Are We There Yet? Bridging the Knowledge Acquisition Gap in Automated Planning

Roman Barták[1], **Lukáš Chrpa**[2,3], **Simona Ondrčková**[1], **Kristýna Pantůčková**[1],

[1] Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
[2] Czech Technical University, Czech Institute of Informatics, Robotics and Cybernetics, Prague, Czech Republic
[3] Filuta AI, Inc., Austin, TX, United States
bartak@ktiml.mff.cuni.cz, chrpaluk@cvut.cz, ondrckova@ktiml.mff.cuni.cz, pantuckova@ktiml.mff.cuni.cz

Abstract

Automated planning addresses reasoning about agents' actions. It is a model-based approach requiring a formal model of actions and their interaction with the environment. The model is frequently expressed in a formal language such as PDDL (Planning Domain Definition Language), and the models are usually designed by human experts who are experienced in both automated planning technology and the problem domain. This requirement is a significant bottleneck for the practical application of automated planning (and other model-based approaches), hence it is often referred to as a knowledge acquisition gap. In this paper, we propose a pipeline that bridges this gap for automated planning, presenting the individual steps within the context of existing and future research.

Introduction

Classical symbolic Artificial Intelligence (AI) is often based on an explicit formal model that abstracts the problem to be solved. Having such a model brings several advantages. First, it formally separates the "nasty" real world from the solving techniques. The model encodes the proper information needed for problem solving (abstraction), and researchers can focus on improving (the efficiency of) the solving techniques independently of the real world. Having an explicit, human-understandable model also provides a basis for explaining solutions, which contributes to the trustworthiness of AI systems. Finally, artificial agents, such as robots, require precise instructions on what to do, particularly in high-stakes areas where their behaviour must be trustworthy, reliable, and verifiable. Subsymbolic approaches cannot plan and reason to the required quality. Formal reasoners can do that, but they require the formal model.

On the other hand, the formal model is a bottleneck for symbolic problem-solving, as solvers cannot operate without a (good) model. Creating and maintaining models is a tedious task, frequently performed by experienced humans who are proficient in both the solving technique and the problem domain. This gap between the real world and abstract solving techniques is often referred to as a *knowledge acquisition gap*. Its existence has contributed to the development of machine learning techniques (ML) that can automatically construct formal models. In fact, ML can learn directly the mapping from the problem (even non-abstracted) to a solution, eliminating the need for an explicit formal model and a solver for it, so-called *model-free approaches*. Recent successes of sub-symbolic neural network techniques in areas traditionally complex for symbolic approaches, such as natural language processing and computer vision, support the idea that model-free approaches can solve any problem without substantial human assistance or an explicit model. It seems that (a lot of) data substituted knowledge (expressed through the formal model). We argue that this is not the case; having explicit models remains helpful and necessary for handling complex situations, particularly in high-stakes areas.

Automated planning is a traditional model-based approach to reasoning about the agent's future actions. The explicit symbolic model describes, in a human-readable form, how actions modify the world. Significant progress has been made in recent years in improving the efficiency of automated planners. This progress can be attributed, at least in part, to the formalization of a method for describing the problem domain formally. This model is very general, capturing "physics-only" properties of the world, that is, how actions interact with the environment. There exist methods that provide planners with additional information to improve their efficiency, such as hierarchical task networks or control rules; however, these methods further exacerbate the knowledge gap. These hints are therefore kept separate from the core domain model, which further challenges the developers of automated planners. Note also that large language models, the recent horsepower of AI, cannot compete with automated planners (Valmeekam, Stechly, and Kambhampati 2024), which advocates the power of formal models.

However, having (many) planning problems formulated in these formal models, typically designed for international planning competitions (McDermott 2000; Long and Fox 2003; Vallati et al. 2015), suppressed the research community's interest in how to obtain such formal models, thereby negatively affecting the practical applicability of automated planning. Like any approach based on formal models, the use of human modelers restricts the scalability of the technology, and there is a need for automated modeling. Recent years have shown an increasing number of publications on automated modeling of planning domains (see below).

However, these approaches are usually standalone or single-step, in which a complete domain model is obtained from input data under various assumptions about the input content (Arora et al. 2018).

In this paper, we propose a pipeline that spans from raw data to a formal planning model and beyond. The holy grail is that the agent observes the environment and automatically builds a model of the environment and its own capabilities for modifying it, together with knowledge on how to use these models efficiently when reasoning about future actions (i.e., planning). The primary motivation for this paper is to identify and separate the steps required in the knowledge acquisition process, discuss relevant approaches from the literature, as well as to identify gaps and open research problems. Having such a pipeline explicitly places existing techniques within the domain model acquisition process. It integrates them to create a modular system, allowing researchers to improve individual components separately while maintaining the overall vision of the process.

Preliminaries

Numerous formal models of planning problems exist, which are best reflected in various versions and variants of the Planning Domain Definition Language (PDDL). This language (Ghallab et al. 1998) has been developed for International Planning Competitions (McDermott 2000), and standardization had a very positive effect on the development of planning technology. For the purposes of this paper, specifically to understand the steps of the proposed pipeline, we focus on the core concepts of the planning domain model, which originated in the STRIPS formalism (Fikes and Nilsson 1971) and are captured in PDDL 1.2.

