HDDLGym: A Tool for Studying Multi-Agent Hierarchical Problems Defined in HDDL with OpenAI Gym

Ngoc La[1], **Ruaridh Mon-Williams**[2], and **Julie A. Shah**[1]
[1]MIT
[2]University of Edinburgh
ntmla@mit.edu, ruaridh.mw@ed.ac.uk, julie.a.shah@csail.mit.edu

Abstract

In recent years, reinforcement learning (RL) methods have been widely tested using tools like OpenAI Gym, though many tasks in these environments could also benefit from hierarchical planning. However, there is a lack of a tool that enables seamless integration of hierarchical planning with RL. Hierarchical Domain Definition Language (HDDL), used in classical planning, introduces a structured approach well-suited for model-based RL to address this gap. To bridge this integration, we introduce HDDLGym, a Python-based tool that automatically generates OpenAI Gym environments from HDDL domains and problems. HDDLGym serves as a link between RL and hierarchical planning, supporting multi-agent scenarios and enabling collaborative planning among agents. This paper provides an overview of HDDLGym's design and implementation, highlighting the challenges and design choices involved in integrating HDDL with the Gym interface, and applying RL policies to support hierarchical planning. We also provide detailed instructions and demonstrations for using the HDDLGym framework, including how to work with existing HDDL domains and problems from International Planning Competitions, exemplified by the Transport domain. Additionally, we offer guidance on creating new HDDL domains for multi-agent scenarios and demonstrate the practical use of HDDLGym in the Overcooked domain. By leveraging the advantages of HDDL and Gym, HDDLGym aims to be a valuable tool for studying RL in hierarchical planning, particularly in multi-agent contexts.

Code — https://github.com/HDDLGym/HDDLGym

1 Introduction

Hierarchical planning is essential for addressing complex, long-horizon planning problems by decomposing them into smaller, manageable subproblems. In reinforcement learning (RL), hierarchical strategies can guide exploration along specific pathways, potentially enhancing learning efficiency. However, implementing RL policies within hierarchical frameworks often requires custom modifications to the original environments to incorporate high-level actions (Wu et al. 2021; Liu et al. 2017; Xiao, Hoffman, and Amato 2020). For example, in a Bayesian inference study using the Overcooked game, subtasks are integrated as high-level actions through specific rules embedded in the system codebase (Wu et al. 2021). Similarly, several RL studies use author-defined high-level actions, or macro-actions, to organize complex tasks (Liu et al. 2017; Xiao, Hoffman, and Amato 2020). While these studies highlight the benefits of hierarchical approaches in complex scenarios, the additional programming required to integrate hierarchical layers can make it challenging for external users to modify or implement alternative high-level strategies. This limitation reduces users' flexibility to implement diverse hierarchical strategies tailored to their specific requirements.

The Hierarchical Domain Definition Language (HDDL) (Höller et al. 2020) is an extension of Planning Domain Definition Language (PDDL) (McDermott et al. 1998) that incorporates hierarchical task networks (HTN) (Erol, Hendler, and Nau 1994). HDDL provides a standardized language for hierarchical planning systems and is supported by extensive documentation as well as a variety of domains and problems. Many of these resources are sourced from the hierarchical task network tracks of the International Planning Competitions (IPC-HTN) (IPC 2023 HTN Tracks). HDDL's intuitive and flexible design also allows users to define or modify problem-solving approaches by adjusting the hierarchical task networks to suit their specific needs. To leverage HDDL's strengths in studying RL within hierarchical planning problems, we present HDDLGym — a framework that integrates HDDL with OpenAI Gym (Brockman et al. 2016), a standardized RL interface. HDDLGym is a Python-based tool that automatically generates Gym environments from HDDL domain and problem files.

Multi-agent contexts are a key area in automated planning and hierarchical planning research. While HDDL is not inherently designed for multi-agent systems, multi-agent features have been explored in planning formalisms like MA-PDDL (Kovacs 2012) and MA-HTN (Cardoso and Bordini 2017). However, to utilize the extensive, well-documented HDDL domains and problems from IPC-HTN, HDDLGym is designed to work closely with the HDDL defined by Höller et al. (2020). We introduce a new protocol for extending HDDL domains and problems to support HDDLGym with multi-agent features. This includes making minor modifications to existing HDDL files from IPC-HTN.

Main contributions This paper makes the following three key contributions:

- We introduce HDDLGym, a novel framework that automatically bridges reinforcement learning and hierarchical planning by automatically generating Gym environments from HDDL domains and problems.
- We provide a protocol for modifying HDDL domains to support multi-agent configurations within HDDLGym, thereby extending hierarchical planning techniques to complex multi-agent environments.
- We detail HDDLGym's design and usage, demonstrating its effectiveness with examples from the Transport domain (in IPC-HTN) and the Overcooked environment (as shown in Figures 1a and 1b, respectively).

Core features HDDLGym offers five core features, which are detailed throughout the paper.

- Support verifying and adapting HDDL files to the tool (Section 4);
- Support multi-agent collaborations (Sections 5.5 and 6);
- Support centralized and decentralized planning (Section 5.5);
- Support modifying design of RL policy in hierarchical planning (Sections 5.3 and 5.4);
- Support training and deploying trained RL policies with various evaluation metrics and visualization (Sections 6).

The remainder of this paper is organized as follows. Section 2 provides background information on HDDL and OpenAI Gym, the two foundational frameworks on which our system is built. Section 3 discusses relevant prior work and thus highlights our contributions to the field. Section 4 then introduces the formal framework of HDDLGym, detailing how HDDL is modified to align with the agent-centric design of this tool. Section 5 covers the design and implementation details of HDDLGym. Following this, section 6 demonstrates the use of HDDLGym with examples from the Transport domain, representing domains from IPC-HTN, and Overcooked, representing customized environments. Section 7 discusses the key benefits and current limitations of the HDDLGym tool, along with future developments to address these limitations and expand its applications within artificial intelligence research. Finally, Section 8 concludes the paper.

