Q: What is "sometime visit a c b h and d" in LTL?
A: Fa & Fb & Fc & Fd & Fh

Q: What is "in strictly this order visit a then
   eventually visit b and finally eventually c" in LTL
   ?
A: (!b U a) & (!c U b) & Fc
```

```
Q: What is "find b but not until you find a first" in
   LTL?
A: (!b U a) & Fb

Q: What is "go to a only after that go to b only then
   go to c and only then go to d" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & Fd

Q: What is "go to a and only go to b only after a is
   visited and then go to c only after a and b are
   both visited and then go to d only after a b and c
   are all visited and then go to h only after a b c
   and d are all visited" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh

Q:
```

C.1.8 Type Holdout Plain Prompt Fold 1

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "visit a and then b" in LTL?
A: F(a & F(a & Fb))

Q: What is "in some sequence visit a b and c in that
   order" in LTL?
```

A: F(a & F(b & Fc))

Q: What is "visit a then visit b then visit c then visit d" in LTL?
A: F(a & F(b & F(c & Fd)))

Q: What is "visit a b c d and h sequentially" in LTL?
A: F(a & F(b & F(c & F(d & Fh))))

Q: What is "go to a exactly once while avoiding b then go to b" in LTL?
A: (!b U a) & Fb & (!a U (a U (!a U b)))

Q: What is "visit a exactly once while avoiding b and c then visit b exactly once while avoiding c finally visit c" in LTL?
A: (!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))

Q: What is "visit a then b then c and then d you can only visit each landmark once" in LTL?
A: (!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))

Q: What is "visit a then b then c then d and then h visit each landmark only once" in LTL?

```
A: (!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!
   a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U
    (c U (!c U d))) & (!d U (d U (!d U h)))

Q: What is "go to a an infinite number of times" in
   LTL?
A: GFa

Q: What is "frequent a b and c infinitely in any order
   " in LTL?
A: GFa & GFb & GFc

Q: What is "frequent a b c and d infinitely in any
   order" in LTL?
A: GFa & GFb & GFc & GFd

Q: What is "frequent a b c d and h infinitely in any
   order" in LTL?
A: GFa & GFb & GFc & GFd & GFh

Q: What is "go to a and b infinitely often" in LTL?
A: GFa & GFb

Q:
```

C.2 Final Chain of Thought Prompts

C.2.1 Instruction Holdout CoT Prompt

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "reach"
   specifies we must "eventually visit a". Examining
   each part, "eventually visit a" is "Fa". There is 1
    landmark so we are done. The answer is "Fa".

Q: What is "visit c a and b in no specific order" in
   LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "visit"
   specifies we must "eventually visit a, b, and c".
   Examining each part, "eventually visit a" is "Fa".
   "eventually visit b" is "Fb". Repeat for the
   remaining parts. The answer is "Fa & Fb & Fc".

Q: What is "move to a b c and d in any order" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "move to
   " specifies we must "eventually visit a, b, c, and
   d". Examining each part, "eventually visit a" is "
```

Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd".

Q: What is "make sure a b c d and h are all visited" in LTL?

A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "are all visited" specifies we must "eventually visit a, b, c, d, and h". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd & Fh".

Q: What is "visit a and then b" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must "visit a and then visit b, eventually". Working backwards, "eventually, visit b" is "Fb". "eventually, visit a and then visit b" is "F(a & Fb)". We reached the first landmark so we are done. The answer is "F(a & F(a & Fb))".

Q: What is "in some sequence visit a b and c in that order" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "in that order" specifies we must visit "a and then visit b

and then visit c, eventually". Working backwards, "eventually, visit c" is "Fc". "eventually, visit b and then visit c" is "F(b & Fc)". "eventually, visit a and then b and then visit c" is "F(a & F(b & Fc))". We reached the first landmark so we are done. The answer is "F(a & F(b & Fc))".

Q: What is "head to a and then to b after a is visited and then to c after b is visited and then to d after c is visited" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and then visit c and then visit d, eventually". Working backwards, "eventually, visit d" is "Fd". " eventually, visit c and then visit d" is "F(c & Fd)". "eventually, visit b and then c and then visit d" is "F(b & F(c & Fd))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & Fd)))".

Q: What is "go to a then go to b then go to c then go to d finally go to h" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and then visit c and then visit d and then visit h,

eventually". Working backwards, "eventually, visit h" is "F h". "eventually, visit d and then visit h" is "F(d & F h)". "eventually, visit c and then d and then visit h" is "F(c & F(d & Fh))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & F(d & Fh))))".

Q: What is "do not go to c until you visit b and do not go to b until you visit a and make sure you visit c" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "until" specifies we only visit one landmark after visiting another. "We eventually visit c but not until we visit b and we visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "eventually visit c" is "Fc". The answer is "(!b U a) & (!c U b) & Fc".

Q: What is "go to a only once but keep away from b and then visit b but keep away from a" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we

never visit it again. Eventually is the "F" LTL operator. "keep away" and "then" specify we only visit one landmark after visiting another. "only once" specifies we visit a landmark only once. "We eventually visit b but not until we visit a. We visit each landmark only once except for the last landmark, b". Examining each part, "visit b but not until visit a" is "(!b U a)". "eventually visit b" is "Fb". "visit a only once" is (!a U (a U (!a U b))). There are no other landmarks remaining besides the last landmark b so we are done. The answer is "(!b U a) & Fb & (!a U (a U (!a U b)))".

Q: What is "make your way to c only after going to b before that make your way to b only after going to a do not go back to the previous location you have visited while heading to the next" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "only after" specifies we only visit one landmark after visiting another. "do not go back to previous" specifies that we visit a landmark only once. "We eventually visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last

landmark, c". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "eventually visit c" is "Fc". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). There are no other landmarks remaining besides the last landmark c so we are done. The answer is "(!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))".

Q: What is "make your way to d only after going to c before that make your way to c only after going to b before that make your way to b only after going to a do not go back to the previous location you have visited while heading to the next" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "only after" specifies we only visit one landmark after visiting another. "do not go back to previous" specifies that we visit a landmark only once. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, d". Examining each part, "visit b but not until visit

a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "eventually visit d" is "Fd". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark d. The answer is "(!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))".

Q: What is "do not go to h until d is visited and do not go to d until c is visited and do not go to c until b is visited and do not go to b until a is visited along the path never return to previous location you have been to" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "until" specifies we only visit one landmark after visiting another. "never return to previous" specifies that we visit a landmark only once. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, h". Examining

each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "visit h but not until we visit d" is "(!h U d)". "eventually visit h" is "Fh". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for the landmark h. The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))".

Q: What is "find b but not until you find a first" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "until" specifies we only visit one landmark after visiting another. "We eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "eventually visit b" is "Fb". The answer is "(!b U a) & Fb".

Q: What is "make sure at any point in time you will visit a at least once in the future" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. "at least once" specifies we must "eventually visit a forever". Examining each part, "eventually visit a forever" is "GFa". There is 1 landmark so we are done. The answer is "GFa".

Q: What is "visit a as many times as you want while avoiding b c and d then visit b as many times as you want while avoiding c and d then visit c as many times as you want while avoiding d finally visit d" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit d" is "Fd". The answer is "(!b U a) & (!c U b) & (!d U c) & Fd".

Q: What is "do not go to h until d is visited and do not go to d until c is visited and do not go to c until b is visited and do not go to b until a is visited" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, " visit b but not until visit a" is "(!b U a)". " visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit h " is "Fh". The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh".

Q: What is "keep visiting a b and c in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. "keep visiting" specifies we must "eventually visit a, b, and c forever". Examining each part, "eventually visit a forever" is "GFa". "eventually visit b