The properties of the world are described with a set of propositions p, relations over entities (objects) of the world. For example, proposition *(at R1)* describes that an agent is in a room $R1$ (we use the LISP syntax that is common in PDDL). Objects may have a type, for example, $R1$ is of type $room$, and there may be an entire hierarchy of types and their subtypes, for example, $room$ is a subtype of a more general type $location$. A world state is modeled as a set of propositions that represent what is true at a given moment. The action is then modeled as a four-tuple: $(\texttt{pre}^+(a), \texttt{pre}^-(a), \texttt{eff}^+(a), \texttt{eff}^-(a))$. The sets $\texttt{eff}^+(a)$, $\texttt{eff}^-(a)$ represent effects describing how an action changes the world, what propositions become true (a positive effect) and false (a negative effect) in the state right after executing the action. The state after the action is obtained by removing negative effects and adding positive effects to the current state: $(s \setminus \texttt{eff}^-(a)) \cup \texttt{eff}^+(a)$. This approach naturally solves the *frame problem* (McCarthy and Hayes 1969). The sets $\texttt{pre}^+(a)$, $\texttt{pre}^-(a)$ represent the preconditions of an action which must be true (or false) in the state preceding the action to allow the execution of the action ($\texttt{pre}^+(a) \subset s$, $\texttt{pre}^-(a) \cap s = \emptyset$). Actions basically describe how the agent modifies the world. They are obtained by substituting object names into attributes (parameters) of planning operators that can be seen as templates for actions. The operator attributes may be of a certain type (the variable $?to$ must be of type $room$).

```
(:action move
   :parameters (?from - room ?to - room)
   :precondition ((at ?from))
   :effect (and (at ?to)
                (not (at ?from))))
```

To specify a particular planning problem, one needs to describe an initial state, which also specifies all the entities in the world, and a goal condition that specifies the desired properties of the world. The task for the planner is to find an executable sequence of actions (executable means that the precondition of each action is satisfied in the state preceding it), modifying the world from its initial state to a state satisfying the goal condition.

The above model describes the core planning task that will be reflected in the knowledge acquisition pipeline proposed in the next section. Numerous model extensions to cover numerical fluents, explicit time (durative actions), probabilistic effects, rewards, and so on can be naturally added to the suggested pipeline.

Knowledge Acquisition Pipeline for Planning

Due to a focus on physics-only models, the problem modelling does not have the same tradition in automated planning as in related areas such as constraint programming (CP). While in CP and similar areas, the model plays a crucial role in efficient problem solving, and hence good modeling practices are important there; the role of problem modeling, with some exceptions (Barták and Vodrázka 2016), has not been the primary focus of the planning community. Although Knowledge Engineering in Planning and Scheduling (KEPS) has not attracted as much attention as the development of planning solvers, some important works merit mention. Tools such as GIPO (Simpson, Kitchin, and McCluskey 2007), ItSimple (Vaquero et al. 2013) or `planning.domains` provide a useful assistance in human-driven domain modelling. Shah et al. (2013) studied how the use of modelling tools (ItSimple, in that case) affects the KEPS process in a Traffic Incident Management domain. An important step of the process is a conceptualisation of the (real-world) requirements into a formal description of the model, from which the PDDL model is then refined (by hand or with a tool). McCluskey, Vaquero, and Vallati (2017) introduce and elaborate on properties that a good quality domain model should have. Although those properties are mostly qualitative, they provide an important measure for human modellers. A recent work elaborates on the KEPS process in a real-world application concerning traffic signal control (Bhatnagar et al. 2022). It is also good to mention that recent editions of the International Competition on KEPS were focused on domain modeling with requirements distilled from real/realistic scenarios (Chrpa et al. 2017).

In a nutshell, the major modeling decision concerns the set of propositions describing the properties of world states and actions. Once this decision is made, the action model is straightforward. On the other hand, it helps keep knowledge of how the world evolves through actions and how good plans should look (and can be obtained) clearly separated,

hence making it easier to manage. We will reflect this in the proposed knowledge acquisition pipeline.

Automated acquisition of planning domain models is not a new topic (Yang, Wu, and Jiang 2007), (Amir and Chang 2008). However, the vast majority of research focuses on a single, monolithic step that leads directly from raw data to a formal model (in PDDL). Frequently, the raw data already define the propositions modeling world states, and the traces of states determine the actions (Callanan et al. 2022). In this paper, we propose smaller steps, following the knowledge engineering process but focusing on a fully automated pipeline rather than a human modeler. These steps are motivated by components of a planning domain model, so it is not only about learning an action model but also about identifying appropriate propositions that model world states (including relevant entities/objects in the state). Moreover, the efficiency of planning and the agent's autonomy are also assumed. Splitting the pipeline allows us to use the best technology for each step, and we hope it will help the research community bridge the knowledge acquisition gap more quickly.

From Raw to Symbolic

As the symbolic planning model is based on entities represented by constants (for example, $R1$ modeling a particular room), the first obvious step in domain modeling is to identify (and name) these entities in real-world states. This step is often neglected in the planning community, as it is assumed that the object-type hierarchy (objects, their types, and subtypes) is already provided. However, when approaching a new problem area, the modeler must decide which types of objects to include in the model. Hence, we explicitly include object extraction and type classification as the first step in the proposed pipeline.