2 Background

2.1 HDDL

HDDL (Höller et al. 2020) is an extension of PDDL (McDermott et al. 1998). Höller et al. (2020) define the domain and problem as follows.

Definition of Planning Domain: A planning domain D is a tuple (L, T_P, T_C, M) defined as follows.

- L is the underlying predicate logic.
- T_P and T_C are finite sets of primitive and compound tasks, respectively.
- M is a finite set of decomposition methods with compound tasks from T_C and task networks over the set $T_P \cup T_C$.

Definition of Planning Problem: A planning problem $\mathcal{P}$ is a tuple (D, s_I, tn_I, g), where:

- $s_I \in S$ is the initial state, a ground conjunction of positive literals over the predicates assuming the closed world assumption.
- tn_I is the initial task network that may not necessarily be grounded.
- g is the goal description, being a first-order formula over the predicates (not necessarily ground).

In other words, beyond the action definition in PDDL, which establishes the rules of interaction with the environment, HDDL introduces two additional operators: *task* and *method*. In HDDL, a *task* represents a high-level action, while a *method* is a strategy to accomplish a task. Multiple methods can exist to perform a single task. Essentially, a method is a task network that decomposes a high-level task into a partially or totally ordered list of tasks and actions.

In HDDL, a task is defined with its parameters, and a method is defined with parameters, the associated task, preconditions, a list of subtasks with their ordering or a list of ordered subtasks. Examples of task and method definitions from the original HDDL work (Höller et al. 2020) are:

```
(:task get-to :parameters (?l - location))
(:method m-drive-to-via
    :parameters (?li ?ld - location)
    :task (get-to ?ld)
    :precondition ()
    :subtasks (and
        (t1 (get-to ?li))
        (t2 (drive ?li ?ld)))
    :ordering (and
        (t1 < t2)))
```

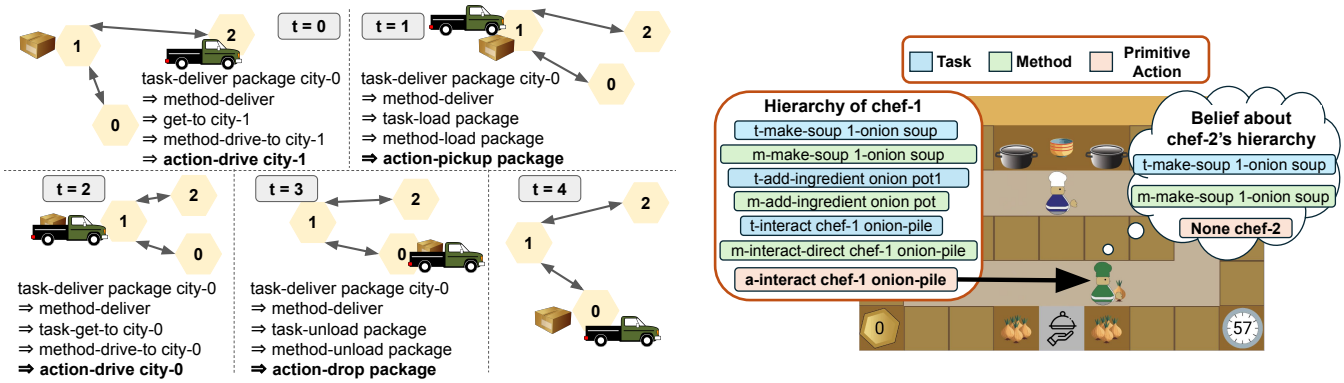
In HDDL, state-based goal definition is optional. Goals are instead defined as a list of goal tasks in the HDDL problem file. An example of the goal in a transport problem is as follows.

```
(:htn
    :tasks (and
        (deliver package-0 city-loc-0)
        (deliver package-1 city-loc-2))
    :ordering ())
```

More details about HDDL domain and problem files can be found in Höller et al. (2020). In addition to the original format of HDDL, some modifications are required to make the HDDL domains and problems work smoothly with HDDLGym. Details of the modifications are in Section 4.

2.2 OpenAI Gym

OpenAI Gym (Brockman et al. 2016) is a widely adopted toolkit that provides a standardized interface for benchmarking and developing reinforcement learning (RL) algorithms. Its consistent API includes methods for environment initialization, resetting, and interaction, allowing researchers to focus on RL algorithm development without dealing with environment-specific details. With a diverse range of environments, from simple tasks to complex simulations like Atari games, Gym enhances reproducibility



(a) Transport scenario

(b) Overcooked scenario

Figure 1: Examples of the Transport and Overcooked environments in HDDLGym

and enables direct comparisons across RL methodologies. Therefore, integrating OpenAI Gym with HDDL creates a unified framework for designing and evaluating hierarchical RL approaches, combining RL's adaptive learning with the structured decision-making of hierarchical planning.

3 Related Work

PDDLGym (Silver and Chitnis 2020) constructs Gym environments from PDDL domains and problems, serving as a valuable reference for our work. However, HDDL significantly differs from PDDL, particularly in managing hierarchical task networks or task and method operators. Additionally, PDDLGym operates under a single-action-per-step model, which suits many PDDL domains but lacks the complexity needed for advanced applications, such as multi-agent contexts. In contrast, our framework, HDDLGym, is designed to accommodate multi-agent environments, enabling the study of RL policies in more complex settings.

Similarly, pyRDDLGym (Taitler et al. 2023) integrates a planning domain language, Relational Dynamic Influence Diagram Language (RDDL) (Sanner 2010), with Gym. RDDL is adept at modeling probabilistic domains with intricate relational structures. However, it does not inherently support multi-level actions. This limitation requires significant adjustments when defining hierarchical problems within PyRDDLGym. Users must creatively structure RDDL descriptions to represent sequences of actions, which can complicate the modeling of hierarchical tasks.

NovelGym is a versatile platform that supports hybrid planning and learning agents in open-world environments (Goel et al. 2024). It effectively combines hierarchical task decomposition with modular environmental interactions to facilitate agent adaptation in unstructured settings. Nevertheless, its hierarchical structure is relatively straightforward, primarily relying on primitive and parameterized actions defined in PDDL. Conversely, HDDLGym offers more advanced hierarchical capabilities through HDDL, granting users greater flexibility and complexity in specifying high-level strategies and problem-solving approaches.