```
   forever" is "GFb". Repeat for the remaining parts.
   The answer is "GFa & GFb & GFc".

Q: What is "visit a b h d and c with no limits" in LTL
   ?
A: We must eventually visit some landmarks in any
   order and do so forever. Eventually is the "F" LTL
   operator. Forever is the "G" LTL operator. "no
   limits" specifies we must "eventually visit a, b, c
   , d, and h forever". Examining each part, "
   eventually visit a forever" is "GFa". "eventually
   visit b forever" is "GFb". Repeat for the remaining
    parts. The answer is "GFa & GFb & GFc & GFd & GFh"
   .

Q:
```

C.2.2 Formula Holdout CoT Prompt Fold 0

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "reach"
   specifies we must "eventually visit a". Examining
   each part, "eventually visit a" is "Fa". There is 1
```

landmark so we are done. The answer is "Fa".

Q: What is "visit b and a" in LTL?

A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "visit" specifies we must "eventually visit a and b". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". There are 2 landmarks so we are done. The answer is "Fa & Fb".

Q: What is "visit c a and b in no specific order" in LTL?

A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "visit" specifies we must "eventually visit a, b, and c". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc".

Q: What is "in some sequence visit a b and c in that order" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "in that order" specifies we must visit "a and then visit b and then visit c, eventually". Working backwards, "eventually, visit c" is "Fc". "eventually, visit b and then visit c" is "F(b & Fc)". "eventually,

visit a and then b and then visit c" is "F(a & F(b & Fc))". We reached the first landmark so we are done. The answer is "F(a & F(b & Fc))".

Q: What is "visit a b c d and h sequentially" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. " sequentially" specifies we must visit "a and then visit b and then visit c and then visit d and then visit h, eventually". Working backwards, " eventually, visit h" is "F h". "eventually, visit d and then visit h" is "F(d & F h)". "eventually, visit c and then d and then visit h" is "F(c & F(d & Fh))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & F(d & Fh))))".

Q: What is "in strictly this order visit a then eventually visit b and finally eventually c" in LTL ?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "strictly this order" specifies we only visit one landmark after visiting another. "We eventually visit c but not until we visit b and we visit b but not until we visit a". Examining each

part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". The answer is "(!b U a) & (!c U b) & Fc".

Q: What is "go to a exactly once while avoiding b then go to b" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit b but not until we visit a. We visit each landmark only once". Examining each part, "visit b but not until visit a" is "(!b U a)". "eventually visit b" is "Fb". "visit a only once" is (!a U (a U (!a U b))). The answer is "(!b U a) & Fb & (!a U (a U (!a U b)))".

Q: What is "visit a exactly once while avoiding b and c then visit b exactly once while avoiding c finally visit c" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we

never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies that we visit a landmark only once. "We eventually visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once". Examining each part, "visit b but not until visit a" is "(!b U a)" . "visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). The answer is "(!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))".

Q: What is "visit a then b then c and then d you can only visit each landmark once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only visit...once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit d but not until we

visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, d ". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "eventually visit d" is "Fd ". "visit a only once" is (!a U (a U (!a U b))). " visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark d. The answer is "(!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))".

Q: What is "visit a then b then c then d and then h visit each landmark only once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c

but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, h". Examining each part, "visit b but not until visit a" is "(!b U a)" . "visit c but not until visit b" is "(!c U b)". " visit d but not until we visit c" is "(!d U c)". " visit h but not until we visit d" is "(!h U d)". " eventually visit h" is "Fh". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark h. The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))".

Q: What is "find b but not until you find a first" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "until" specifies we only visit one landmark after visiting another. "We eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)" . "eventually visit b" is "Fb". The answer is "(!b U a) & Fb".

Q: What is "go to a an infinite number of times" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinite" specifies we must "eventually visit a forever". Examining each part, "eventually visit a forever" is "GFa". There is 1 landmark so we are done. The answer is "GFa".

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" and "only then" specify we only visit one landmark after visiting another. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit d" is "Fd". The answer is "(!b U a) & (!c U b) & (!d U c) & Fd".