While the formal model determines the output of this step – it is the object-type hierarchy – the input may vary significantly. The input could be a visual observation of the scene, i.e., a set of images or videos, a textual description of the domain, a stream of sensory data, knowledge of a human expert, or a combination of more types of input. Each type of input significantly influences the extraction method ranging from image recognition, natural language processing, and planning expert involvement. It is, however, crucial, in this step, to determine the appropriate level of abstraction as the identified types of objects are important in the following steps of the pipeline. We refer to this step as "from raw to symbolic" to emphasize the role of symbol extraction from raw data. A possible output of this stage in terms of PDDL can be seen in Figure 1 (a type hierarchy for the Logistics domain).

We should now highlight that, even though we present the pipeline as a linear sequence of smaller steps, later steps may influence the decisions of earlier steps. For example, the action model constructed later in the pipeline naturally influences which entities in the scene are relevant to the domain model. In this first step, a natural approach is to identify as many entities as possible and to filter out irrelevant entities (or to group entities to abstract them) later in the pipeline.

An example of a work with text as an input is work by Yang, Wu, and Yue (2025), which learns Prolog facts. Krafczyk, El-Sharkawy, and Schmid (2018) proposed an approach to transform conditions with integer variables into propositional formulas. Other approaches focus on translating continuous information into symbolic representation (Konidaris, Kaelbling, and Lozano-Perez 2014; Nevens, Van Eecke, and Beuls 2020; Konidaris, Kaelbling, and Lozano-Perez 2018).

```
(:types
    package location vehicle - object
    truck airplane - vehicle
    city airport - location)
(:objects
    package1 - package
    package2 - package
    airplane1 - airplane
    airplane2 - airplane
    bos - city
    la - city
    bos-truck - truck
    la-truck - truck
    bos-po - location
    la-po - location
    bos-central - location
    la-central - location
    bos-airport - (either airport location)
    la-airport - (either airport location))
```

Figure 1: Type hierarchy for the logistics domain

From Symbolic to Structural

Planning domain models use propositions to describe relations among objects; the world state is then modelled using propositions that are true at that state, and actions modify the validity of some propositions. Again, many domain-learning approaches assume a given state model, but in practice, the modeler must decide which properties of the environment to capture. Hence, we explicitly include this step, which we refer to as the "from symbolic to structural" step, to highlight the role of relations. The output of this (and also previous) step is a formal model of world states appropriate for planning. Note that automated planning is usually assumed to use factored representation of states (Russell and Norvig 2020), but this is the case of grounded models. In contrast, the lifted models with variables are closer to the structured representation.

Again, other steps in the pipeline influence this step. One can extract some relations directly from a static scene, such as the relative locations of objects. However, for other relations, one may need to observe the evolution of a property in time. Typically, relations whose validity changes in time, so-called *fluents*, are important for the later definition of actions. However, some properties, which are not changing in time, might also be important to define preconditions of actions (so-called *rigid* predicates). An example could be the neighborhood relation between two locations x and y that allows the agent to move directly from x to y. This relation between locations does not change over time, so it may be hard to recognize its importance for the action model (ac-

tions describe changes in the environment). However, if this relation is not included in the model, the agent may plan to teleport to any location. Rigid predicates make learning of action models more complicated (Cresswell, McCluskey, and West 2013a).

The situation is even more complicated, as the action model may require relations (predicates) derived from other (primal) relations. In the well-known Blockworld problem, where the agent is moving blocks to build towers, the primal relations could be relative locations of blocks, e.g., $on(A, B)$ indicating that block A lies on block B. The action to pick up the block x requires x to be clear, that is, to be free of any other block. This property is usually modeled using relation $clear(x)$, which is actually derived from the primal relations: $clear(x) \equiv \nexists y : on(y, x)$. These derived relations might be even more important for learning control knowledge (see below), for example, the relations $goodtower$ and $badtower$ used in the Blockworld domain (Chrpa et al. 2022). $goodtower(x)$ means that the tower below the block x does not need to be touched (deconstructed) to achieve the goal so the agent should not pickup block x in the plan as the $pickup(x)$ action is not necessary to achieve the goal (even though the action might be applicable in a given state). This relation is actually defined across the states as it requires information about relations valid in the goal state. Figure 2 explains the above predicates.

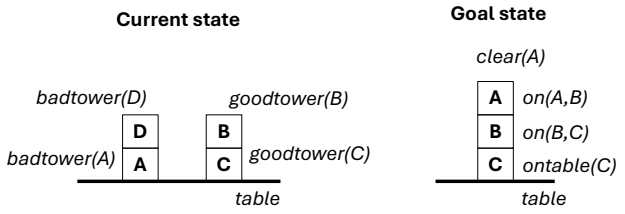

Figure 2: Two states from the Blockworld problem with relevant entities (A, B, C, D, table) and propositions (on, ontable, clear, goodtower, badtower).

The open question is how to find out which relations/propositions are important for domain modeling. Similarly to the first step, one may generate as many propositions as possible and later filter out those not relevant to the action model (derived predicates may be added later as needed by other steps). A possible output of this stage in PDDL may look like this (predicates from the Blockworld domain without types):

```
(:predicates (clear ?x)
             (on-table ?x)
             (arm-empty)
             (holding ?x)
             (on ?x ?y))
```

An example of an automatic technique in this area is predicate invention, which can create new predicates to structure a domain. The current work using this technique is covered in a survey by Kramer and Bessiere (2020).

