HELMUT SCHMIDT UNIVERSITÄT
Universität der Bundeswehr Hamburg

AI@CPS
Informatik im Maschinenbau

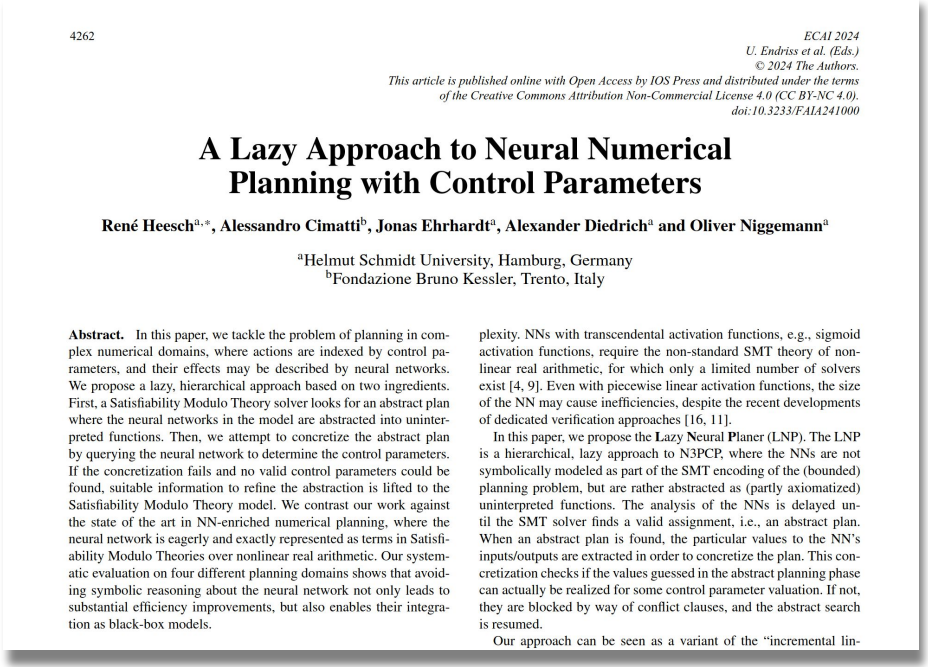
Integrating Neural Networks into SMT for AI Planning in Real-World Applications

Jonas Ehrhardt & René Heesch

ICAPS Tutorial,
November, 11 2025

icaps

Motivation



ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA241000

A Lazy Approach to Neural Numerical Planning with Control Parameters

René Heesch[a,*], Alessandro Cimatti[b], Jonas Ehrhardt[a], Alexander Diedrich[a] and Oliver Niggemann[a]

[a]Helmut Schmidt University, Hamburg, Germany
[b]Fondazione Bruno Kessler, Trento, Italy

Abstract. In this paper, we tackle the problem of planning in complex numerical domains, where actions are indexed by control parameters, and their effects may be described by neural networks. We propose a lazy, hierarchical approach based on two ingredients. First, a Satisfiability Modulo Theory solver looks for an abstract plan where the neural networks in the model are abstracted into uninterpreted functions. Then, we attempt to concretize the abstract plan by querying the neural network to determine the control parameters. If the concretization fails and no valid control parameters could be found, suitable information to refine the abstraction is lifted to the Satisfiability Modulo Theory model. We contrast our work against the state of the art in NN-enriched numerical planning, where the neural network is eagerly and exactly represented as terms in Satisfiability Modulo Theories over nonlinear real arithmetic. Our systematic evaluation on four different planning domains shows that avoiding symbolic reasoning about the neural network not only leads to substantial efficiency improvements, but also enables their integration as black-box models.

plexity. NNs with transcendental activation functions, e.g., sigmoid activation functions, require the non-standard SMT theory of nonlinear real arithmetic, for which only a limited number of solvers exist [4, 9]. Even with piecewise linear activation functions, the size of the NN may cause inefficiencies, despite the recent developments of dedicated verification approaches [16, 11].

In this paper, we propose the **L**azy **N**eural **P**laner (LNP). The LNP is a hierarchical, lazy approach to N3PCP, where the NNs are not symbolically modeled as part of the SMT encoding of the (bounded) planning problem, but are rather abstracted as (partly axiomatized) uninterpreted functions. The analysis of the NNs is delayed until the SMT solver finds a valid assignment, i.e., an abstract plan. When an abstract plan is found, the particular values to the NN's inputs/outputs are extracted in order to concretize the plan. This concretization checks if the values guessed in the abstract planning phase can actually be realized for some control parameter valuation. If not, they are blocked by way of conflict clauses, and the abstract search is resumed.

Our approach can be seen as a variant of the "incremental lin-

Heesch, R., Cimatti, A. Ehrhardt, J., Diedrich, A. & Niggemann, O. (2024). A Lazy Approach to Neural Numerical Planning with Control Parameters, 27TH European Conference on Artificial Intelligence (ECAI), Santiago de Compostela, Spain.

Motivation

Why should we use Neural Networks in planning?

- Though classical planners are powerful, when the planning domain model is correct, complete and at the right level of abstraction …
- … engineering such planning domain models is tedious, error-prone.
- Neural Networks are a powerful modeling tool, which can reduce the manual modelling effort.

Where can we use Neural Networks in planning?

- There is an action model learning community which learns formal action models (preconditions and effects) from data.
- Other approaches learn heuristics (domain dependent or domain independent).
- But we could also think about directly integrating Neural Networks into the solving process.

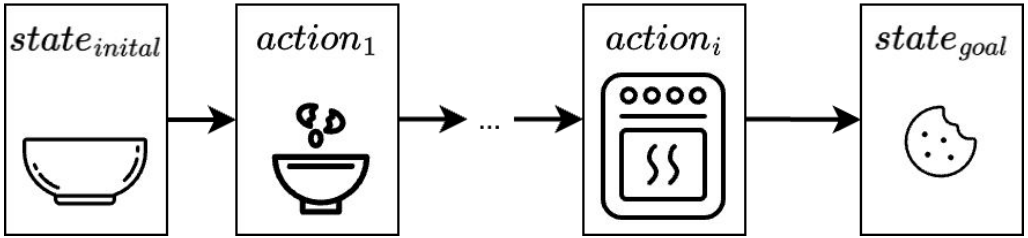
How do we propose to use ML in planning?

- We propose substituting the manually expensive modelling tasks with learned models.
- We propose learning preconditions and action effects in a non-formalized way from data.
- We propose a novel way of integrating such non-formalized models into a planning algorithm.

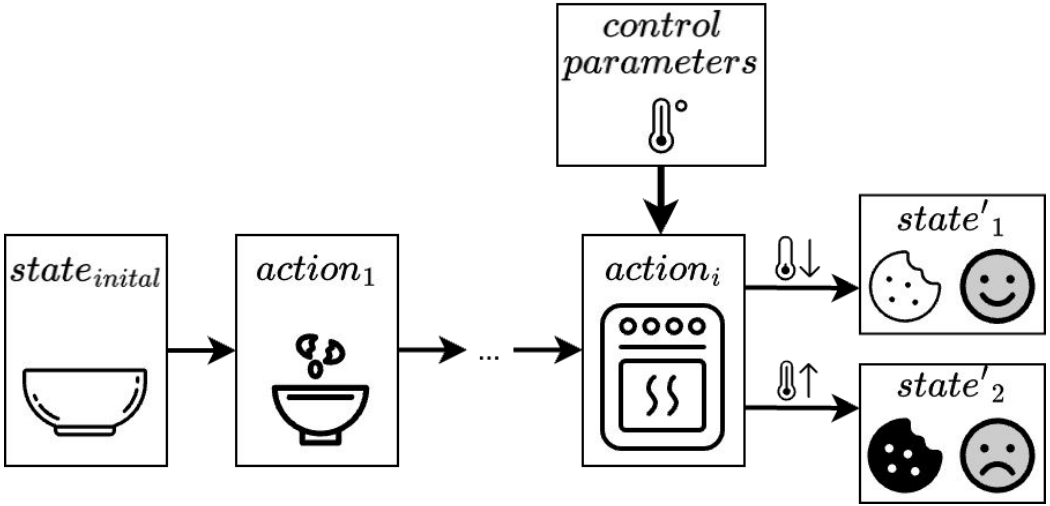
A. Mordoch, B. Juba, and R. Stern, “Learning Safe Numeric Action Models,” AAAI, vol. 37, no. 10, pp. 12079–12086, June 2023, doi: 10.1609/aaai.v37i10.26424.

M. Grand, D. Pellier, and H. Fiorino, “TempAMLSI: Temporal Action Model Learning Based on STRIPS Translation,” ICAPS, vol. 32, pp. 597–605, June 2022, doi: 10.1609/icaps.v32i1.19847.

Motivation – A Simple Planning Problem?

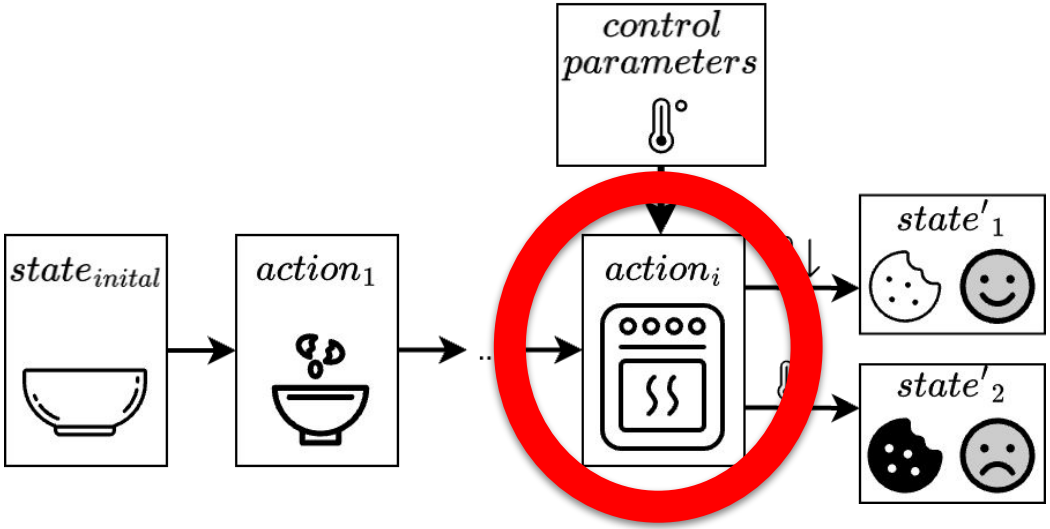


Motivation – A Simple Planning Problem?



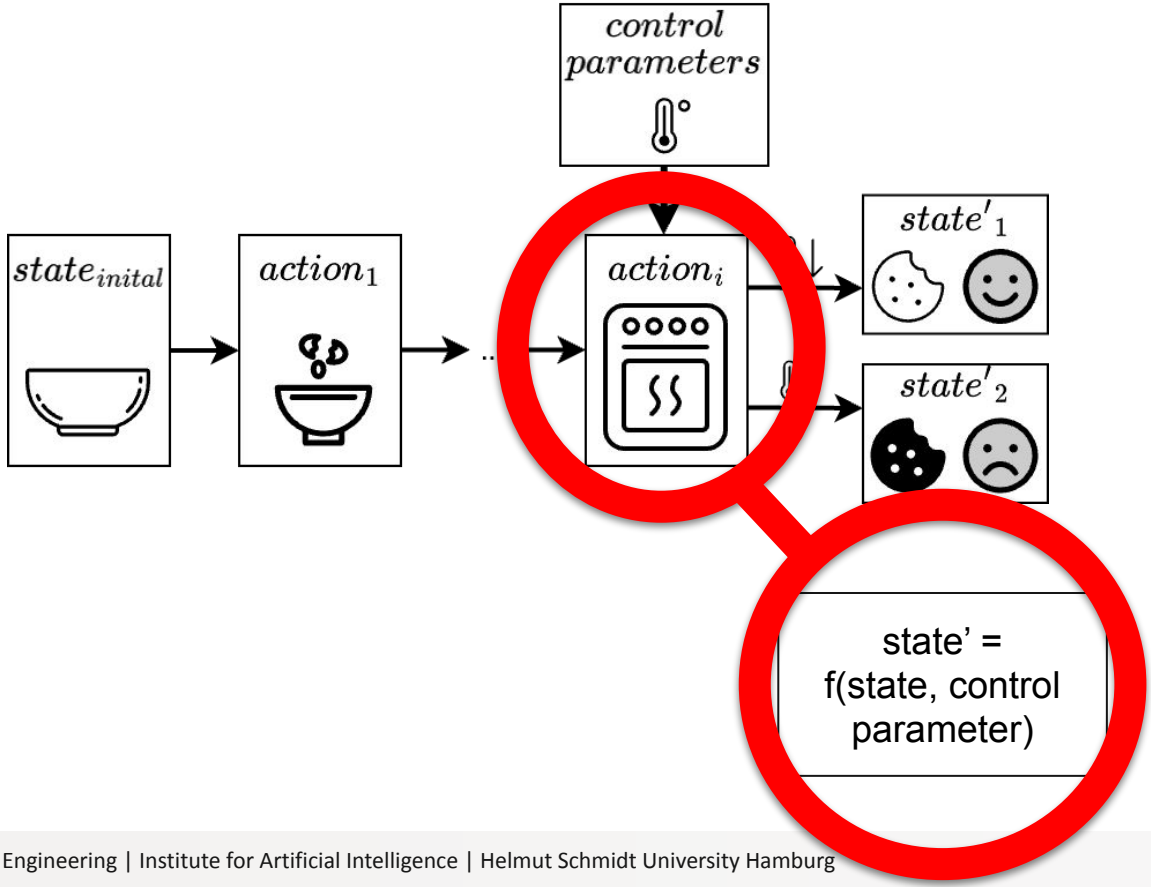
Savaş, E., Fox, M., Long, D., & Magazzeni, D. (2016). Planning using actions with control parameters. In ECAI 2016 (pp. 1185-1193)

Motivation – A Simple Planning Problem?