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" specify we only visit one landmark after visiting another. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit h" is "Fh". The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh".

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, and c forever". Examining each part, "eventually

visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc".

Q: What is "frequent a b c d and h infinitely in any order" in LTL?
A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, d, and h forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd & GFh" .

Q: What is "go to a and b infinitely often" in LTL?
A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a and b forever". Examining each part, "eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". There are 2 landmarks so we are done. The answer is "GFa & GFb".

Q:

C.2.3 Formula Holdout CoT Prompt Fold 1

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "reach"
   specifies we must "eventually visit a". Examining
   each part, "eventually visit a" is "Fa". There is 1
    landmark so we are done. The answer is "Fa".

Q: What is "visit b and a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "visit"
   specifies we must "eventually visit a and b".
   Examining each part, "eventually visit a" is "Fa".
   "eventually visit b" is "Fb". There are 2 landmarks
    so we are done. The answer is "Fa & Fb".

Q: What is "visit c a and b in no specific order" in
   LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "visit"
   specifies we must "eventually visit a, b, and c".
   Examining each part, "eventually visit a" is "Fa".
   "eventually visit b" is "Fb". Repeat for the
```

remaining parts. The answer is "Fa & Fb & Fc".

Q: What is "go to four locations a b d and c" in LTL?
A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "go to" specifies we must "eventually visit a, b, c, and d". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd".

Q: What is "sometime visit a c b h and d" in LTL?
A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "sometime visit" specifies we must "eventually visit a, b, c, d, and h". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd & Fh".

Q: What is "visit a and then b" in LTL?
A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must "visit a and then visit b, eventually". Working backwards, "eventually, visit b" is "Fb". "eventually, visit a and then visit b" is "F(a & Fb)". We reached the first landmark so we are done. The answer is "F(a & F(a & Fb))".

Q: What is "in some sequence visit a b and c in that order" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "in that order" specifies we must visit "a and then visit b and then visit c, eventually". Working backwards, "eventually, visit c" is "Fc". "eventually, visit b and then visit c" is "F(b & Fc)". "eventually, visit a and then b and then visit c" is "F(a & F(b & Fc))". We reached the first landmark so we are done. The answer is "F(a & F(b & Fc))".

Q: What is "visit a then visit b then visit c then visit d" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and then visit c and then visit d, eventually". Working backwards, "eventually, visit d" is "Fd". " eventually, visit c and then visit d" is "F(c & Fd)". "eventually, visit b and then c and then visit d" is "F(b & F(c & Fd))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & Fd)))".

Q: What is "in strictly this order visit a then eventually visit b and finally eventually c" in LTL ?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "in strictly this order" specifies we only visit one landmark after visiting another. "We eventually visit c but not until we visit b and we visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)" . "visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". The answer is "(!b U a) & (!c U b) & Fc".

Q: What is "go to a exactly once while avoiding b then go to b" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit b but not until we visit a. We visit each landmark only once". Examining each part, "visit b but not until visit a" is "(!b

U a)". "eventually visit b" is "Fb". "visit a only once" is (!a U (a U (!a U b))). The answer is "(!b U a) & Fb & (!a U (a U (!a U b)))".

Q: What is "visit a exactly once while avoiding b and c then visit b exactly once while avoiding c finally visit c" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once". Examining each part , "visit b but not until visit a" is "(!b U a)". " visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). The answer is "(!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))".

Q: What is "visit a then b then c and then d you can only visit each landmark once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only visit...once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, d ". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "eventually visit d" is "Fd ". "visit a only once" is (!a U (a U (!a U b))). " visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark d. The answer is "(!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))".

Q: What is "visit a then b then c then d and then h visit each landmark only once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, h". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". " visit d but not until we visit c" is "(!d U c)". " visit h but not until we visit d" is "(!h U d)". " eventually visit h" is "Fh". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark h. The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))".

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" specify we only visit one landmark after visiting another. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit h" is "Fh". The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh".

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, and c forever". Examining each part, "eventually

visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc".

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, and d forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd".

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, d, and h forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd & GFh" .

```
Q:
```

C.2.4 Formula Holdout CoT Prompt Fold 2

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "visit c a and b in no specific order" in
   LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "visit"
   specifies we must "eventually visit a, b, and c".
   Examining each part, "eventually visit a" is "Fa".
   "eventually visit b" is "Fb". Repeat for the
   remaining parts. The answer is "Fa & Fb & Fc".

Q: What is "go to four locations a b d and c" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "go to"
   specifies we must "eventually visit a, b, c, and d"
   . Examining each part, "eventually visit a" is "Fa"
   . "eventually visit b" is "Fb". Repeat for the
   remaining parts. The answer is "Fa & Fb & Fc & Fd".

Q: What is "sometime visit a c b h and d" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "
```

sometime visit" specifies we must "eventually visit a, b, c, d, and h". Examining each part, " eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd & Fh".

Q: What is "visit a and then b" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must "visit a and then visit b, eventually". Working backwards, "eventually, visit b" is "Fb". "eventually, visit a and then visit b" is "F(a & Fb)". We reached the first landmark so we are done. The answer is "F(a & F(a & Fb))".

Q: What is "in some sequence visit a b and c in that order" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "in that order" specifies we must visit "a and then visit b and then visit c, eventually". Working backwards, "eventually, visit c" is "Fc". "eventually, visit b and then visit c" is "F(b & Fc)". "eventually, visit a and then b and then visit c" is "F(a & F(b & Fc))". We reached the first landmark so we are done. The answer is "F(a & F(b & Fc))".

Q: What is "visit a then visit b then visit c then visit d" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and then visit c and then visit d, eventually". Working backwards, "eventually, visit d" is "Fd". " eventually, visit c and then visit d" is "F(c & Fd)". "eventually, visit b and then c and then visit d" is "F(b & F(c & Fd))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & Fd)))".