From Structural to Temporal

Given a model of world states, the next step is to describe the evolution of world states through actions to construct the action model. We focus primarily on actions rather than modeling state transitions outside the agent's control. Actions describe how the agent changes the environment; in classical planning, an action changes the state's properties, so one needs to identify the state before and after the action to see the difference. This itself is an interesting problem of identifying the key frames in a flow of states. See Figure 3 for the example of several frames/real states captured with some frequency when playing the sliding-tile puzzle. Only the first and last frames are important for describing the abstract action of moving tile 4, and the question is how to distinguish these key frames. This example again demonstrates the dependence between the pipeline steps. If the previous step uses an abstraction of tile locations to only 9 grid positions rather than exact (x, y) coordinates, then deciding the key frames for action-model learning is easier. The keyframes are those where the validity of any predicate is modified, which are only the first and last frames in Figure 3.

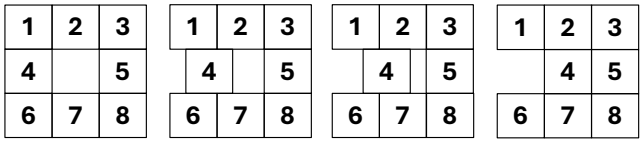

Figure 3: Four real states captured from the sliding-tile puzzle with only the first and last states being relevant to define the move action.

The output of this stage in PDDL is a complete action model, that is, describing actions, or more precisely planning operators with preconditions and effects.

This stage of formulating/learning an action model is where the majority of research in the automated planning domain model acquisition falls (Arregui, Celorrio, and de la Rosa Turbides 2010). The output of learning methods is typically a comprehensive domain model (in PDDL), and the methods differ in their assumptions about the input to learn from. We refer to this step as "from structural to temporal" to emphasize the temporal aspect. Note that temporal relations between actions could be more complex than simple effect-precondition (causal) relations. This is an example of a planning operator from the Blockworld domain:

```
(: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))))
```

Many techniques for learning planning domain models have been proposed: SAM (Juba, Le, and Stern 2021),

COND-SAM (Mordoch et al. 2024), LOCM (Cresswell, McCluskey, and West 2013b), POLOCM (Liu and Haslum 2025), NOLAM (Lamanna and Serafini 2024), ARMS (Yang, Wu, and Jiang 2007), SLAF (Amir and Chang 2008), FAMA (Aineto García 2022), OffLAM (Lamanna et al. 2025), OLAM (Lamanna et al. 2021), and L1 (Balyo et al. 2024), to name a few. Learning a PDDL domain model usually means learning action preconditions and effects from plan traces. Plan traces are sequences of actions with parameters combined with some state information. The state information can range from complete information about each state in between the actions (SAM (Juba, Le, and Stern 2021), COND-SAM (Mordoch et al. 2024)) to no information about the states at all (including no initial state and no goal state information) (LOCM (Cresswell, McCluskey, and West 2013b), POLOCM (Liu and Haslum 2025)). Some algorithms may assume that given information could be incorrect (NOLAM (Lamanna and Serafini 2024)) or that it is only partially observed (SLAF (Amir and Chang 2008), ARMS (Yang, Wu, and Jiang 2007), FAMA (Aineto, Celorrio, and Onaindia 2019), OffLAM (Lamanna et al. 2025)). The L1 (Balyo et al. 2024) algorithm requires complete state information, but besides learning preconditions and effects, it also learns the parameters of actions. There is an algorithm that handles online learning by alternating between learning and exploration phases (OLAM (Lamanna et al. 2021)).

Latplan (Asai et al. 2022), and its extension R-Latplan (Barbin, Cerutti, and Gerevini 2024) learn a PDDL action model with a unique form of input – an unlabeled set of image pairs that shows a transition from one image to the next, where each image describes a state. It also receives information about actions that the agent may make. The authors have also created a planner that takes an image pair of an initial state and a goal state and creates a visualized plan. Note that the first and last picture in Figure 3 could be an example of an input for Latplan. The output would be some representation of an action slide: *slide(4,middle-left,middle-middle)*, that represents sliding the number 4 from the *middle-left* position to the *middle-middle* position. The ROSAME (Xi, Gould, and Thiébaux 2024) algorithm learns preconditions and effects from probabilistic plan traces. The authors have also combined it with a deep learning computer vision model and learn action models from visual plan traces.

One might ask why the slide action would have three parameters, could it not be represented with two (the number and the final destination): *slide(4, middle-middle)*? This simply depends on the input. As mentioned before, most of the algorithms above take planning traces as an input (which contain action description - name and parameters). But an interesting area for future work for algorithms that would not take any action information as input could be a discussion of what represents a good action. Which of the two slide actions is better?

Just because a model is generated does not mean it is necessarily correct. Several potential errors can occur in such a model. We can split them into two main groups: syntax errors and semantic errors. Syntax errors are, for example, inconsistent parameter use, undefined entities (like nonexistent predicates), duplicated definitions, or duplicated/-cyclic ordering (Sleath and Bercher 2023). Semantic errors may be an action, whose preconditions could never be satisfied, redundant effects, immutable predicate (a predicate that never occurs in action effects so it remains the same through any plan and therefore is redundant) and complementary effects – an action has a positive and negative effect of the same predicate (though some planners can handle this by giving a positive effect priority) (Sleath and Bercher 2023). Sleath and Bercher (2023) evaluated existing PDDL and HTN parsers and tested how well they can handle domains with these flaws. This method can be used to test whether an AI-generated model is free of these flaws. Oswald et al. (2024) evaluated LLM's (large language model's) ability to create PDDL. Aside from syntax and semantic errors, they also compared the generated model to the original human-created model. Once the models are evaluated, the results can be used to correct the models (Lin et al. 2025) or to improve them (Smirnov et al. 2024).

