Proc2PDDL: Predicting Domain Definitions Based on Natural Language for Symbolic Planning

Anonymous EMNLP submission

Abstract

A symbolic planner such as a PDDL solver produces an executable and interpretable plan, based on a domain file and a problem file. Prior work assumes provided domain files, which are costly to create in practice. This paper breaks the assumption by introducing the domain file action prediction problem where preconditions and effects of each action are predicted. Towards this, we propose the first dataset containing open-domain procedural articles from wikiHow paired with annotated PDDL representations. We then show that LLMs are capable of generating plausible PDDL actions, but more than half are incorrect, resulting in failure to solve problems. We provide an in-depth error analysis of why LLMs fail, and are still far lagging behind humans.

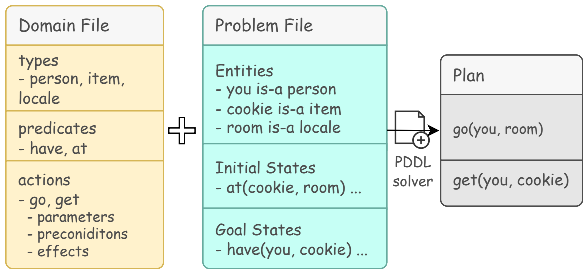

Figure 1: A PDDL solver produces a plan based on a minimal domain file and problem file. Previous work assumes the domain file as given, while we predict the action definitions in the domain file.

1 Introduction

Planning is the task of finding a sequence of actions to achieve a goal in a given environment (Fikes and Nilsson, 1971; LaValle, 2006). For both humans and machines, planning is critical for solving problems such as math questions, multi-hop reasoning questions, or even high-level problems such as cooking, litigation, or policy-making. To assist problem-solving for both humans and machines, researchers have explored ways to create plans either expressed as natural language (NL) (Sakaguchi et al., 2021; Lyu et al., 2021) or symbolic language (SL) (Silver et al., 2022; Huang et al., 2022, 2023; Lin et al., 2023). SL plans have many advantages over NL plans. First, the ambiguous and highly-granular nature of NL instructions results in a lack of tractability and interpretability, while the predefined symbolic patterns (e.g., <pick, subject, object>) makes SL plans easy to validate, explain, and revise. Second, SL plans are necessary for machines to execute, while NL plans without any grounding fail in this regard.

In this work, we treat symbolic planning as the task of translating procedural text to symbolic representations using large language models (LLMs) and create the plan with a symbolic solver. This approach, as opposed to generating SL plans by LLMs directly, has been shown to be more effective and faithful (Lyu et al., 2023; Xie et al., 2023; Liu et al., 2023). Following the past research, we employ PDDL (Planning Domain Definition Language) (Aeronautiques et al., 1998). PDDL planners require one domain file and at least one problem file (see an example in Figure 1). The domain file describes the universal context and rules of the environment, while the problem file defines the initial and goal states. Given a domain file and a problem file, a PDDL solver will attempt to produce a plan, namely a sequence of actions, to achieve the specified goal states.

There has been a growing interest in using LLMs to generate PDDL. Existing work (Lyu et al., 2023; Xie et al., 2023; Liu et al., 2023) assumes the availability of a complete domain file and a partial problem file including the initial states, while a model *translates* a textual query into a PDDL-representation of the goal states. Due to the high cost of creating such domain files, the scope is often closed-domain, such as simulations (Puig et al., 2018; Shridhar et al., 2020; Wang et al., 2022; Park et al., 2023). In contrast, we are interested in generating a **domain file** based on natural language