HDDLGym implements several critical extensions to support hierarchical and multi-agent planning. As outlined in the Introduction, its five key features enable users to systematically design and study hierarchical planning in conjunction with RL approaches. In addition, the framework integrates visualization tools and evaluation metrics to facilitate both qualitative and quantitative policy analysis. These enhancements allow HDDLGym to capture the full complexity of hierarchical decision-making in multi-agent environments, enabling capabilities that prior frameworks have not fully supported.

4 Formal Framework

Due to various differences in the original formalities and purposes between HDDL and Gym, some modifications in HDDL domain files are required to enable HDDLGym to work smoothly. In this section, we introduce the agent-centric extension of HDDL, modified from the standard HDDL by Höller et al. (2020). The agent-centric extension only includes changes to the HDDL domain. The agent-centric planning domain is defined below:

Definition 1. *An agent-centric planning domain $\mathcal{D}$ is a tuple $\mathcal{D} = \langle t_a, L, T_P, T_C, M \rangle$, where:*

- *t_a is an agent type hierarchy in the domain.*
- *L is the underlying predicate logic.*
- *T_P is a finite set of primitive tasks, also known as actions. Actions can be further classified into agent actions and environment actions.*
- *T_C is a finite set of compound tasks.*
- *M is a finite set of decomposition methods with compound tasks from T_C and task networks over the name $T_P \cup T_C$.*

We next discuss the elements in Def. 1 that are different from the definition of planning domain in Sec. 2.

Agent type hierarchy t_a One major difference compared to the standard HDDL (Höller et al. 2020) is the addition of t_a. t_a is used to specify which types are classified as agent types within the domain. In an HDDL domain, this classification is done by defining the type "agent" within the `:types` block. For instance, in the Transport domain, the

"vehicle" is designated as an agent type, as shown in the line 5 of the types block below. This approach allows the domain to clearly differentiate agent types from other entities, enabling more structured interactions within hierarchical planning tasks.

```
1  (:types
2          location target locatable - object
3          vehicle package - locatable
4          capacity-number - object
5          vehicle - agent)
```

Primitive Task Set T_P The primitive task set, T_P, encompasses all actions defined within the domain, classified as either agent actions or environment actions. Agent actions include one or more agents as parameters, while some actions - initially defined without agent parameters due to the nature of their predicates - must be modified to include agents if these actions are performed on behalf of agents. Additionally, in RL, particularly in multi-agent settings, it is essential to ensure that the domain includes a *none* action for each agent, enabling an agent to choose no action for a given step. Therefore, the HDDL domain file should incorporate the following action block to support the *none* action functionality.

```
1  (:action none
2      :parameters (?agent - agent)
3      :precondition ()
4      :effect ())
```

On the other hand, environment actions exclude agents from their parameters, making them non-agent actions. These actions execute automatically as soon as their preconditions are met immediately after all agents have completed their actions, enabling flexible environment dynamics.

Compound Task Set T_C The compound task set, T_C, includes all high-level tasks, aligning with the standard HDDL structure as described by Höller et al. (2020). However, in HDDLGym's implementation, additional task definition details are required. Specifically, to ensure task completion, HDDLGym checks the current world state against the defined task effects. Thus, task definitions must include explicit effects. In the following example from the Transport domain, the text in bold highlights the revisions made to the original HDDL task definition.

```
1  (:task get-to
2      :parameters (?agent - agent ?dest - location)
3      :effect (at ?agent ?destination))
```

The remaining components in the tuple, *L* and *M*, are consistent with the standard HDDL formulation as defined by Höller et al. (2020). Note that to facilitate the modification and verification of HDDL domains for compatibility with HDDLGym, the codebase includes an interactive platform featuring autonomous task effect generation, agent parameter augmentation for actions, and related capabilities.

5 HDDLGym Framework

This section explains the design and implementation of HDDLGym. It covers (1) details of HDDLEnv as a Gym environment, (2) the definition of the Agent class, (3) observation and action space details, (4) RL policy, (5) planning in multi-agent scenarios, and (6) the HDDLGym architecture.

5.1 Gym and HDDLEnv

In the HDDLGym framework, we introduce HDDLEnv, a Python class that extends the Gym environment to support hierarchical planning with HDDL.

Initialize and reset functions HDDLEnv is initialized using HDDL domain and problem files, together with an optional list of policies for all agents. During initialization, the `main_parser` function converts HDDL files into an environment dictionary, setting up the initial state and goals. Agents are then initialized with their associated policies.

The reset function optionally accepts a new or updated list of agents' policies and resets the environment to its initial state and goal tasks as specified in the HDDL problem file. It also re-initializes agents with their associated policies.

Step function The step function in HDDLEnv accepts an action dictionary from the agents and returns the new state, reward, 'done' flag (indicating win or loss), and debug information, similar to the format of OpenAI Gym's step function. After executing agents' actions, it also checks and applies any valid environment actions. Environment actions are any actions that are not associated with any agent. This design enables the environment to change independently from agents' behaviors. If the current state of the environment satisfies the precondition of an environment action, that action is executed automatically.

5.2 Agent

HDDLGym is designed as an agent-centric system. It inherently focuses on the interactions and actions of agents within the environment. Therefore, defining an Agent class, as in Definition 2 below, is critical in implementing HDDLEnv.

Definition 2. *An agent A is defined with a tuple $\langle N, P, B, H, U \rangle$ where:*

- *N is agent name,*
- *P is a policy,*
- *B is set of agents, representing the agent's belief about other agents' configuration in the environment,*
- *H is a list of tasks, methods, and action, representing the action hierarchy of the agent,*
- *U is a function to update the action hierarchy of agent based on the current state of the world.*

Initialize an agent All agents in the environment are initialized when an HDDLEnv instance is created or reset. Each agent is initialized with a name N and a policy P. The agent's name N is derived directly from the HDDL domain and problem files. The policy P refers to an RL strategy that the agent employs to support its hierarchical planning process. This initialization configures the agent to operate within the hierarchical planning framework.

