Addressing the Challenges of Planning Language Generation

Prabhu Prakash Kagitha **Andrew Zhu** **Li Zhang**
Drexel University   University of Pennsylvania
prabhuprakash.k@drexel.edu   harry.zhang@drexel.edu

Abstract

Using LLMs to generate formal planning languages such as PDDL that invokes symbolic solvers to deterministically derive plans has been shown to outperform generating plans directly. While this success has been limited to closed-sourced models or particular LLM pipelines, we design and evaluate 8 different PDDL generation pipelines with open-source models under 50 billion parameters previously shown to be incapable of this task. We find that intuitive approaches such as using a high-resource language wrapper or constrained decoding with grammar decrease performance, yet inference-time scaling approaches such as revision with feedback from the solver and plan validator more than double the performance.[1]

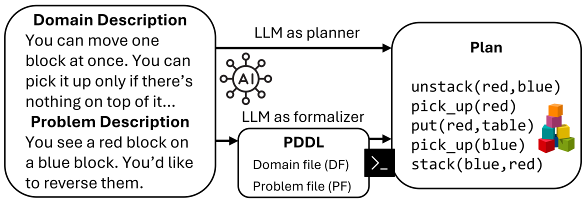

Figure 1: An illustration of using LLM as a planner or a formalizer in planning.

1 Introduction

Recently, Large Language Models (LLMs) have been extensively applied to planning tasks. Prominently, LLMs are given a description of the planning domain and problem, and are utilized as planners to directly *generate* a plan (Parmar et al., 2025; Majumder et al., 2023; Silver et al., 2024), or as formalizers to generate a formal language that is input into a formal solver to *calculate* a plan (Li et al., 2024; Hu et al., 2025; Zuo et al., 2024; Zhang et al., 2024a,b). LLM-as-formalizer (Figure 1) has been widely advocated in literature due to its reportedly better performance and formal guarantees compared to LLM-as-planner.

Although LLM-as-formalizer could be instantiated with several planning languages including satisfiability modulo theories (SMT) (Hao et al., 2025), linear temporal logic (LTL) (Li et al., 2024), Answer Set Programming (Lin et al., 2024), among others (Ishay and Lee, 2025; Guo et al., 2024), we follow most work and focus on the planning domain definition language (PDDL) (Li et al., 2024; Hu et al., 2025; Zuo et al., 2024; Zhang et al., 2024a,b) due to its dominant popularity, though our experiments apply to any language. Previous work evaluating PDDL generation focused primarily on closed-course and huge LLMs over 100 billion parameters such as gpt-4o (OpenAI et al., 2024) or DeepSeek-R1 (DeepSeek-AI et al., 2025) using some particular LLM pipeline. Moreover, much work concluded even large, closed-source models have limited ability to generate syntactically and semantically correct PDDL due to its specificity and lack of training data (Huang and Zhang, 2025; Zuo et al., 2024), while small, open-source models achieve close to zero performance. This greatly hinders progress in automatic planning.

This work is the first to evaluate mid-size open-source LLMs less than 50 billion parameters on zero-shot PDDL generation. We experiment with 8 different modular pipelines, including prompting techniques such as providing extensive PDDL knowledge as a prefix, or pre-inference techniques such as generating a natural language summary before the PDDL, sequentially generating domain and problem files, using a Python wrapper of PDDL, or constraining decoding with PDDL grammar. We also consider inference-time techniques such as generating multiple responses, revising generated PDDL with feedback from the formal solver or the plan validator. Our best performing pipeline decreases Qwen-3 32B model's syntax errors by 97% and semantic errors by 47% on the common

[1] Our resources are at https://github.com/prakashkagitha/llm-pddl-gen.

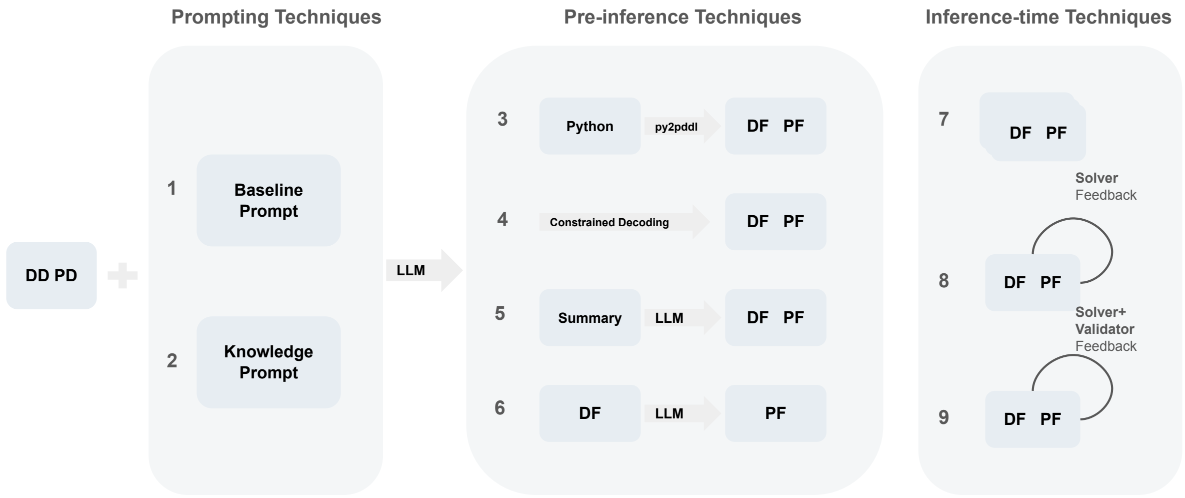

Figure 2: Our modular approach that includes prompting (1. Baseline prompt, 2. Knowledge prompt), pre-inference (3. Python wrapper, 4. Constrained decoding, 5. Summary, 6. Sequential), and inference-time techniques (7. Pass@N, 8. Revision with solver feedback, 9. Revision with solver + validator feedback.)