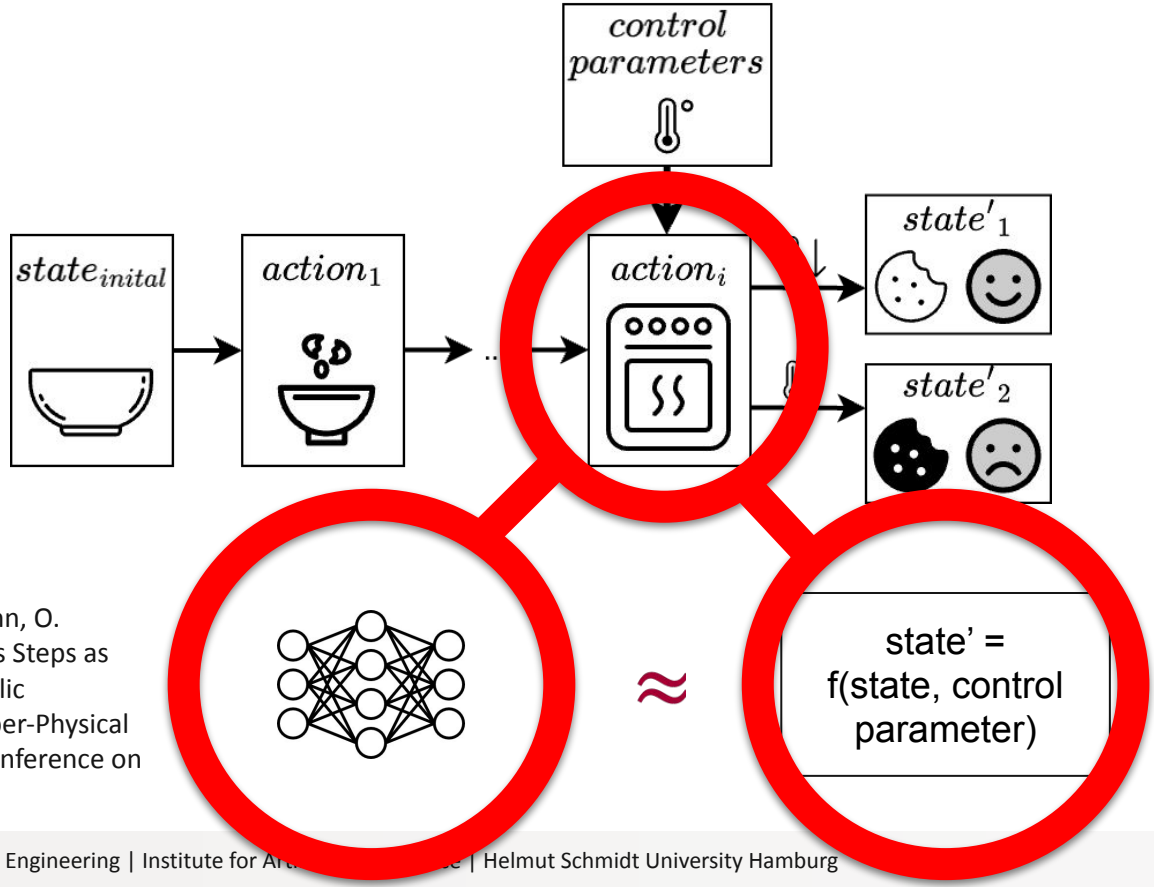


Savaş, E., Fox, M., Long, D., & Magazzeni, D. (2016). Planning using actions with control parameters. In ECAI 2016 (pp. 1185-1193)

Motivation – A Simple Planning Problem?

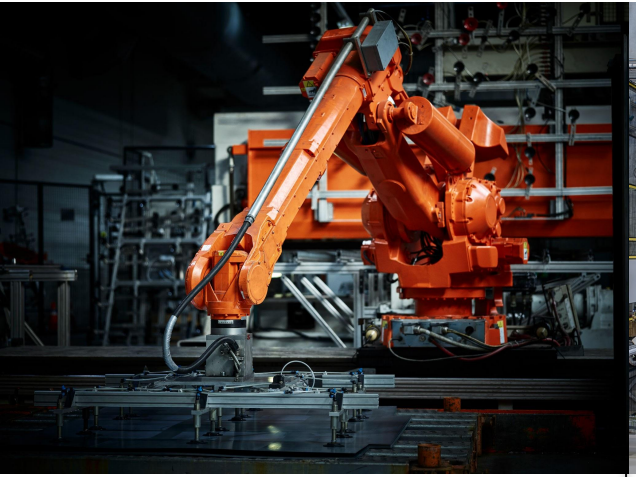


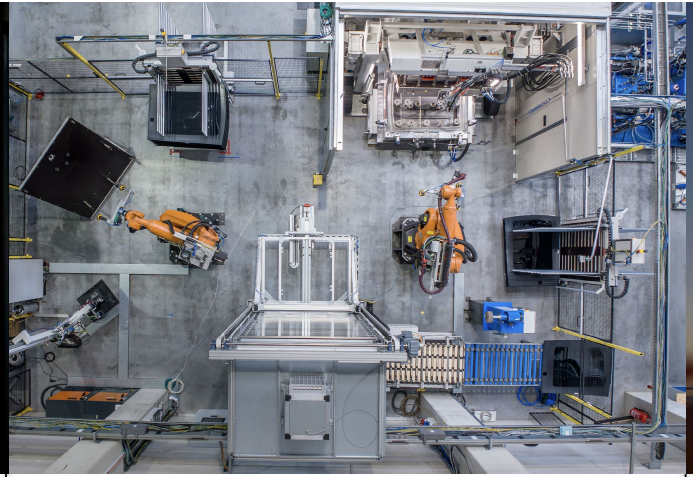
Motivation – A Simple Planning Problem?



Ehrhardt, J., Heesch, R., & Niggemann, O. (2023, September). Learning Process Steps as Dynamical Systems for a Sub-Symbolic Approach of Process Planning in Cyber-Physical Production Systems. In European Conference on Artificial Intelligence (pp. 332-345).

Back to Reality - Numerical Planning with Control Parameters





Exemplary Actions:

Action:	***pick***	**Action:**	***paint***
Parameter:	*coordinates,* *gripper angle,* *gripper force,* *…*	**Parameter:**	*paint thickness,* *flash off,* *duration,* *…*

Action:	***recharge***
Parameter:	*voltage,* *current,* *time,* *…*

Challenges

- Hybrid domains:
  - Discrete / propositional state variables
  - Numerical state variables

- Control parameters influence action effects

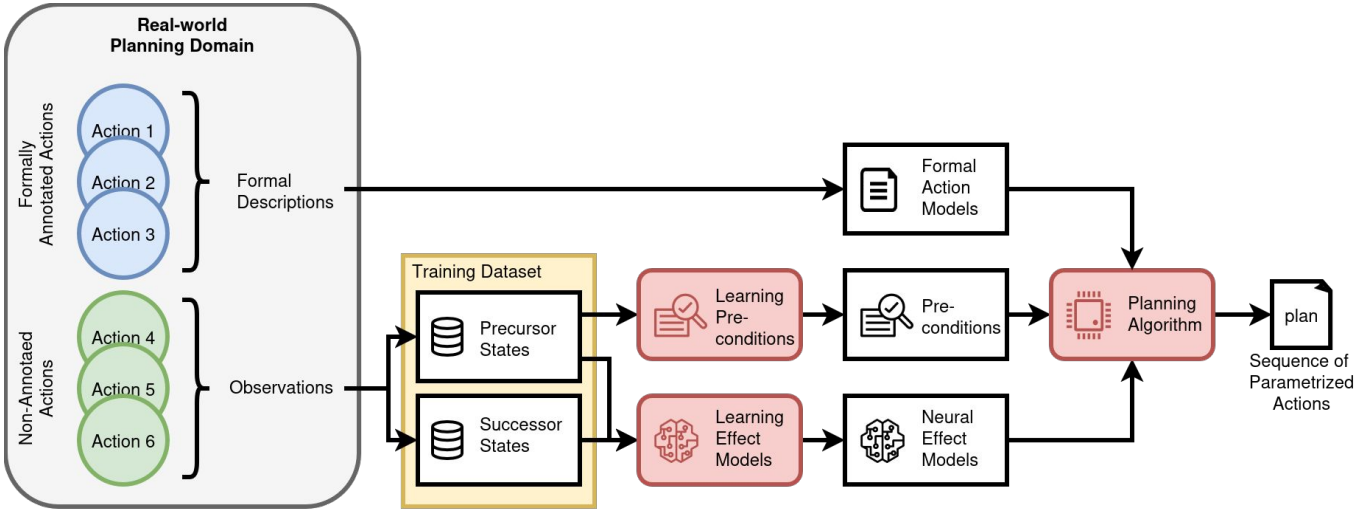
- We do not always have formal models of the different actions

- We do have observations of the system, e.g., data on the states an action has successfully been applied to and the corresponding successor state, traces from the planning domain.

Agenda

1. Problem Formalization
2. A Joint State Space Representation
3. Learning Preconditions
4. Learning Neural Effect Models
5. Eager Planning with Neural Effect Models
6. Lazy Planning with Neural Effects Models
7. Estimating Control Parameters via Concretization
8. Outlook

1. Problem Formalization



Heesch, R., Ehrhardt, J., & Niggemann, O. (2023, September). Integrating Machine Learning into an SMT-Based Planning Approach for Production Planning in Cyber-Physical Production Systems. In *European Conference on Artificial Intelligence* (pp. 318-331).

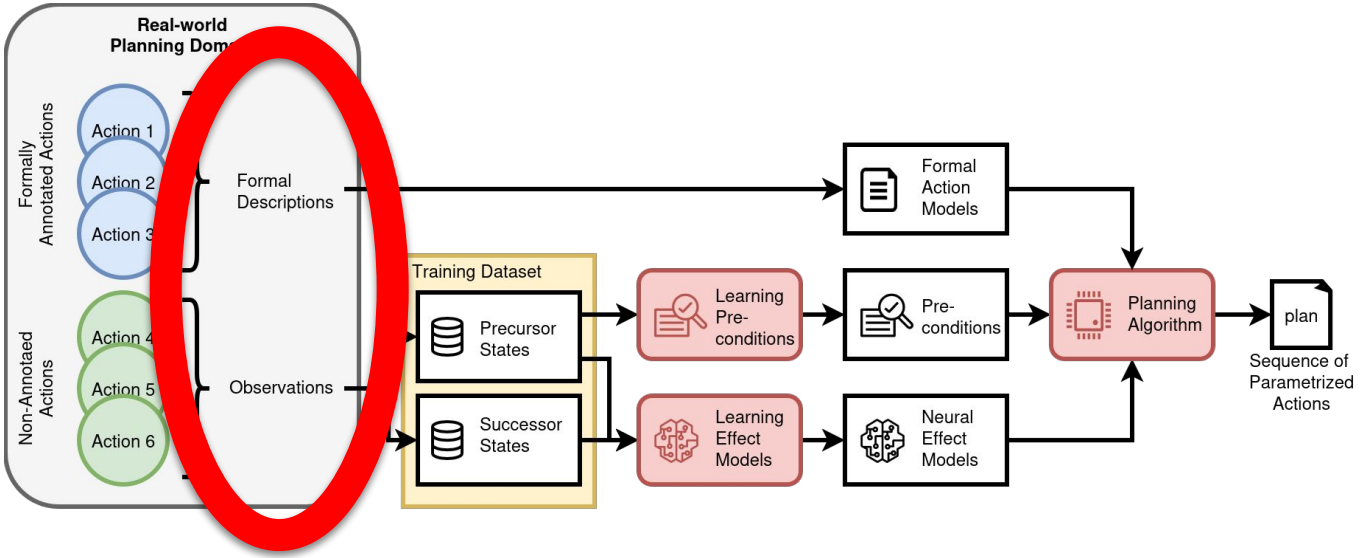
Problem Formalization

An NN-enriched Numerical Planning with Control Parameters (N3PCP) problem is a tuple:

$$P = \langle B, N, A, \Psi, T, M, I, G \rangle$$

- B denotes the set of propositional state variables
- N denotes the set of numerical state variables
- A denotes the set of actions
- Ψ denotes the set of control parameters
- T denotes the set of symbolic transition functions
- M denotes the set of executable transition functions
- I denotes the initial state
- G represents the set of goal conditions

2. A Joint State Space Representation



Heesch, R., Ehrhardt, J., & Niggemann, O. (2023, September). Integrating Machine Learning into an SMT-Based Planning Approach for Production Planning in Cyber-Physical Production Systems. In *European Conference on Artificial Intelligence* (pp. 318-331).

