Learning Sound and Complete Preconditions in Complex Real-World Domains

René Heesch[1,2,*], Björn Ludwig[1,2], Jonas Ehrhardt[1,2], Alexander Diedrich[1,2] and Oliver Niggemann[1,2]

[1]HSU-AI Institute for Artificial Intelligence, Helmut-Schmidt-University, Hamburg, Germany
[2]Institute of Automation, Helmut-Schmidt-University, Hamburg, Germany

Abstract

In this paper, we address the problem of learning sound and complete action preconditions in complex real-world planning domains, the remaining bottleneck in the N3PCP pipeline. Such planning domains involve hybrid state spaces including discrete and numerical variables. We propose a dependency-aware learning approach that captures interdependencies between discrete and numerical variables by constructing distinct convex hulls over the numerical subspace for each discrete state configuration. This poses a more accurate representation of hybrid preconditions in real-world domains than existing approaches for learning preconditions. We empirically compare our method against two other approaches: an exact baseline method that ensures soundness but lacks completeness, and a generalized variant of N-SAM, that achieves completeness but compromises soundness. We evaluate our approach across multiple planning problems and domains which are based on a real-world industrial system, demonstrating the practical benefits of our approach. An additional theoretical analysis confirms that, under standard convexity assumptions and sufficient coverage of discrete configurations within the training data, our proposed dependency-aware method guarantees both completeness and soundness.

Keywords

Planning, N3PCP, SMT, Neural Networks, Action Model Learning

1. Introduction

Automated planning is a key capability for the autonomy of intelligent systems [1]. Planning enables system to generate plans, which are action-sequences that transition a system from an initial state to a goal state [1]. In many real-world domains, this involves reasoning over hybrid state spaces that combine symbolic (discrete) and continuous (numerical) components, e.g., in robotics, autonomous systems, and industrial manufacturing [2, 3, 4]. In such settings, the complexity of planning lies not exclusively in the branching factor of the search space, but also in the need to determine valid control parameters for actions and the difficulty of modeling system behavior mathematically [5].

Traditional planning approaches depend on manually specified domain models, which define action preconditions, effects, and causal relations. However, constructing such models in complex, dynamic, and heterogeneous domains is time-consuming and error-prone [6]. This modeling bottleneck significantly limits the scalability and adaptability of planning systems in complex real-world domains.

To address this bottleneck, *Neural Network–enriched Numerical Planning with Control Parameters* (N3PCP) was recently introduced as a hybrid planning framework that leverages neural networks to approximate the effects of actions based on empirical data [7, 2]. By learning effect models from observations or simulations, N3PCP eliminates the need for explicit, hand-crafted transition models. While early approaches employed eager and exact evaluation strategies [7], later work adopted lazy evaluation methods that trade off completeness for generality and computational efficiency [4]. However, both existing methods still rely on manually defined action preconditions, limiting their generality and automation.

In this paper, we address this remaining bottleneck in the N3PCP pipeline by proposing an approach for learning action preconditions from data. We focus on domains where only observational data are

CAIPI'25: ECAI Workshop on AI Planning in Complex Real-World Applications, October, 2025, Bologna, Italy
*Corresponding author.

available, i.e., states to which actions have been successfully applied, without access to examples of invalid plans. This leads to our first research question:

RQ1: Is it possible to learn action preconditions in real-world domains exclusively from observations of the states in which an action was applied?

To motivate this question, we consider a concrete and challenging application: production planning in modular manufacturing systems (cf. Figure 1). In particular, we focus on a modular real-world system that refines automotive glass components via polyurethane foaming. Modular manufacturing systems require the coordinated execution of production steps, each guided by machine-level control parameters, to reliably transform raw materials into finished products [8, 7]. The hybrid nature of the task, by combining symbolic reasoning and numerical parameter identification, makes it a typical real-world use case for automated planning frameworks like N3PCP [7]. Moreover, rapid reconfiguration is essential to cope with increasing product variability and reduced batch sizes [9, 10].

In such hybrid domains, two properties are essential: soundness, which ensures that any generated plan is executable, and completeness, which ensures that all valid plans can be found [11]. These properties depend not only on the planner but also on the quality of the domain model, i.e., the learned action preconditions. This raises our second research question:

RQ2: How does precondition learning affect the soundness and completeness of planning algorithms in N3PCP domains?

To address these questions, we propose a novel approach to learn dependency-aware preconditions from observational data. We compare this approach with two other approaches to precondition learning. An exact, memorization baseline, assuring soundness and a variation of the N-SAM [12], aiming for completeness. We assess the methods two-fold: Empirically, using data from an industrial production system, and theoretically, via formal analysis of their soundness and completeness.

2. Related Work

Automated planning relies on formal domain descriptions. These description include action models, which specify the conditions that must be satisfied in a state to make an action applicable to this state, i.e., the preconditions, and how this action is transforming the state, i.e., the effects. As manually creating such descriptions is tedious and error-prone [6], multiple approaches proposed learning action models based on observations from plan executions [13, 12, 14, 15, 16, 6]. Yet, most existing approaches are restricted to propositional representations and only limited work focus on learning action models in numerical domains.

PlanMiner [16, 15] infers action models with numeric preconditions and effects from noisy or partial plan traces by employing a combination of preprocessing, regression, and classification techniques to learn logical and arithmetic relations among state variables. However, the approach lacks formal guarantees on soundness: a plan found using the learned model may not be valid in the real world.

Addressing this limitation, N-SAM [12] extends the propositional SAM framework to numeric domains by learning *safe* numeric action models, adopting a two-stage learning process. The *safe* action models guarantee that generated plans execute as expected, with a total deviation in numeric effects bounded by at most a threshold $\epsilon \in \mathbb{R}$. The algorithm first infers propositional preconditions using techniques for classical planning problems [17]. In detail, to ensure soundness, the propositional preconditions are conservatively defined, i.e., as the intersection of all observed applicable states, which guarantees that no invalid plans are produced. Then it defines numeric preconditions by constructing a convex hull over the numeric values observed in all states where the action was applied and expressing this hull as a set of linear inequalities. The convex hull defines a conservative approximation of the region in which the action is known to be applicable. However, learning a convex hull for an action with d numeric parameters requires at least $d + 1$ independent samples. Furthermore, the algorithm requires information about the set of numeric state variables involved in each action's preconditions and effects.