Q: What is "visit a b c d and h sequentially" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. " sequentially" specifies we must visit "a and then visit b and then visit c and then visit d and then visit h, eventually". Working backwards, " eventually, visit h" is "F h". "eventually, visit d and then visit h" is "F(d & F h)". "eventually, visit c and then d and then visit h" is "F(c & F(d & Fh))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & F(d & Fh))))".

Q: What is "in strictly this order visit a then eventually visit b and finally eventually c" in LTL ?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "in strictly this order" specifies we only visit one landmark after visiting another. "We eventually visit c but not until we visit b and we visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)" . "visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". The answer is "(!b U a) & (!c U b) & Fc".

Q: What is "go to a exactly once while avoiding b then go to b" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit b but not until we visit a. We visit each landmark only once". Examining each part, "visit b but not until visit a" is "(!b

U a)". "eventually visit b" is "Fb". "visit a only once" is (!a U (a U (!a U b))). The answer is "(!b U a) & Fb & (!a U (a U (!a U b)))".

Q: What is "visit a exactly once while avoiding b and c then visit b exactly once while avoiding c finally visit c" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once". Examining each part , "visit b but not until visit a" is "(!b U a)". " visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). The answer is "(!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))".

Q: What is "visit a then b then c then d and then h visit each landmark only once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, h". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". " visit d but not until we visit c" is "(!d U c)". " visit h but not until we visit d" is "(!h U d)". " eventually visit h" is "Fh". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark h. The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))".

Q: What is "find b but not until you find a first" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "until" specifies we only visit one landmark after visiting another. "We eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "eventually visit b" is "Fb". The answer is "(!b U a) & Fb".

Q: What is "go to a an infinite number of times" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinite" specifies we must "eventually visit a forever". Examining each part, "eventually visit a forever" is "GFa". There is 1 landmark so we are done. The answer is "GFa".

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL

operator. "only after" and "only then" specify we only visit one landmark after visiting another. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit d" is "Fd". The answer is "(!b U a) & (!c U b) & (!d U c) & Fd".

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" specify we only visit one landmark after visiting another. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the

remaining parts. "eventually visit h" is "Fh". The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh".

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, and d forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd".

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, d, and h forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd & GFh" .

```
Q: What is "go to a and b infinitely often" in LTL?
A: We must eventually visit some landmarks in any
   order and do so forever. Eventually is the "F" LTL
   operator. Forever is the "G" LTL operator. "
   infinitely" specifies we must "eventually visit a
   and b forever". Examining each part, "eventually
   visit a forever" is "GFa". "eventually visit b
   forever" is "GFb". There are 2 landmarks so we are
   done. The answer is "GFa & GFb".

Q:
```

C.2.5 Formula Holdout CoT Prompt Fold 3

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "reach"
   specifies we must "eventually visit a". Examining
   each part, "eventually visit a" is "Fa". There is 1
    landmark so we are done. The answer is "Fa".

Q: What is "visit b and a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "visit"
```

specifies we must "eventually visit a and b". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". There are 2 landmarks so we are done. The answer is "Fa & Fb".

Q: What is "visit c a and b in no specific order" in LTL?
A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "visit" specifies we must "eventually visit a, b, and c". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc".

Q: What is "go to four locations a b d and c" in LTL?
A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "go to" specifies we must "eventually visit a, b, c, and d". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd".

Q: What is "sometime visit a c b h and d" in LTL?
A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "sometime visit" specifies we must "eventually visit a, b, c, d, and h". Examining each part, "

eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd & Fh".

Q: What is "visit a and then b" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must "visit a and then visit b, eventually". Working backwards, "eventually, visit b" is "Fb". "eventually, visit a and then visit b" is "F(a & Fb)". We reached the first landmark so we are done. The answer is "F(a & F(a & Fb))".

Q: What is "in some sequence visit a b and c in that order" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "in that order" specifies we must visit "a and then visit b and then visit c, eventually". Working backwards, "eventually, visit c" is "Fc". "eventually, visit b and then visit c" is "F(b & Fc)". "eventually, visit a and then b and then visit c" is "F(a & F(b & Fc))". We reached the first landmark so we are done. The answer is "F(a & F(b & Fc))".

Q: What is "visit a then visit b then visit c then visit d" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and then visit c and then visit d, eventually". Working backwards, "eventually, visit d" is "Fd". " eventually, visit c and then visit d" is "F(c & Fd)". "eventually, visit b and then c and then visit d" is "F(b & F(c & Fd))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & Fd)))".

Q: What is "visit a b c d and h sequentially" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. " sequentially" specifies we must visit "a and then visit b and then visit c and then visit d and then visit h, eventually". Working backwards, " eventually, visit h" is "F h". "eventually, visit d and then visit h" is "F(d & F h)". "eventually, visit c and then d and then visit h" is "F(c & F(d & Fh))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & F(d & Fh))))".

Q: What is "visit a then b then c and then d you can only visit each landmark once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only visit...once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, d ". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "eventually visit d" is "Fd ". "visit a only once" is (!a U (a U (!a U b))). " visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark d. The answer is "(!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))".

Q: What is "visit a then b then c then d and then h visit each landmark only once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, h". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". " visit d but not until we visit c" is "(!d U c)". " visit h but not until we visit d" is "(!h U d)". " eventually visit h" is "Fh". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark h. The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))".

Q: What is "find b but not until you find a first" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "until" specifies we only visit one landmark after visiting another. "We eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "eventually visit b" is "Fb". The answer is "(!b U a) & Fb".

Q: What is "go to a an infinite number of times" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinite" specifies we must "eventually visit a forever". Examining each part, "eventually visit a forever" is "GFa". There is 1 landmark so we are done. The answer is "GFa".

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL

operator. "only after" and "only then" specify we only visit one landmark after visiting another. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit d" is "Fd". The answer is "(!b U a) & (!c U b) & (!d U c) & Fd".

Q: What is "frequent a b and c infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. "infinitely" specifies we must "eventually visit a, b, and c forever". Examining each part, "eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc".

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. "

infinitely" specifies we must "eventually visit a, b, c, and d forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd".

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, d, and h forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd & GFh" .

Q: What is "go to a and b infinitely often" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a and b forever". Examining each part, "eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". There are 2 landmarks so we are done. The answer is "GFa & GFb".