BlocksWorld benchmark, enabling planning in low-compute scenarios that require safety, and privacy, and domain-specific finetuning.

Our key findings include:

- Qwen-3 32B model is capable of generating correct PDDL but Qwen-3 8B model is not.
- Inference-time scaling approaches such as revision with feedback from solver and validator roughly doubles semantic accuracy.
- Generating PDDL with a Python wrapper and constrained decoding with PDDL grammar decrease performance.
- Modularly generating a summary before PDDL or generating domain and problem files sequentially do not improve semantic accuracy compared to baseline.

2 Methodology

To address challenges of zero-shot PDDL generation, for medium size open-source models (Huang and Zhang, 2025), we identify techniques grouped into three stages in a pipeline (Figure 2).

First, we consider two *prompting techniques*. A **baseline prompt** which is just minimal instruction to generate PDDL. In contrast, a **knowledge prompt** first introduces PDDL components including the domain file ($\mathbb{DF}$, types, predicates, action declaration, action semantics) and the problem file ($\mathbb{PF}$, object initialization, initial states, and goal states), along with a domain-agnostic example.

Second, we implement an array of *pre-inference* techniques, including:

Summary. The LLM is first prompted to generate a textual summary with all necessary information before it generates the PDDL accordingly.

Sequential generation. LLM is prompted to first only generate the $\mathbb{DF}$ before the $\mathbb{PF}$.

Python wrapper. LLM is prompted to generate PDDL in a Python wrapper[2], following success of generating low-resource languages with high-resource wrappers (Cassano et al., 2024).

Constrained decoding. We translate the formal BNF definition of PDDL 3.1[3] into a LALR(1)-compatible EBNF grammar used to limit LLMs' decoding to trivially syntactically correct PDDL.

Third, we consider several *inference-time* techniques, including:

Pass@N. We evaluate N independent LLM generations, counted as correct if any is correct.

Revision with solver feedback. LLM is prompted to generate PDDL and to revise based on the solver's error feedback.

Revision with solver + validator feedback. Same as the above, but additional revision is performed based on the feedback of a plan validator.

Prompts and example outputs are provided in the Appendix. Baseline prompt: Figure 9, Knowledge prompt: Figure 10, Python wrapper prompt: Figure 12, Python wrapper model response: Figure 14, and Python translated to PDDL: Figure 16.

[2] https://github.com/remykarem/py2pddl

[3] Kovacs, 2011: http://pddl4j.imag.fr/repository/wiki/BNF-PDDL-3.1.pdf

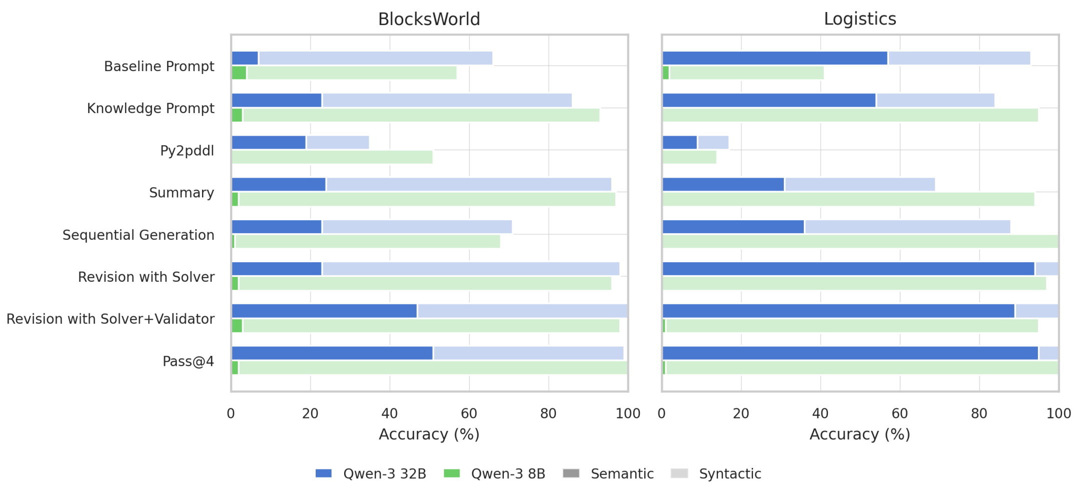

Figure 3: Performance of all the techniques implemented. DF: Domain File, PF: Problem File, DD: Domain Description, PD: Problem Description.

3 Evaluation: Datasets, Metrics, Models

We adopt datasets and metrics from Huang and Zhang (2025).