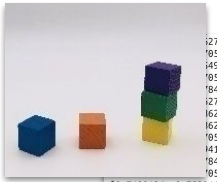
A Joint State Space Representation

Representing the state space in PDDL

```
(define (problem example)
  (:domain Blocksworld)
  (:objects a b c - smallblock)
            d e - block
            f - blueblock)
  (:init (clear a) (clear b) (clear c)
         (clear d) (clear e) (clear f)
         (ontable a) (ontable b) (ontable c)
         (ontable d) (ontable e) (ontable f))

  (:goal (and (on a d) (on b e) (on c f)))
)
```

Representing the state space for ML



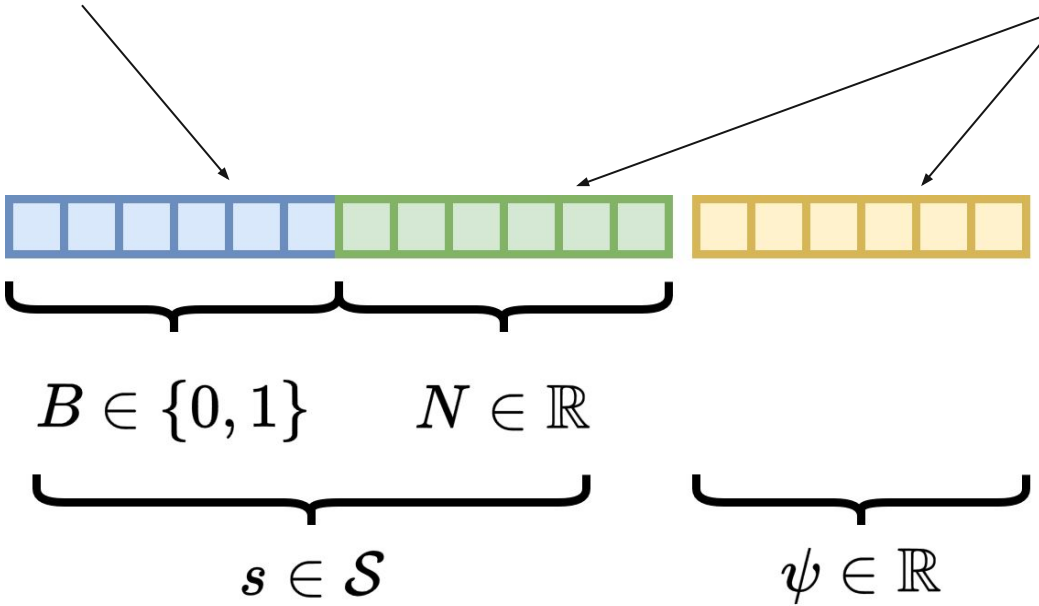
```
2746 0.77254903 0.77254903 0.77254903 0.76862746
059  0.76862746 0.7764706  0.78039217 0.7764706
4903 0.7647059  0.75686276]
059  0.7647059  0.7647059  0.7607843  0.7607843
843  0.7607843  0.7647059  0.76862746 0.76862746
2746 0.7607843  0.7529412 ]
6276 0.75686276 0.75686276 0.7529412  0.75686276
6276 0.75686276 0.7607843  0.7647059  0.7647059
059  0.7607843  0.75686276]
412  0.75686276 0.75686276 0.7529412  0.75686276
843  0.7607843  0.7607843  0.7647059  0.7647059
059  0.7607843  0.7529412 ]
[0.7490196  0.7529412  0.75686276 0.75686276 0.7529412  0.75686276
 0.7607843  0.7607843  0.7607843  0.7607843  0.7647059  0.77254903
 0.77254903 0.77254903 0.76862746 0.7607843 ]
[0.74509805 0.7490196  0.7529412  0.7529412  0.7529412  0.75686276
 0.7607843  0.7647059  0.7647059  0.76862746 0.77254903 0.78039217
 0.78039217 0.78431374 0.78431374 0.78039217]
[0.74509805 0.7529412  0.75686276 0.7607843  0.7647059  0.77254903
 0.78039217 0.78431374 0.7882353  0.7921569  0.79607844 0.8
 0.8039216  0.80784315 0.8039216  0.8       ]
[0.8039216  0.80784315 0.8156863  0.81960785 0.827451   0.83137256
 0.8352941  0.8352941  0.8392157  0.84313726 0.84313726 0.84313726
 0.84705883 0.84705883 0.84313726 0.8392157 ]
[0.85490197 0.85882354 0.85882354 0.8627451  0.8666667  0.8666667
 0.87058824 0.87058824 0.8745098  0.8745098  0.8745098  0.8745098
 0.8627451  0.85490197 0.87058824 0.8666667 ]
[0.87058824 0.8745098  0.8745098  0.8745098  0.8745098  0.8745098
 0.8784314  0.8784314  0.8784314  0.8784314  0.88235295 0.8980392
 0.46666667 0.2509804  0.6392157  0.90588236]
[0.8745098  0.8784314  0.8784314  0.8784314  0.88235295 0.88235295
 0.8862745  0.8862745  0.8862745  0.8862745  0.8862745  0.8901961
```

- Fixed layout & total ordering in order to maintain the semantics for ML models
- Mixed type handling (Boolean, integer, real-valued)
- Roundtrip compatibility between SMT and ML models

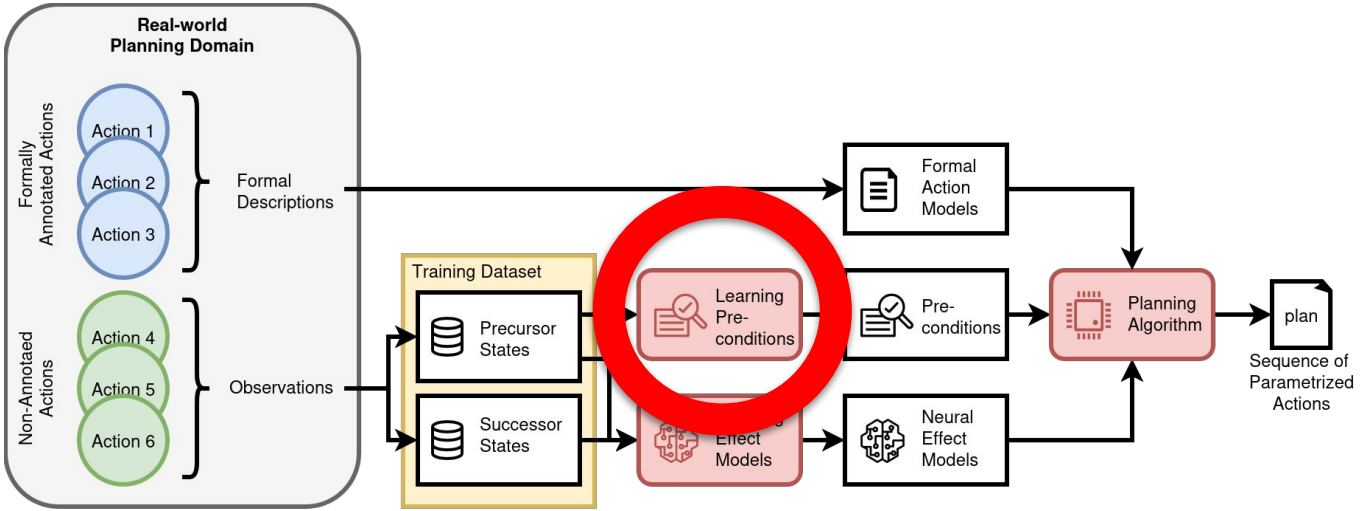
A Joint Feature Vector State Space Representation

transform discrete variables into zero or one encodings

Keep numerical variables and control parameters as real valued entries of the vector

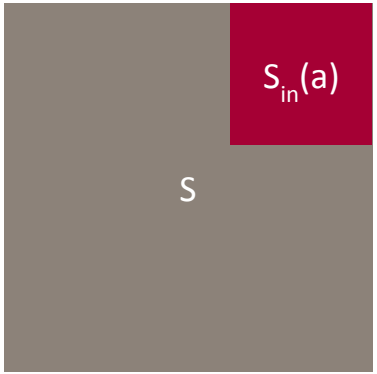


3. Learning Preconditions



Heesch, R., Ludwig, L., Ehrhardt, J., Diedrich, A. & Niggemann, O. (2025, October). Learning Sound and Complete Preconditions in Complex Real-World Planning Domains. ECAI Workshop on AI-based Planning for Complex Real-World Applications (CAIPI'25).

Learning Preconditions – Problem Description



S: State Space

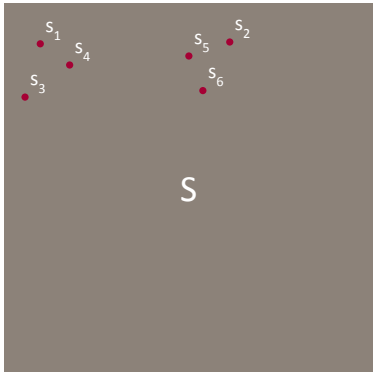
$S_{in}(a)$: Space of states, where action *a* is applicable

ϕ_a: Preconditions, such that
$S_{in}(a) = \{s \in S \mid \phi_a(s) = \text{true}\}$

- Data (τ_a) on the states an action has successfully been applied to
  - All valid discrete configurations
  - Not all valid numerical configurations

How can we derive preconditions that are sound and complete from these data?

Learning Exact Preconditions



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

$$\tau_a = \{s_1, \ldots, s_j\} \subseteq S_{\text{in}}(a)$$

Learning Exact Preconditions

Algorithm 1 Learn Exact Preconditions

Require: Dataset $\tau^a = \{s_1, s_2, \ldots, s_j\}$ 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}

Sound and complete with respect to the training data.

Learning Exact Preconditions

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

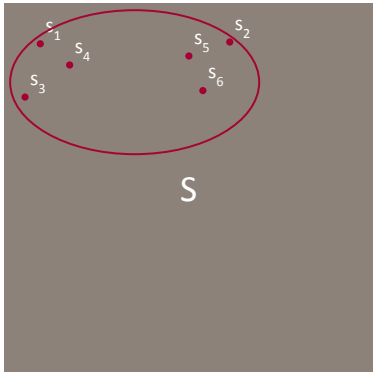
Proof 1. *Let S_a be the complete set of applicable states for action a. For the training set $\tau^a \subseteq S_a$ holds and $\forall s \in \phi_a^{\text{exact}} : s \in \tau^a$. Thus, $\forall s \in \phi_a^{\text{exact}} : s \in S_a$.*

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 $S_{in}(a)$.

Learning Generalized Preconditions



$$\tau_a = \{s_1, \ldots, s_j\} \subseteq S_{in}(a)$$

- Split propositional and numerical space (Mordoch et al.[1])
- Propositional space:
  - Disjunction of all observed propositional configurations
- Numerical space:
  - following Mordoch et al.[1]
  - Compute convex hull
  - Derive set of linear inequalities

[1] *Mordoch, A., Juba, B., Stern, R., Learning safe numeric action models, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, 2023, pp. 12079–12086.*

Learning Generalized Preconditions

Algorithm 2 Learn Generalized Preconditions

Require: Dataset $\tau^a = \{s_1, s_2, \ldots, s_j\}$ 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:

3:

4:

5:

6:

7:

8:

9:

10: Construct $\phi_a \leftarrow \bigvee_{s_B \in \Phi_B} s_B$

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}

Sound and complete under convexity assumptions and if boundary states are represented in the training data.

But what if different discrete configurations allow for different numerical spaces?

Learning 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* $\exists s^* \in \phi_a^{\text{general}} : s^* \notin S_a$.

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* $\exists s \in S_a : s \notin \phi_a^{\text{general}}$.

Learning Generalized 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.*

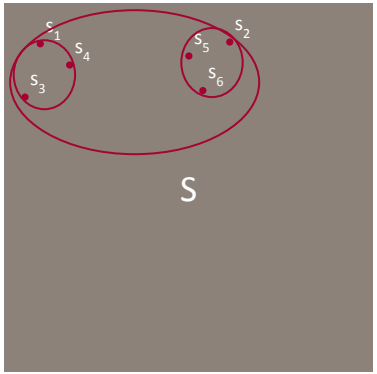
Proof 5. *Let $\forall s^* \in S_a : s^*(B) = s(B)$, then $\forall s^* \in \tau^a : s^*(B) = s(B)$ and therefore $\forall s^* \in \phi_a^{\text{general}} : s^*(B) = s(B)$. By assumption, $S_a(N)$ is convex and $\forall s(N) \in \tau^a(N) : s(N) \in S_a(N)$, so $H(N) := \text{Conv}(\tau^a(N)) \subseteq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

What if there are different propositional configurations within the training set τ_a ?

Learning Dependency-Aware Preconditions



- Propositional space:
  - Disjunction of all observed propositional configurations
- Numerical space:
  - Compute convex hull for each of the observed propositional configurations
  - Derive set of linear inequalities

$$\tau_a = \{s_1, \ldots, s_j\} \subseteq S_{in}(a)$$

Learning Dependency-Aware Preconditions

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}

Initialize $\phi_a^{\text{depend}} \leftarrow \emptyset$
Initialize map $groups : s_B \mapsto$ List of numeric vectors
for all $s \in \tau^a$ **do**
    $s_B \leftarrow s[\mathcal{ID}_B]$
    $n \leftarrow s[\mathcal{ID}_N]$
    Append n to $groups[s_B]$
end for
Initialize list $clauses \leftarrow [\,]$
for all $s_B \in \text{keys}(groups)$ **do**
    $V \leftarrow groups[s_B]$
    $H \leftarrow \texttt{ConvexHull}(V)$
    $\phi_{s_B}^{(N)} \leftarrow \texttt{GetHalfspaceInequalities}(H)$
    $\phi_{s_B}^{(B)} \leftarrow s_B$
    Append $(\phi_{s_B}^{(B)} \wedge \phi_{s_B}^{(N)})$ to $clauses$
end for
return $\phi_a^{\text{depend}} \leftarrow \bigvee_{c \in clauses} c$