Another way to evaluate a model is to examine its predictive ability or problem-solving ability (Stern et al. 2025). This can be done if an environment can be created with an agent. The agent's actions are considered the plan traces. The predictive ability then checks the applicability of actions. So one can compare whether the action in the learned model can be applied in the same states as in the environment. Similarly, one can compare the effects of an action. The problem-solving metrics then focus on the solvability. For example, one may calculate a false plan ratio, which is the fraction of some set of plans that represents the plans that the learned model solved but cannot be solved by the agent in the environment (Stern et al. 2025). The arguments for comparing the learned model to the environment instead of some human-made reference model are such that it is not certain that the reference model correctly describes the environment. Also, there might be multiple domain models that both accurately represent the environment but are different syntactically. How would one determine which of these is the best model and therefore which model should be compared with the learned model (Stern et al. 2025)?

From Temporal to Efficient

As mentioned in the introduction, pure planning domain models focus solely on modeling the physics of the world, specifically how actions modify the state of the world. However, to improve the efficiency of planning and also to make plans more predictable (which is important when artificial agents operate in a human-inhabited environment), there exist various extensions of a pure model, for example, hierarchical task networks grouping actions into meaningful tasks, or control rules restricting the sequences of actions beyond causal relations. As these extensions typically operate on top of the action model, we refer to this step as "from temporal to efficient."

The goal of this stage is to enhance the action model with extra information describing good plans. This information is again obtained from state/action traces and may focus on extra conditions for action applicability (control knowledge) or typical action sequences (macro-actions, higher-level tasks). For example, in the Blockworld domain, if the block is al-

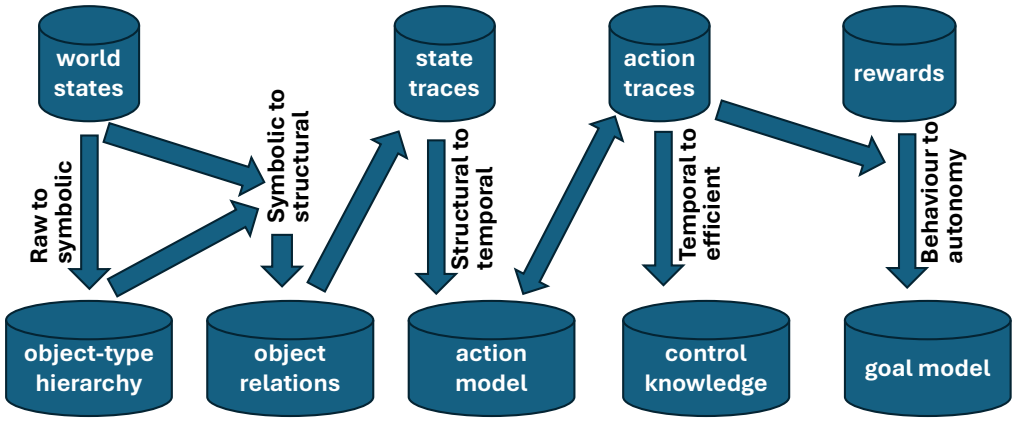

Figure 4: Schematic description of pipeline for automated knowledge acquisition in automated planning.

ready properly placed, we should not unstack it, as such an action is not needed in any plan leading to a given goal (recall that the property of being properly placed also depends on the goal state). This knowledge can be expressed directly in the action model by adding an extra precondition $badtower(x)$ to the action $unstack(x, underx)$. However, a more common approach is to keep this information separate, for example, as control rules (Minton and Carbonell 1987), to maintain the flexibility of the action model for different goals.

Similarly, *hierarchical task networks* (Erol, Hendler, and Nau 1996) capture typical sequences of actions and give them a name of the task. Then, when the agent performs that task, it can decompose it directly into the action sequence rather than generating it from scratch. This is an example of the task decomposition method in HDDL (Höller et al. 2020) for the task of delivering a package to some location:

```
(:method m_deliver_ordering_0
    :parameters (?l1 - location
            ?l2 - location
            ?p - package
            ?v - vehicle)
  :task (deliver ?p ?l2)
  :subtasks (and
     (task0 (get_to ?v ?l1))
     (task1 (load ?v ?l1 ?p))
     (task2 (get_to ?v ?l2))
     (task3 (unload ?v ?l2 ?p)))
  :ordering (and
    (< task0 task1)
    (< task1 task2)
    (< task2 task3)))
```

The performance of classical planning can be improved by leveraging domain control knowledge. Multiple approaches have been proposed to extract domain knowledge automatically (Fern, Yoon, and Givan 2004; Fuentetaja and Borrajo 2006; Yoon, Fern, and Givan 2008; Segovia Aguas, Jimenez Celorrio, and Jonsson 2016). Other approaches focus on learning hierarchical control knowledge, such as learning recursive control programs (Langley et al. 2006), teleoreactive logic programs (Nejati, Langley, and Konik 2006), or hierarchical task networks (HTN) – either directly from PDDL (Lotinac and Jonsson 2016), from learning examples of successful task decompositions (Li et al. 2024), or from extended information about decomposable tasks (Hogg 2011; Hogg, Muñoz-Avila, and Kuter 2016; Li et al. 2022). Some approaches extend HTN decomposition models by additional derived information to further increase planning performance (Olz 2024).