3.1 Datasets

We consider three simulated planning environments, BlocksWorld, Logistics, and Barman from the International Planning Competition (IPC, 1998). Each dataset comes with ground-truth PDDL domain ($\mathbb{DF}$) and problem files ($\mathbb{PF}$) that can be used to validate a predicted plan. The input to the model is a natural language description of the domain ($\mathbb{DD}$) that includes the names and parameters of the actions, and the problem ($\mathbb{PD}$). An example of $\mathbb{DD}$ and $\mathbb{PD}$ is provided in Figure 7 and Figure 8 respectively. The output of an LLM-as-formalizer is the predicted $\mathbb{DF}$ and $\mathbb{PF}$, which are used with a planner to search for a plan. The dataset of each environment have 100 problems with varying levels of complexity. We use moderately templated descriptions from Huang and Zhang (2025) which are common in literature.

3.2 Metrics

We use syntactic and semantic accuracy to assess the $\mathbb{DF}$ and $\mathbb{PF}$ generated by an LLM. *Syntactic accuracy* is the percentage of problems where no syntax error are returned by the planning solver. *Semantic accuracy* is the percentage of problems where a plan is not only found but also correct. We use the `dual-bfws-ffparser` planner (Muise, 2016) to solve for the plan and the VAL[4] (Howey et al., 2004) to validate the plan against the ground-truth $\mathbb{DF}$ and $\mathbb{PF}$.

3.3 Models

We evaluate two recent and best performing open-source LLMs, Qwen-3 32B and Qwen-3 8B[5], for their small size and strong performance in other tasks. While Huang and Zhang (2025) report zero performance with 8B and 70B DeepSeek-R1 (Guo et al., 2025) and Llama-3.1 (Dubey et al., 2024) models, we attempt to push their limits via our techniques and inference-time scaling. We follow previous work to only consider zero-shot prompting for to emulate real-life application with minimal user interference and need for training data.((Huang and Zhang, 2025)) We use vLLM (Kwon et al., 2023) to speed up inference and set temperature of 0.4 for all our experiments. For constrained decoding we used HugginFace Transformers (Wolf et al., 2020) backend with Outlines (Willard and Louf, 2023). All of our experiments for the main results are run for approximately 72 hours on 4 H100 GPUs.

4 Results and Observations

The results are shown in Figure 3. Using the baseline prompt without no techniques, Qwen-3 32B can generate correct PDDL, while Qwen-3 8B struggled with semantic accuracy near zero. None of the techniques were able to improve 8B model's performance informing the lower bound of reasoning capabilities needed for PDDL generation.

[4] `nms.kcl.ac.uk/planning/software/val.html`

[5] `https://github.com/QwenLM/Qwen3`

Prompting with PDDL knowledge is helpful, but no pre-inference techniques help. PDDL knowledge prompt improves semantic accuracy from 7% to 23% in BlocksWorld but decreases in Logistics from 57% to 54% where the baseline itself is comparatively stronger. Multi-stage LLM pipelines such as summary before PDDL and separate domain and problem files does not improve semantic accuracy in BlocksWorld and significantly decreases performance in Logistics compared to single-stage LLM pipelines such as baseline and knowledge prompt.

Python wrapper decreases performance. As LLMs are adept at generating Python in general, one may expect generating PDDL via a Python wrapper would greatly decrease syntax errors. In contrast, we conclude that generating code in Py2PDDL format before converting it to PDDL to be compatible with the planner performs worse than directing generating PDDL. The generated python code failed to be converted to PDDL more than half of the time even with extensive Py2PDDL documentation in the prompt. We suspect Py2PDDL couldn't exploit better python generation capability of LLMs as it is too similar to PDDL syntax than Python syntax.

Constrained decoding with PDDL grammar decrease performance. To be compatible with constrained decoding, we evaluate non-reasoning model, Qwen-2.5-32B-Instruct (Team, 2024) with blocksworld domain. There were no syntactic errors, by definition, but 98% of the generated $\mathbb{DF}$ and $\mathbb{PF}$ have semantic errors. The strict PDDL grammar might have suppressed the "semantic" tokens to get semantics of generation correct.

Test-time scaling techniques greatly improve performance. As seen in Figure 3, pass@N and revision methods improved performance of 32B model upto 2x and 1.5x in BlocksWorld and Logistics respectively. Interestingly, revision with solver, in Logistics domain, with feedback only on syntactic errors, reached the semantic accuracy of pass@N and revision with solver+VAL which are informed by both syntactic and semantic errors. To assess the performance improvement through revision, we contrasted against pass@4 performance which needs the same inference budget. As seen in Figure 4, three rounds of revision with the solver and validator recovered the performance of pass@4 in both domains.

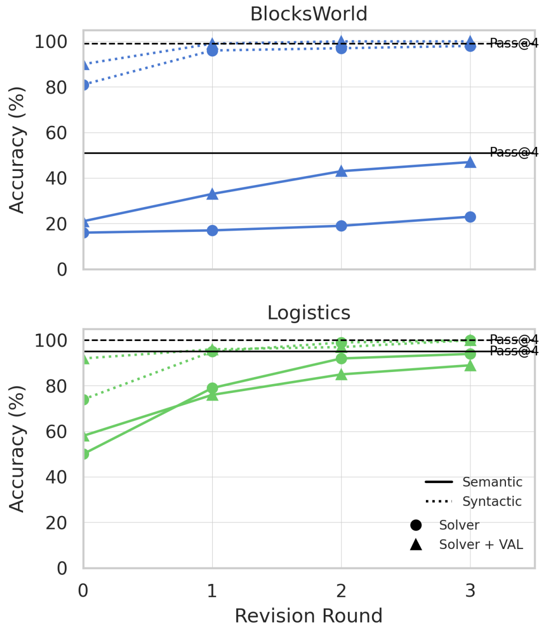

Figure 4: Performance improvement by revision with feedback for 3 rounds and comparison against Pass@4.