Formulation	$\mathbb{PF}$ prediction (previous work)	$\mathbb{DF}$-action prediction **(this work)**	full $\mathbb{DF}$ prediction (our ongoing work)	$\mathbb{PF}$ and $\mathbb{DF}$ prediction (our ongoing work)
Assumes	$\mathbb{T}$, $\mathbb{DF}$ (H,A), partial $\mathbb{PF}$	$\mathbb{T}$, H, $\mathbb{PF}$ for eval	$\mathbb{T}$, $\mathbb{PF}$ for eval	$\mathbb{T}$
Predicts	goal states in $\mathbb{PF}$	A	H, A	H, A, $\mathbb{PF}$
Scenario	A robot has full access to an env. and actions.	A robot is in an unfamiliar env.; does not know how its actions affect the env.	The env. is ungrounded with descriptions to carry out known tasks.	Nothing is grounded; tasks are undefined.
Difficulty	*	**	***	****
Well-defined	****	***	**	*

Table 1: Possible ways of formulating the task of translating NL to PDDL. Previous work assumes the $\mathbb{DF}$ and partial $\mathbb{PF}$, merely predicting the goal states in the $\mathbb{PF}$. In contrast, we predict action definitions in the $\mathbb{DF}$ based on procedural texts.

descriptions of any open-domain environment. See Appendix A for a comprehensive comparison with related work.

To this end, we propose a dataset coined PROC2PDDL consisting of 27 domain files and 81 problem files manually annotated based on open-domain WikiHow articles. An LLM is then provided with textual descriptions along with the domain file scaffolding, and predicts the actions in the domain file. We experiment on various prompt designs, prompting the LLM with different level of procedural contexts from whole articles to sentence-long summarizations, while considering methods such as chain-of-thought (Nye et al., 2021). Our evaluation shows that the latest LLMs can plan based on our formulation, despite previous claims that they cannot (Valmeekam et al., 2022).

2 Task

A PDDL example contains a domain file $\mathbb{DF}$ and one or more problem files $\mathbb{PF}$.

A $\mathbb{DF}$ defines the following elements:

- a header H, which consists of
  - types of entities (e.g., *object*, *location*, *player*)
  - predicates (e.g., if object is *at* a location)
  - names of possible actions (e.g., *boil water*)
- definitions of actions A, which consist of
  - parameters (e.g., water, pot) as a list of types
  - precondition (e.g., water and pot belongs to player; water is not treated) as a conjunctive normal form of predicates
  - effect (e.g., water is treated) as a conjunctive normal form of predicates

A $\mathbb{PF}$ defines the following elements:

- objects and their type (e.g., rainwater is water)
- initial states (e.g., bucket is empty)
- goal states (e.g., bucket is filled with rainwater; rainwater is treated)

We say that a $\mathbb{DF}$ can *solve* a $\mathbb{PF}$ if there exists a sequence of actions $A_1, \ldots, A_n$ that propels the object states to transition from initial to goal.

We are concerned with translating some procedural text $\mathbb{T}$ to a $\mathbb{DF}$. A successfully generated $\mathbb{DF}$ can thus solve $\mathbb{PF}$s defined accordingly. As shown in Table 1, different components of PDDL can be predicted. Among them, we focus on predicting action definitions A in the $\mathbb{DF}$. With the types and predicates H specified, predicted A can be expected to be consistent with the naming convention in the $\mathbb{PF}$, leading to a well-defined evaluation, A typical approach is shown in Figure 2.

3 Dataset

We propose the PROC2PDDL dataset of 27 different $\mathbb{T}$-$\mathbb{DF}$-$\mathbb{PF}$s tuples, drawing procedural texts from WikiHow articles of various topics (see Appendix B). A class of graduate students in a U.S. university with prior knowledge on PDDL are each given a WikiHow article $\mathbb{T}$ and annotate a $\mathbb{DF}$ and multiple corresponding $\mathbb{PF}$s from the article, each with a gold plan to solve it. Each $\mathbb{T}$ consists of step paragraphs that may or may not be used in defining the actions in the $\mathbb{DF}$. Hence, a mapping between actions and steps is also annotated. On average, there are 13.33 defined actions in a $\mathbb{DF}$ and 8.07 instantiated actions in a gold plan.

We partition the 27 examples into a 5:6:16 train-development-test splits. In this work, the train split is unused as all our methods are zero-shot; only the development set is used for error analysis; the test set is strictly held out for evaluation.