From Behaviour to Autonomy

When the formal model of agent behavior is given, the final question is initiating the planning process, that is, to prompt the agent to act. The classical planning problem is defined by the initial state, which can be obtained from the results of steps 1 and 2 of the proposed pipeline (i.e., objects and their relations in the world state), and by the goal condition. The goal condition and how to obtain and formulate it are the topics of this last step of the pipeline.

An owner of the agent can specify the goal condition, but a research problem is how to communicate it to the agent while assuming the learned model of behaviour. More complex goal conditions (not just a desired property of the world state) can be formulated using task networks in the context of HTN. Currently, there exist approaches that propose techniques for decomposing the given planning goal into subproblems, thereby improving the agent's planning efficiency in its environment (Núñez-Molina, Fernández-Olivares, and Pérez 2022; Drexler, Seipp, and Geffner 2022, 2023).

An even more interesting and challenging problem is when the agent itself defines its own goals, sometimes referred to as *agentic AI*. Automated planning employs a goal-based approach to agent design; however, a utility-based agent, as used, for example, in Markov Decision Processes and Reinforcement Learning, provides a more general concept of agentic AI. The question is whether and how to transform rewards and utility into goals, which are more efficient for planning. Finally, the open problem is determining the utility of the agent that allows the agent to behave desirably (and what that behavior should be).

Beyond the Linear Pipeline

In the previous section, we proposed a modular pipeline for obtaining formal planning models from raw data. We intentionally presented the steps separately, though in practice they influence each other, and it might be more efficient to combine them into higher-level blocks. We already mentioned several backward influences: later steps, such as learning an action model, influence earlier steps, such as the choice of predicates that model world-state properties.

Let us now discuss some proposed and existing pipelines that span several, and even all, steps of the proposed pipeline. A great example might be the work by Mokhtari, Lopes, and Pinho (2016). They create a learning system that allows users to verbally instruct a robot to perform complex tasks, such as serving a guest coffee. The robot has predetermined actions it knows how to perform, such as moving and picking up objects, and it learns an activity schema, which is a goal-directed task model (Mokhtari, Lopes, and Pinho 2016).

Fine-Morris et al. (2025) suggest a different type of pipeline, where input in natural language (description of a goal) and symbolic input (definition of actions in PDDL) are combined to generate HTN domain models. Another approach of Fine-Morris et al. (2022) represents another type of pipeline, where translation from a numeric representation to a purely logical representation is skipped, and hierarchical decompositions are learnt for domains with numeric effects and subgoals.

Smirnov et al. (2024) created a pipeline that uses LLMs to learn a domain and generate a plan. It takes as input a natural-language description of an initial state and a goal state. Then it uses LLMs to generate a more detailed problem description. Then it generates a domain model (note that this model is not in PDDL but in JSON with a PDDL-like structure, which is later transformed into PDDL). Then it performs a variety of reachability checks and consistency checks to improve the domain. Then they use a Fastdownward planner (Helmert 2006) to solve the problem. The main drawback is that the pipeline does not guarantee the semantic correctness of the plan (i.e., that all necessary actions are included). This has to be checked by a human before the plan can be regenerated.

Note, finally, that the pipeline is not meant to be a one-way process. Though most learning techniques use batch processing, where data to learn from is provided as input and a model is learnt from it, in practice, a more desirable approach is *never-ending learning* (Mitchell et al. 2015), where the agent continually improves the model using new data. Therefore, we would rather talk about *model correction* than model learning. The output of each step is actually available for that step, and the task is updating it based on new data. The pipeline is ready for that because, as we mentioned, each step might be influenced by future steps. All steps run in parallel, and data from previous and future steps feed them. For example, the step identifying predicates in world states may get a new state (or a new entity discovered in the state), a set of existing predicates, and a partial action model, and based on that information, it may generate new predicates (or may remove some predicates that are found irrelevant). Designing techniques that work incrementally would further increase the applicability of the proposed pipeline in the design of autonomous agents. There are already approaches for correction planning models (Bercher, Sreedharan, and Vallati 2025).

Conclusions

The knowledge acquisition gap is one of the most significant limitations of model-based AI techniques, as it restricts the technology's practical applicability. This gap is even more pronounced in automated planning, where little effort has been devoted to proper modeling techniques and to the knowledge engineering aspects of planning. Despite the growing number of techniques for learning action models, they remain limited in usability due to various assumptions and fail to cover all aspects of the knowledge acquisition process.

In the paper, we proposed a complete pipeline (Figure 4) that specifically addresses the problems appearing prior to learning action models, namely, obtaining the object-type hierarchy and learning relationships describing properties of world states that are important for planning, and after having an action model, namely, learning control structures to improve the efficiency of planning, and formulating goals for agents. In particular, these post-action-model stages are still in their infancy, with limited research results.

We intentionally used the STRIPS model as the core modeling approach to highlight the core components of the planning domain models and to explain the motivation for individual steps in the proposed pipeline. These steps are suggested as independent modules with a straightforward interface between them, and the pipeline is described as a simplified linear process. However, the practical implementation may require tighter integration and bidirectional communication, with later steps influencing decisions made earlier.