```
Q:
```

C.2.6 Formula Holdout CoT Prompt Fold 4

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "reach"
   specifies we must "eventually visit a". Examining
   each part, "eventually visit a" is "Fa". There is 1
    landmark so we are done. The answer is "Fa".

Q: What is "visit b and a" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "visit"
   specifies we must "eventually visit a and b".
   Examining each part, "eventually visit a" is "Fa".
   "eventually visit b" is "Fb". There are 2 landmarks
    so we are done. The answer is "Fa & Fb".

Q: What is "go to four locations a b d and c" in LTL?
A: We must eventually visit some landmarks in any
   order. Eventually is the "F" LTL operator. "go to"
   specifies we must "eventually visit a, b, c, and d"
```

. Examining each part, "eventually visit a" is "Fa" . "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd".

Q: What is "sometime visit a c b h and d" in LTL?
A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. " sometime visit" specifies we must "eventually visit a, b, c, d, and h". Examining each part, " eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd & Fh".

Q: What is "visit a and then b" in LTL?
A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must "visit a and then visit b, eventually". Working backwards, "eventually, visit b" is "Fb". "eventually, visit a and then visit b" is "F(a & Fb)". We reached the first landmark so we are done. The answer is "F(a & F(a & Fb))".

Q: What is "visit a then visit b then visit c then visit d" in LTL?
A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and

then visit c and then visit d, eventually". Working backwards, "eventually, visit d" is "Fd". "eventually, visit c and then visit d" is "F(c & Fd)". "eventually, visit b and then c and then visit d" is "F(b & F(c & Fd))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & Fd)))".

Q: What is "visit a b c d and h sequentially" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "sequentially" specifies we must visit "a and then visit b and then visit c and then visit d and then visit h, eventually". Working backwards, "eventually, visit h" is "F h". "eventually, visit d and then visit h" is "F(d & F h)". "eventually, visit c and then d and then visit h" is "F(c & F(d & Fh))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & F(d & Fh))))".

Q: What is "in strictly this order visit a then eventually visit b and finally eventually c" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL

operator. "in strictly this order" specifies we only visit one landmark after visiting another. "We eventually visit c but not until we visit b and we visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "eventually visit c" is "Fc". The answer is "(!b U a) & (!c U b) & Fc".

Q: What is "go to a exactly once while avoiding b then go to b" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. "exactly once" specifies we visit a landmark only once. "We eventually visit b but not until we visit a. We visit each landmark only once except for the last landmark, b". Examining each part, "visit b but not until visit a" is "(!b U a)". "eventually visit b" is "Fb". "visit a only once" is (!a U (a U (!a U b))). There are no other landmarks remaining besides the last landmark b so we are done. The answer is "(!b U a) & Fb & (!a U (a U (!a U b)))".

Q: What is "visit a exactly once while avoiding b and c then visit b exactly once while avoiding c finally visit c" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, c". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "eventually visit c" is "Fc". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). There are no other landmarks remaining besides the last landmark c so we are done. The answer is "(!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))".

Q: What is "visit a then b then c and then d you can only visit each landmark once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we

visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only visit...once" specifies that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, d". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "eventually visit d" is "Fd". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark d. The answer is "(!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))".

Q: What is "find b but not until you find a first" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL

operator. "until" specifies we only visit one landmark after visiting another. "We eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)" . "eventually visit b" is "Fb". The answer is "(!b U a) & Fb".

Q: What is "go to a an infinite number of times" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinite" specifies we must "eventually visit a forever". Examining each part, "eventually visit a forever" is "GFa". There is 1 landmark so we are done. The answer is "GFa".

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" and "only then" specify we only visit one landmark after visiting another. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each

part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit d" is "Fd". The answer is "(!b U a) & (!c U b) & (!d U c) & Fd".

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" specify we only visit one landmark after visiting another. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit h" is "Fh". The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh".

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, and c forever". Examining each part, "eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc".

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, and d forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd".

Q: What is "go to a and b infinitely often" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a

```
    and b forever". Examining each part, "eventually
    visit a forever" is "GFa". "eventually visit b
    forever" is "GFb". There are 2 landmarks so we are
    done. The answer is "GFa & GFb".

Q:
```

C.2.7 Type Holdout CoT Prompt Fold 0

```
Your task is to translate English instructions into
    linear temporal logic (LTL) formulas.

Q: What is "reach a" in LTL?
A: We must eventually visit some landmarks in any
    order. Eventually is the "F" LTL operator. "reach"
    specifies we must "eventually visit a". Examining
    each part, "eventually visit a" is "Fa". There is 1
     landmark so we are done. The answer is "Fa".

Q: What is "visit b and a" in LTL?
A: We must eventually visit some landmarks in any
    order. Eventually is the "F" LTL operator. "visit"
    specifies we must "eventually visit a and b".
    Examining each part, "eventually visit a" is "Fa".
    "eventually visit b" is "Fb". There are 2 landmarks
     so we are done. The answer is "Fa & Fb".
```

Q: What is "visit c a and b in no specific order" in LTL?

A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "visit" specifies we must "eventually visit a, b, and c". Examining each part, "eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc".

Q: What is "go to four locations a b d and c" in LTL?

A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. "go to" specifies we must "eventually visit a, b, c, and d" . Examining each part, "eventually visit a" is "Fa" . "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd".