Undefined symbols and action semantics are corrected during revisions. We qualitatively observe that most of the syntax errors that are corrected are imbalanced parenthesis or undefined symbols and most of the semantic errors that are corrected are mistakes in generating $\mathbb{DF}$s, especially, action semantics: either missing necessary parameters or incorrect logical expressions for precondition and effects. Examples of corrected syntax and corrected semantics by revision are provided in Figure 17 and Figure 18 respectively.

5 Conclusion

We identify different LLM pipelines to address the challenges of PDDL generation and evaluate medium-size open source models on different domains. We find that python wrapper and constrained decoding with PDDL grammar do not work, but inference-time techniques improve the performance for both the domains. This work shows feasibility of using mid-size open source models to generate planning formalisms in PDDL and provides scope for extending LLM-as-formalizer to multiple formalisms.

6 Limitations

Due to limitation of compute resources, all experiments are performed with a single run, which might not be as reliable as multiple runs, reporting average and standard deviation.

Some of the techniques implemented show mixed results on the two datasets we considered. For example, revision with solver feedback is better than revision with solver and validator in Logistics but not in BlocksWorld. Having more datasets will likely drawn out the clear pattern on which technique is better comparatively and how much.

We work with moderately templated description as input that is easier than more natural version of the data. While we observe increased performance with some of the techniques implemented, it is yet to be determined whether there will be the same performance increase with natural data, which is more applicable to the real world.

Acknowledgments

We thank Cassie Huang for support on the datasets. The code for the experiments in this work is written partially with the help of AI coding tools to autocomplete boilerplate code and occasionally few methods. Authors used these tools to accelerate the experimentation but not to get ideas on different LLM pipelines or evaluation strategies.

References

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. 2024. Knowledge transfer from high-resource to low-resource programming languages for code llms. *Preprint*, arXiv:2308.09895.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Weihang Guo, Zachary Kingston, and Lydia E. Kavraki. 2024. Castl: Constraints as specifications through llm translation for long-horizon task and motion planning. *Preprint*, arXiv:2410.22225.

Yilun Hao, Yang Zhang, and Chuchu Fan. 2025. Planning anything with rigor: General-purpose zero-shot planning with llm-based formalized programming. *Preprint*, arXiv:2410.12112.

R. Howey, D. Long, and M. Fox. 2004. Val: automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301.

Mengkang Hu, Tianxing Chen, Yude Zou, Yuheng Lei, Qiguang Chen, Ming Li, Yao Mu, Hongyuan Zhang, Wenqi Shao, and Ping Luo. 2025. Text2world: Benchmarking large language models for symbolic world model generation. *Preprint*, arXiv:2502.13092.

Cassie Huang and Li Zhang. 2025. On the limit of language models as planning formalizers. *Preprint*, arXiv:2412.09879.

IPC. 1998. International planning competition. https://www.icaps-conference.org/competitions.

Adam Ishay and Joohyung Lee. 2025. Llm+al: Bridging large language models and action languages for complex reasoning about actions. *Preprint*, arXiv:2501.00830.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, et al. 2024. Embodied agent interface: Benchmarking llms for embodied decision making. *arXiv preprint arXiv:2410.07166*.

Xinrui Lin, Yangfan Wu, Huanyu Yang, Yu Zhang, Yanyong Zhang, and Jianmin Ji. 2024. Clmasp: Coupling large language models with answer set programming for robotic task planning. *Preprint*, arXiv:2406.03367.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. 2023. Clin: A continually learning language agent for rapid task adaptation and generalization. *Preprint*, arXiv:2310.10134.

Christian Muise. 2016. Planning.Domains. In *The 26th International Conference on Automated Planning and Scheduling - Demonstrations*.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, Chitta Baral, Chen-Yu Lee, Tomas Pfister, and Hamid Palangi. 2025. Plangen: A multi-agent framework for generating planning and reasoning trajectories for complex problem solving. *Preprint*, arXiv:2502.16111.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20256–20264.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Brandon T Willard and Rémi Louf. 2023. Efficient guided generation for llms. *arXiv preprint arXiv:2307.09702*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024a. PDDLEGO: Iterative planning in textual environments. In *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, pages 212–221, Mexico City, Mexico. Association for Computational Linguistics.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024b. PROC2PDDL: Open-domain planning representations from texts. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ACL 2024)*, pages 13–24, Bangkok, Thailand. Association for Computational Linguistics.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L Littman, and Stephen H Bach. 2024. Planetarium: A rigorous benchmark for translating text to structured planning languages. *arXiv preprint arXiv:2407.03321*.