N-SAM* [14] enhance the N-SAM algorithm by reducing the sample requirements to one observation

of every action and the requirements of prior knowledge on the involved state variables, while still maintaining their previous introduced safety guarantee. The authors build a linear combination of using the Gram-Schmidt process and thereby enabling to learn preconditions from less than $d + 1$ samples for an action with d numerical parameters.

Both neural networks and control parameters affect only action effects, not preconditions. Hence, although N-SAM [12] and N-SAM* [14] were designed for numeric planning domains, their precondition-learning methods could, in principle, be adapted to N3PCP problems.

However, these approaches learn models expressed in the Planning Domain Definition Language (PDDL), which has been shown to lack the expressiveness required for complex real-world applications such as production planning [18]. Moreover, their methodology treats propositional and numerical variables independently during precondition learning. As a result, they cannot capture cross-dependencies between these variable types, e.g., for cases where the feasible numeric state space is further constrained by specific propositional configurations. Additionally, the conservative strategy for learning the propositional preconditions leads to a loss of completeness, potentially excluding valid behaviors that were simply not observed in all instances.

3. Background

All planning approaches to Neural Network-enriched Numerical Planning with Control Parameters (N3PCP) are grounded in the planning as satisfiability paradigm, specifically by using Satisfiability Modulo Theories (SMT) as the underlying computational framework.

3.1. Logical Preliminaries

We operate within the setting of many-sorted first-order logic. Terms are either constants, individual variables, or applications of n-ary function symbols to n terms. Atoms are either propositional variables or n-ary predicate symbols applied to n terms. Formulae are constructed from atoms using standard Boolean connectives ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$) and quantifiers ($\forall, \exists$) applied to one or more variables and a subformula. We follow the standard semantic terminology of interpretation, model, satisfiability, and validity [19].

In SMT, the interpretation of a given set of symbols is constrained by a given background theory, e.g., the theory of nonlinear arithmetic with transcendental functions (NTA). To solve N3PCP problems, numerical constants, real-valued variables, standard function symbols $+, -, \times, \div$, comparison predicates $<, \leq, >, \geq$, and transcendental functions such as tan are considered [4].

For planning, we employ a symbolic representation of infinite-state transition systems [20]. A transition system is defined as the tuple $\langle V, I(V), T(V, V') \rangle$, where:

- V is a set of state variables;
- $I(V)$ is an SMT formula characterizing the set of initial states (i.e., a state s is initial if $s \models I$);
- $T(V, V')$ is the transition relation, with V denoting the state before and V' denoting the state after the transition.

A transition from state s to s' exists if the combined valuation over $V \cup V$' that assigns V according to s and V' according to s' satisfies T, i.e., $(s, s') \models T$. In this case, s' is a successor of s, and s a predecessor of s'. In a transition system Γ, a trace is a sequence of states $s_0, s_1, \ldots$ such hat $s_0 \models I$ and for all i, s_{i+1} is a successor of s_i. To express properties over such traces, we use Linear Temporal Logic [21]. The model checking problem $\Gamma \models \phi$ states, that all traces of Γ satisfy a temporal formula ϕ.

Among various verification techniques, we focus on *Bounded Model Checking* (BMC) [22], which reduces the reachability problem to satisfiability over a bounded horizon. BMC proceeds by unrolling the transition relation for k steps. The following SMT formula is constructed:

$$I(V_0) \wedge T(V_0, V_1) \wedge \ldots \wedge T(V_{k-1}, V_k) \wedge G(V_k), \tag{1}$$

where $G(V_k)$ specifies the goal condition. The formula is satisfiable if there exists a trace of k transitions from the initial state to a state satisfying G. A satisfying assignment to the variables $V_0, V_1, \ldots, V_k$ represents the corresponding plan.

3.2. Real-World State Spaces

In AI planning with propositional and numeric variables, the state space factors as the Cartesian product of a propositional valuation space and a numeric assignment space [23]. Let B be the set of propositional variables and N the set of numeric variables. Define

$$\mathrm{Val}(B) \;:=\; \{\alpha \mid \alpha : B \to \{\bot, \top\}\} \qquad \text{and} \qquad \Omega \;:=\; \prod_{i \in N} I_i \;\subseteq\; \mathbb{R}^N, \tag{2}$$

where each $I_i \subseteq \mathbb{R}$ is typically a closed interval. A state is a pair (α, u) with $\alpha \in \mathrm{Val}(B)$ and $u \in \Omega$, and the state space is

$$S \;=\; \mathrm{Val}(B) \times \Omega. \tag{3}$$

If the real-world state space also includes finite discrete variables $D = \{D_j\}_{j \in J}$ with domains D_j, these are encoded into the propositional part via a standard injective encoding, obtaining an expanded set B' and a bijection $\prod_{j \in J} D_j \;\times\; \mathrm{Val}(B) \;\cong\; \mathrm{Val}(B')$. Hence, w.l.o.g., we use Eq. (3) and refer only to the propositional part.

Definition 1. *For $u, v \in \mathbb{R}^N$, the line segment $\overline{uv}$ is $\overline{uv} \;:=\; \{(1-t)u + tv \mid t \in [0,1]\}$. For states $(\alpha, u), (\alpha, v) \in S$ with the* same *propositional valuation α, the corresponding segment in S is $\{(\alpha, w) \mid w \in \overline{uv}\}$. If the propositional parts differ, no convex combination is defined in S.*

Definition 2. *Let $S \subseteq \mathrm{Val}(B) \times \mathbb{R}^N$. For $\alpha \in \mathrm{Val}(B)$, define the numeric fiber $F(\alpha) \;:=\; \{u \in \mathbb{R}^N \mid (\alpha, u) \in S\}$. We say that S is* fiberwise convex *if each $F(\alpha)$ is convex in $\mathbb{R}^N$.*

In many applications, numeric variables are bounded physical quantities, i.e., $u \in \Omega = \prod_{i \in N} I_i$ with intervals $I_i = [\ell_i, u_i]$. Each I_i is convex, and thus Ω is an axis-aligned box and convex. Accordingly, we assume throughout that for every fixed propositional assignment α, the feasible numeric fiber $F(\alpha)$ is convex. This guarantees that for any two feasible numeric states under the same α, their linear interpolations remain feasible.