Learning Dependency-Aware Preconditions

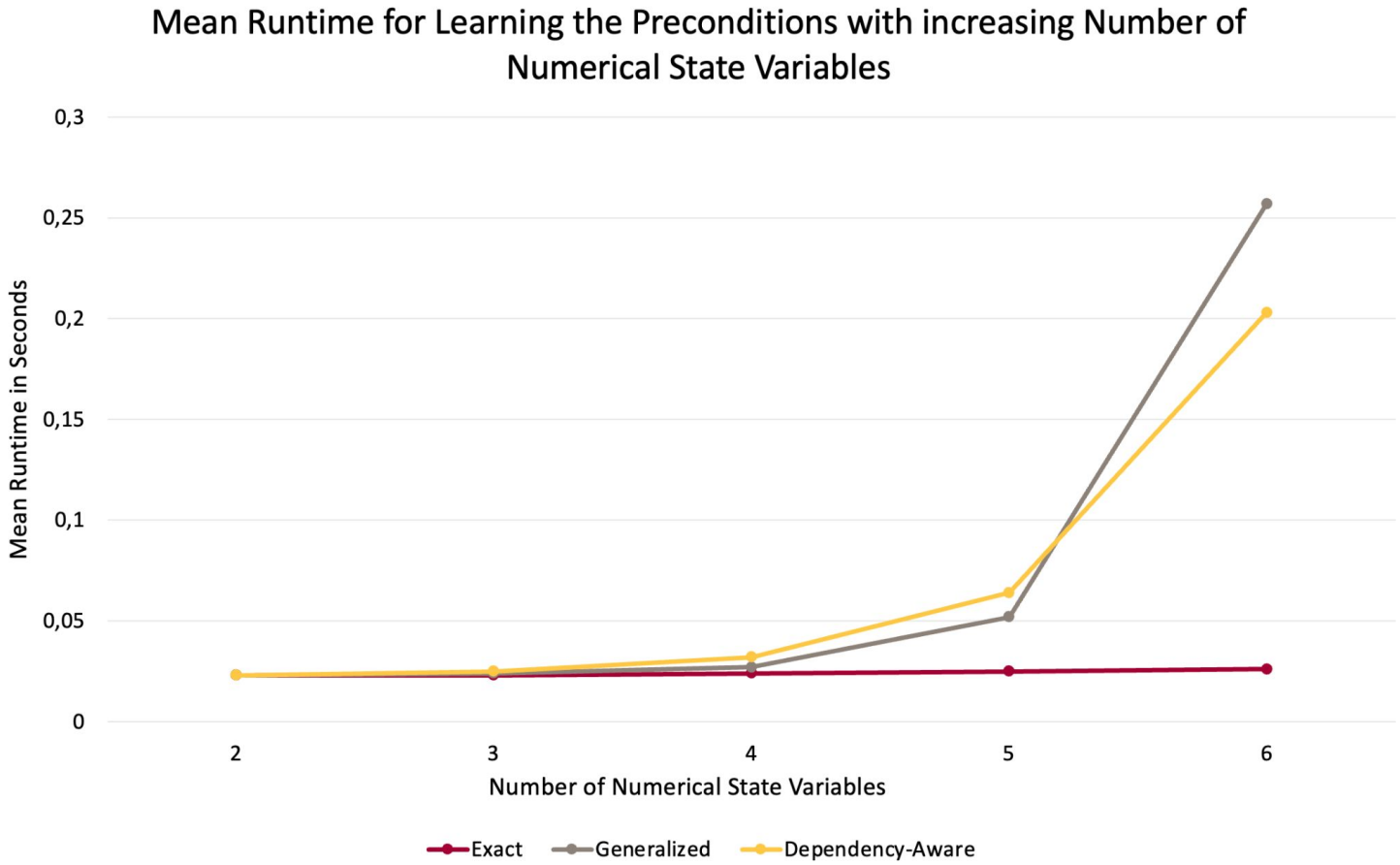
Theorem 7. *The dependency-aware preconditions are also sound for non-globally convex hulls, where different propositional configurations* $s(B) \in Conf(B)$ *allow for different convex hulls* $H_{s(B)}$*, when* $\forall s(B) \in \tau^a : \mathrm{Conv}(\tau_{a,s(B)}) \subseteq S_{a,s(B)}$.

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

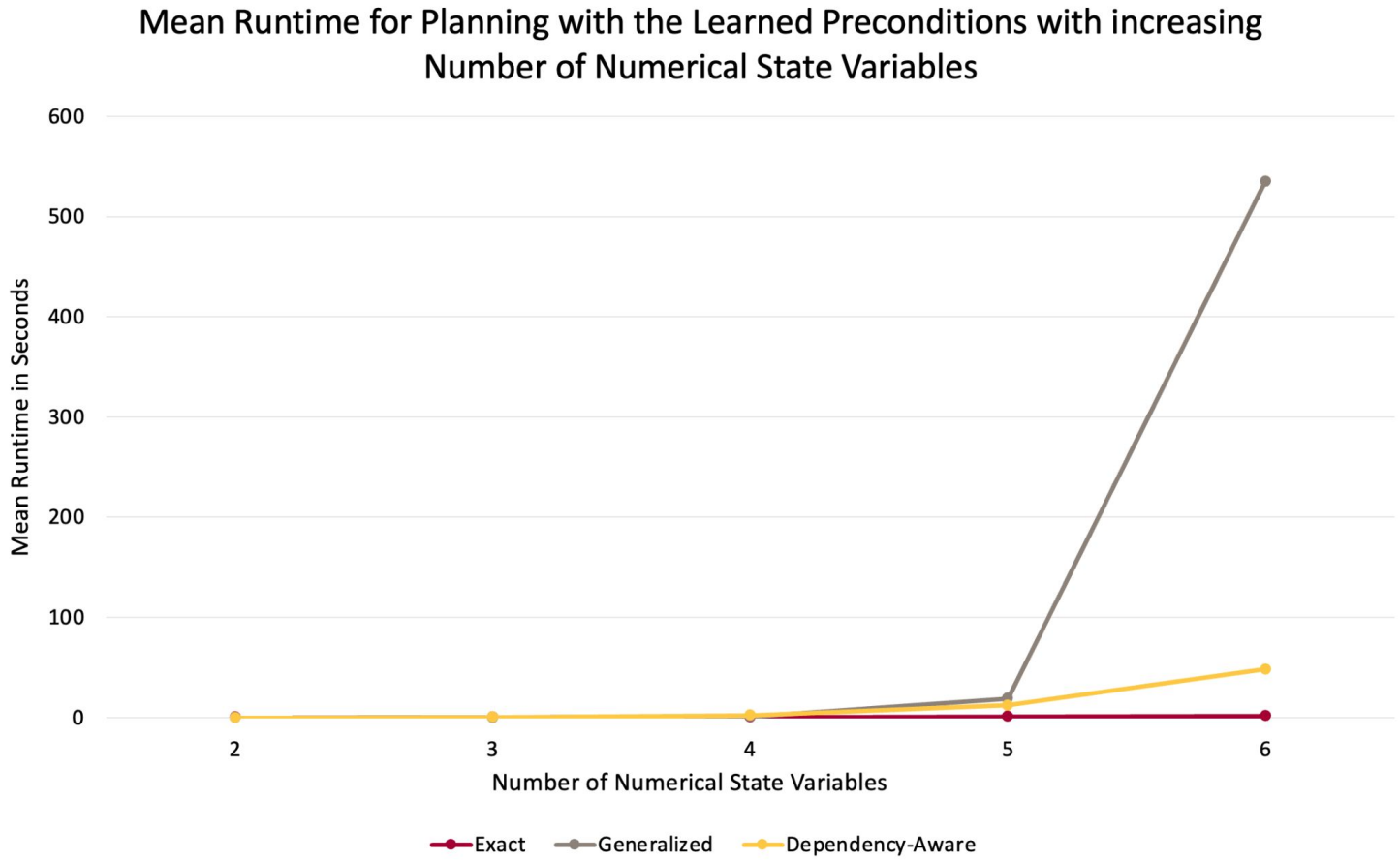
Theorem 8. *The dependency-aware preconditions are complete for non-globally convex hulls, where different propositional configurations* $s(B) \in Conf(B)$ *allow for different convex hulls* $H_{s(B)}$*, when the dataset* τ^a *represents these hulls.*

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

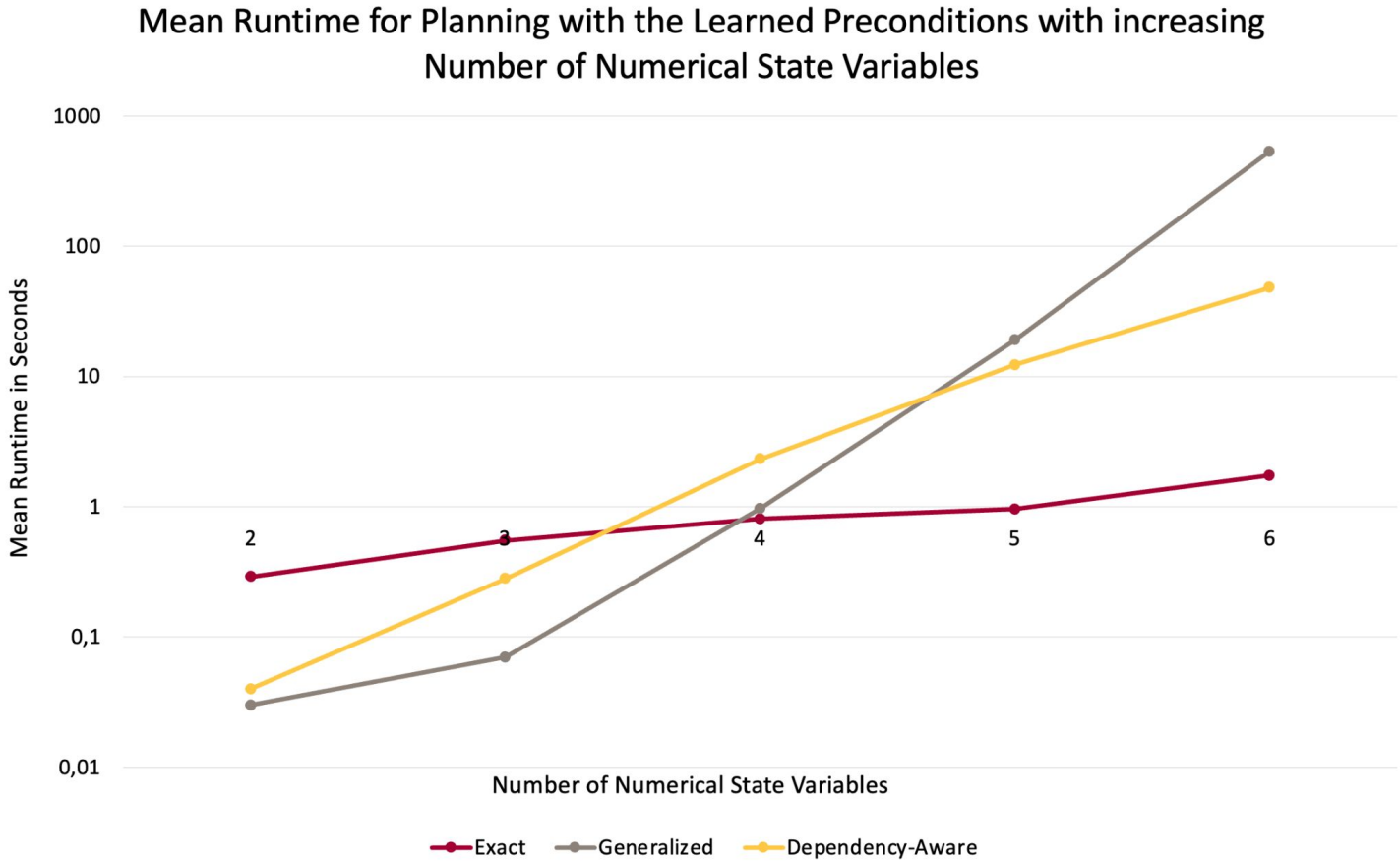
Empirical Evaluation – Learning Preconditions



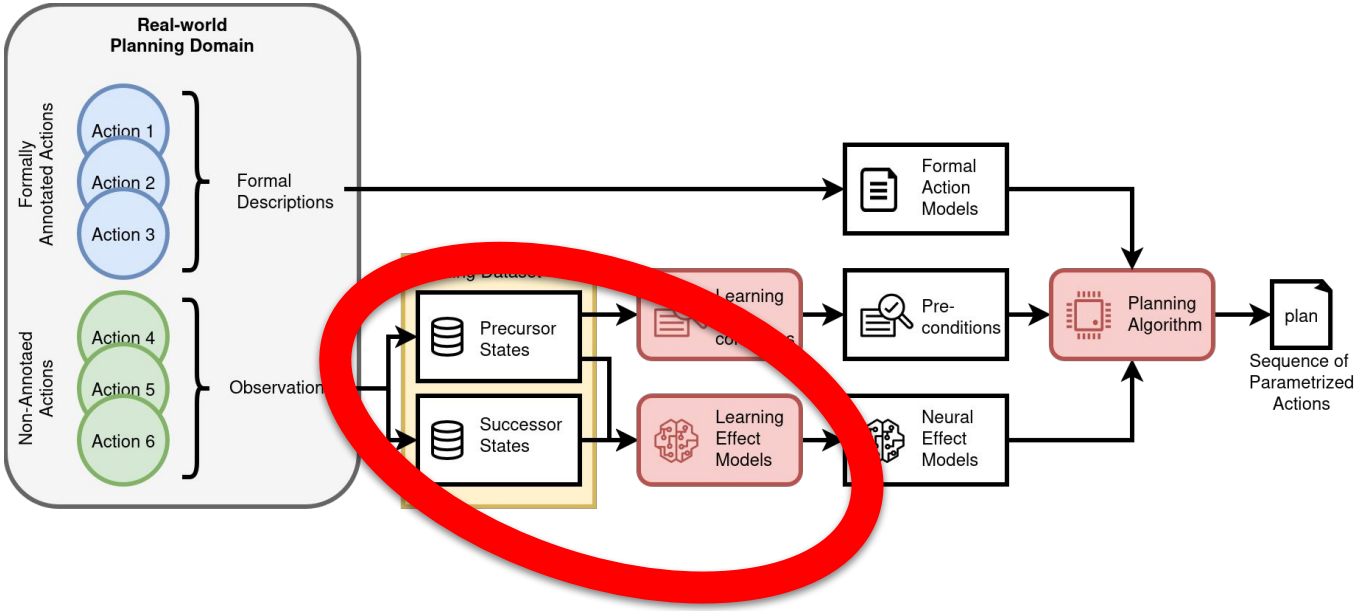
Empirical Evaluation – Learning Preconditions



Empirical Evaluation – Learning Preconditions



4. Learning Neural Effect Models



Ehrhardt, J., Heesch, R., & Niggemann, O. (2023, September). Learning Process Steps as Dynamical Systems for a Sub-Symbolic Approach of Process Planning in Cyber-Physical Production Systems. In *European Conference on Artificial Intelligence* (pp. 332-345).

Why Learning a Neural Effect Model?