4 Method

To predict action definitions A in $\mathbb{DF}$ based on the header H and a wikiHow article $\mathbb{T}$ containing steps, we prompt `gpt-4-32k` in a zero-shot manner (for prompt details and examples, see Appendix C). We divide our approach into three sequential stages:

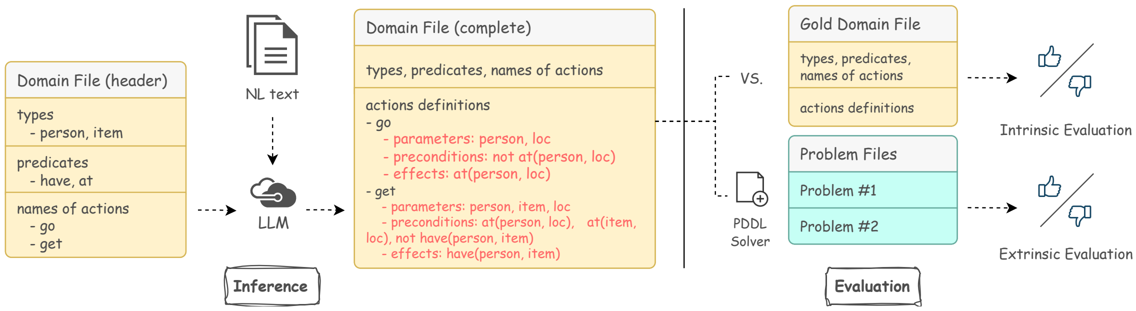

Figure 2: Our approach to the $\mathbb{DF}$-action prediction task. Given types, predicates, and action names in a domain file, our model predicts action definitions including parameters, preconditions, and effects based on textual descriptions. The predicted domain file is compared with the gold one, and used to solve the corresponding problem files.

First, **Identification** of the action relevant steps in a wikiHow article.
Second, **Extraction** of entities and states from those steps. This also needs inference on implicit entity states (e.g., a cloth gets wet if soaked).
Third, **Translation** of said entity states to PDDL. This requires paraphrasing of entity states in NL to given predicates in SL (e.g., a cloth getting soaked might translate to `(submerged ?cloth)`).

We consider the following families of prompts.
Prompt without text (w/o $\mathbb{T}$) is an ablation baseline where the model predicts A solely based on H. Naturally, none of the three aforementioned stages are involved.

Prompt with text (w/ $\mathbb{T}$) additionally provides the model with four different portions of $\mathbb{T}$, involving the three aforementioned stages, as follows:
($\mathbb{T}$ = all) all steps in a wikiHow article
($\mathbb{T}$ = rel) relevant steps to all actions in a $\mathbb{DF}$ based on the annotated mapping in PROC2PDDL
(e.g., 1. Find fresh water... 2. Collect food...
7. Set up camp...)
($\mathbb{T}$ = map) each action mapped with steps based on the annotated mapping in PROC2PDDL
(e.g., clean_water: 1. Find fresh water...)
($\mathbb{T}$ = sum) one-line summaries of actions annotated in PROC2PDDL
(e.g., clean_water; boil water to clean it)
The four prompts are increasingly more brief and less coherent in supplementary text. The former ones demand accurate information extraction, while, the later one brings model challenges to infer implicit entity states.

Prompt with text, chain-of-thought (text+CoT) explicitly has the model following our proposed three stages predict: 1. a summary of the action and the result, 2. the needed entities, their states before and after the action, 3. their PDDL representation.