Function U: updates agent's action hierarchy H An important method in the Agent class is the update hierarchy function U. This function checks whether any tasks or actions in the agent's hierarchy H have been completed by comparing their effects with the current state of the world. Once tasks or actions are completed, they are removed from both the agent's hierarchy H and the agent's belief about other agents' action hierarchies (B). U is called for each agent after the environment's step function is executed, ensuring that the agents are prepared to plan the next step.

5.3 Observation and Action Spaces

Observation space In general multi-agent problems, each agent can be assumed to have knowledge about the current state of the world, its own hierarchical actions, and other agents' previous actions. Different RL methods have different designs for which information should be included in the inputs and outputs of the RL policies. For example, in the default setup, we set the observation of each agent to include information on (1) current state of the world, (2) goal tasks, (3) the agent's action hierarchy, and (4) other agents' previous primitive actions. Meanwhile, the RL model should return information about the action hierarchy of each agent.

In our current design, we use dynamic grounded predicates to represent the current state of the world. Dynamic grounded predicates represent a subset of all possible grounded predicates within the environment. In HDDL, and PDDL more broadly, predicates can either be static or dynamic. Static predicates define unchanging world conditions (e.g., spatial relationships between locations), while dynamic predicates represent changing world conditions (e.g., agent positions). Dynamic predicates can be added or removed from the world state by actions.

Our default setup focuses on using dynamic grounded predicates, rather than the full set of grounded predicates, to reduce the observation space. This scalability trade-off is illustrated in Table 1, specifically in Overcooked domain. However, this design choice may limit the generalizability of the RL policy, as it is tailored to a specific set of HDDL problem instances and may not transfer well to problems with different agents, objects, and/or static world conditions. To address this, HDDLGym also allows users to customize the state representations to accommodate diverse needs across domains, for example, using grounded predicates in the Transport domain, where their dimensions remain manageable as the problem size increases (Table 1).

Action space Unlike PDDL or non-hierarchical planning problems, HDDLGym aims to provide not only primitive actions but also the full action hierarchies that reflect high-level strategies guiding agent behavior. As shown in Table 1, the set of all possible grounded operators can grow prohibitively large in complex domains, while lifted operators offer a more compact alternative. A middle-ground approach uses lifted operators with associated objects to retain contextual information. Our default setup uses the middle-ground approach. The action hierarchy of each agent is one-hot encoded over all lifted operators and objects, reducing observation and action space sizes by omitting subtask orders and specific object-operator links. HDDLGym also supports flexible state and action space designs, including multiple RL models for different operator types and customizable encoding schemes for diverse domains and experiments.

Domain	**Transport**			**Overcooked**	
# of Agents	**1**	**2**	**3**	**2**	**3**
G. predicates	35	79	269	937,158	1,186,066
G. dynamic pred.	13	38	132	90	101
G. operators	184	1570	11,979	200,597	300,860
G. actions	58	610	4,093	175,791	263,668
L. operators	14	14	14	18	18
L. actions	4	4	4	5	5
Objects	8	13	23	18	19

Table 1: **Dimensions of Lifted and Grounded Representations in Transport and Overcooked Problems.** *G.* denotes Grounded, and *L.* denotes Lifted. The large number of grounded predicates in Overcooked highlights the need to use dynamic grounded predicates for state representation. Likewise, grounded operators are impractical for defining the action space in RL model training due to their scale.

5.4 RL Policy

The RL policy plays a crucial role in the HDDLGym framework by supporting the search for an optimal hierarchical plan for each agent. In the default setup, the policy takes the observation as input, which includes information about dynamic grounded predicates, goal tasks, and previous action hierarchies. Its output is the probabilities of lifted operators and objects, which are then used to compute the probabilities of grounded operators. These probabilities guide the search for action hierarchies within the HDDLGym planner, as discussed in Sec. 5.5.

In this work, we implemented Proximal Policy Optimization (PPO) (Schulman et al. 2017) for discrete domains to effectively explore the application of RL in hierarchical planning problems. HDDLGym is designed to enable users to flexibly integrate their preferred RL models, including multi-layer perceptrons (MLPs), recurrent neural networks (RNNs), and others. Comprehensive guidance and detailed examples are provided in the tutorial Python notebook included in the codebase.

5.5 Planning for Multi-agent Scenarios

HDDLGym is designed to work in multi-agent settings; therefore, the planner also considers collaboration between agents. The HDDLGym planner is designed in a centralized format. In decentralized planning, each agent runs the centralized planner using its own information and beliefs about the other agents.

Algorithm 1 outlines the approach of the HDDLGym Planner, where agents determine their action hierarchies by iteratively updating through valid operator combinations. Particularly, HDDLGym Planner's inputs are list $\mathcal{A}$ of all agents with uncompleted hierarchies, policy P, and deterministic flag d. The HDDLGym planner is a centralized

planner. In case of decentralized planning, the list $\mathcal{A}$ include a real agent and that agent's belief about other agents. The deterministic flag d determines whether the selection process should follow a deterministic or probabilistic approach when choosing operators to form agents' action hierarchies. The policy *P* guides the search for a suitable hierarchy based on the flag d. The HDDLGym Planner outputs an updated list of agent instances, $\mathcal{A}_{updated}$, with each agent's action hierarchy terminating in a grounded primitive action.

The planner begins by initializing an empty list, *Done*, to keep track of agents whose hierarchies end with an action (line 1). The while loop from lines 2 to 28 continues until all agents have completely updated their hierarchies. Within this loop, an empty list, $O_{\mathcal{A}}$, is initialized to store the valid operators of all agents in $\mathcal{A}$ (line 3). Next, the for-loop from lines 4 to 17 iterates to find all valid operators O_a for each agent *a*, for $a \in \mathcal{A}$. To do this, the algorithm first checks if *a* is in *Done*, meaning its hierarchy is complete (line 5). If so, then O_a is set as a list containing the agent *a*'s primitive action (line 6). Otherwise, the while loop from lines 8 to 14 runs until it finds a non-empty O_a. In this while loop, the list of valid operators for *a* is validated in line 9; if no valid operators are found (line 10), the last operator in *a*'s hierarchy is removed, and the loop is rerun. However, if *a*'s hierarchy is already empty, indicating that no valid operator can be found for *a*, the `none` action is added to O_a (line 12).