Q: What is "sometime visit a c b h and d" in LTL?

A: We must eventually visit some landmarks in any order. Eventually is the "F" LTL operator. " sometime visit" specifies we must "eventually visit a, b, c, d, and h". Examining each part, " eventually visit a" is "Fa". "eventually visit b" is "Fb". Repeat for the remaining parts. The answer is "Fa & Fb & Fc & Fd & Fh".

Q: What is "in strictly this order visit a then eventually visit b and finally eventually c" in LTL ?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "in strictly this order" specifies we only visit one landmark after visiting another. "We eventually visit c but not until we visit b and we visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)" . "visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". The answer is "(!b U a) & (!c U b) & Fc".

Q: What is "find b but not until you find a first" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "until" specifies we only visit one landmark after visiting another. "We eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)" . "eventually visit b" is "Fb". The answer is "(!b U a) & Fb".

Q: What is "go to a only after that go to b only then go to c and only then go to d" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" and "only then" specify we only visit one landmark after visiting another. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we eventually visit b but not until we visit a". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". Repeat for the remaining parts. "eventually visit d" is "Fd". The answer is "(!b U a) & (!c U b) & (!d U c) & Fd".

Q: What is "go to a and only go to b only after a is visited and then go to c only after a and b are both visited and then go to d only after a b and c are all visited and then go to h only after a b c and d are all visited" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. Eventually is the "F" LTL operator. "only after" specify we only visit one landmark after visiting another. "We eventually visit h but not until we visit d and we visit d but

```
    not until we visit c and we visit c but not until
    we visit b and we eventually visit b but not until
    we visit a". Examining each part, "visit b but not
    until visit a" is "(!b U a)". "visit c but not
    until visit b" is "(!c U b)". Repeat for the
    remaining parts. "eventually visit h" is "Fh". The
    answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d
    ) & Fh".

Q:
```

C.2.8 Type Holdout CoT Prompt Fold 1

```
Your task is to translate English instructions into
   linear temporal logic (LTL) formulas.

Q: What is "visit a and then b" in LTL?
A: We eventually visit some landmarks in a certain
   order. Eventually is the "F" LTL operator. "then"
   specifies we must "visit a and then visit b,
   eventually". Working backwards, "eventually, visit
   b" is "Fb". "eventually, visit a and then visit b"
   is "F(a & Fb)". We reached the first landmark so we
    are done. The answer is "F(a & F(a & Fb))".

Q: What is "in some sequence visit a b and c in that
   order" in LTL?
```

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "in that order" specifies we must visit "a and then visit b and then visit c, eventually". Working backwards, "eventually, visit c" is "Fc". "eventually, visit b and then visit c" is "F(b & Fc)". "eventually, visit a and then b and then visit c" is "F(a & F(b & Fc))". We reached the first landmark so we are done. The answer is "F(a & F(b & Fc))".

Q: What is "visit a then visit b then visit c then visit d" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. "then" specifies we must visit "a and then visit b and then visit c and then visit d, eventually". Working backwards, "eventually, visit d" is "Fd". " eventually, visit c and then visit d" is "F(c & Fd)". "eventually, visit b and then c and then visit d" is "F(b & F(c & Fd))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & Fd)))".

Q: What is "visit a b c d and h sequentially" in LTL?

A: We eventually visit some landmarks in a certain order. Eventually is the "F" LTL operator. " sequentially" specifies we must visit "a and then

visit b and then visit c and then visit d and then visit h, eventually". Working backwards, " eventually, visit h" is "F h". "eventually, visit d and then visit h" is "F(d & F h)". "eventually, visit c and then d and then visit h" is "F(c & F(d & Fh))". Keep doing this until we reach the first landmark. The answer is "F(a & F(b & F(c & F(d & Fh))))".

Q: What is "go to a exactly once while avoiding b then go to b" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit b but not until we visit a. We visit each landmark only once". Examining each part, "visit b but not until visit a" is "(!b U a)". "eventually visit b" is "Fb". "visit a only once" is (!a U (a U (!a U b))). The answer is "(!b U a) & Fb & (!a U (a U (!a U b)))".

Q: What is "visit a exactly once while avoiding b and c then visit b exactly once while avoiding c

finally visit c" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "while avoiding" and "then" specify we only visit one landmark after visiting another. " exactly once" specifies we visit a landmark only once. "We eventually visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once". Examining each part , "visit b but not until visit a" is "(!b U a)". " visit c but not until visit b" is "(!c U b)". " eventually visit c" is "Fc". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). The answer is "(!b U a) & (!c U b) & Fc & (!a U (a U (!a U b))) & (!b U (b U (!b U c)))".

Q: What is "visit a then b then c and then d you can only visit each landmark once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only visit...once" specifies

that we visit a landmark only once. Combining this logic, we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, d ". Examining each part, "visit b but not until visit a" is "(!b U a)". "visit c but not until visit b" is "(!c U b)". "visit d but not until we visit c" is "(!d U c)". "eventually visit d" is "Fd ". "visit a only once" is (!a U (a U (!a U b))). " visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark d. The answer is "(!b U a) & (!c U b) & (!d U c) & Fd & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d)))".

Q: What is "visit a then b then c then d and then h visit each landmark only once" in LTL?