4. Problem Definition

Following the formalization of [4] the Neural Network-enriched Numerical Planning with Control Parameters (N3PCP) problem is defined as a tuple:

$$P = \langle B, N, A, \Theta, T, M, I, G \rangle, \tag{4}$$

where each component contributes to describing hybrid planning tasks that involve both symbolic and numerical reasoning under parametric control. This formalism is particularly motivated by real-world applications such as production planning in modular manufacturing systems [8, 7], where tasks involve discrete process steps and machine-level parameter tuning.

Let B denote the set of propositional (Boolean) state variables and N the set of numerical state variables. Each state $s_i \in S := B \times N, i \in \mathbb{N}$ is characterized by a valuation of the propositional variables in B and real-valued (or bounded-precision) variables in N. We assume a fixed ordering of s_i.

A denotes the finite set of actions, and Θ the set of control parameters associated with these actions. Each action models a transition of the system from one state to another, influenced by a specific valuation of the parameters Θ. In the context of production systems, actions represent different production steps such as cutting, assembling, or coating, while the control parameters correspond to machine-level settings like feed rate, temperature, or pressure [7, 2].

Each action $a \in A$ is specified in one of two ways:

- A symbolic transition function $T_a \in T$, represented as an SMT formula $T_a(s_i, \Theta, s_{i+1})$ over the current state s_i, the control parameters Θ, and the successor state s_{i+1}.
- An executable transition function $M_a \in M$, which is a loop-free imperative program that computes a successor state s_{i+1} given (s_i, Θ). In practice, this may be implemented by a Neural Network, trained to approximate physical or empirical system dynamics or a simulation of a particular production step.

The symbolic representation allows for logical reasoning, constraint satisfaction, and artificial data generation. The executable form enables forward simulation, including black-box models where analytical descriptions are infeasible.

The control parameters Θ are subject to domain constraints. Let $ValidCP_a(\Theta)$ denote the admissible space of control parameter valuations for action a. These constraints may be, for instance, interval bounds:

$$lb \leq \theta \leq ub, \quad \forall \theta \in \Theta, \tag{5}$$

encoding physical or safety limitations, e.g., torque limits and temperature ranges in the context of production systems, where lb describes the lower bound and ub the upper bound, $lb, ub \in \mathbb{R}$.

Additionally, not all actions are applicable in all states. We define the applicability set of an action $a \in A$ as:

$$S_{in}{}^{(a)} = \{s \in S \mid \phi_a(s) = \text{true}\}, \tag{6}$$

where $\phi_a : S \to \{\text{true}, \text{false}\}$ is a precondition predicate that determines whether the action a can be applied in state s. These predicates may be manually specified or learned from empirical execution data, such as successful and failed application attempts of a across different states.

Moreover, we assume that the symbolic and executable transition functions T_a and M_a are defined and valid only on their respective domains of applicability:

$$\text{dom}(T_a), \text{dom}(M_a) \subseteq S_{in}{}^{(a)} \times ValidCP_a. \tag{7}$$

This ensures that logical inference and simulation are only performed under feasible conditions.

I denotes the set of initial state conditions, encoded as an SMT formula over B and N, describing feasible system configurations at the outset of planning. The goal condition G is similarly an SMT formula over B and N, specifying the desired properties of the final state. Unlike classical planning with discrete goals, N3PCP problems allow goal descriptions and preconditions of actions over continuous state variables. These are not expressed as equalities but as a set of inequalities, where the numerical values have to lay within a certain range, reflecting the properties of real-world domains, e.g., target dimensions, quality metrics.

5. Solution

In this section, we introduce our novel solution to learn dependency-aware preconditions solely from observations of the planning domain. We derive our dependency-aware approach from two baseline approaches, one emphasizing exclusively on soundness of the learned preconditions, and the other emphasizing on completeness of the learned preconditions. While the first baseline is the most restrictive and conservative one with the highest possible incompleteness, the second baseline is the most optimistic approach, aiming for completeness at the cost of soundness. Our proposed dependency-aware approach is based on reasonable assumptions both, sound and complete.

For all approaches, a set τ^a of observed applicable states $\{s_1, \ldots, s_j\}$ for an action a is given. Following the vector-based state-space representation presented in Heesch et al. [7], we represent the assignments to all variables in B and N, i.e., the states, in the form of vectors.

5.1. Soundness First: Learning Exact Preconditions

Our first baseline is sound with respect to the training data. Given the observed applicable states $\tau^a = \{s_1, \dots, s_j\} \subseteq S_a$, we define the *exact* precondition as a pure enumeration of those states:

$$\phi_a^{\text{exact}} = \bigvee_{i=1}^{j} s_i \tag{8}$$

This DNF exactly accepts the training examples and rejects all others. The corresponding algorithm is given in Alg. 1.

Algorithm 1 Learn Exact Preconditions

Require: Dataset $\tau^a = \{s_1, s_2, \dots, s_j\}$ of states $s \in S$ where action a was applied
Ensure: Symbolic precondition ϕ_a^{exact}

1: Initialize $\phi_a^{\text{exact}} \leftarrow \emptyset$
2: **for all** $s_i \in \tau^a$ **do**
3: Construct clause $\phi_i \leftarrow s_i$
4: $\phi_a^{\text{exact}} \leftarrow \phi_a^{\text{exact}} \vee \phi_i$
5: **end for**
6: **return** ϕ_a^{exact}

The algorithm first initializes an empty clause ϕ_a^{exact} (cf. Alg. 1 l. 1). Subsequently, the algorithm iterates through the whole dataset τ^a. For each state s_i a clause ϕ_i is constructed (cf. Alg. 1 l. 2). The clause ϕ_i is a conjunction, assigning each variable in B and N the corresponding value of s_i (cf. Alg. 1 l. 3). This clause is then added to the clause ϕ_a^{exact} via a disjunction (cf. Alg. 1 l. 4). At the end, the clause ϕ_a^{exact} is returned (cf. Alg. 1 l. 6).

As Alg. 1 may be reasonable in propositional domains, it becomes problematic in continuous or hybrid domains. In such domains, this method is unlikely to provide a complete representation, as it only enumerates the observed states from the data set. It cannot cover the infinite space of possible continuous values. As a result, ϕ_a^{exact} is sound, but its completeness due to possible unseen but valid states is limited to the dataset and propositional domains.

5.2. Completeness First: Generalized Preconditions

Our second baseline approximates the preconditions by generalizing from observed data emphasizing rather on completeness than soundness. This involves inferring broader parts of the state space from sampled data,

e.g., intervals or convex regions, over the numerical variables N. This relaxation may come at the expense of soundness, when the space of applicable states is not convex, e.g., some states which satisfy ϕ_a^{general} may not allow action a in practice.