The operator list O_a for each agent is then added to $O_{\mathcal{A}}$, the list of operators for all agents (line 16). This list, $O_{\mathcal{A}}$, is subsequently used to generate all combinations of joint operators, *C* (line 18). Line 19 details the pruning of invalid combinations in *C*. A combination is invalid if it violates either of two conditions: first, no agent should perform multiple different actions; and second, no action in the combination should have effects that conflict with the preconditions of other actions. After this pruning, *C* contains only valid operator combinations.

Lines 20 to 25 describe how the policy *P* is applied to select a combination *c* from the list of valid combinations, *C*. The probability list, P_O, corresponding to *C* is generated using policy *P*. Depending on the deterministic flag *d*, the chosen combination *c* is selected in either deterministically (line 22) or probabilistically (line 24).

With the combination of operators determined, the next step is to use it to update each agent's hierarchy (line 26). The list *Done* is then updated if any agents have completed hierarchies (line 27). This process is repeated until all agents in $\mathcal{A}$ have completed their hierarchies. At this point, the HDDLGym planner returns the list of fully updated agents, $\mathcal{A}_{updated}$, as shown on line 29.

Algorithm 1: HDDLGym Planner

Input: list of agents $\mathcal{A}$, deterministic flag *d*, policy *P*
Output: updated list of agents $\mathcal{A}$

1: Initialize an empty list *Done* to keep track of agents whose hierarchies reached action.
2: **while** not all agents in *Done* **do**
3: Initialize an empty list $O_{\mathcal{A}}$ for valid operators of $\mathcal{A}$
4: **for** agent *a* in $\mathcal{A}$ **do**
5: **if** *a* in *Done* **then**
6: $O_a \leftarrow$ [action of agent *a*]
7: **else**
8: **while** O_a not empty **do**
9: $O_a \leftarrow$ a list of valid operators for *a*
10: **if** O_a is empty **then**
11: Remove the last operator of agent *a* hierarchy from its hierarchy
12: If no more operator from *a*'s hierarchy to remove, add `none` action to O_a
13: **end if**
14: **end while**
15: **end if**
16: Add O_a to $O_{\mathcal{A}}$
17: **end for**
18: $C \leftarrow$ Combinations of joint operators from $O_{\mathcal{A}}$
19: Remove any invalid combinations from *C*
20: $P_O \leftarrow$ Probability of combinations in *C* with *P*
21: **if** *d* is True **then**
22: $c \leftarrow argmax_{c \in C} P_O$
23: **else**
24: $c \leftarrow$ Randomly from C with weights be P_O
25: **end if**
26: Update hierarchies of all agent $\mathcal{A}$ with operators in c
27: Check each agent's hierarchy and update *Done* if any hierarchy ends with action
28: **end while**
29: **return** $\mathcal{A}_{updated}$

5.6 HDDLGym Architecture

The high-level architecture of HDDLGym is demonstrated in Figure 2. As discussed in Section 5.3, The RL policy, described in Section 5.4, takes an observation as input and outputs a probability distribution over action representations (see Section 5.3). These probabilities guide the HDDLGym planner in selecting the most appropriate action hierarchy for each agent, as outlined in Algorithm 1 and Section 5.5. Primitive actions are then extracted from the updated hierarchies and executed in the environment, resulting in a new world state via the step function (Section 5.1). Completed tasks or actions are archived and removed from the each agent's action hierarchy H with method U (Section 5.2) before proceeding to the next cycle. This integrated process supports dynamic and adaptive agent behavior based on both learned policies and hierarchical planning.

6 Applications

Table 2 lists the domains, primarily from IPC-HTN (IPC 2023 HTN Tracks) and custom designs, that have been studied with HDDLGym and are included in the codebase[1]. We highlight two representative examples: Transport from IPC-HTN and Overcooked, a popular multi-agent problem in OpenAI Gym.

[1]This list will be updated as the tool evolves. For the most recent version of this work, please refer to the latest arXiv version or visit: https://ngocla.github.io/files/HDDLGym.pdf

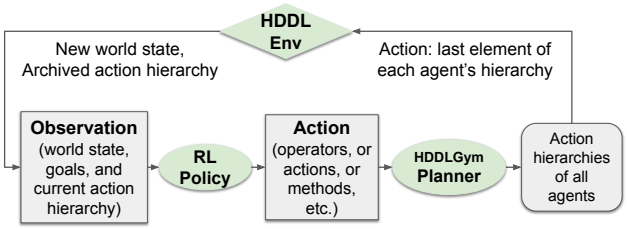


Figure 2: **HDDLGym high-level architecture.** Outputs of RL policy help HDDLGym Planner update the action hierarchy of each agent. Then, primitive actions are extracted from the hierarchies, and applied to the environment.

Domain	Source	Agent-centric	Collab
Transport	IPC-HTN	Yes	No
Transport Collab.	Modified IPC-HTN	Yes	Yes
Overcooked	Ours	Yes	Yes
Rover	IPC-HTN	Yes	No
Satellite	IPC-HTN	Yes	No
Depots	IPC-HTN	Hidden	No
Minecraft-Player	IPC-HTN	Hidden	No
Barman-BDI	IPC-HTN	Hidden	Yes
Search and Rescue	Ours	Yes	Yes
ZenoTravel	IPC-HTN	Yes	No
Taxi	Ours	Yes	No
Factories-simple	IPC-HTN	No	No

Table 2: **List of Domains Included in the HDDLGym Codebase (As of June 2025)**. *Agent-centric* domains are those in which the agent can be explicitly identified. *Hidden* indicates that the domain is agent-centric, but the agent is not specified as an explicit parameter.