A Input, Prompts, and Examples

```
(define (domain blocksworld)
  (:requirements :strips)
(:predicates (clear ?x)
             (on-table ?x)
             (arm-empty)
             (holding ?x)
             (on ?x ?y))

(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
               (not (arm-empty))))

(:action putdown
  :parameters (?ob)
  :precondition (holding ?ob)
  :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
               (not (holding ?ob))))

(:action stack
  :parameters (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))

(:action unstack
  :parameters (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
  :effect (and (holding ?ob) (clear ?underob)
               (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-empty)))))
```

Figure 5: $\mathbb{DF}$ for the BlocksWorld domain.

```
(define (problem blocksworld-p01)
  (:domain blocksworld)
  (:objects block1 block2 block3 block4 )
  (:init
    (on-table block3)
    (clear block3)
    (on-table block4)
    (clear block4)
    (on-table block1)
    (clear block1)
    (on-table block2)
    (clear block2)
    (arm-empty)
  )
  (:goal (and
    (on-table block4)
    (on-table block2)
    (on-table block1)
    (on-table block3)
  ))
)
```

Figure 6: $\mathbb{PF}$ for the BlocksWorld domain.

```
I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

  Pick up a block
  Unstack a block from on top of another block
  Put down a block
  Stack a block on top of another block

  I have the following restrictions on my actions:
  I can only pick up or unstack one block at a time.
  I can only pick up or unstack a block if my hand is empty.
  I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
  I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
  I can only unstack a block from on top of another block if the block I am unstacking is clear.
  Once I pick up or unstack a block, I am holding the block.
  I can only put down a block that I am holding.
  I can only stack a block on top of another block if I am holding the block being stacked.
  I can only stack a block on top of another block if the block onto which I am stacking the block is clear.
  Once I put down or stack a block, my hand becomes empty.
  Once you stack a block on top of a second block, the second block is no longer clear.
```

Figure 7: $\mathbb{DD}$ for the BlocksWorld domain.

```
As initial conditions I have that, block 1 is clear, block 2 is clear, block 3 is clear, block 4 is clear, the hand is empty, block 1 is on the table, block 2 is on the table, block 3 is on the table, and block 4 is on the table.
My goal is to have that block 1 is on the table, block 2 is on the table, block 3 is on the table, and block 4 is on the table.
```

Figure 8: Problem Description for the BlocksWorld domain.

```
Domain description:
{domain_description}

Problem description:
{problem_description}

Write the domain and problem files in minimal PDDL.
Wrap PDDL domain file inside <domain file>...</domain file> and PDDL problem file inside <problem file>...</problem file>.
<think>
```

Figure 9: Baseline Prompt

```
PDDL domain file contains domain name, requirements, types of objects in the domain, predicates, and actions.
Based on the natural language domain description, identify the actions that are possible.
Identify action sematics i.e. understand the preconditions under which that action could be done and the effects of the action.
Then identify appropriate predicates that could enable action semantics i.e. preconditions and effects.
PDDL domain file has a definitive syntax that must be followed for any domain. An abstract example PDDL domain file is given below:

<domain_file>
(define
	(domain domain_name)
	(:requirements :strips :typing)
	(:types
		type1
		type2
	)
	(:predicates
		(predicate1 ?arg1 - type1 ?arg2 - type2)
		(predicate2 ?arg1 - type1 ?arg2 - type2)
	)
	(:action action1
		:parameters (?arg1 - type1 ?arg2 - type2 ?arg3 - type2)
		:precondition (predicate1 ?arg1 ?arg2)
		:effect (and (predicate1 ?arg1 ?arg2) (predicate2 ?arg1 ?arg3))
	)
	(:action action2
		:parameters (?arg1 - type1 ?arg2 - type2 ?arg3 - type2)
		:precondition (and (predicate1 ?arg1 ?arg2) (predicate2 ?arg1 ?arg3))
		:effect (predicate2 ?arg1 ?arg3)
	)
)
</domain_file>

Notes for generating domain file:
- type1 & type2 are only representative and should be replaced with appropriate types. There could be any number of types.
- predicate1 & predicate2 are only representative and should be replaced with appropriate predicates. There could be any number of predicates.
- action1 & action2 are only representative and should be replaced with appropriate actions. There could be any number of actions.
- arg1 & arg2 are only representative and should be replaced with appropriate arguments for predicates and in preconditions and effects.
- predicates with proper arguments could be combined to combine complex boolean expression to represent predicondition and effect
The braces should be balanced for each section of the PDDL program
- Use predicates with arguments of the right type as declared in domain file
- All the arguments to any :precondition or :effect of an action should be declared in :parameters as input arguments


PDDL problem file contains problem name, domain name, objects in this problem instance, init state of objects, and goal state of objects.
Based on the natural language problem description, identify the relevant objects for this problems with their names and types.
Represent the initial state with the appropriate predicates and object arguments. Represent the goal state with the appropriate predicates and object arguments.
PDDL problem file has a definitive syntax that must be followed for any problem. An abstract example PDDL problem file is given below.
```

Figure 10: Knowledge Prompt

```
<problem_file>
(define
	(problem problem_name)
	(:domain domain_name)
	(:objects
		obj1 obj2 - type1
		obj3, obj4 - type2
	)
	(:init (predicate1 obj1 obj3) (predicate2 obj2 obj3))
	(:goal (and (predicate1 obj1 obj4) (predicate2 obj2 obj3)))
)
</problem_file>

Notes for generating problem file:
- obj1, obj2, ... are only representative and should be replaced with appropriate objects. There could be any number of obects with their types.
- init state with predicate1 & predicate2 is only representative and should be replaced with appropriate predicates that define init state
- goal state with predicate1 & predicate2 is only representative and should be replaced with appropriate predicates that define goal state
- predicates with proper arguments could be combined to combine complex boolean expression to represent init and goal states
- The braces should be balanced for each section of the PDDL program
- Use predicates with arguments of the right type as declared in domain file
- All the objects that would be arguments of predicates in init and goal states should be declared in :objects

Domain description:
{domain_description}

Problem description:
{problem_description}

Write the domain and problem files in minimal PDDL.
Wrap PDDL domain file inside <domain_file>...</domain_file> and PDDL problem file inside <problem_file>...</problem_file>.
<think>
```