Let $n_k \in N$ be a continuous state variable. If the action a was applied in states s_0 and s_1 where all other variables are equal and only x_k differs, and if a was successfully executed in both cases, we assume a is applicable to any state s satisfying:

$$x_k \in \overline{x_k^{s_0} x_k^{s_1}} \tag{9}$$

where x_k denotes an assignment to n_k in state s.

Building on this idea and inspired by Mordoch et al. [12], we separate propositional and numerical variables when learning preconditions. For the numerical variables, we calculate the convex hull of the numerical components of the states in the data set τ^a of successful executions of a. The linear inequalities of the hull's supporting hyperplanes then define the numerical precondition. Unlike Mordoch et al. [12], our method does not require prior manual identification of action-relevant numeric variables. For propositional variables, we use a straightforward disjunctive strategy to increase completeness: we

collect the distinct propositional assignments under which a was observed, and take their disjunction as the propositional precondition.

Algorithm 2 Learn Generalized Preconditions

Require: Dataset $\tau^a = \{s_1, s_2, \dots, s_j\}$ of states where action a was applied; index sets $\mathcal{ID}_B$ for propositional variables, $\mathcal{ID}_N$ for numerical variables

Ensure: Symbolic precondition ϕ_a^{general}

1: Initialize $\Phi_B \leftarrow \emptyset$
2: Initialize matrix $V \leftarrow [\,]$
3: **for all** $s \in \tau^a$ **do**
4: Extract propositional subvector $s_B \leftarrow s[\mathcal{ID}_B]$
5: **if** $s_B \notin \Phi_B$ **then**
6: $\Phi_B \leftarrow \Phi_B \cup \{s_B\}$
7: **end if**
8: Append $s[\mathcal{ID}_N]$ to V
9: **end for**
10: Construct $\phi_a^{(B)} \leftarrow \bigvee_{s_B \in \Phi_B} s_B$
11: Compute convex hull $H \leftarrow \texttt{ConvexHull}(V)$
12: Derive set of linear inequalities $\phi_a^{(N)}$ from hull faces
13: $\phi_a^{\text{general}} \leftarrow \phi_a^{(B)} \wedge \phi_a^{(N)}$
14: **return** ϕ_a^{general}

The algorithm begins by initializing two data structures: an empty set Φ_B to collect all unique propositional configurations observed in the dataset τ^a (cf. Alg. 2, l. 1), and an empty matrix V to store the numerical components of the corresponding states (cf. Alg. 2, l. 2).

For each state $s \in \tau^a$, the propositional subvector s_B is extracted (cf. Alg. 2, l. 4). If this configuration has not yet been encountered, it is added to Φ_B (cf. Alg. 2, l. 5–7). Simultaneously, the numerical subvector is appended to the matrix V for subsequent processing (cf. Alg. 2, l. 8).

Once all states have been processed, the propositional precondition $\phi_a^{(B)}$ is constructed as a disjunction over the elements of Φ_B (cf. Alg. 2, l. 10):

$$\phi_a^{(B)} = \bigvee_{s_B \in \Phi_B} s_B, \tag{10}$$

ensuring that the resulting precondition evaluates to true precisely for the propositional configurations under which the action has been previously observed. This avoids the overly conservative strategy of assuming propositional variables must be fixed across all samples, as in [12].

Subsequently, a convex hull H is computed over the matrix V containing the numerical state components (cf. Alg. 2, l. 11). The set of defining linear inequalities for H is then extracted and used to define the numerical precondition $\phi_a^{(N)}$ (cf. Alg. 2, l. 12). The final action precondition ϕ_a^{general} is returned as the conjunction of its propositional and numerical components (cf. Alg. 2, l. 13-14).

Using a convex hull to describe the numerical component reduces incompleteness by generalizing over observed numeric values, while maintaining soundness under the assumption that preconditions are independent of unobserved propositional contexts. However, this assumption introduces a potential loss of soundness when the propositional and numerical components are *not* independent. Specifically, by applying a single convex hull over all numerical data points, not accounting for the associated propositional configurations, the approach cannot capture cross-dependencies, such as when certain propositional conditions constrain the feasible numeric subspace. Consequently, the combined precondition $\phi_a^{\text{general}} = \phi_a^{(B)} \wedge \phi_a^{(N)}$ may admit states that were never observed together in the data, potentially leading to plans with action that could not be applied.

5.3. Dependency-Aware Preconditions

To address the drawbacks our two baselines, we introduce our novel *dependency-aware precondition* formulation. Instead of treating propositional and continuous constraints independently, we model the precondition as a conditional structure. Let s_B and s_N denote the propositional and numeric subvectors of the state s, respectively. Let $\Phi_B \subseteq \text{dom}(s_B)$ be the set of observed propositional configurations (from states where action a was applied). For each $s_B \in \Phi_B$, we fit a convex region $\phi_{s_B}^{(N)}$ over the numeric components observed under configuration s_B.

Formally, the dependency-aware precondition for action a is defined as:

$$\phi_a^{\text{depend}} = \bigvee_{s_B \in \Phi_B} \Big(\underbrace{s_B}_{\text{propositional}} \wedge \underbrace{\phi_{s_B}^{(N)}}_{\text{numeric}} \Big). \tag{11}$$

The numeric condition $\phi_{s_B}^{(N)}$ is defined by the set of linear inequalities (i.e., halfspaces) that describe the convex hull over all numeric vectors observed in states with the propositional configuration s_B.