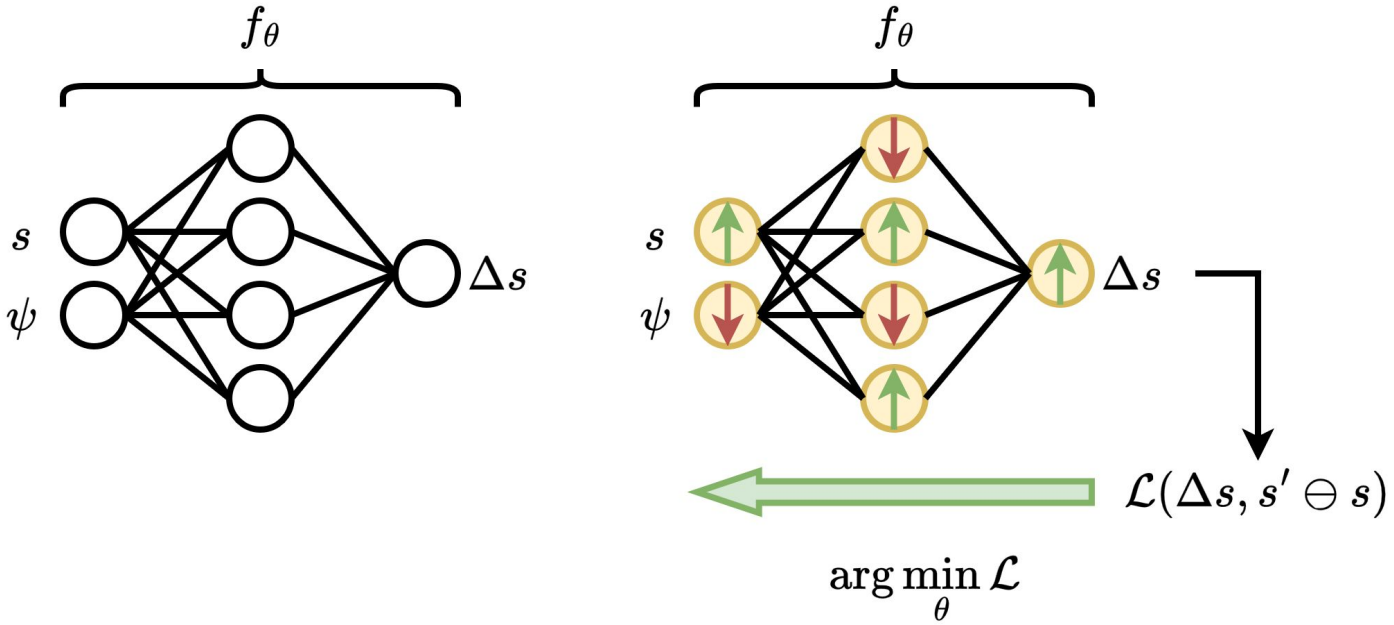
- ~~There is always an existing model of a planning domain.~~
- ~~If the planning domain model is not existing, one can easily model it with the available formalisms.~~
- ~~Everybody knows how to model a planning domain with the available formalisms.~~

BUT … there is hope

- If there is a functional dependency between an (input state, control parameter) tuple and an output state
- The Universal Approximation Theorem (Leshno et al. 1993) tells us, that we can approximate any function with a MLP.

M. Leshno, V. Ya. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” Neural Networks, vol. 6, no. 6, pp. 861–867, Jan. 1993, doi: 10.1016/s0893-6080(05)80131-5.

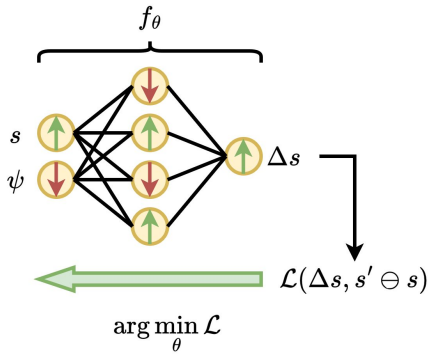
Fundamentals of Neural Effect Models



Fundamentals of Neural Effect Models

Let $\mathcal{S}$ be the state space, where $\mathcal{S} \subseteq \mathbb{R}^n$.

Let $\psi \in \Psi$ be a vector of real-valued control parameters, where $\Psi \subseteq \mathbb{R}^m$.



Let $\oplus : \mathcal{S} \times \Delta\mathcal{S} \to \mathcal{S}$ an abstract addition and $\ominus : \mathcal{S} \times \mathcal{S} \to \Delta\mathcal{S}$ an abstract difference, so that

$$s' = s \oplus \Delta s, \quad \Delta s = s' \ominus s, \quad s, s' \in \mathcal{S}.$$

Let $\mathcal{D} = \{(s_i, \psi_i, s_i')\}_{i=1}^{M}$ be a dataset of M observed state transitions from previously executed plans.

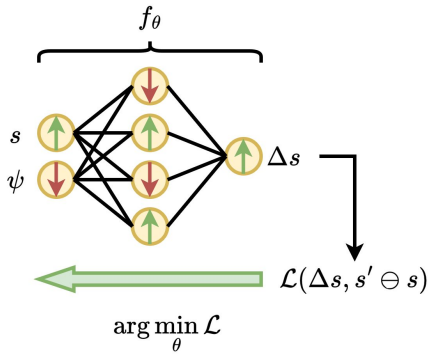
Fundamentals of Neural Effect Models

A Neural Effect Model is a parametric function

$$f_\theta : \mathcal{S} \times \Psi \to \Delta\mathcal{S}, \quad \widehat{\Delta s} = f_\theta(s, \psi),$$

where θ are the parameters of f, and the successor state can be calculated as

$$\hat{s}' = s \oplus f_\theta(s, \psi).$$



Predicting Δs with the Neural Effect Model helps to keep the gradients well scaled, by shrinking the the target magnitudes.

Fundamentals of Neural Effect Models

Finding the best θ is done by empirical risk minimization.

Let $\mathcal{L} : \Delta\mathcal{S} \times \Delta\mathcal{S} \to \mathbb{R}$ be a pointwise loss function (e.g. Mean-Squared Error).

The population risk then is

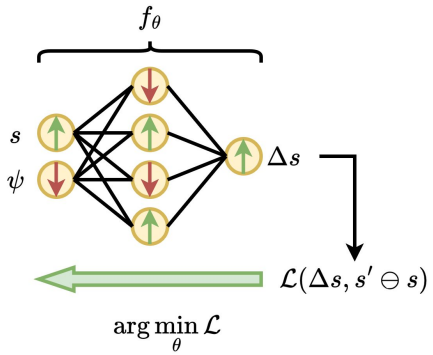
$$\mathcal{R}(\theta) = \mathbb{E}_{(s,\psi,s')}[\mathcal{L}(f_\theta(s,\psi), s' \ominus s)].$$

The empirical risk then is

$$\hat{\mathcal{R}}(\theta) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}(f_\theta(s_i,\psi_i), s'_i \ominus s)$$

To minimize the empirical counterpart, we can formulate

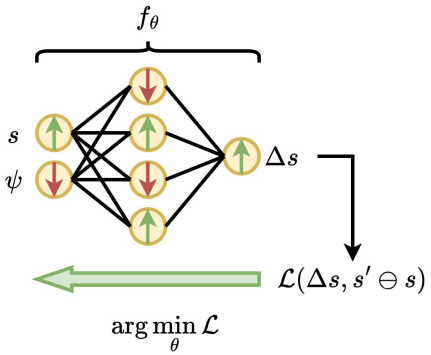
$$\hat{\theta} \in \arg\min_\theta \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}(f_\theta(s_i,\psi_i), s'_i \ominus s_i)$$



Fundamentals of Neural Effect Models

We minimize $\hat{\mathcal{R}}(\theta)$ by Stochastic Gradient Descent on mini-batches $\mathcal{B}_t \subset \mathcal{D}$,

$$g_t = \nabla_\theta [\frac{1}{|\mathcal{B}_t|} \sum_{(s,\psi,s') \in \mathcal{B}_t} \mathcal{L}(f_\theta(s,\psi), s' \ominus s)],$$

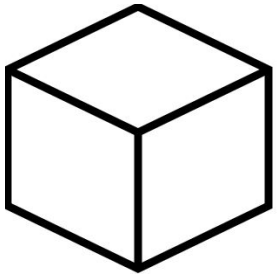


and the SGD update is

$$\theta_{t+1} = \theta_t - \alpha g_t,$$

where $\alpha > 0$ is the learning rate.

Neural Effect Models – Thinking inside Boxes

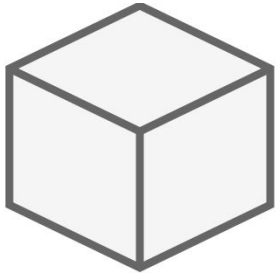


Whitebox Models

Model architecture is known

Model weights are known

Model weights are accessible



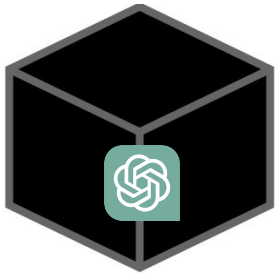
Greybox Models

~~Model architecture is known~~

~~Model weights are known~~

~~Model weights are accessible~~

Functionalities over the model are provided as a service
(e.g. backprop)



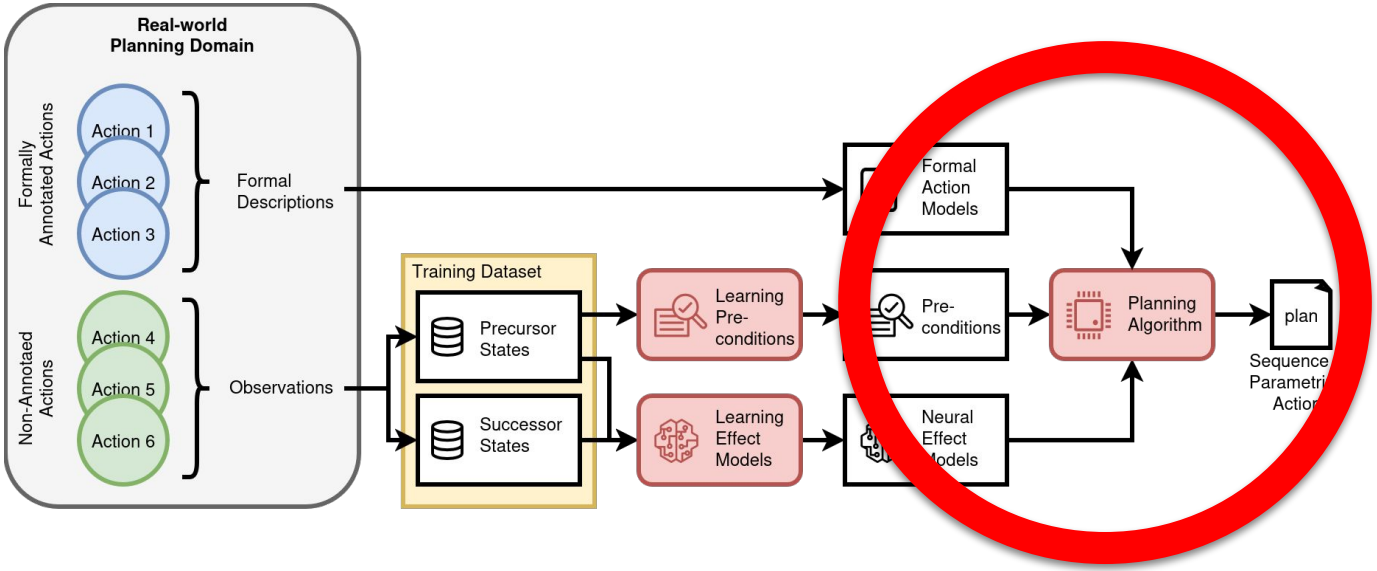
Blackbox Models

~~Model architecture is known~~

~~Model weights are known~~

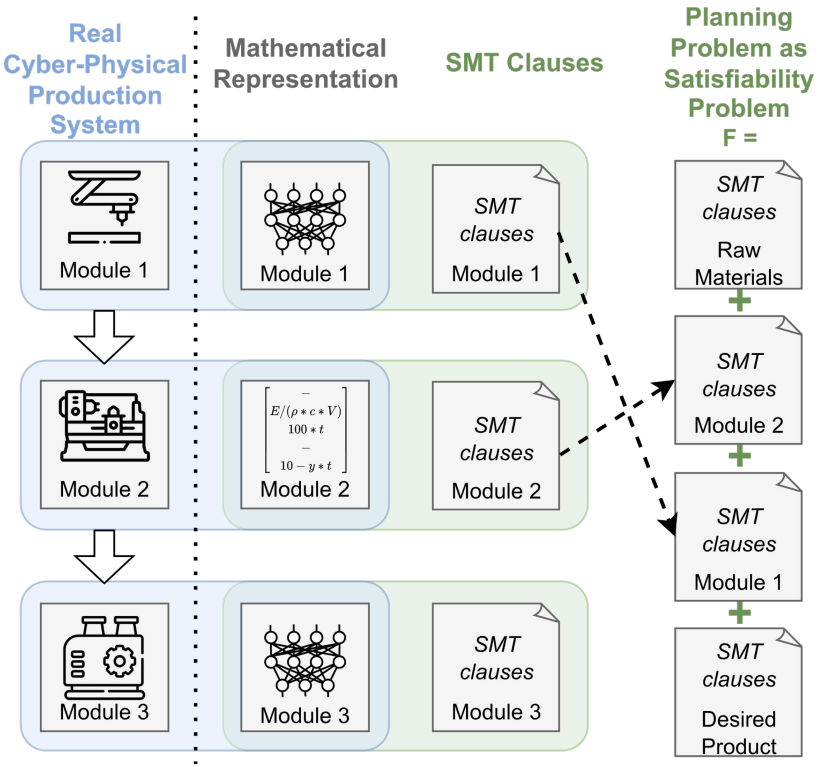
~~Model weights are accessible~~

5. Eager Planning with Neural Actions

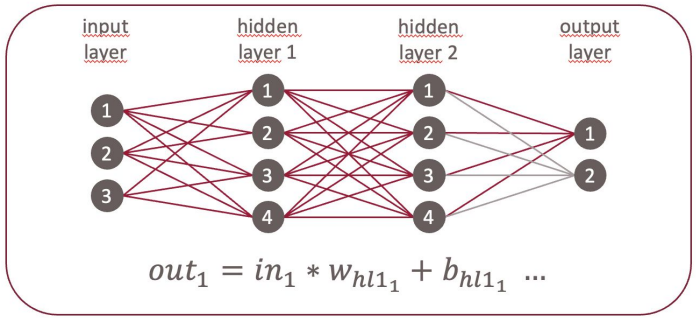


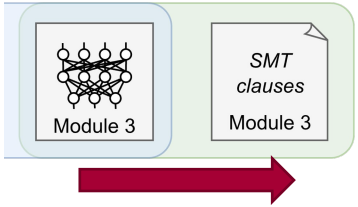
Heesch, R., Ehrhardt, J., & Niggemann, O. (2023, September). Integrating Machine Learning into an SMT-Based Planning Approach for Production Planning in Cyber-Physical Production Systems. In *European Conference on Artificial Intelligence* (pp. 318-331).