6.1 IPC-HTN Domains

As previously discussed, Gym defines interactions between agents and the environment. Therefore, not all HDDL domains from IPC-HTN (IPC 2023 HTN Tracks) are directly compatible with HDDLGym. Since agent specification within a domain is necessary, this requirement may not be feasible or appropriate for every IPC-HTN domain. HDDLGym is particularly well-suited to domains with agent-centric systems, such as Transport (where the vehicle serves as the agent), Rover (with the rover as the agent), and Satellite (with the satellite as the agent). To better illustrate the applications of these agent-centric environments, we provide a detailed discussion on Transport domain as follows.

Transport domain The goal of a Transport problem is to deliver one or more packages from their original locations to designated locations.

The resulting action hierarchy of an 1-agent Transport problem is illustrated in Figure 1a. In this scenario, the truck completes the `delivery package` goal task after four actions. At each step, the truck's action hierarchy begins with the goal task and concludes with a primitive action. The action hierarchy updates after each step, following a sequence of subtasks in `method-deliver` to accomplish the `delivery package` goal.

To evaluate the capability of handling collaborative interactions in Transport domain, we embed the collaborative task, method, and action to the Transport domain. Specifically, task `transfer`, method `m-deliver-collab`, and action `transfer-package` are added in the domain to enable the packages to be transferred from one vehicle to another when the vehicles are at adjacent locations. Details of these collaborative operators can be found in the codebase. Following this template, users can explore more interesting interactions and modifying Transport domain to study heterogeneous multi-agent problems.

Above is an example of how to modify an existing IPC-HTN domain to study with HDDLGym and explore more interesting features for multi-agent hierarchical planning. A similar process can be applied to other domains such as Rover, Satellite, and Barman-BDI, to plan with HDDLGym in single or multi-agent contexts.

6.2 OpenAI Gym-based Domains

Writing HDDL domains and problems for an environment is not trivial, especially domains with complicated interaction rules. While there are many ways to do so, we suggest starting with the goal task, then designing methods to achieve the goal task, then coming up with other intermediate tasks and methods for them, and gradually working to the primitive actions. Here is an example of how HDDLgym is applied in Overcooked environment (Carroll et al. 2019).

Overcooked Overcooked (Carroll et al. 2019) is a popular Gym-based environment for studying RL, modeled after the cooperative, fast-paced mechanics of the original game. Players, acting as chefs (agents), collaborate to prepare an onion soup by placing onions in a pot, cooking, pouring the soup into a bowl, and serving it (see Figure 1b).

In typical Overcooked scenarios, each agent can perform six primitive actions: moving in a 2D gridworld (up, down, left, right), interacting with objects, or doing nothing. Although the whole Overcooked scenarios could be fully defined using HDDL, we found it more efficient to utilize HDDLGym for high-level planning and then apply A* (Duchoň et al. 2014) for motion planning to find the primitive actions as listed above. The core HTN for Overcooked domain is entailed in Figure 3. In the HDDL domain, we define following tasks: make-soup, add-ingredient, cook, deliver, wait, and task-interact. Each of them has one or more method to complete the tasks. Figure 3 only lists several key HTNs of the domain, though all HDDL domain and problem files of Overcooked environment can be found from the codebase. Additionally, Figure 1b shows an example of a hierarchy of an agent and its belief about the other agent's hierarchy.

The following videos help visualize the result of combining HDDLgym in task planning and use A* for motion planning in various Overcooked layouts:

Bottleneck — https://tinyurl.com/hddlBottleNeckRoom
Coord. ring — https://tinyurl.com/hddlCoordinationRing
Left isle — https://tinyurl.com/hddlLeftIsle
Counter circuit — https://tinyurl.com/hddlCounterCircuit
Cramped room — https://tinyurl.com/hddlCrampedRoom

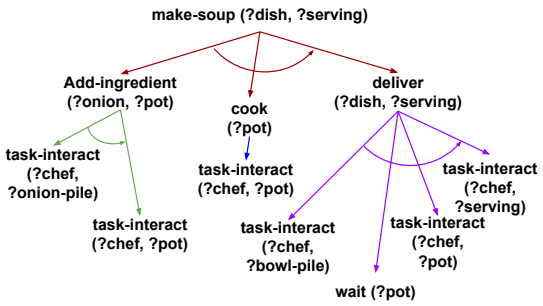


Figure 3: HTNs of the Overcooked domain.

6.3 Evaluation Metrics

Complexity and Difficulty HDDLGym provides a range of metrics to evaluate RL models within hierarchical planning contexts, with particular focus on plan complexity and planning difficulty. Plan complexity can be assessed by examining the dimensionality of the elements defining a problem, as summarized in Table 1. Complementing this, planning difficulty can be estimated prior to training by measuring planning time, number of planning steps, and success rate using random exploration. Table 3 presents such evaluations across the Transport, Overcooked, Rover, and Satellite domains as the number of agents and problem size increase.

Several domains show a 0% success rate with three or more agents, highlighting scalability issues during training. Low success probabilities from exploration increase the risk of low-quality data, which can degrade model performance.

These metrics also enable comparisons between different methods, also known as hierarchical task networks (HTNs). For example, in the Transport domain, adding a collaborative method in simple scenarios with one or two homogeneous agents can unnecessarily increase planning time. However, in more complex settings with heterogeneous agents, collaborative strategies improve efficiency, reducing plan lengths (e.g., 25 vs. 18 steps).

In summary, evaluating the complexity and difficulty of the plan in HDDLGym prior to training provides valuable information on the scalability of the domain. This helps users better tailor HTN structures and adjust training strategies, such as reward shaping or extending exploration horizons, to improve convergence and model performance.

Training Several metrics are implemented to evaluate the RL training process. The first is the loss value, which monitors the convergence rate and is illustrated in plots A–C of Figure 4 for the Transport 1-agent, 2-agent, and 3-agent problems. Loss plots are generated and saved at specified intervals during training to help users track the learning progress. Secondly, the policy is periodically evaluated using quantitative metrics such as cumulative discounted reward, success rate, planning time, and planning steps to assess whether the RL policy is converging toward an optimal solution. These evaluation results are recorded in two graphs, as shown for the Transport 1-agent problem in parts D and E of Figure 4.