Algorithm 3 Dependency-Aware Preconditions

Require: Dataset τ^a of states s where action a was applied; propositional indices $\mathcal{ID}_B$, numeric indices $\mathcal{ID}_N$

Ensure: Symbolic precondition ϕ_a^{depend}

1: Initialize $\phi_a^{\text{depend}} \leftarrow \emptyset$
2: Initialize map $groups : s_B \mapsto$ List of numeric vectors
3: **for all** $s \in \tau^a$ **do**
4: $\quad s_B \leftarrow s[\mathcal{ID}_B]$
5: $\quad n \leftarrow s[\mathcal{ID}_N]$
6: $\quad$ Append n to $groups[s_B]$
7: **end for**
8: Initialize list $clauses \leftarrow [\,]$
9: **for all** $s_B \in \text{keys}(groups)$ **do**
10: $\quad V \leftarrow groups[s_B]$
11: $\quad H \leftarrow \texttt{ConvexHull}(V)$
12: $\quad \phi_{s_B}^{(N)} \leftarrow \texttt{GetHalfspaceInequalities}(H)$
13: $\quad \phi_{s_B}^{(B)} \leftarrow s_B$
14: $\quad$ Append $(\phi_{s_B}^{(B)} \wedge \phi_{s_B}^{(N)})$ to $clauses$
15: **end for**
16: **return** $\phi_a^{\text{depend}} \leftarrow \bigvee_{c \in clauses} c$

The algorithm begins by partitioning the dataset τ^a into groups of states that exhibit identical propositional configurations (cf. Alg. 3, l.2–7). For each such group, a convex hull is computed over the continuous components of the state vectors (cf. Alg.3, l.10-11), and the corresponding set of linear inequalities $\phi_{s_B}^{(N)}$ is derived (cf. Alg.3, l. 12). These inequalities define the feasible numeric region associated with propositional configuration s_B.

The propositional part $\phi_{s_B}^{(B)}$ is constructed as a conjunction of variable assignments that match configuration b (cf. Alg. 3, l.13). The two components are then conjoined into a single clause that is added, and added to ϕ_a^{depend} as a disjunction (cf. Alg.3, l. 14-16).

By explicitly associating each propositional configuration with a corresponding region in the numeric state space, this approach avoids the need for conservative treatment of propositional variables of other approaches [17, 12]. As a result, it mitigates the incompleteness typically introduced by requiring propositional preconditions to hold uniformly across all observed states. Simultaneously, under the assumption that the numeric values observed under each configuration form a convex region, the

approach preserves soundness, ensuring that plans generated using the learned model remain executable in the real domain.

We recognize several limitations of this method. Most notably, constructing convex hulls requires a sufficient number of observed state samples for each propositional configuration in which the action was applied. Furthermore, the complexity of the resulting preconditions can increase the computational complexity, both during model learning and at planning time, where more detailed applicability conditions must be evaluated.

Nonetheless, in domains where complexity arises from the of actions rather than the number of objects, e.g., production planning [8], capturing these dependencies is critical for deriving both sound and complete action models. Importantly, we assume that precondition learning is performed entirely offline, prior to planning. This design choice alleviates concerns about runtime overhead, making it feasible to deploy expressive, dependency-aware preconditions in practical planning systems without compromising online efficiency.

6. Evaluation

Our evaluation is twofold. To address the first research question, we analyze the performance of our proposed approache empirically. To address the second research question, we perform a theoretical analysis of the presented methods in terms of soundness and completeness.

6.1. Empirical Evaluation

To investigate whether data-driven methods can effectively infer action preconditions based solely on observations of states in which the action was executed (RQ1), we focus on a representative action from an industrial production system. Thereby, we do not only focus on learning preconditions, but also the subsequent planning task, to account for the increasing complexity of the planning problem by learning complex precondition structures as in the generalized and the dependency-aware approach.

As evaluation metric, we chose the algorithmic runtime. Specifically, we included both, the learning of the action preconditions and the planning with these learned preconditions in the recording of runtime. Soundness and completeness are proven theoretically in Section 6.2.

6.1.1. Experimental Setup

We evaluated the algorithms on a real-world production system (cf. Figure 1). The production system refines automotive glass components using polyurethane foaming. In the foaming cell, a robot positions mechanical inserts onto a pretreated glass pane fixed in a foaming mold. This insert placement is a critical step before the polyurethane injection and must satisfy strict constraints related to temperature and configuration to ensure both product integrity and functional reliability.

Our evaluation centers on the robot's action model. An expert-defined model exists for each phase of the process, including an action for placing the inserts. Our objective is to infer the preconditions of this specific action exclusively from 1000 observations of states where the action was applied. These observations were generated by randomly sampling state transitions from the expert model. In the minimal complexity level, the states are described by fifteen state variables: thirteen propositional and two numerical variables[1]. To assess the scalability, we introduced up to four additional numerical state variables, incrementally adding additional preconditions to the planning domain, resulting in five increasingly complex planning problems and domains. Following the experimental design guidelines for Machine Learning research by [24], we generated the data for each level of domain complexity with eight different seeds. For avoiding random effects, we repeated each experiment ten times. The source

[1]We removed the control parameters and the neural network which is usually representing the actions' effect in N3PCP problems to reduce the complexity of the planning problems. Including these components would significantly increase computational complexity, and thereby obscure a meaningful comparison of the approaches to learn preconditions.

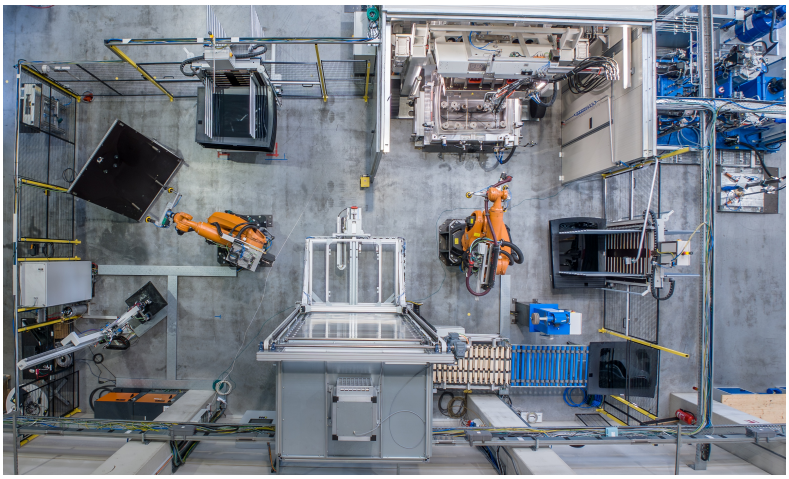

Figure 1: We perform our empirical evaluation on a planning domain of an automated production system for the refinement of automotive glass components using polyurethane foaming.

code and the action-model modeled by experts are available at https://github.com/RHeesch/Learning_Hybrid_Preconditions.

We ran our experiments on a system with an AMD Ryzen 9 9950X 16-core processor and 96 GB RAM. The planning problems, including the learned preconditions, were encoded in the Unified Planning Framework (UPF)[25] and solved using the LPG planner [26]. We evaluated the resulting plans, which were generated based on the learned preconditions, against the expert-defined domain model using the UPF and the TAMER planner [27].