Eager Approach – Overview



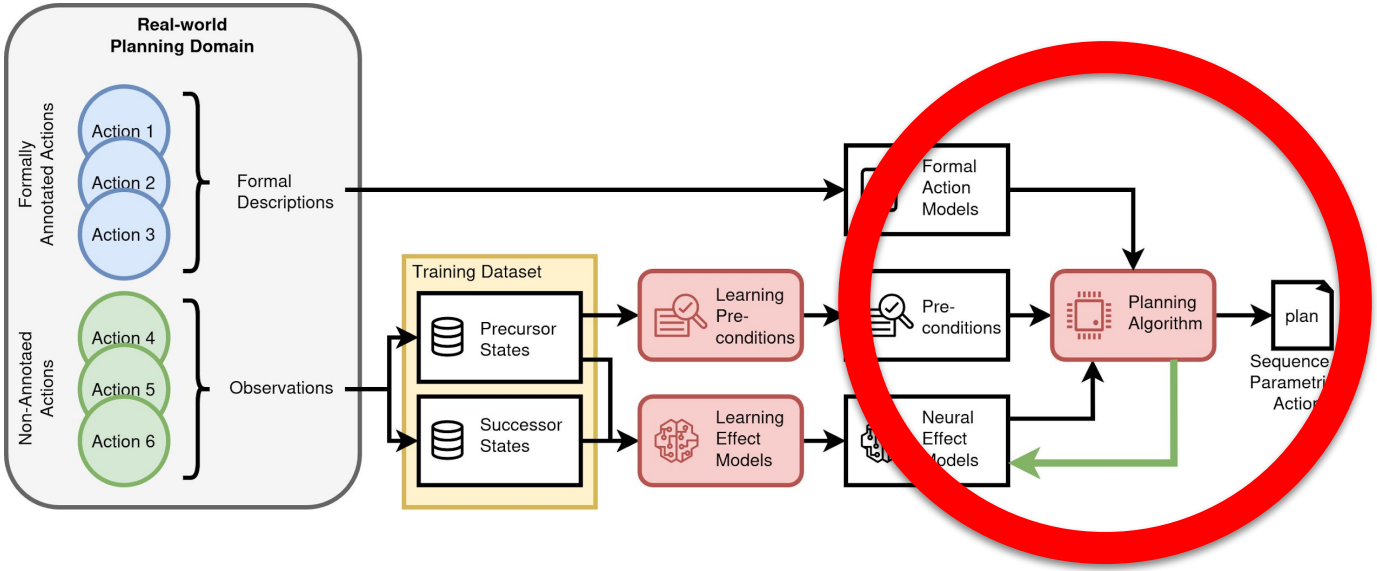
Eager Approach – Integration of NNs





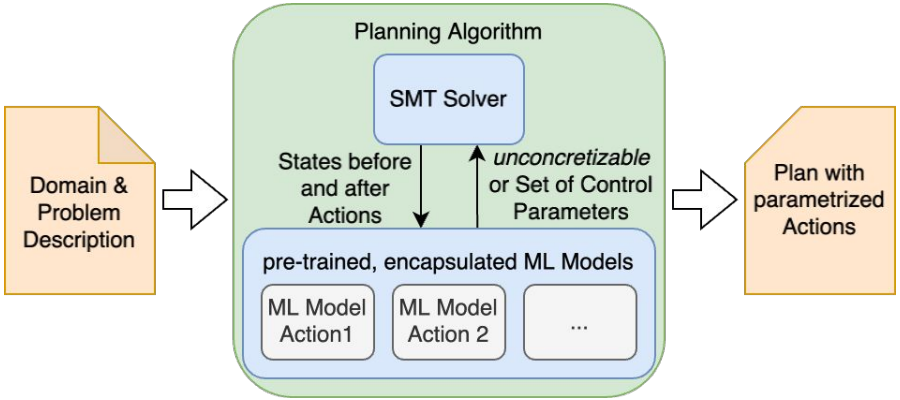
- long runtime
- architecture, weights and biases must be known
- network types are limited to the capabilities of the SMT solver

6. Lazy Planning with Neural Effect Models



Heesch, R., Cimatti, A. Ehrhardt, J., Diedrich, A. & Niggemann, O. (2024). A Lazy Approach to Neural Numerical Planning with Control Parameters, 27TH European Conference on Artificial Intelligence (ECAI), Santiago de Compostela, Spain.

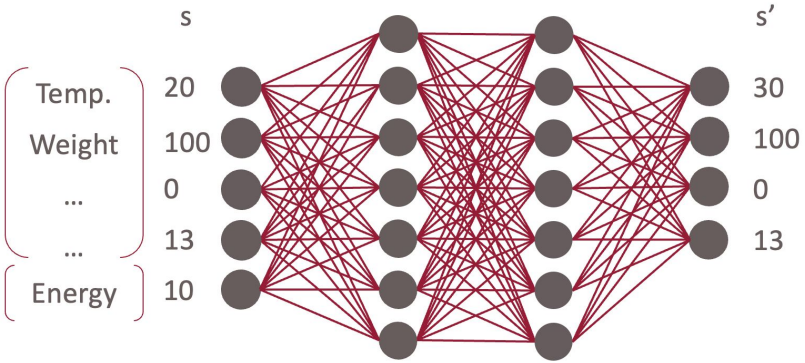
Lazy Approach – Overview



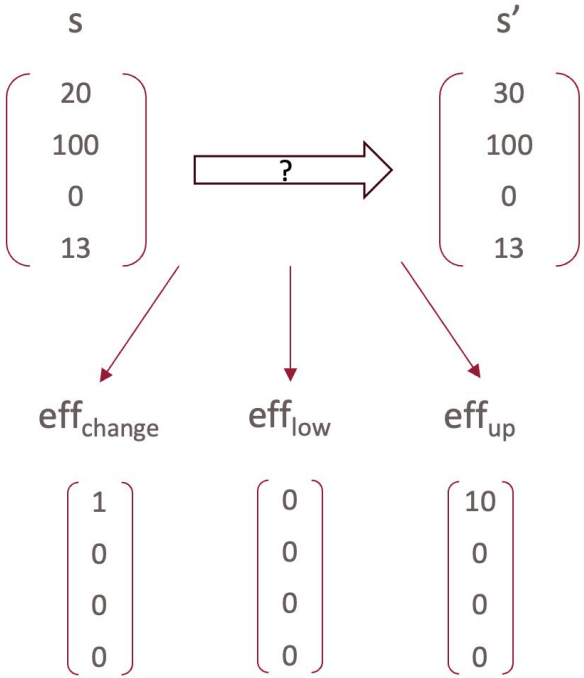
- Lifted information about neural actions is integrated into the satisfiability problem
- Analysing the Neural Network is delayed until an abstract plan is found by the SMT solver
- Either valid concretization is found or blocking lemma added

Lazy Approach – Lifted Information

Action *Heat* increases the temperature by the value of the control parameter *energy*.

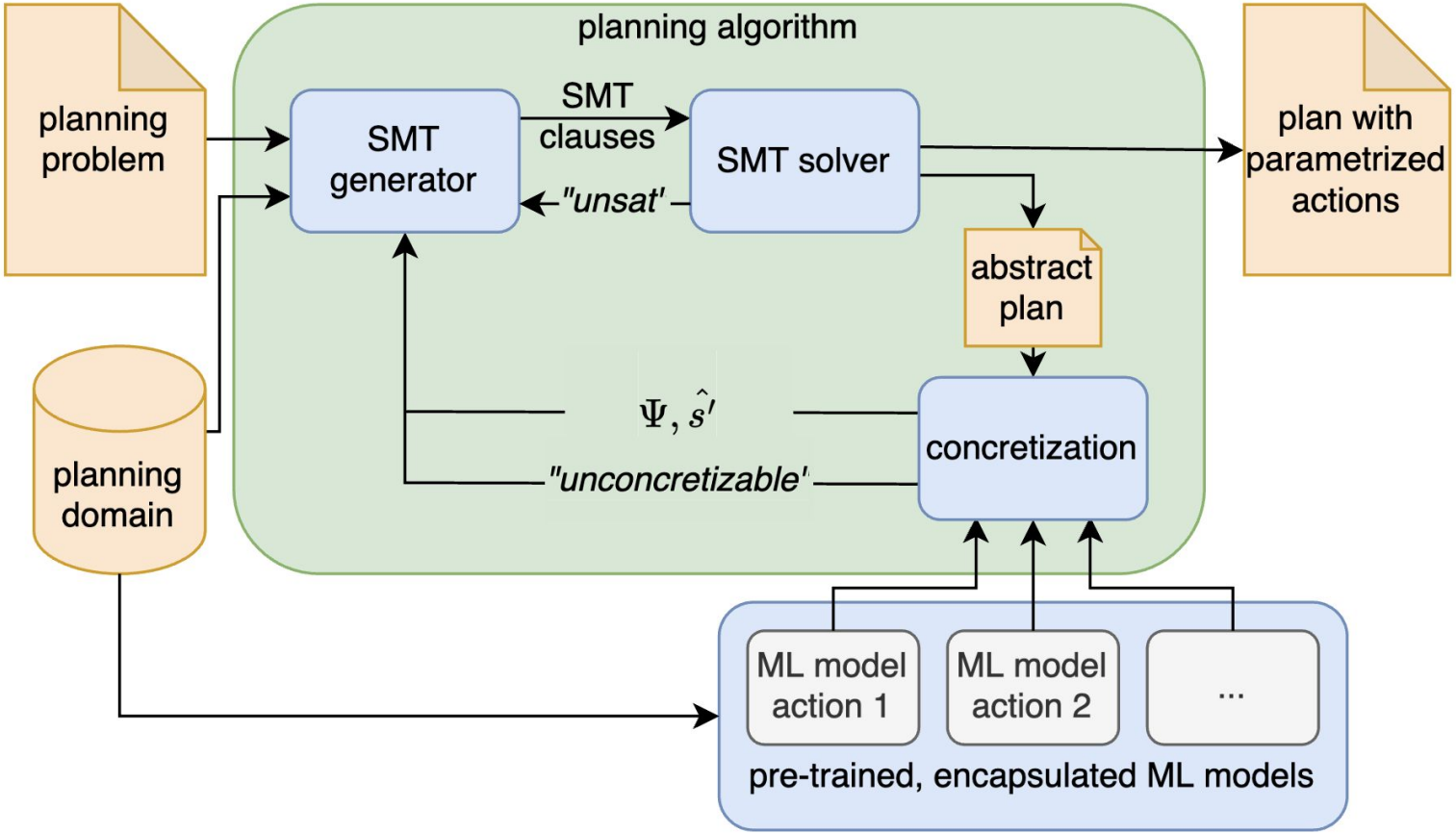


training of the Neural Network

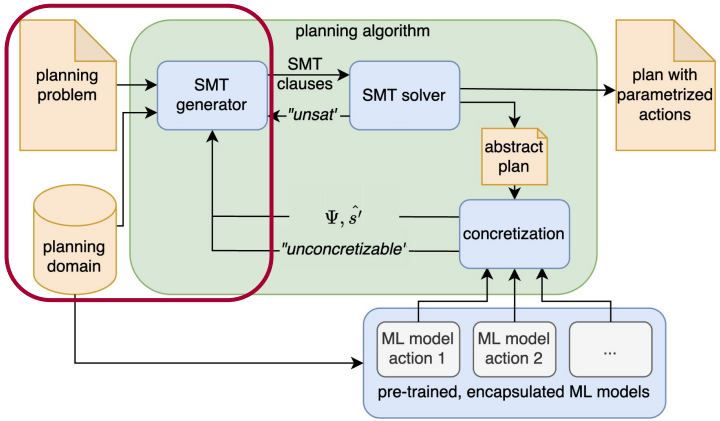


observer function

Lazy Approach – **L**azy **N**eural **P**laner



Lazy Approach – Generate the Satisfiability Problem

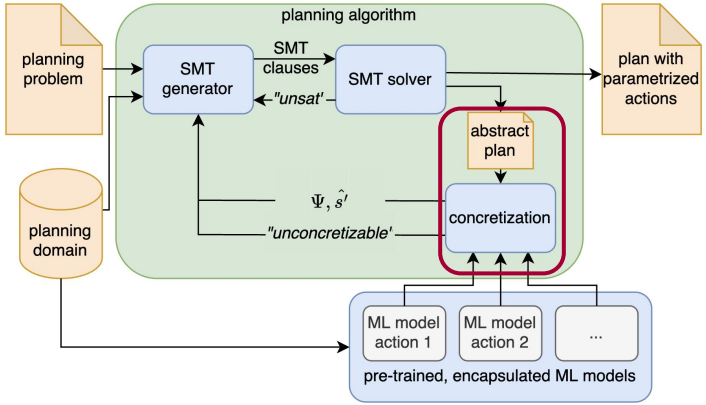


Lifted Information

$$\text{eff}_{\text{change}} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \text{eff}_{\text{low}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \text{eff}_{\text{up}} = \begin{pmatrix} 100 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