Figure 11: Knowledge Prompt (continued)

```
Python representation of PDDL domain file contains domain name, requirements, types of objects in the domain, predicates, and actions.
Based on the natural language domain description, identify the actions that are possible.
Identify action sematics i.e. understand the preconditions under which that action could be done and the effects of the action.
Then identify appropriate predicates that could enable action semantics i.e. preconditions and effects.
Python representation of PDDL domain file has a definitive syntax that must be followed for any domain. An abstract example is given below:

In the following python domain file, the AirCargoDomain class has been created. The structure of the class is similar to how a PDDL domain should be defined.

Name of the domain is the name of the Python class (DomainName).
Types are defined as class variables at the top (Type1, Type2).
Predicates are defined as instance methods decorated with @predicate.
Actions are defined as instance methods decorated with @action

The positional arguments of @predicate and @action decorators are the types of the respective arguments.
Methods decorated with @predicate should have empty bodies.
Methods decorated with @action return a tuple of two lists

<domain_file>
# imports stays exactly same for all domain files
from py2pddl import Domain, create_type
from py2pddl import predicate, action

class DomainName(Domain):

    Type1 = create_type("Type1")
    Type2 = create_type("Type2")

    @predicate(Type1, Type2)
    def predicate1(self, arg1, arg2):
        """Complete the method signature and specify
        the respective types in the decorator"""

    @predicate(Type1)
    def predicate2(self, arg1):
        """Complete the method signature and specify
        the respective types in the decorator"""


    @action(Type1, Type2, Type2)
    def action1(self, arg1, arg2, arg3):
        precond = [self.predicate1(arg1, arg3), self.predicate2(arg1)]
        effect = [~self.predicate1(arg1, arg2), self.predicate2(arg3)]
        return precond, effect

    @action(Type1)
    def action2(self, arg1):
        precond = [self.predicate2(arg1)]
        effect = [~self.predicate2(arg1)]
        return precond, effect
</domain_file>

Notes for generating domain file:
- the above example file is only for understanding the syntax
- type1 & type2 are only representative and should be replaced with appropriate types. There could be any number of types.
- predicate1 & predicate2 are only representative and should be replaced with appropriate predicates. There could be any number of predicates.
- action1 & action2 are only representative and should be replaced with appropriate actions. There could be any number of actions.
- arg1 & arg2 are only representative and should be replaced with appropriate arguments for predicates and in preconditions and effects.
```

Figure 12: Prompt for Py2PDDL

```
Python representation of PDDL problem file contains problem name, domain name, objects in this problem instance, init state of objects, and goal state of objects.
Based on the natural language problem description, identify the relevant objects for this problems with their names and types.
Represent the initial state with the appropriate predicates and object arguments. Represent the goal state with the appropriate predicates and object arguments.
Python representation of PDDL problem file has a definitive syntax that must be followed for any problem. An abstract example is given below.

<problem_file>
# imports stays the same for all problem files
# Assume DomainName is declared just before the problem file
from py2pddl import goal, init

class ProblemName(DomainName):

    def __init__(self):
        super().__init__()
        self.type1Objs = DomainName.Type1.create_objs([1, 2], prefix="type1Obj")
        self.type2Objs = DomainName.Type2.create_objs(["type2Obj1", "type2Obj2"])

    @init
    def init(self):
        at = [self.predicate1(self.type1Objs[1], self.type2Objs["type2Obj1"]),
              self.predicate2(self.type1Objs[1]),]
        return at

    @goal
    def goal(self):
        return [self.predicate1(self.type1Objs[1], self.type2Objs["type2Obj2"]),
                self.predicate2(self.type1Objs[2])]
</problem_file>

Notes for generating problem file:
- No need to import DomainName for python problem file. Python Domain file and python problem file would be merged and executed.
- type1Objs, type2Oobjs, ... are only representative and should be replaced with appropriate objects. There could be any number of obects with their types.
- init state with predicate1 & predicate2 is only representative and should be replaced with appropriate predicates that define init state
- goal state with predicate1 & predicate2 is only representative and should be replaced with appropriate predicates that define goal state
```

Figure 13: Prompt for Py2PDDL (continued)

```python
from py2pddl import Domain, create_type
from py2pddl import predicate, action

class BlocksWorld(Domain):

    Block = create_type("Block")

    @predicate(Block, Block)
    def On(self, block1, block2):
        """Block1 is on top of Block2"""

    @predicate(Block)
    def Clear(self, block):
        """Block has nothing on top and is not being held"""

    @predicate(Block)
    def OnTable(self, block):
        """Block is on the table"""

    @predicate(Block)
    def Holding(self, block):
        """Robot is holding Block"""

    @predicate()
    def HandEmpty(self):
        """Robot's hand is empty"""


    @action(Block)
    def Pickup(self, block):
        precond = [self.HandEmpty(), self.OnTable(block), self.Clear(block)]
        effect = [self.Holding(block), ~self.HandEmpty(), ~self.Clear(block)]
        return precond, effect

    @action(Block, Block)
    def Unstack(self, block, from_block):
        precond = [self.HandEmpty(), self.On(block, from_block), self.Clear(block)]
        effect = [self.Holding(block), ~self.HandEmpty(), ~self.On(block, from_block), ~self.Clear(block), self.Clear(from_block)]
        return precond, effect

    @action(Block)
    def Putdown(self, block):
        precond = [self.Holding(block)]
        effect = [self.OnTable(block), self.Clear(block), ~self.Holding(block), self.HandEmpty()]
        return precond, effect

    @action(Block, Block)
    def Stack(self, block, on_block):
        precond = [self.Holding(block), self.Clear(on_block)]
        effect = [self.On(block, on_block), ~self.Clear(on_block), ~self.Holding(block), self.HandEmpty(), ~self.Clear(block)]
        return precond, effect

from py2pddl import goal, init

class BlockProblem(BlocksWorld):

    def __init__(self):
        super().__init__()
        self.blocks = BlocksWorld.Block.create_objs([1,2,3,4,5,6,7,8,9,10,11], prefix="block")
```