6.1.2. Results

Table 1 summarizes the runtimes for learning action preconditions. We report mean and standard deviation of the runtime for each algorithm across the increasing domain complexities. Regarding precondition learning (cf. Table 1), all approaches exhibited similar runtimes in the domain with two numerical state variables. The *exact* method showed only modest increases as domain complexity grew, whereas the *generalized* and *dependency-aware* variants scaled exponentially, likely due to the overhead of constructing convex hulls and deriving the corresponding systems of linear inequalities.

Table 1
Mean runtime and standard deviation in seconds for learning action preconditions of the proposed algorithms across increasingly complex planning domains. Format: Runtime [s] ± Standard Deviation [s]

Prop. Variables	Num. Variables	Exact	Generalized	Dep-aware
13	2	0.023 ± 0.0006	0.023 ± 0.0004	0.023 ± 0.0004
13	3	0.023 ± 0.0003	0.024 ± 0.0003	0.025 ± 0.0003
13	4	0.024 ± 0.0002	0.027 ± 0.0004	0.032 ± 0.0003
13	5	0.025 ± 0.0003	0.052 ± 0.0012	0.064 ± 0.0008
13	6	0.026 ± 0.0003	0.257 ± 0.0084	0.203 ± 0.0049

Table 2 reports the mean planning times and standard deviation. All plans are valid with respect to expert-modeled domain description. Planning runtimes increased exponentially with domain complexity across all approaches. For the 15-variable domain, planning was initially much faster using the *generalized* and *dependency-aware* preconditions compared to the *exact* preconditions. However, as complexity increased, planning times for both variants soon exceeded those of the *exact* method, with the *generalized* approach exhibiting the highest planning times in the most complex domain. The *dependency-aware* variant initially performed less favorably than the *generalized* one, but at last surpassed it in the two most complex domains.

Table 2
Mean runtime and standard deviation in seconds for solving planning problems using the proposed algorithms for learning preconditions across increasingly complex planning domains. Format: Runtime [s] ± Standard Deviation [s].

Prop. Variables	Num. Variables	Exact	Generalized	Dep-aware
13	2	0.29 ± 0.0380	0.03 ± 0.0014	0.04 ± 0.0019
13	3	0.55 ± 0.3705	0.07 ± 0.0044	0.28 ± 0.1768
13	4	0.81 ± 1.0654	0.97 ± 1.2460	2.33 ± 1.5783
13	5	0.96 ± 1.2912	19.06 ± 3.0970	12.30 ± 3.1482
13	6	1.74 ± 5.0838	535.38 ± 44.7695	48.19 ± 7.0255

6.2. Theoretical Evaluation

For evaluating the soundness and completeness of the different approaches, we consider a hybrid state space $S = B \times N$, where B denotes the propositional component and N the continuous (numeric) component. For any action a, we define its true applicability region as $S_a = \{s \in S \mid a \text{ is truly applicable in } s\}$. We assume access to a finite training set $\tau^a = \{s_1, \dots, s_j\} \subseteq S_a$, representing observed states where action a was applicable.

The learned preconditions are sound, if the planning algorithm can find a plan that is executable in the real world, based on learned preconditions. The learned preconditions are complete, if the planning algorithm can find every single plan, that is an executable plan in the real world, based on the learned preconditions [11].

Exact Preconditions:

Theorem 1. *The exact preconditions are sound, i.e., for all $s \in S$, with ϕ_a^{exact} holds $s \in S_a$.*

Proof 1. *Let S_a be the complete set of applicable states for action a. For the training set $\tau^a \subseteq S_a$ holds and $\forall s \in \phi_a^{\text{exact}} : s \in \tau^a$. Thus, $\forall s \in \phi_a^{\text{exact}} : s \in S_a$.*

Since the exact preconditions only allow the action to be applied to states to which the action was previously applied, the learned exact preconditions are sound by construction.

Theorem 2. *The exact preconditions are complete with respect to the training set τ^a.*

Proof 2. *Proof by contradiction: We assume $s^* \in \tau^a$ exists with a is applicable for s^*, where $s^* \notin \phi_a^{\text{exact}}$, but by definition it holds $\forall s \in \tau^a : s \in \phi_a^{\text{exact}}$.*

Exact preconditions are complete only if the training set τ^a fully enumerates S_a. However, in general, when S_a is continuous or infinite, this condition is not met, making exact preconditions incomplete in practical scenarios.

Generalized Preconditions:

Theorem 3. *In general, the generalized preconditions are not sound, i.e., $\exists s^* \in \phi_a^{\text{general}} : s^* \notin S_a$.*

Proof 3. *Proof by contradiction: Let $S_a := \{s_1, s_2\}$ with $s_1 = [true, false, 10], s_2 = [true, true, 15]$. We assume $\tau^a = \{s_1, s_2\}$. Creating ϕ_a^{general} from τ^a gives the set $\phi_a^{\text{general}} = \{[true, b, z] \mid b \in \{true, false\}, z \in [10, 15]\}$. Let $s_3 = [true, false, 15]$, then $s_3 \in \phi_a^{\text{general}}$, but $s_3 \notin S_a$. So $\exists s^* \in \phi_a^{\text{general}} : s^* \notin S_a$.*

The generalized preconditions are not, in general, sound. In particular, generalization within the numerical space across all propositional configurations may introduce additional states into the precondition set that do not lie within the true applicability region of the action.

Theorem 4. *In general, the generalized preconditions are not complete, i.e.,* $\exists s^* \in S_a : s^* \notin \phi_a^{\text{general}}$.

Proof 4. *Proof by contradiction: Let* $S_a := \{s_1, s_2, s_3\}$ *with* $s_1 = [true, false], s_2 = [true, true]$ *and* $s_3 = [false, true]$. *We assume* $\tau^a = \{s_1, s_2\}$. *Creating* ϕ_a^{general} *from* τ^a *gives the set* $\phi_a^{\text{general}} = \{s_1, s_2\}$. *Then* $s_3 \notin \phi_a^{\text{general}}$, *but* $s_3 \in S_a$. *So* $\exists s \in S_a : s \notin \phi_a^{\text{general}}$.

The generalized preconditions are not generally complete, as propositional configurations that fall within the true applicability region may be absent from the training data and, consequently, excluded from the learned preconditions.