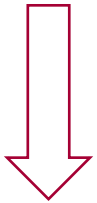
$$0 \leq \Delta^{Heat}_{Temp.} \leq 100$$

Lazy Approach – Abstract Plan



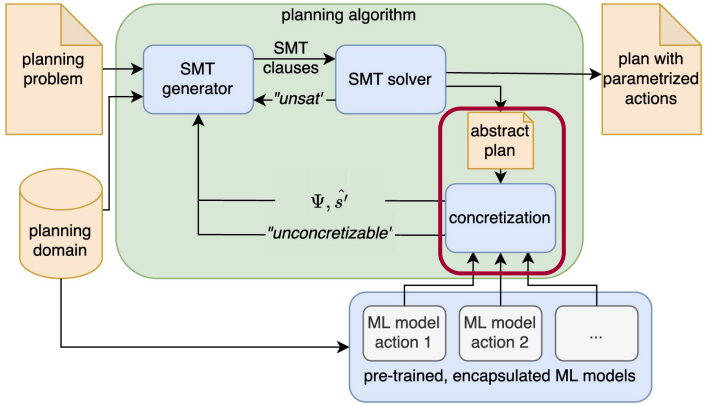
abstract plan:

- sequence of actions leading from the initial state to a state satisfying the goal conditions
- values for all variables replacing the effects of neural actions



explicit states the neural actions are applied to and the resulting states

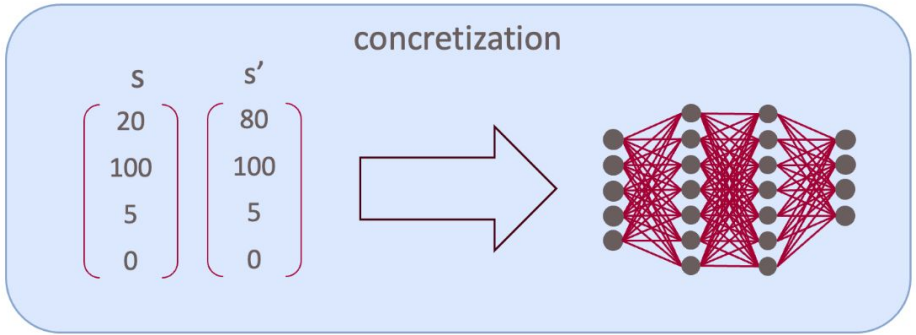
Lazy Approach – Abstract Plan



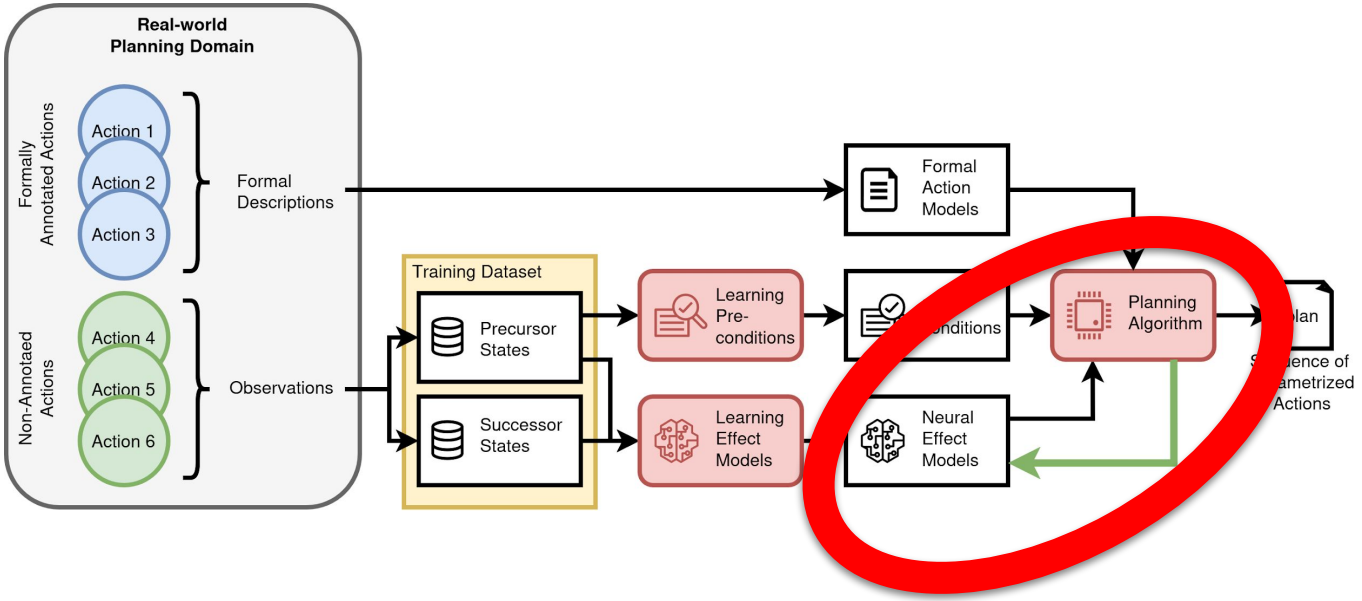
I	G
20	80
100	100
5	0
0	13

Abstract plan:

1. $Heat, \Delta^{Heat}_{Temp.} = 60$
2. ...
3. ...



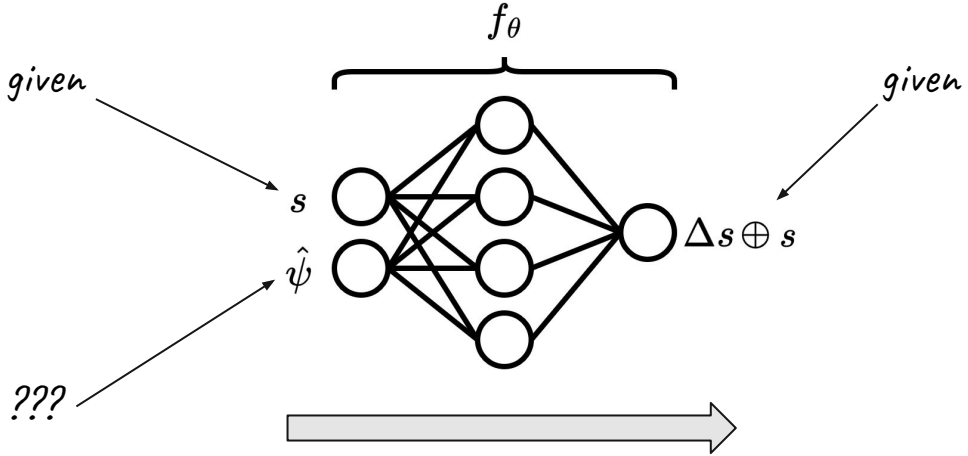
7. Efficiently Estimating Control Parameters via Concretization (*paramOpt*)



Ehrhardt, J., Schmidt, J., Heesch, R. & Niggemann, O. (2025, October). Using Gradient-based Optimization for Planning with Deep Q-Networks in Parametrized Action Spaces. ECAI Workshop on AI-based Planning for Complex Real-World Applications (CAIPI'25).

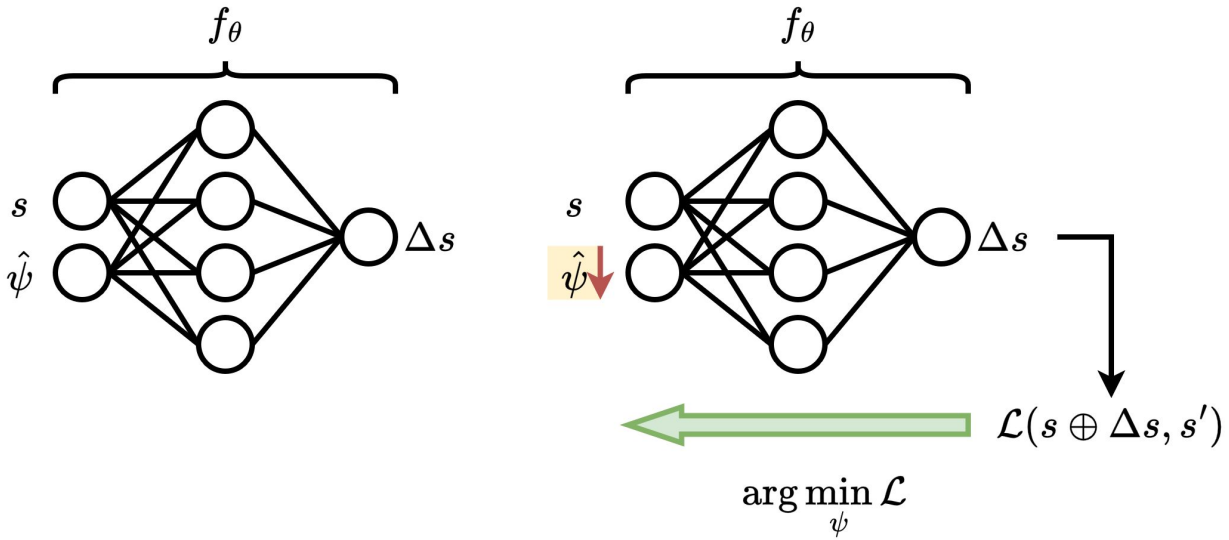
Estimating Parameters – the Problem

- Neural Networks are unidirectional (there are exceptions, but they are not well suited).
- We need a way to determine a part of the input to the Neural Network with contextual information on parts of the input and the output.



- Search is not an option, as we are dealing with high-dimensional and continuous state and parameter spaces.
- We propose an optimization-based approach for whitebox, greybox, and blackbox Neural Networks.

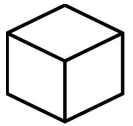
Estimating Parameters Gradient-based – *paramOpt*$_{GB}$

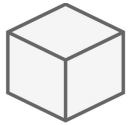


D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, “Semi-Supervised Learning with Deep Generative Models,” 2014, arXiv. doi: 10.48550/ARXIV.1406.5298.

Ehrhardt, J., Schmidt, J., Heesch, R. & Niggemann, O. (2025, October). Using Gradient-based Optimization for Planning with Deep Q-Networks in Parametrized Action Spaces. ECAI Workshop on AI-based Planning for Complex Real-World Applications (CAIPI’25).

Estimating Parameters *paramOpt*





Let (s, s') be a state tuple given by the lifted planner, and $\Delta s^\star := s' \ominus s$ the desired effect.

We aim to find the control parameters $\psi^\star$ that minimize the instance loss, starting from a parameter guess $\hat{\psi}$

$$J(\psi; s, s') := \mathcal{L}(f_\theta(s, \hat{\psi}), \Delta s^\star).$$

The optimization problem is

$$\psi^\star \in \arg\min_{\psi \in \Psi} J(\hat{\psi}; s, s')$$

To allow keeping ψ in legal ranges, we can use projected gradient descent

$$g_t = \nabla_\psi J(\psi_t; s, s'), \quad \psi_{t+1} = \Pi_\Psi(\psi_t - \beta g_t)$$

where Π is the projection function onto the feasible set Ψ, and $\beta > 0$ the learning rate.

Estimating Parameters Gradient-based – $paramOpt_{WB}$

Algorithm 1: $paramOpt_{wb}$ algorithm for white and grey box models, which allow access to gradients.

Require: s, s' // state, subsequent state
α // learning rate
γ // stopping threshold
$\hat{\psi} \leftarrow \text{init}()$ // initial guess
$\triangle_d \leftarrow +\infty$
$d_{++} \leftarrow +\infty$
while $\triangle_d > \gamma$ **do**
  $d \leftarrow \mathcal{L}(f_\theta(s, \psi) \oplus s, s')$
  $\nabla_{\hat{\psi}}\, d \leftarrow \text{backprop}(d, \hat{\psi})$
  $\hat{\psi} \leftarrow \hat{\psi} + \alpha \nabla_{\hat{\psi}}\, d$
  $\triangle_d \leftarrow d_{++} - d$
  $d_{++} \leftarrow d$
return $\tilde{\psi}^\star \leftarrow \hat{\psi}$ // optimized parameters

paramOpt with Gradient Estimates – $paramOpt_{BB}$

In cases where we cannot compute the gradient (e.g. no access to the model), we can work with approximated gradient estimates.

Finite difference method is one way of approximating gradients.