A: We eventually visit some landmarks in a certain order while we cannot visit some landmarks until we visit others first. After we visit a landmark, we never visit it again. Eventually is the "F" LTL operator. "then" specifies we visit landmarks in a specific order and "only once" specifies that we visit a landmark only once. Combining this logic,

we only visit one landmark after visiting another and never go back to previously visited landmarks. "We eventually visit h but not until we visit d and we visit d but not until we visit c and we visit c but not until we visit b and we visit b but not until we visit a. We visit each landmark only once except for the last landmark, h". Examining each part, "visit b but not until visit a" is "(!b U a)" . "visit c but not until visit b" is "(!c U b)". " visit d but not until we visit c" is "(!d U c)". " visit h but not until we visit d" is "(!h U d)". " eventually visit h" is "Fh". "visit a only once" is (!a U (a U (!a U b))). "visit b only once" is (!b U (b U (!b U c))). Repeat for the remaining landmarks except for landmark h. The answer is "(!b U a) & (!c U b) & (!d U c) & (!h U d) & Fh & (!a U (a U (!a U b))) & (!b U (b U (!b U c))) & (!c U (c U (!c U d))) & (!d U (d U (!d U h)))".

Q: What is "go to a an infinite number of times" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinite" specifies we must "eventually visit a forever". Examining each part, "eventually visit a forever" is "GFa". There is 1 landmark so we are

done. The answer is "GFa".

Q: What is "frequent a b and c infinitely in any order " in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, and c forever". Examining each part, "eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc".

Q: What is "frequent a b c and d infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, and d forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd".

Q: What is "frequent a b c d and h infinitely in any order" in LTL?

A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a, b, c, d, and h forever". Examining each part, " eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". Repeat for the remaining parts. The answer is "GFa & GFb & GFc & GFd & GFh" .

Q: What is "go to a and b infinitely often" in LTL?
A: We must eventually visit some landmarks in any order and do so forever. Eventually is the "F" LTL operator. Forever is the "G" LTL operator. " infinitely" specifies we must "eventually visit a and b forever". Examining each part, "eventually visit a forever" is "GFa". "eventually visit b forever" is "GFb". There are 2 landmarks so we are done. The answer is "GFa & GFb".

Q:

Appendix D

Pattern Types

	Description	Example	Formula
Past avoidance	A condition has been fulfilled in the past.	If the robot enters location l_1, then it should have not visited location l_2 before. The trace $l_3 \to l_4 \to l_1 \to l_2 \to l_4 \to l_3 \to (l_2 \to l_3)^\omega$ satisfies the mission requirement since location l_2 is not entered before location l_1.	$(\neg(l_1))\,\mathcal{U}\,p$, where $l_1 \in L$ and $p \in M$
Global avoidance	An avoidance condition globally holds throughout the mission.	The robot should avoid entering location l_1. Trace $l_3 \to l_4 \to l_3 \to l_2 \to l_4 \to l_3 \to (l_3 \to l_2 \to l_3)^\omega$, satisfies the mission requirement since the robot never enters l_1.	$\mathcal{G}(\neg(l_1))$, where $l_1 \in L$
Future avoidance	After the occurrence of an event, avoidance has to be fulfilled.	If the robot enters l_1, then it should avoid entering l_2 in the future. The trace $l_3 \to l_4 \to l_3 \to l_1 \to l_4 \to l_3 \to (l_3 \to l_2 \to l_3)^\omega$ does not satisfy the mission requirement since l_2 is entered after l_1.	$\mathcal{G}((c) \to (\mathcal{G}(\neg(l_1))))$, where $c \in M$ and $l_1 \in PL$
Upper Rest. Avoidance	A restriction on the maximum number of occurrences is desired.	A robot has to visit l_1 at most 3 times. The trace $l_1 \to l_4 \to l_1 \to l_3 \to l_1 \to l_4 \to l_1 \to (l_3)^\omega$ violates the mission requirement since l_1 is visited four times. The trace $l_4 \to l_3 \to l_1 \to l_2 \to l_4 \to (l_3)^\omega$ satisfies the mission requirement.	$\neg\,\mathcal{F}(\underbrace{l_1 \wedge \mathcal{X}(\mathcal{F}(l_1 \wedge \ldots \mathcal{X}(\mathcal{F}(l_1))))}_{\text{n}})$, where $l_1 \in L$
Lower Rest. Avoidance	A restriction on the minimum number of occurrences is desired.	A robot to enter location l_1 at least 3 times. The trace $l_4 \to l_3 \to l_2 \to l_2 \to l_4 \to (l_3)^\omega$ violates the mission requirement since location 1 is never entered. The trace $l_1 \to l_4 \to l_3 \to l_1 \to l_4 \to l_1 \to (l_3)^\omega$ satisfies the mission requirement.	$\mathcal{F}(\underbrace{l_1 \wedge \mathcal{X}(\mathcal{F}(l_1 \wedge \ldots \mathcal{X}(\mathcal{F}(l_1))))}_{\text{n}})$, where $l_1 \in L$
Exact Rest. Avoidance	The number of occurrences desired is an exact number.	A robot must enter location l_1 exactly 3 times. The trace $l_4 \to l_3 \to l_2 \to l_2 \to l_4 \to (l_3)^\omega$ violates the mission requirement. The trace $l_1 \to l_4 \to l_3 \to l_1 \to l_4 \to l_1 \to (l_3)^\omega$ satisfies the mission requirement since location l_1 is entered exactly 3 times.	$\underbrace{(\neg(l1))\,\mathcal{U}(l1 \wedge (\mathcal{X}((\neg l1)\,\mathcal{U}(l1 \ldots \wedge (\mathcal{X}((\neg l1)\,\mathcal{U}(l1}_{\text{n}}$ $\wedge(\mathcal{X}(\mathcal{G}(\neg l1))))))))))$, where $l_1 \in L$
Inst. Reaction	The occurrence of a stimulus instantaneously triggers a counteraction.	When location l_2 is reached action a must be executed. The trace $l_1 \to l_3 \to \{l_2, a\} \to \{l_2, a\} \to l_4 \to (l_3)^\omega$ satisfies the mission requirement since when location l_2 is entered condition a is performed. The trace $l_1 \to l_3 \to l_2 \to \{l_1, a\} \to l_4 \to (l_3)^\omega$ does not satisfy the mission requirement since when l_2 is reached a is not executed.	$\mathcal{G}(p_1 \to p_2)$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Delayed Reaction	The occurrence of a stimulus triggers a counteraction some time later	When c occurs the robot must start moving toward location l_1, and l_1 is subsequently finally reached. The trace $l_1 \to l_3 \to \{l_2, c\} \to l_1 \to l_4 \to (l_3)^\omega$ satisfies the mission requirement, since after c occurs the robot starts moving toward location l_1, and location l_1 is finally reached. The trace $l_1 \to l_1 \to \{l_2, c\} \to l_3 \to (l_3)^\omega$ does not satisfy the mission requirement since c occurs when the robot is in l_2, and l_1 is not finally reached.	$\mathcal{G}(p_1 \to \mathcal{F}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Prompt Reaction	The occurrence of a stimulus triggers a counteraction promptly, i.e. in the next time instant.	If c occurs l_1 is reached in the next time instant. The trace $l_1 \to l_3 \to \{l_2, c\} \to l_1 \to l_4 \to (l_3)^\omega$ satisfies the mission requirement, since after c occurs l_1 is reached within the next time instant. The trace $l_1 \to l_3 \to \{l_2, c\} \to l_4 \to l_1 \to (l_3)^\omega$ does not satisfy the mission requirement.	$\mathcal{G}(p_1 \to \mathcal{X}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Bound Reaction	A counteraction must be performed every time and only when a specific location is entered.	Action a_1 is bound though a delay to location l_1. The trace $l_1 \to l_3 \to \{l_2, c\} \to \{l_1, a_1\} \to l_4 \to \{l_1, a_1\} \to (l_3)^\omega$ satisfies the mission requirement. The trace $l_1 \to l_3 \to \{l_2, c\} \to \{l_1, a_1\} \to \{l_4, a_1\} \to \{l_1, a_1\} \to (l_3)^\omega$ does not satisfy the mission requirement since a_1 is executed in location l_4.	$\mathcal{G}(p_1 \leftrightarrow p_2)$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Bound Delay	A counteraction must be performed, in the next time instant, every time and only when a specific location is entered	Action a_1 is bound to location l_1. The trace $l_1 \to l_3 \to \{l_2, c\} \to \{l_1\} \to \{l_4, 1_1\} \to \{l_1\} \to \{l_4, a_1\} \to (l_3)^\omega$ satisfies the mission requirement. The trace $l_1 \to l_3 \to \{l_2, c\} \to \{l_1\} \to \{l_4, 1_1\} \to \{l_1, a_1\} \to \{l_4\} \to (l_3)^\omega$ does not satisfy the mission requirement.	$\mathcal{G}(p_1 \leftrightarrow \mathcal{X}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Wait	Inaction is desired till a stimulus occurs.	The robot remains in location l_1 until condition c is satisfied. The trace $l_1 \to l_3 \to \{l_2, c\} \to l_1 \to l_4 \to (l_3)^\omega$ violates the mission requirement since the robot left l_1 before condition c is satisfied. The trace $l_1 \to \{l_1, c\} \to l_2 \to l_1 \to l_4 \to (l_3)^\omega$ satisfies the mission requirement.	$(l_1)\,\mathcal{U}(p)$, where $l_1 \in L$ and $p \in PE \cup PA$