Model %	Internal action acc.	External $\mathbb{PF}$ solve	External exact plan
w/o $\mathbb{T}$ (baseline)	13.7	26.3	3.2
$\mathbb{T}$ =sum	15.9	33.7	4.2
$\mathbb{T}$ =sum, CoT	**18.1**	**35.8**	**6.3**
$\mathbb{T}$ =map	11.8	13.7	2.1
$\mathbb{T}$ =map, CoT	8.9	26.3	1.1
$\mathbb{T}$ =rel	11.6	27.4	0.0
$\mathbb{T}$ =rel, CoT	12.2	21.1	4.2
$\mathbb{T}$ =all	12.1	28.4	0.0
$\mathbb{T}$ =all, CoT	12.1	31.6	0.0

Table 2: Performance of the $\mathbb{DF}$-action prediction on the concatenation of the development and test set of PROC2PDDL. Metrics include action-wide accuracy, average edit distance of action definitions, the proportion of $\mathbb{PF}$s that can be solved, and the proportion of generated plans that exactly match the gold plans.

This form provides the model a scaffold to achieve the task, and makes it more attentive to the entities and changes in entity states, even implicit ones (e.g., a soaking action causes an entity to go from dry to wet). Thus, the model is facilitated to think about the conditions completely.

5 Evaluation and Analysis

Now that a model generates the parameters, preconditions, and effects of actions A, we have a complete $\mathbb{DF}$. We evaluate it in two ways (Figure 2). **Intrinsically**, we semantically compare the predicted A with the ground-truth provided by our PROC2PDDL and report an action-wide accuracy, where equivalence of two action definitions does not depend on the naming of variables and the order within conjunctions (see Appendix D). **Extrinsically**, to measure actions' coherence, we use a BFS-based PDDL solver[1] to attempt to solve

[1] https://github.com/pucrs-automated-planning/pddl-parser

	Unsolved			Solved	
	Syntax Error	Bad Action	Good Action	Bad Plan	Good Plan
$\mathbb{T}$ =sum	3	7	2	0	3
$\mathbb{T}$ =all	0	10	0	3	2

Table 3: Statistics of error types on the development set.

ground-truth $\mathbb{PF}$s with the predicted $\mathbb{DF}$ and report a success rate. An unsolved $\mathbb{PF}$ is caused by (1.) no plan can be found, or (2.) the solver runs for more than 30 seconds, or (3.) the solver returns an error (usually a syntax error in generated PDDL).

5.1 Evaluation

The results (Intrinsic & Extrinsic) on the dev & test set are in Table 2. In w/o text setting, the model already has a good knowledge of PDDL syntactically and semantically. Using a sentence-long description for each action provided by PROC2PDDL, the model achieves the best performance among all, showing a strong deduction ability with the limited but precise NL input. In contrast, longer and more coherent texts (**`all/rel/map`**) lead to worse results, indicating its extraction shortage in a long context. This shortage is less from extracting relevant entities (e.g., fish, spear in hunt_fish), but more from extracting the relation between actions (e.g., make_spear to hunt_fish) which may be explicitly expressed in PROC2PDDL annotation. CoT, overall, is helpful since it explicitly spells out many implicit entities and state changes (see example outputs of w/ and w/o CoT in Appendix C). Thus, the improvement is most salient in **`sum`** where higher inference ability is desired. However, even with CoT, there are cases of identified entity states being ignored in the translation stage, likely because of the task complexity. We also notice CoT leads to omitted actions in longer outputs. To emphasize the simplicity of the task, and the brevity and coherence of NL text, an obvious next step is to separate our joint model into a two-episode pipeline: first summarize action, entity, and states, then translate into PDDL.

5.2 Error Analysis

To provide deeper insights into model performance, we manually inspect the model output of 2 best-performing prompts (**`sum`** and **`all`**) of all 6 examples (18 $\mathbb{PF}$s) in the development set. We consider the following scenarios.

Syntax Error Model output may contain illegal expressions that cannot be parsed. For example, `(inventory ?player (clean ?strips))` is unacceptable because the arguments to a predicate must be atomic types, not another predicate.

Unsolved In case that the predicted $\mathbb{DF}$ cannot solve a $\mathbb{PF}$, we identify the first problematic action that differs with the ground-truth. For example, if the action `cut_plant` misses a critical effect of `(inventory ?player ?stalk)`, then other actions such as `graft_stalk` requiring it cannot be executed. However, at times, there could be false negatives where the predicted action definitions are in fact reasonable, but nonetheless cannot lead to a solution.