Domain	# agents	Plan time (sec)	Avg steps.	Success rate
Transport	1	0.059	21	100%
Transport Collab	1	0.077	22	100%
Transport	2	0.374	41	38%
Transport Collab.	2	1.317	35	46%
Transport Collab.	hetero. 2	0.795	18	1%
Transport	hetero. 2	0.367	26	2%
Transport	3	NA	NA	0%
Transport Collab.	3	NA	NA	0%
Overcooked	2	102.267	26	80%
Overcooked	3	194.794	28	40%
Rover	1	21.360	39	59%
Rover	2	87.305	38	11%
Rover	3	120.732	49	2%
Rover	4	NA	NA	0%
Satellite	1	0.036	12	100%
Satellite	2	1.600	21	78%
Satellite	3	6.915	25	44%
Satellite	4	NA	NA	0%

Table 3: **Plan Difficulty of Transport, Overcooked, Rover, and Satellite Problems.** *Hetero.* indicates heterogeneous agents, meaning agents have different capabilities. *NA* means no successful run occur within the maximum episode steps of 100.

Deployment After training, the next step is to deploy and evaluate the RL models. HDDLGym supports a variety of standard quantitative evaluation metrics, including the execution time of the RL-assisted planner, the number of planning steps needed to achieve the specified goals, and the success rate, similar to those reported in Table 3. For qualitative assessment, HDDLGym provides a visualization tool with detailed usage instructions in the codebase, allowing users to examine the action hierarchy of each agent step by step. This visualization can be integrated with the domain renderer, for example, in Overcooked demonstration videos discussed in Section 6.2, to facilitate a more comprehensive evaluation of agent performance during deployment. Together, these quantitative and qualitative tools support in-depth analysis, helping users refine their models and compare performance across different training settings and domain configurations.

7 Discussion and Future Work

We introduced HDDLGym, which transforms HDDL-defined hierarchical problems into Gym environments, enabling the use of RL policies in hierarchical planning systems. By prioritizing scalability in observation and action spaces, HDDLGym makes trade-offs that enhance complexity handling at the cost of slight accuracy loss in RL models. This flexibility is crucial for tackling intricate tasks. Additionally, HDDLGym supports multi-agent environments, enriching the framework for studying collaborative dynamics in hierarchical planning and offering engaging RL research scenarios.

HDDLGym currently operates under certain limitations that we aim to address in future developments. First, it can only handle discrete state and action spaces, which restricts its application to scenarios that require continuous or hybrid spaces. Furthermore, HDDLGym assumes a deterministic transition function, meaning that action effects are pre-

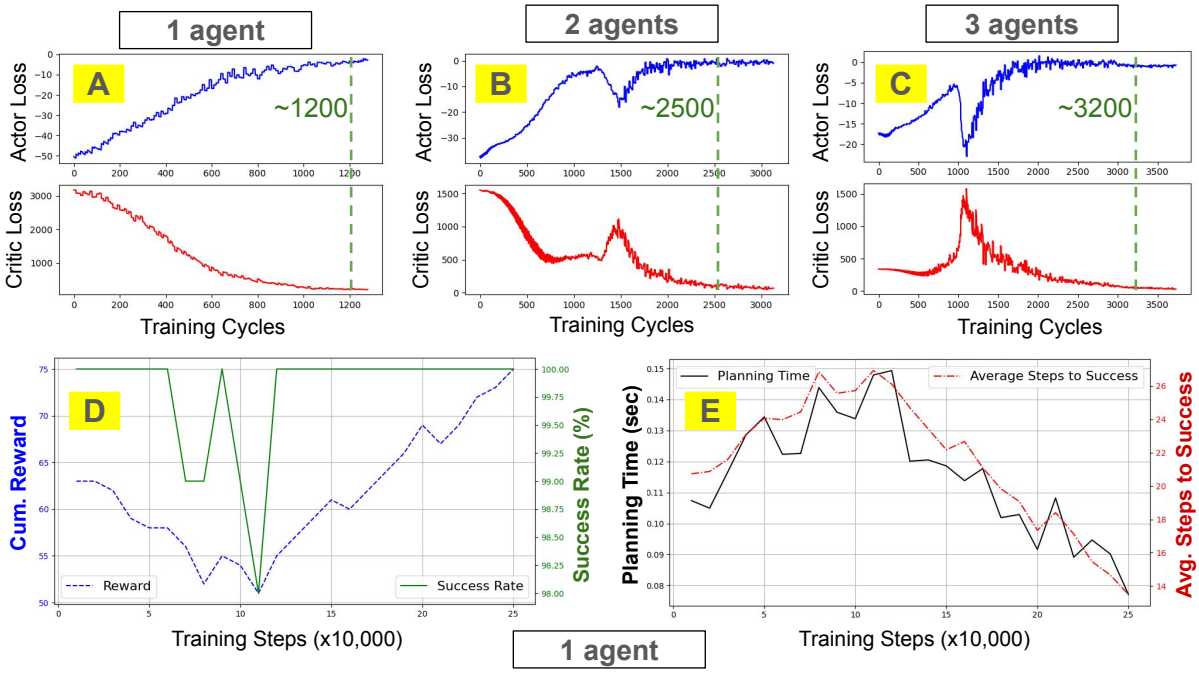


Figure 4: **Training Dynamics Analysis.** Figures A, B, and C show the training losses for the Transport domain with 1, 2, and 3 agents, highlighting longer convergence times as the number of agents increases. Figures D and E display the PPO policy's training progression for the 1-agent Transport problem, including cumulative reward, success rate, planning time, and steps.

dictable and do not account for probabilistic outcomes. This limits its applicability to environments where uncertainty and stochastic outcomes are common. Lastly, the use of one-hot encoding for observations as input to an RL policy restricts its applicability to problems involving similar objects. In many cases, when only dynamic predicates are used for observations, the RL policy is confined to a fixed static condition. Changes like varying agent numbers, adding/removing objects, or altering static conditions require a different RL policy, limiting scalability and adaptability across scenarios within the same domain. Overcoming these challenges will be crucial for expanding HDDLGym's applicability to complex, real-world settings.