To justify the proposed steps, we conducted an extensive literature survey that demonstrates progress in each step. Obviously, the reference techniques were not designed with the proposed pipeline in mind, so they may not entirely fit into a specific step, as they typically span multiple steps or cover only part of a step. Nevertheless, we are not aware of any approach that covers the entire knowledge acquisition pipeline. We are not there yet, but the knowledge gap is narrowing, and we believe the proposed pipeline gives some structure to narrow it further.

Acknowledgments

This research is supported by the project 25-18003S of the Czech Science Foundation and by the Twist project number FY01010074.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Aineto García, D. 2022. *Inference and learning with planning models*. Ph.D. thesis, Universitat Politècnica de València.

Amir, E.; and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33: 349–402.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *The Knowledge Engineering Review*, 33: e20.

Arregui, S. F.; Celorrio, S. J.; and de la Rosa Turbides, T. 2010. Improving automated planning with machine learning. In *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques*, 599–620. IGI Global Scientific Publishing.

Asai, M.; Kajino, H.; Fukunaga, A.; and Muise, C. 2022. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74: 1599–1686.

Balyo, T.; Suda, M.; Chrpa, L.; Safránek, D.; Gocht, S.; Dvorák, F.; Barták, R.; and Youngblood, G. M. 2024. Planning Domain Model Acquisition from State Traces without Action Parameters. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*.

Barbin, A.; Cerutti, F.; and Gerevini, A. E. 2024. Learning reliable pddl models for classical planning from visual data. In *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, 722–728. IEEE.

Barták, R.; and Vodrázka, J. 2016. An Experimental Study of Influence of Modeling and Solving Techniques on Performance of a Tabled Logic Programming Planner. *Fundam. Informaticae*, 149(1-2): 35–60.

Bercher, P.; Sreedharan, S.; and Vallati, M. 2025. A survey on model repair in AI planning. In *34th International Joint Conference on Artificial Intelligence*, 26. IJCAI Organization.

Bhatnagar, S.; Mund, S.; Scala, E.; McCabe, K.; McCluskey, T. L.; and Vallati, M. 2022. On-the-Fly Knowledge Acquisition for Automated Planning Applications: Challenges and Lessons Learnt. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence, ICAART 2022*, volume 2, 387–397.

Callanan, E.; De Venezia, R.; Armstrong, V.; Paredes, A.; Chakraborti, T.; and Muise, C. 2022. MACQ: a holistic view of model acquisition techniques. In *2022 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Chrpa, L.; Barták, R.; Vodrážka, J.; and Vomlelová, M. 2022. Attributed transition-based domain control knowledge for domain-independent planning. *IEEE Transactions on Knowledge and Data Engineering*, 34(9): 4089–4101.

Chrpa, L.; McCluskey, T. L.; Vallati, M.; and Vaquero, T. 2017. The Fifth International Competition on Knowledge Engineering for Planning and Scheduling: Summary and Trends. *AI Magazine*, 38(1): 104–106.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013a. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013b. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 62–70.

Drexler, D.; Seipp, J.; and Geffner, H. 2023. Learning hierarchical policies by iteratively reducing the width of sketch rules. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 208–218.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Annals of Mathematics and AI*, 18(1): 69–93.

Fern, A.; Yoon, S. W.; and Givan, R. 2004. Learning Domain-Specific Control Knowledge from Random Walks. In *ICAPS*, 191–199.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *IJCAI 1971*, 608–620.

Fine-Morris, M.; Floyd, M.; Auslander, B.; Pennisi, G.; Gupta, K. M.; Roberts, M.; Heflin, J.; and Munoz-Avila, H. 2022. Learning Decomposition Methods with Numeric Subtasks. *Proc. of the Advances in Cognitive Systems. Fairfax, VA*.

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

Fuentetaja, R.; and Borrajo, D. 2006. Improving control-knowledge acquisition for planning by active learning. In *European Conference on Machine Learning*, 138–149. Springer.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; Wilkins, D.; Barrett, A.; Christianson, D.; et al. 1998. Pddl—the planning domain definition language. *Technical Report*.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2016. Learning hierarchical task models from input traces. *Computational Intelligence*, 32(1): 3–48.

Hogg, C. M. 2011. *Learning Hierarchical Task Networks From Traces and Semantically Annotated Tasks*. Ph.D. thesis, Lehigh University.

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 AAAI conference on artificial intelligence*, 9883–9891.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe learning of lifted action models. *arXiv preprint arXiv:2107.04169*.

Konidaris, G.; Kaelbling, L.; and Lozano-Perez, T. 2014. Constructing symbolic representations for high-level planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Krafczyk, A.; El-Sharkawy, S.; and Schmid, K. 2018. Reverse engineering code dependencies: converting integer-based variability to propositional logic. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 2*, 34–41.

Kramer, S.; and Bessiere, C. 2020. A brief history of learning symbolic higher-level representations from data (and a curious look forward). In *IJCAI*, 4868–4876.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; Traverso, P.; et al. 2021. Online Learning of Action Models for PDDL Planning. In *IJCAI*, 4112–4118.

Lamanna, L.; and Serafini, L. 2024. Action model learning from noisy traces: a probabilistic approach. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 342–350.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted action models learning from partial traces. *Artificial Intelligence*, 339: 104256.

Langley, P.; Choi, D.; Olsson, R.; and Schmid, U. 2006. Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, 7(3).

Li, R.; Nau, D.; Roberts, M.; and Fine-Morris, M. 2024. Automatically Learning HTN Methods from Landmarks. *The International FLAIRS Conference Proceedings*, 37(1).