Solved The predicted $\mathbb{DF}$ may solve a $\mathbb{PF}$, but the plan may be different from the gold plan. It is naturally possible that the predicted plan is a fluke made possible by under-specified preconditions or over-exaggerated effects, as well as loopholes in the $\mathbb{PF}$ leading to unreasonable shortcuts. For the example in Figure 1, a model could *cheat* by defining the action `get` by not requiring the person and object to be in the same location; thus, the predicted plan would unreasonably omit the action `go`. However, at times, the predicted plan could also be a reasonable alternative.

The statistics of these errors made by all prompts on the development set is shown in Table 3. When no solution can be found, true negative is highly likely as the model indeed makes aforementioned mistakes during action prediction. When some solution is found, false positive is still possible as the predicted plan may be unreasonable. See attached materials for a complete error analysis of these examples. Our aforementioned future pipeline that separates summarization and translation would likely mitigate these errors.

6 Conclusion

We propose PROC2PDDL, the first dataset that pairs procedural texts with annotated PDDL representations. We are the first to attempt open-domain $\mathbb{DF}$-action prediction as a means to symbolic planning with LLMs, transcending previous works' restriction of only predicting the goal states in the problem file. We show that state-of-the-art LMs are capable of the task but still have a large room for improvement. This work paves the path for more ambitious settings such as predicting both full $\mathbb{PF}$ and $\mathbb{DF}$; leading to a neuro-symbolic automatic planning that is verifiable through a PDDL solver.

Limitations

Any planning language, including PDDL that we consider in this work, is an approximation of planning in the real world and cannot accurately reflect its complexity. Due to the consideration for simplicity in the annotation process, we use the primitive version of PDDL instead of newer planning languages, with restricted expressions and syntax.

Annotating PROC2PDDL is extremely costly as it requires knowledge of PDDL and much effort to translate procedural texts to PDDL. Thus, our dataset is relatively small with a limited range of topics. Due to the highly complex and subjective nature of the annotation process, each annotated example may reflect idiosyncratic though processes and biases of the individual annotator.

As many similar works, there is a known gap between high-level planning such as ours (with high-level actions like "boil) and the present robots (with low-level motor functions like "move"). However, like similar works, we believe our efforts can see more practical application in the near future.

Our modeling efforts so far have mainly considered options of zero-shot prompting. There of course exists many other approaches even in a few-shot setting, such as a big-model-teaches-small-model paradigm, that we plan to experiment with in the future. Moreover, our evaluation is imperfect in that even a well-annotated DF-PF pair might have multiple successful plans. Manual inspection is still necessary to accurately gauge models.

Ethics statement

Does not apply.

References

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. 1998. Pddl| the planning domain definition language. Technical Report, Tech. Rep.

Richard E Fikes and Nils J Nilsson. 1971. Strips: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2(3-4):189–208.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In International Conference on Machine Learning, pages 9118–9147. PMLR.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. 2023. Grounded decoding: Guiding text generation with grounded models for robot control. arXiv preprint arXiv:2303.00855.

Steven M LaValle. 2006. Planning algorithms. Cambridge university press.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2023. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. arXiv preprint arXiv:2305.17390.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+p: Empowering large language models with optimal planning proficiency.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning.

Qing Lyu, Li Zhang, and Chris Callison-Burch. 2021. Goal-oriented script construction. In Proceedings of the 14th International Conference on Natural Language Generation, pages 184–200, Aberdeen, Scotland, UK. Association for Computational Linguistics.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. arXiv preprint arXiv:2112.00114.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 8494–8502.

Keisuke Sakaguchi, Chandra Bhagavatula, Ronan Le Bras, Niket Tandon, Peter Clark, and Yejin Choi. 2021. proScript: Partially ordered scripts generation. In Findings of the Association for Computational Linguistics: EMNLP 2021, pages 2138–2149, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10740–10749.

Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. Pddl planning with pretrained large language models. In NeurIPS 2022 Foundation Models for Decision Making Workshop.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). arXiv preprint arXiv:2206.10498.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? arXiv preprint arXiv:2203.07540.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023. Translating natural language to planning goals with large-language models.

A Comparison of Related Work

As discussed extensively before, our work can be differentiated with related works in three regards (see Table 4) in the context of using LLMs for planning.[2] First, a family of works focus on generating NL plans, where steps are sentences; the ungrounded nature of these plans leads to a lack of executability and interpretability, as a trade-off for NL's flexibility in open-domain texts. Second, another family of works focus on generating SL plans directly using LLMs[3]; the black-box nature of LLMs leads to a lack of interpretability and faithfulness of how the plan is arrived at. Third, the most recent family of works, all contemporaneous, do not use LLMs to generate plans, but instead translate queries to planning specifications such as PF in PDDL. While we identify with the "translation" approach, we break out of their assumption that the rules of the environment, namely DF, are given. In contrast, we translate descriptions of the environment, namely procedural texts, to DF, while evaluating generated DFs using PFs.

Additionally, our work may be the first of its kind, or at least one of a few, to study open-domain symbolic planning, made possible by LLMs.

[2] Note that non-LLM methods for planning are historic and abundant. However, the potency of LLMs has recently been shown great potential in planning.

[3] Regardless of whether the output directly resembles some SL, or the output is some structured NL that is later converted to SL, the end-goal is the same – to generate SL plans.

B WikiHow Topics

create secret society
throw an anime party
open a coconut
calculate pi by throwing frozen hot dogs
hack
get out of quicksand
make a detective kit
lock picking
make papyrus
survive on a desert island
survive in the jungle
survive a war
survive a comet hitting earth
survive a nuclear attack
survive in the woods
survive deserted island
survive shark attack
survive emp attack

C Prompts

C.1 Prompt without text (w/o $\mathbb{T}$)

Prompt:
could you fill out the below pddl actions with the predicates?
All fields: parameters, precondition and effect, should have predicates.
For each action, do NOT change the name and do NOT drop the action and do NOT add more actions.
The output should be in correct pddl format.

here are the actions I want:
<insert_action_names>

here are the requirements I have:
<insert_requirements>

here are the types I have:
<insert_types>

here are the predicates I have:
<insert_predicates>

Example Completion:
(:action clean_water
:parameters (?player - human ?water - water)
:precondition (inventory ?player ?water)
:effect (treated ?water)
)

	How to plan	Dataset	Domain
Ours	LM translates to **DF**	WikiHow	**Open**
(Lyu et al., 2023)	LM translates to PF	SayCan	Closed/simulated
(Xie et al., 2023)	LM translates to PF	Blocksworld, Alfred	Closed/simulated
(Liu et al., 2023)	LM translates to PF	Blocksworld, etc.	Closed/simulated
(Huang et al., 2023)	LM generates SL plan	Tabletop rearrangement	Closed/simulated/real-word
(Huang et al., 2022)	LM generates SL plan	VirtualHome	Closed/simulated
(Silver et al., 2022)	LM generates SL plan	Blocksworld, etc.	Closed/simulated
(Valmeekam et al., 2022)	LM generates SL plan	Blocksworld	Closed/simulated
(Lyu et al., 2021)	LM generates NL plan	WikiHow	Open
(Sakaguchi et al., 2021)	LM generates NL plan	proScript	Open

Table 4: Comparison with related works.

C.2 Prompt with text (part/whole)

Prompt:
could you fill out the below pddl actions with the predicates based on the text?
All fields: parameters, precondition and effect, should have predicates.
For each action, do NOT change the name and do NOT drop the action and do NOT add more actions.
The output should be in correct pddl format.

here are the actions I want:
<insert_action_names>

here are the requirements I have:
<insert_requirements>

here are the types I have:
<insert_types>

here are the predicates I have:
<insert_predicates>

here are the texts containing steps to <insert_goal>:
<insert_text>[4]

Example Completion:
(:action clean_water
:parameters (?player - human ?water - water)
:precondition (inventory ?player ?water)
:effect (treated ?water)
)