Figure D.1: Avoidance and Trigger Patterns taken from [16].

Bibliography

[1] M. Berg, D. Bayazit, R. Mathew, A. Rotter-Aboyoun, E. Pavlick, and S. Tellex. Grounding language to landmarks in arbitrary outdoor environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 208–215, 2020.

[2] M. Berg, D. Bayazit, R. Mathew, A. Rotter-Aboyoun, E. Pavlick, and S. Tellex. Grounding language to landmarks in arbitrary outdoor environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 208–215, 2020.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[4] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov, E. H. Chi,

J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei. Scaling instruction-finetuned language models, 2022.

[5] S. Diao, X. Li, Y. Lin, Z. Huang, and T. Zhang. Black-box prompt learning for pre-trained language models. *CoRR*, abs/2201.08531, 2022.

[6] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. Lauko. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, Aug. 2022.

[7] N. Gopalan, D. Arumugam, L. L. S. Wong, and S. Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. *Robotics: Science and Systems XIV*, 2018.

[8] N. Gopalan, D. Arumugam, L. L. S. Wong, and S. Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. *Robotics: Science and Systems XIV*, 2018.

[9] J. Gu, Z. Lu, H. Li, and V. O. K. Li. Incorporating copying mechanism in sequence-to-sequence learning. *CoRR*, abs/1603.06393, 2016.

[10] K. Guu, P. Pasupat, E. Z. Liu, and P. Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood, 2017.

[11] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751, 2019.

[12] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners, 2023.

[13] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. *CoRR*, abs/2104.08691, 2021.

[14] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190, 2021.

[15] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. GPT understands, too. *CoRR*, abs/2103.10385, 2021.

[16] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger. Specification patterns for robotic missions. *CoRR*, abs/1901.02077, 2019.

[17] A. Pnueli. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977.

[18] R. M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *CoRR*, abs/1912.05911, 2019.

[19] R. Shin, C. H. Lin, S. Thomson, C. Chen, S. Roy, E. A. Platanios, A. Pauls, D. Klein, J. Eisner, and B. V. Durme. Constrained language models yield few-shot semantic parsers. *CoRR*, abs/2104.08768, 2021.

[20] L. R. Tang and R. J. Mooney. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141, Hong Kong, China, Oct. 2000. Association for Computational Linguistics.

[21] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford$_a$*lpaca*, 2023.

[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[24] C. Wang, C. Ross, Y. Kuo, B. Katz, and A. Barbu. Learning a natural-language to LTL executable semantic parser for grounded robotics. *CoRR*, abs/2008.03277, 2020.

[25] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022.

[26] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[27] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *CoRR*, abs/1207.1420, 2012.