Li, R.; Roberts, M.; Fine-Morris, M.; and Nau, D. 2022. Teaching an HTN Learner. *HPlan Workshop*.

Lin, S.; Grastien, A.; Shome, R.; and Bercher, P. 2025. Told You That Will Not Work: Optimal Corrections to Planning Domains Using Counter-Example Plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 26596–26604.

Liu, C.; and Haslum, P. 2025. Learning Action Models from Partially Ordered Action Traces. In *KEPS Workshop*.

Long, D.; and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

Lotinac, D.; and Jonsson, A. 2016. Constructing hierarchical task models using invariance analysis. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 1274–1282.

McCarthy, J.; and Hayes, P. 1969. Some Philosophical Problems From the Standpoint of Artificial Intelligence. In Meltzer, B.; and Michie, D., eds., *Machine Intelligence 4*, 463–502. Edinburgh University Press.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *Proceedings of the Knowledge Capture Conference, K-CAP 2017, Austin, TX, USA, December 4-6, 2017*, 14:1–14:8. ACM.

McDermott, D. M. 2000. The 1998 AI planning systems competition. *AI magazine*, 21(2): 35–35.

Minton, S.; and Carbonell, J. G. 1987. Strategies for Learning Search Control Rules: An Explanation-based Approach. In *Proceedings of IJCAI*, 228–235.

Mitchell, T.; Cohen, W.; Hruschka, E.; Talukdar, P.; Betteridge, J.; Carlson, A.; Dalvi Mishra, B.; Gardner, M.; Kisiel, B.; Krishnamurthy, J.; Lao, N.; Mazaitis, K.; Mohamed, T.; Nakashole, N.; Platanios, E.; Ritter, A.; Samadi, M.; Settles, B.; Wang, R.; Wijaya, D.; Gupta, A.; Chen, X.; Saparov, A.; Greaves, M.; and Welling, J. 2015. Never-Ending Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1).

Mokhtari, V.; Lopes, L. S.; and Pinho, A. J. 2016. Experience-based robot task learning and planning with goal inference. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, 509–517.

Mordoch, A.; Scala, E.; Stern, R.; and Juba, B. 2024. Safe Learning of PDDL Domains with Conditional Effects–Extended Version. *arXiv preprint arXiv:2403.15251*.

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *Proceedings of the 23rd international conference on Machine learning*, 665–672.

Nevens, J.; Van Eecke, P.; and Beuls, K. 2020. From continuous observations to symbolic concepts: A discrimination-based strategy for grounded concept learning. *Frontiers in Robotics and AI*, 7: 84.

Núñez-Molina, C.; Fernández-Olivares, J.; and Pérez, R. 2022. Learning to select goals in Automated Planning with Deep-Q Learning. *Expert Systems with Applications*, 202: 117265.

Olz, C. 2024. *Exploring the Hierarchy: Extracting and Exploiting State Information of Compound Tasks in HTN Planning*. Ph.D. thesis, Ulm University.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large Language Models as Planning Domain Generators. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1): 423–431.

Russell, S.; and Norvig, P. 2020. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson. ISBN 9780134610993.

Segovia Aguas, J.; Jimenez Celorrio, S.; and Jonsson, A. 2016. Generalized Planning with Procedural Domain Control Knowledge. *Proceedings of the International Conference on Automated Planning and Scheduling*, 26(1): 285–293.

Shah, M. M. S.; Chrpa, L.; Kitchin, D. E.; McCluskey, T. L.; and Vallati, M. 2013. Exploring Knowledge Engineering Strategies in Designing and Modelling a Road Traffic Accident Management Domain. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2373–2379. IJCAI/AAAI.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowl. Eng. Rev.*, 22(2): 117–134.

Sleath, K.; and Bercher, P. 2023. Detecting AI Planning Modelling Mistakes – Potential Errors and Benchmark Domains. In *Proceedings of the 20th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2023)*, 448–454. Springer.

Smirnov, P.; Joublin, F.; Ceravola, A.; and Gienger, M. 2024. Generating consistent pddl domains with large language models. *arXiv preprint arXiv:2404.07751*.

Stern, R.; Lamanna, L.; Mordoch, A.; Benyamin, Y.; Lauer, P.; Juba, B.; Behnke, G.; Muise, C.; Bercher, P.; Vallati, M.; Xi, K.; Wattad, O.; and Eliyahu, O. 2025. Evaluating Planning Model Learning Algorithms. In *Proceedings of the 14th Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2025)*.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *Ai Magazine*, 36(3): 90–98.

Valmeekam, K.; Stechly, K.; and Kambhampati, S. 2024. LLMs Still Can't Plan; Can LRMs? A Preliminary Evaluation of OpenAI's o1 on PlanBench. arXiv:2409.13373.

Vaquero, T. S.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2013. itSIMPLE: towards an integrated design system for real planning applications. *Knowl. Eng. Rev.*, 28(2): 215–230.

Xi, K.; Gould, S.; and Thiébaux, S. 2024. Neuro-symbolic learning of lifted action models from visual traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 653–662.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.

Yang, Y.; Wu, J.; and Yue, Y. 2025. Robust Hypothesis Generation: LLM-Automated Language Bias for Inductive Logic Programming. *arXiv preprint arXiv:2505.21486*.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning Control Knowledge for Forward Search Planning. *Journal of Machine Learning Research*, 9(4).