Theorem 5. *If there is only one single propositional configuration* $s(B)$ *within* S_a *and the numerical state space represented in* $\tau^a(N) \subseteq S_a(N)$ *and* $S_a(N)$ *is convex, then the generalized preconditions are sound.*

Proof 5. *Let* $\forall s^* \in S_a : s^*(B) = s(B)$, *then* $\forall s^* \in \tau^a : s^*(B) = s(B)$ *and therefore* $\forall s^* \in \phi_a^{\text{general}} : s^*(B) = s(B)$. *By assumption,* $S_a(N)$ *is convex and* $\forall s(N) \in \tau^a(N) : s(N) \in S_a(N)$, *so* $H(N) := \text{Conv}(\tau^a(N)) \subseteq S_a(N)$. *Then* $\forall s_a^*(N) \in H(N) : s_a^*(N) \in S_a(N)$. *Therefore,* $\forall s \in \phi_a^{\text{general}} : s \in S_a$.

Theorem 6. *If there is only one single propositional configuration* $s(B)$ *within* S_a *and the numerical state space represented in* $\tau^a(N)$ *is the convex hull of* $S_a(N)$, *then the generalized preconditions are complete.*

Proof 6. *Let* $\forall s^* \in S_a : s^*(B) = s(B)$, *then* $\forall s^* \in \tau^a : s^*(B) = s(B)$ *and therefore* $\forall s^* \in \phi_a^{\text{general}} : s^*(B) = s(B)$. *By assumption,* $S_a(N)$ *is convex and represented by* $\tau^a(N)$, *so* $H(N) := \text{Conv}(\tau^a(N)) = S_a(N)$. *Then* $\forall s^*(N) \in S_a(N) : s^*(N) \in \text{Conv}(\tau^a(N))$. *Therefore,* $\forall s \in S_a : s \in \phi_a^{\text{general}}$.

This generalization admits all convex combinations of observed numeric values. Assuming that the true numeric applicability region is globally convex, the generalized preconditions are both sound and complete. However, in non-convex domains, particularly where different propositional configurations admit distinct convex regions, soundness may be violated.

Dependency-Aware Preconditions:

Theorem 7. *The dependency-aware preconditions are also sound for non-globally convex hulls, where different propositional configurations* $s(B) \in Conf(B)$ *allow for different convex hulls* $H_{s(B)}$, *when* $\forall s(B) \in \tau^a : \text{Conv}(\tau_{a,s(B)}) \subseteq S_{a,s(B)}$.

Proof 7. *By definition* $\forall s(B) \in \tau^a : \text{Conv}(\tau_{a,s(B)}) \subseteq S_{a,s(B)}$. *Therefore, for each fixed* $s(B) \in \tau^a$ *it holds Theorem 5.*

Theorem 8. *The dependency-aware preconditions are complete for non-globally convex hulls, where different propositional configurations* $s(B) \in Conf(B)$ *allow for different convex hulls* $H_{s(B)}$, *when the dataset* τ^a *represents these hulls.*

Proof 8. *For each fixed* $s(B) \in \tau^a$ *by assumption,* $S_{a,s(B)}(N)$ *is convex and* $\forall s(B) \in \tau^a : \text{Conv}(\tau_{a,s(B)}(N)) = S_{a,s(B)}(N)$. *Therefore, for each fixed* $s(B) \in \tau^a$ *it holds Theorem 6.*

By exploiting the structure of propositional configurations to construct independent convex approximations of applicability regions, the dependency-aware preconditions also achieve soundness and completeness under only localized convexity assumptions.

7. Discussion

The empirical and theoretical evaluations offer complementary insights for answering our two posed research questions.

The experimental results show that it is possible to learn action preconditions in a real-world domain exclusively from observational data in which the action has previously been successfully applied. All methods supported both learning and subsequent planning based on the learned preconditions, confirming the efficiency of data-driven precondition learning in hybrid domains without expert input (cf. RQ1). Nevertheless, there were clear differences between the three approaches in terms of runtime performance and scalability (cf. Table 1).

Interestingly, for smaller domains, the *generalized* approach exhibited slower runtime growth than the *dependency-aware* variant. Yet, in the most complex domain, the *dependency-aware* method outperformed the *generalized* one in terms of runtime. This suggests that the overhead introduced by dependency modeling may become advantageous in higher-complexity settings.

The results regarding the planning runtimes (cf. Table 2) reflect a core limitation of convex hull-based representations: as numerical preconditions are encoded via systems of linear inequalities derived from hulls, computational demands grow with state space complexity. This can be mitigated by incorporating domain knowledge, e.g., by constraining the set of considered variables during learning, as proposed by [12]. In addition, preconditions can be represented more efficiently by bounded intervals in areas with axis- or box-shaped spaces, as it is often the case in most real-world applications.

It is also notable that planning runtimes exhibited high standard deviation, indicating sensitivity to the random seed. This variation is likely due to differences in the distribution of numerical variables, which can influences the structure of the state space.

Two key requirements must be considered when learning action models from data for planning in complex real-world domains: soundness and completeness. To learn preconditions fulfilling both of these requirements, especially in continuous or high-dimensional numerical domains, where it is not possible for training data to completely cover all valid states, is challenging.

The *exact* approach guarantees soundness by declaring an action applicable only in states that were explicitly observed during training. However, the approach does not account for the mentioned problem of missing data for every single state within the numerical state space and is therefore not complete.

By contrast, the *generalized* approach is complete under the assumptions that (i) the dataset includes all valid propositional configurations, (ii) the numerical state space can be approximated by a convex hull, and (iii) boundary states are represented in the dataset. Nonetheless, this method risks overgeneralization in both the propositional and numerical dimensions. In the propositional space, the method does not restrict preconditions to propositions true in all observed states, compromising soundness. In the numerical space, even if the outer shape is convex, the internal structure may contain holes or disallowed subregions, e.g., if certain configurations of the propositional state space only allow for a certain subspace of the numerical state space. Consequently, the generalized approach does not guarantee soundness in either component, which is confirmed by experimental observations where plans were generated that are invalid with respect to the expert-modeled domain.

Our novel *dependency-aware* approach maintains the completeness of the generalized approach under the same assumptions but addresses its key shortcomings by modeling dependencies among propositional variables and possibly resulting constraints in the numerical space. It also guarantees soundness under the reasonable assumption that, for each propositional configuration, the corresponding set of applicable numerical states forms a convex region.

Accordingly, the *dependency-aware* method ensures both soundness and completeness of planning in N3PCP domains, provided that the conditions (i)-(iii) are satisfied (cf. RQ2).