C.3 Prompt with text (pair/desc)

Prompt:
could you fill out the below pddl actions with the predicates based on the text?
All fields: parameters, precondition and effect, should have predicates.
For each action, do NOT change the name and do NOT drop the action and do NOT add more actions.
The output should be in correct pddl format.

here are the action-text pairs I have to <insert_goal>:
<insert_action_text_pairs>

here are the requirements I have:
<insert_requirements>

here are the types I have:
<insert_types>

here are the predicates I have:
<insert_predicates>

C.4 Prompt with text and CoT (text+CoT)

Prompt:
could you fill out the below pddl actions with the predicates based on the text? All fields: parameters, precondition and effect, should have predicates.
For each action, do NOT change the name and do NOT drop the action and do NOT add more actions and:
First, summarize the action in a few sentences based on the text and provide its requirements and its aims if it has.
Next, identify ALL the entities involved in the action and describe whether it changed, unchanged, added, removed in the action in natural language.
Last, translate it into pddl format. Check all the related entities are in the 'parameters'.

Please use this output format:

[4] bold text is distinguished in different prompts

- action name: ...
- summarize action: ...
- what is need to do the action: ...
- what is the result: ...

- entities:
- entity name: ...
- before: ...
- after: ...
...

- describe how to match it to pddl relevant predicates step by step:
1. ...
2. ...

pddl:

here are the actions I want:
<insert_action_names>

here are the requirements I have:
<insert_requirements>

here are the types I have:
<insert_types>

here are the predicates I have:
<insert_predicates>

here are the texts containing steps to <insert_goal>:
<insert_text>

Example Completion:
- action name: clean_water
- summarize action: The player cleans water in their inventory using heat from a fire.
- what is needed to do the action: The player must have untreated water in their inventory and be at a location with fire.
- what is the result: The player has treated water in their inventory.

- entities:
- entity name: player
- before: Having untreated water in inventory.
- after: Having treated water in inventory.
- entity name: water
- before: Untreated.
- after: Treated.

- describe how to match it to pddl relevant predicates step by step:
1. Check if the player has untreated water in their inventory.
2. Check if the player is at a location with a fire.
3. Replace untreated water with treated water in the player's inventory in the effect.

pddl:
(:action clean_water
:parameters (?player - human ?loc - location ?water - water)
:precondition (and (at ?player ?loc) (inventory ?player ?water) (not (treated ?water)) (has_fire ?loc))
:effect (treated ?water)
)

D Calculating Actions Equivalence

The distance between two actions can be divided to two parts:

1. The distance between parameters:

   We don't need to care about the specific parameter names; we only need to consider the parameter types. For each parameter in Action1, we iterate over the parameter list of Action2 to find the first parameter in Action2 with the same type. We use two hash maps, p1 and p2, to record these two parameters and their corresponding types. We increment the counter by 1, remove that parameter from the parameter list of Action2, and break from the current loop. After the iteration, we obtain the number of matching parameters, n. The distance between parameters can be calculated as $|\text{number of parameters in Action1} - n| + |\text{number of parameters in Action2} - n|$.

2. The distance between preconditions/effects:

   For each condition in Action1, we iterate over the condition list of Action2. The conditions can only match if they have the same predicate and the same number of parameters. We iterate over the parameters in these conditions and make the following judgments:

   - If neither of the two current parameters has appeared before (in p1 and p2) and these parameters are not identical, they don't match.
   - If the two parameters have different categories, they don't match.

- If the two parameters have the same categories and don't have an index, we consider them as the same parameter entity and give them the same index. We continue the iteration.
- If the two parameters already have indexes, we check if the indexes are equal. If they are not equal, they don't match. Otherwise, we continue the iteration.
- In any other case, they don't match.

If all parameters of the two conditions match, we increment n by 1. The distance between preconditions/effects can be calculated as $|\text{number of preconditions/effects in Action1} - n| + |\text{number of preconditions/effects in Action2} - n|$.