Figure 14: Py2PDDL Model Response Example - BlocksWorld

```


    @init
    def init(self):
        return [
            self.On(self.blocks[3], self.blocks[6]),
            self.On(self.blocks[5], self.blocks[4]),
            self.On(self.blocks[7], self.blocks[11]),
            self.On(self.blocks[9], self.blocks[5]),
            self.On(self.blocks[10], self.blocks[7]),
            self.OnTable(self.blocks[1]),
            self.OnTable(self.blocks[2]),
            self.OnTable(self.blocks[4]),
            self.OnTable(self.blocks[6]),
            self.OnTable(self.blocks[8]),
            self.OnTable(self.blocks[11]),
            self.Clear(self.blocks[1]),
            self.Clear(self.blocks[2]),
            self.Clear(self.blocks[3]),
            self.Clear(self.blocks[8]),
            self.Clear(self.blocks[9]),
            self.Clear(self.blocks[10]),
            self.HandEmpty()
        ]


    @goal
    def goal(self):
        return [
            self.On(self.blocks[1], self.blocks[8]),
            self.On(self.blocks[2], self.blocks[4]),
            self.On(self.blocks[6], self.blocks[1]),
            self.On(self.blocks[9], self.blocks[6]),
            self.OnTable(self.blocks[3]),
            self.OnTable(self.blocks[4]),
            self.OnTable(self.blocks[5]),
            self.OnTable(self.blocks[7]),
            self.OnTable(self.blocks[8]),
            self.OnTable(self.blocks[10]),
            self.OnTable(self.blocks[11]),
            self.Clear(self.blocks[3]),
            self.Clear(self.blocks[5]),
            self.Clear(self.blocks[7]),
            self.Clear(self.blocks[10]),
            self.Clear(self.blocks[11]),
        ]
```

Figure 15: Py2PDDL Model Response - Correct Example - BlocksWorld (Continued)

```
Domain File:
(define
        (domain blocksworld)
        (:requirements :strips :typing)
        (:types
                block
        )
        (:predicates
                (Clear ?block - block)
                (HandEmpty )
                (Holding ?block - block)
                (On ?block1 - block ?block2 - block)
                (OnTable ?block - block)
        )
        (:action Pickup
                :parameters (?block - block)
                :precondition (and (HandEmpty ) (OnTable ?block) (Clear ?block))
                :effect (and (Holding ?block) (not (HandEmpty )) (not (Clear ?block)))
        )
        (:action Putdown
                :parameters (?block - block)
                :precondition (Holding ?block)
                :effect (and (OnTable ?block) (Clear ?block) (not (Holding ?block)) (HandEmpty ))
        )
        (:action Stack
                :parameters (?block - block ?on-block - block)
                :precondition (and (Holding ?block) (Clear ?on-block))
                :effect (and (On ?block ?on-block) (not (Clear ?on-block)) (not (Holding ?block)) (HandEmpty ) (not (Clear ?block)))
        )
        (:action Unstack
                :parameters (?block - block ?from-block - block)
                :precondition (and (HandEmpty ) (On ?block ?from-block) (Clear ?block))
                :effect (and (Holding ?block) (not (HandEmpty )) (not (On ?block ?from-block)) ( not (Clear ?block)) (Clear ?from-block))
        )
)

Problem File:
(define
        (problem block)
        (:domain blocksworld)
        (:objects
                block1 block2 block3 block4 block5 block6 block7 block8 block9 block10 block11 - block
        )
        (:init (On block3 block6) (On block5 block4) (On block7 block11) (On block9 block5) (On block10 block7) (OnTable block1) (OnTable block2) (OnTable block4) (OnTable block6 ) (OnTable block8) (OnTable block11) (Clear block1) (Clear block2) (Clear block3) (Clear block8) (Clear block9) (Clear block10) (HandEmpty))
        (:goal (and (On block1 block8) (On block2 block4) (On block6 block1) (On block9 block6) ( OnTable block3) (OnTable block4) (OnTable block5) (OnTable block7) (OnTable block8 ) (OnTable block10) (OnTable block11) (Clear block3) (Clear block5) (Clear block7) (Clear block10) (Clear block11)))
)
```