8. Conclusion

In this paper, we addressed the challenge of learning action preconditions in complex real-world domains characterized by hybrid, i.e., discrete and continuous, state spaces, using only observational data from states in which the actions were executed. We introduced a novel dependency-aware approach that constructs separate convex hulls in the numerical subspace for each discrete state configuration, thereby explicitly modeling the dependencies between discrete and continuous components of the state space. In contrast to methods determining the discrete preconditions based on conservative learning paradigms to ensure soundness, the proposed approach guarantees completeness with respect to the discrete state configurations within the training data. Moreover, by capturing structural dependencies between the discrete and continuous components, the approach also guarantees soundness, under the assumption that the valid numerical states associated with each discrete configuration form a convex region.

Future work will focus on addressing the computational challenges introduced by the convex-hull representations, which significantly impact planner performance in large-scale domains. We also plan to conduct a broader empirical evaluation across diverse real-world domains and integrate the learned models into planning systems capable of handling N3PCP problems.

Acknowledgments

This research as part of the projects EKI and LaiLa is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr, which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. Ghallab, D. Nau, P. Traverso, Automated Planning and Acting, Cambridge University Press, 2016.

[2] J. Ehrhardt, R. Heesch, O. Niggemann, Learning process steps as dynamical systems for a sub-symbolic approach of process planning in cyber-physical production systems, in: Artificial Intelligence. ECAI 2023 International Workshops, Springer Nature Switzerland, Cham, 2024, pp. 332–345.

[3] M. Hausknecht, P. Stone, Deep reinforcement learning in parameterized action space, 2016. doi:10.48550/ARXIV.1511.04143.

[4] R. Heesch, A. Cimatti, J. Ehrhardt, A. Diedrich, O. Niggemann, A lazy approach to neural numerical planning with control parameters, in: Frontiers in Artificial Intelligence and Applications, volume Volume 392: ECAI 2024, 2024. doi:10.3233/FAIA241000.

[5] E. Savaş, M. Fox, D. Long, D. Magazzeni, Planning using actions with control parameters, in: Proceedings of the Twenty-second European Conference on Artificial Intelligence, 2016, pp. 1185–1193.

[6] M. Grand, D. Pellier, H. Fiorino, TempAMLSI: Temporal action model learning based on STRIPS translation, Proceedings of the International Conference on Automated Planning and Scheduling 32 (2022) 597–605.

[7] R. Heesch, J. Ehrhardt, O. Niggemann, Integrating machine learning into an SMT-based planning approach for production planning inc yber-physical production systems, in: Artificial Intelligence. ECAI 2023 International Workshops, Springer Nature Switzerland, Cham, 2024, pp. 318–331.

[8] A. Köcher, R. Heesch, N. Widulle, A. Nordhausen, J. Putzke, A. Windmann, O. Niggemann, A research agenda for ai planning in the field of flexible production systems, 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS) (2022) 1–8.
[9] E. Jarvenpaa, N. Siltala, M. Lanz, Formal resource and capability descriptions supporting rapid reconfiguration of assembly systems, in: 2016 IEEE International Symposium on Assembly and Manufacturing (ISAM), IEEE, 2016. doi:10.1109/isam.2016.7750724.
[10] H. Kagermann, J. Helbig, A. Hellinger, W. Wahlster, Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group, Forschungsunion, 2013.
[11] S. J. Russell, P. Norvig, Artificial intelligence: a modern approach, Pearson, 2016.
[12] A. Mordoch, B. Juba, R. Stern, Learning safe numeric action models, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, 2023, pp. 12079–12086.
[13] D. Aineto, E. Scala, Action Model Learning with Guarantees, in: Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, 2024, pp. 801–811. doi:10.24963/kr.2024/75.
[14] A. Mordoch, E. Scala, R. Stern, B. Juba, Safe learning of pddl domains with conditional effects, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 34, 2024, pp. 387–395.
[15] J. Á. Segura-Muros, R. Pérez, J. Fernández-Olivares, Discovering relational and numerical expressions from plan traces for learning action models, Applied Intelligence 51 (2021) 7973–7989.
[16] J. Á. Segura-Muros, J. Fernández-Olivares, R. Pérez, Learning numerical action models from noisy input data, arXiv preprint arXiv:2111.04997 (2021).
[17] B. Juba, H. S. Le, R. Stern, Safe learning of lifted action models, in: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, volume 18, 2021, pp. 379–389.
[18] A. Bunte, P. Wunderlich, N. Moriz, P. Li, A. Mankowski, A. Rogalla, O. Niggemann, Why symbolic ai is a key technology for self-adaption in the context of cpps, in: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2019, pp. 1701–1704.
[19] D. van Dalen, Logic and structure (3. ed.), Universitext, Springer, 1994.
[20] A. Cimatti, A. Griggio, S. Mover, M. Roveri, S. Tonetta, Verification modulo theories, Formal Methods Syst. Des. 60 (2022) 452–481.
[21] Z. Manna, A. Pnueli, The modal logic of programs, in: ICALP, volume 71 of *Lecture Notes in Computer Science*, Springer, 1979, pp. 385–409.
[22] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, Y. Zhu, Bounded model checking, Adv. Comput. 58 (2003) 117–148.
[23] M. Helmert, Decidability and undecidability results for planning with numerical state variables., in: AIPS, 2002, pp. 44–53.
[24] D. Vranješ, J. Ehrhardt, R. Heesch, L. Moddemann, H. S. Steude, O. Niggemann, Design principles for falsifiable, replicable and reproducible empirical machine learning research, in: 35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024, pp. 7–1.
[25] A. Micheli, A. Bit-Monnot, G. Röger, E. Scala, A. Valentini, L. Framba, A. Rovetta, A. Trapasso, L. Bonassi, A. E. Gerevini, L. Iocchi, F. Ingrand, U. Köckemann, F. Patrizi, A. Saetti, I. Serina, S. Stock, Unified planning: Modeling, manipulating and solving ai planning problems in python, SoftwareX 29 (2025) 102012. doi:https://doi.org/10.1016/j.softx.2024.102012.
[26] A. Gerevini, A. Saetti, I. Serina, An empirical analysis of some heuristic features for planning through local search and action graphs, Fundam. Inform. 107 (2011) 167–197. doi:10.3233/FI-2011-399.
[27] A. Valentini, A. Micheli, A. Cimatti, Temporal planning with intermediate conditions and effects, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, 2020, pp. 9975–9982.