In the future, HDDL domains can be learned autonomously through advances in: (i) offline learning of HDDL domains from observations (Grand, Pellier, and Fiorino 2022), (ii) offline learning of HTNs from observations (Li et al. 2014; Zhuo, Muñoz-Avila, and Yang 2014; Li et al. 2022), (iii) online learning of PDDL-like domains (Ng and Petrick 2019; Lamanna et al. 2021; Verma, Marpally, and Srivastava 2021, 2022), and (iv) online learning of HDDL domains from human input with Large Language Models (Fine-Morris et al. 2024; Favier et al. 2025).

As discussed, HDDLGym has limitations that could be addressed to better support complex multi-agent hierarchical problems. An improvement is enabling HDDLGym to handle multiple pairs of HDDL domain and problem files for different agents within a single Gym environment. Inspired by how multi-agent features are added to PDDL and HTNs through MA-PDDL (Kovacs 2012) and MA-HTN (Cardoso and Bordini 2017), respectively, this approach would allow each heterogeneous agent to operate with its own unique pair of HDDL domain and problem files. This capability would enhance HDDLGym's ability to manage complex multi-agent dynamics beyond simple collaboration, supporting scenarios with competition, agent privacy, and distributed context information.

8 Conclusion

In this work, we introduce HDDLGym, a tool for applying RL to hierarchical planning by converting HDDL-defined problems into Gym environments. Its design balances scalability and functionality, enabling multi-agent interactions and complex task structures. We hope HDDLGym opens new possibilities for studying RL in hierarchical planning, especially in multi-agent contexts.

Acknowledgments

We gratefully acknowledge the financial support of the Office of Naval Research (ONR), under grant N000142312883, and sincerely thank Pulkit Verma for his valuable insights and feedback on the project.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

Cardoso, R. C.; and Bordini, R. H. 2017. A Multi-Agent Extension of a Hierarchical Task Network Planning Formalism. *Advances in Distributed Computing and Artificial Intelligence Journal*, 6(2): 5–17.

Carroll, M.; Shah, R.; Ho, M. K.; Griffiths, T.; Seshia, S.; Abbeel, P.; and Dragan, A. 2019. On the Utility of Learning About Humans for Human-AI Coordination. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Duchoň, F.; Babinec, A.; Kajan, M.; Beňo, P.; Florek, M.; Fico, T.; and Jurišica, L. 2014. Path Planning with Modified A Star Algorithm for a Mobile Robot. *Procedia Engineering*, 96: 59–69.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS)*.

Favier, A.; Verma, P.; La, N.; and Shah, J. A. 2025. Leveraging LLMs for Collaborative Human-AI Decision Making. In *Proceedings of the AAAI 2025 Spring Symposium on Current and Future Varieties of Human-AI Collaboration*.

Fine-Morris, M.; Hsiao, V.; Smith, L. N.; Hiatt, L. M.; and Roberts, M. 2024. Leveraging LLMs for Generating Document-Informed Hierarchical Planning Models: A Proposal. In *AAAI 2025 Workshop on Planning in the Era of LLMs (LM4Plan)*.

Goel, S.; Wei, Y.; Lymperopoulos, P.; Churá, K.; Scheutz, M.; and Sinapov, J. 2024. NovelGym: A Flexible Ecosystem for Hybrid Planning and Learning Agents Designed for Open Worlds. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Grand, M.; Pellier, D.; and Fiorino, H. 2022. An Accurate HDDL Domain Learning Algorithm from Partial and Noisy Observations. In *Proceedings of the IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*.

IPC 2023 HTN Tracks. 2023. International Planning Competition 2023 HTN Tracks. Available at https://ipc2023-htn.github.io/.

Kovacs, D. L. 2012. A Multi-Agent Extension of PDDL3.1. In *Proceedings of the ICAPS 2012 Workshop on the International Planning Competition (WS–IPC)*.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021. Online Learning of Action Models for PDDL Planning. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*.

Li, N.; Cushing, W.; Kambhampati, S.; and Yoon, S. 2014. Learning Probabilistic Hierarchical Task Networks as Probabilistic Context-Free Grammars to Capture User Preferences. *ACM Transactions on Intelligent Systems and Technology*, 5(2).

Li, R.; Roberts, M.; Fine-Morris, M.; and Nau, D. 2022. Teaching an HTN Learner. In *Proceedings of the 5th ICAPS Workshop on Hierarchical Planning (HPlan)*.

Liu, M.; Sivakumar, K.; Omidshafiei, S.; Amato, C.; and How, J. P. 2017. Learning for Multi-Robot Cooperation in Partially Observable Stochastic Environments with Macro-Actions. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Ng, J. H. A.; and Petrick, R. P. A. 2019. Incremental Learning of Planning Actions in Model-Based Reinforcement Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): Language Description. *Unpublished ms. Australian National University*, 32: 27.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. In *ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

Taitler, A.; Gimelfarb, M.; Jeong, J.; Gopalakrishnan, S.; Mladenov, M.; Liu, X.; and Sanner, S. 2023. pyRDDLGym: From RDDL to Gym Environments. In *ICAPS 2023 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the Right Questions: Learning Interpretable Action Models Through Query Answering. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2022. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*.

Wu, S. A.; Wang, R. E.; Evans, J. A.; Tenenbaum, J. B.; Parkes, D. C.; and Kleiman-Weiner, M. 2021. Too Many Cooks: Bayesian Inference for Coordinating Multi-Agent Collaboration. *Topics in Cognitive Science*, 13(2): 414–432.

Xiao, Y.; Hoffman, J.; and Amato, C. 2020. Macro-Action-Based Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 3rd Conference on Robot Learning (CoRL)*.

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2014. Learning Hierarchical Task Network Domains from Partially Observed Plan Traces. *Artificial Intelligence*, 212: 134–157.