Figure 16: Py2PDDL Translated PDDL - Correct Example - BlocksWorld.

```
(define
	(domain blocks-world)
	(:requirements :strips :typing)
	(:types
		block
	)
	(:predicates
		(clear ?b - block)
		(on ?b1 - block ?b2 - block)
		(on-table ?b - block)
		(holding ?b - block)
		(hand-empty)
	)
	(:action pickup
		:parameters (?b - block)
		:precondition (and (hand-empty ) (on-table ?b) (clear ?b))
		:effect (and (holding ?b) (not (hand-empty)) (not (on-table ?b)) (not (clear ?b)))
	)
	(:action unstack
		:parameters (?b1 - block)
		:precondition (and (hand-empty ) (on ?b1 ?b2) (clear ?b1))
		:effect (and (holding ?b1) (not (hand-empty)) (not (on ?b1 ?b2)) (not (clear ?b1)) (clear ?b2 ))
	)
	(:action putdown
		:parameters (?b - block)
		:precondition (holding ?b)
		:effect (and (on-table ?b) (clear ?b) (hand-empty) (not (holding ?b)))
	)
	(:action stack
		:parameters (?b1 - block ?b2 - block)
		:precondition (and (holding ?b1) (clear ?b2))
		:effect (and (on ?b1 ?b2) (hand-empty) (not (holding ?b1)) (not (clear ?b2)) (clear ?b1 ))
	)
)
```

(a) Syntax incorrect PDDL

```
(define
	(domain blocks-world)
	(:requirements :strips :typing)
	(:types
		block
	)
	(:predicates
		(clear ?b - block)
		(on ?b1 - block ?b2 - block)
		(on-table ?b - block)
		(holding ?b - block)
		(hand-empty)
	)
	(:action pickup
		:parameters (?b - block)
		:precondition (and (hand-empty ) (on-table ?b) (clear ?b))
		:effect (and (holding ?b) (not (hand-empty)) (not (on-table ?b)) (not (clear ?b)))
	)
	(:action unstack
		:parameters (?b1 - block ?b2 - block)
		:precondition (and (hand-empty ) (on ?b1 ?b2) (clear ?b1))
		:effect (and (holding ?b1) (not (hand-empty)) (not (on ?b1 ?b2)) (not (clear ?b1)) (clear ?b2 ))
	)
	(:action putdown
		:parameters (?b - block)
		:precondition (holding ?b)
		:effect (and (on-table ?b) (clear ?b) (hand-empty) (not (holding ?b)))
	)
	(:action stack
		:parameters (?b1 - block ?b2 - block)
		:precondition (and (holding ?b1) (clear ?b2))
		:effect (and (on ?b1 ?b2) (hand-empty) (not (holding ?b1)) (not (clear ?b2)) (clear ?b1 ))
	)
)
```

(b) Syntax corrected PDDL

Figure 17: Revision by Solver feedback example

```
(define
        (domain blocks-world)
        (:requirements :strips :typing)
        (:types
                block
        )
        (:predicates
                (on ?x - block ?y - block)
                (on-table ?x - block)
                (clear ?x - block)
                (holding ?x - block)
                (hand-empty)
        )
        (:action pickup
                :parameters (?b - block)
                :precondition (and (on-table ?b) (clear ?b) (hand-empty))
                :effect (and (holding ?b) (not (on-table ?b)) (not (clear ?b)) (not (hand-empty)))
        )
        (:action unstack
                :parameters (?b - block ?s - block)
                :precondition (and (on ?b ?s) (hand-empty) (clear ?b))
                :effect (and (holding ?b) (not (on ?b ?s)) (clear ?s) (not (clear ?b)) (not (hand-empty)))
        )
        (:action putdown
                :parameters (?b - block)
                :precondition (holding ?b)
                :effect (and (on-table ?b) (clear ?b) (not (holding ?b)) (hand-empty))
        )
        (:action stack
                :parameters (?b - block ?s - block)
                :precondition (and (holding ?b) (clear ?s))
                :effect (and (on ?b ?s) (not (clear ?s)) (not (holding ?b)) (hand-empty))
                :effect (and (on ?b ?s) (not (clear ?s)) (not (holding ?b)) (hand-empty))
        )
)
```

(a) Semantics incorrect PDDL

```
(define
        (domain blocks-world)
        (:requirements :strips :typing)
        (:types
                block
        )
        (:predicates
                (on ?x - block ?y - block)
                (on-table ?x - block)
                (clear ?x - block)
                (holding ?x - block)
                (hand-empty)
        )
        (:action pickup
                :parameters (?b - block)
                :precondition (and (on-table ?b) (clear ?b) (hand-empty))
                :effect (and (holding ?b) (not (on-table ?b)) (not (clear ?b)) (not (hand-empty)))
        )
        (:action unstack
                :parameters (?b - block ?s - block)
                :precondition (and (on ?b ?s) (hand-empty) (clear ?b))
                :effect (and (holding ?b) (not (on ?b ?s)) (clear ?s) (not (clear ?b)) (not (hand-empty)))
        )
        (:action putdown
                :parameters (?b - block)
                :precondition (holding ?b)
                :effect (and (on-table ?b) (clear ?b) (not (holding ?b)) (hand-empty))
        )
        (:action stack
                :parameters (?b - block ?s - block)
                :precondition (and (holding ?b) (clear ?s))
                :effect (and (on ?b ?s) (clear ?b) (not (clear ?s)) (not (holding ?b)) (hand-empty))
                :effect (and (on ?b ?s) (clear ?b) (not (clear ?s)) (not (holding ?b)) (hand-empty))
        )
)
```

(b) Semantics corrected PDDL

Figure 18: Revision by Solver+validator feedback example