Therefore, we perturb the initial guess $\hat{\psi}$ by $\pm\epsilon \in \mathbb{R}$

$$\psi^{\pm} := \Pi_{\Psi}(\psi_t \pm \epsilon)$$

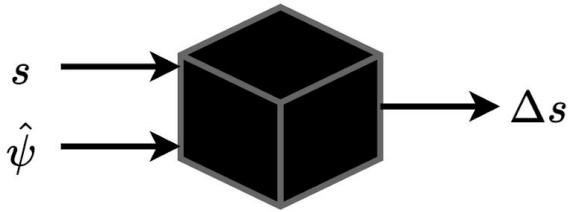
This allows to approximate the gradient as

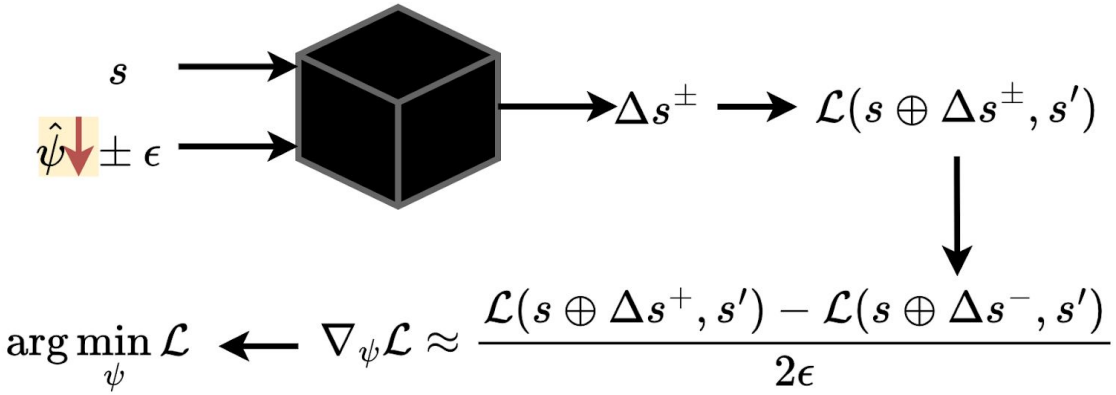
$$\tilde{g}_t := \frac{J(\psi^{+}; s, s') - J(\psi^{-}; s, s')}{2\epsilon}$$

The projected update, then is

$$\psi_{t+1} = \Pi_{\Psi}(\psi_t - \beta \tilde{g}_t)$$

Estimating Parameters Finite-Differences – *paramOpt*$_{BB}$



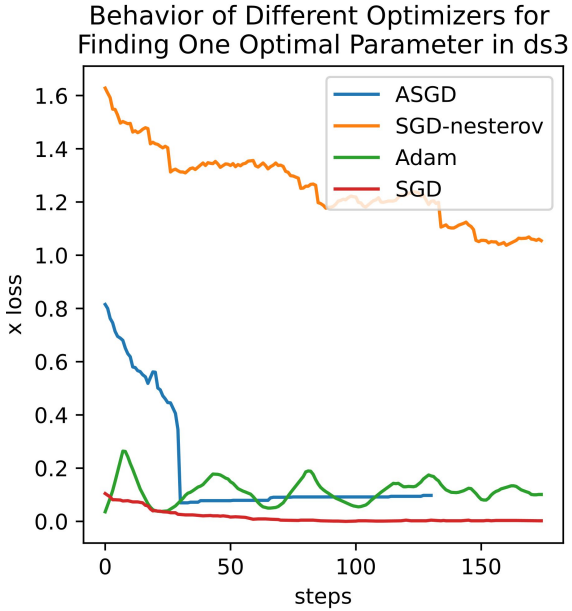


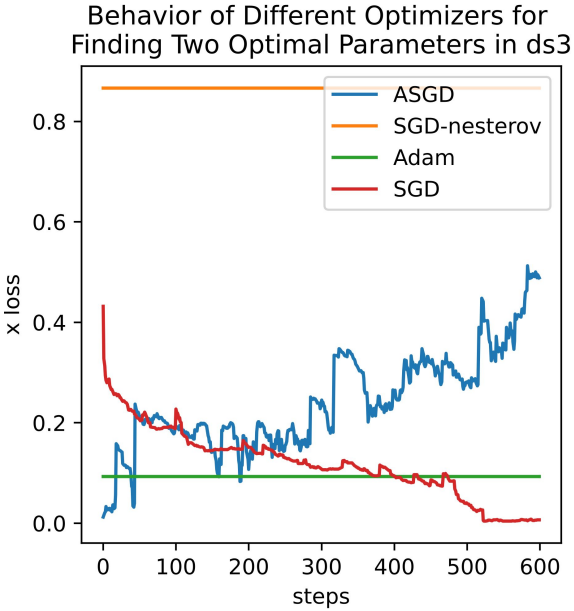
Estimating Parameters Finite-Differences – $paramOpt_{BB}$

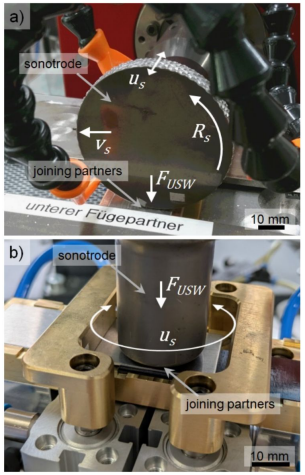
Algorithm 2: $paramOpt_{bb}$ algorithm for black box models, which do not allow access to gradients or weights.

Require: s, s' // state, subsequent state
α // learning rate
γ // stopping threshold
ϵ // difference
$\hat{\psi} \leftarrow \text{init}()$ // initial guess
$\triangle_d \leftarrow +\infty$
$d_{++} \leftarrow +\infty$
while $\triangle_d > \gamma$ **do**
  $\nabla_{\hat{\psi}}\, d \leftarrow 0$
  for $i \in \{1, \ldots, m\}$ **do**
    $\psi^+ \leftarrow \hat{\psi}; \quad \psi_i^+ \leftarrow \psi_i^+ + \epsilon$
    $\psi^- \leftarrow \hat{\psi}; \quad \psi_i^- \leftarrow \psi_i^- - \epsilon$
    $d^+ \leftarrow \mathcal{L}(f_\theta(s, \psi^+) \oplus s, s')$
    $d^- \leftarrow \mathcal{L}(f_\theta(s, \psi^-) \oplus s, s')$
    $\nabla_{\hat{\psi}} d[i] \leftarrow (d^+ - d^-)/(2\epsilon)$
  $\hat{\psi} \leftarrow \hat{\psi} - \alpha \nabla_{\hat{\psi}}\, d$
  $d \leftarrow \mathcal{L}(f_\theta(s, \hat{\psi}) \oplus s, s')$
  $\triangle_d \leftarrow d_{++} - d$
  $d_{++} \leftarrow d$
return $\tilde{\psi}^\star \leftarrow \hat{\psi}$ // optimized parameters

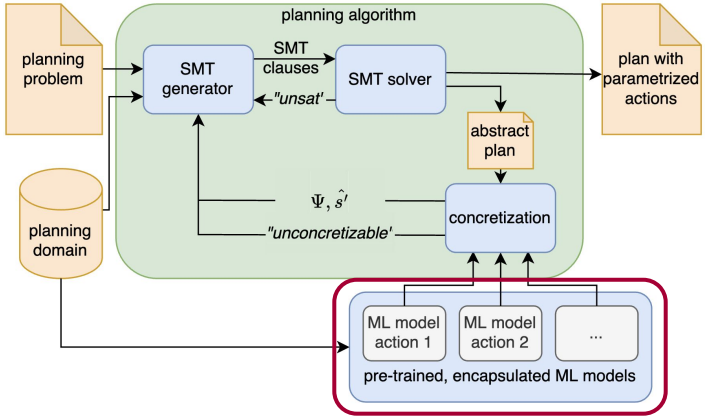
Estimating Parameters







Lazy Approach – Concretization based on Models Access

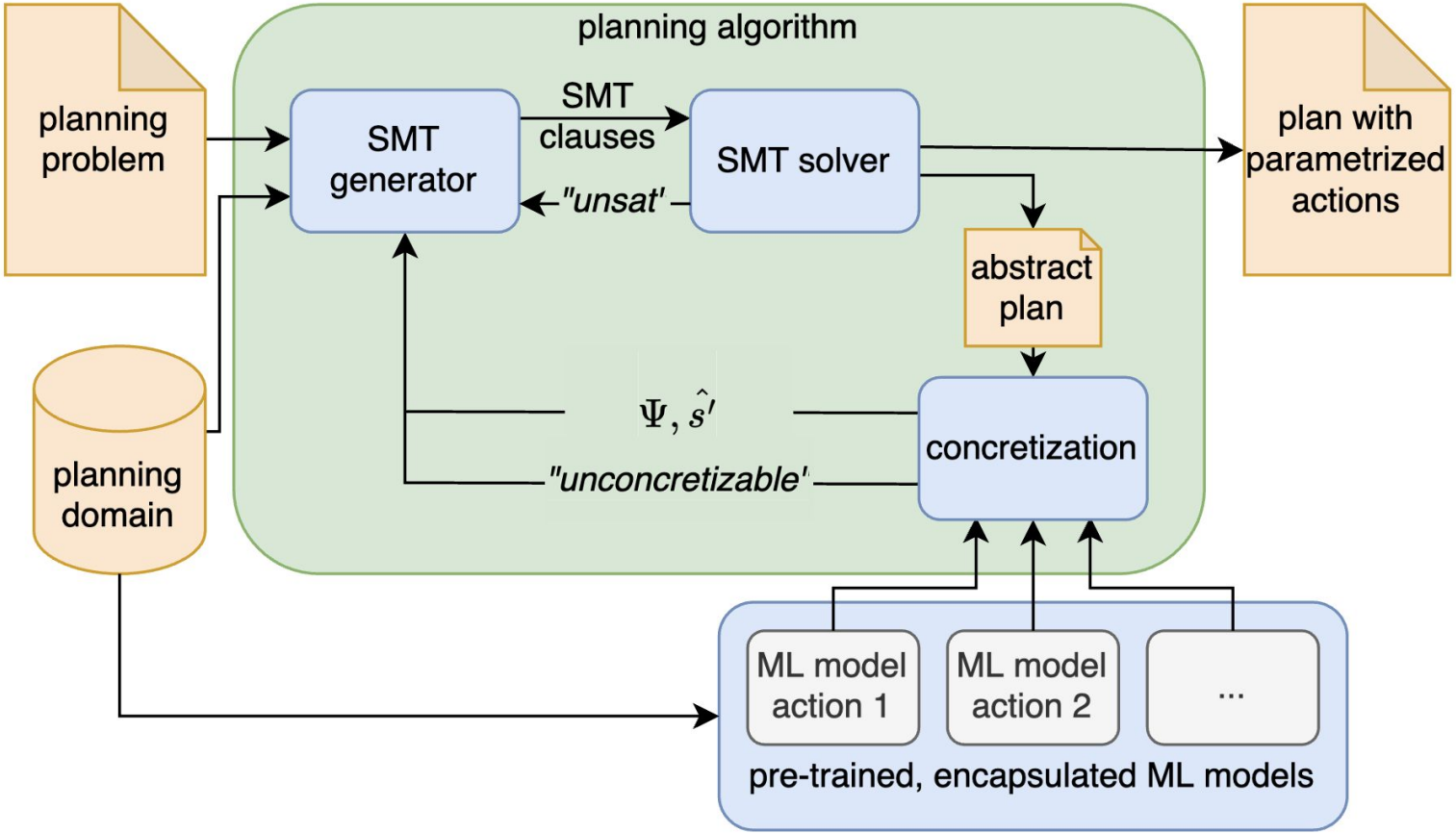


Full Access:
- So-called *white-box models*
- Known architecture
- Known weights and biases

Partial Access:
- Unknown architecture
- Unknown weights and biases
- Access to dedicated functionalities such as backpropagation

No Access:
- So-called *black-box models*
- Unknown architecture
- Unknown weights and biases
- No functionalities available

Lazy Approach – **L**azy **N**eural **P**laner

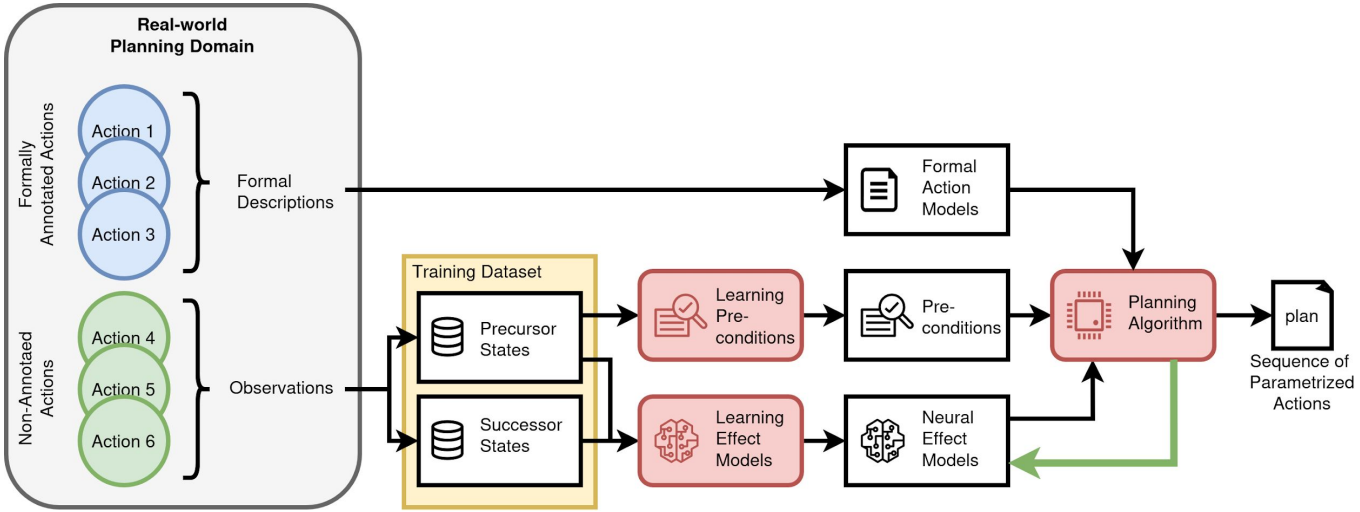


Evaluation – Comparison Lazy and Eager Approach

Table 1. The table shows the computation time in seconds of the eager and the lazy approach for finding an optimal solution in seconds. (TO) denotes that the algorithm was not able to find a plan within seven days (648,000 seconds). The NNs have **ReLU** activation functions and are full-access models.

domain	plan length	Eager	Lazy
FliPSi	2 actions	TO	5.60
	3 actions	TO	5.37
	5 actions	TO	5.55
Drone	2 actions	TO	5.46
	3 actions	TO	5.65
	15 actions	TO	15.69
Zeno	2 actions	TO	5.78
	3 actions	TO	6.06
	9 actions	TO	8.03
Cashpoint	2 actions	TO	5.73
	3 actions	TO	5.88
	6 actions	TO	6.49

8. Outlook


Real-world
Planning Domain
Formally
Annotated Actions
Action 1
Action 2
Action 3
Formal
Descriptions
Non-Annotaed
Actions
Action 4
Action 5
Action 6
Observations
Training Dataset
Precursor
States
Successor
States
Learning
Pre-
conditions
Learning
Effect
Models
Formal
Action
Models
Pre-
conditions
Neural
Effect
Models
Planning
Algorithm
plan
Sequence of
Parametrized
Actions

Limitations & Open Challenges

- At the moment, only the exact values for which the concretization failed, are blocked.

  → possible countless iterations between abstract level and concretization

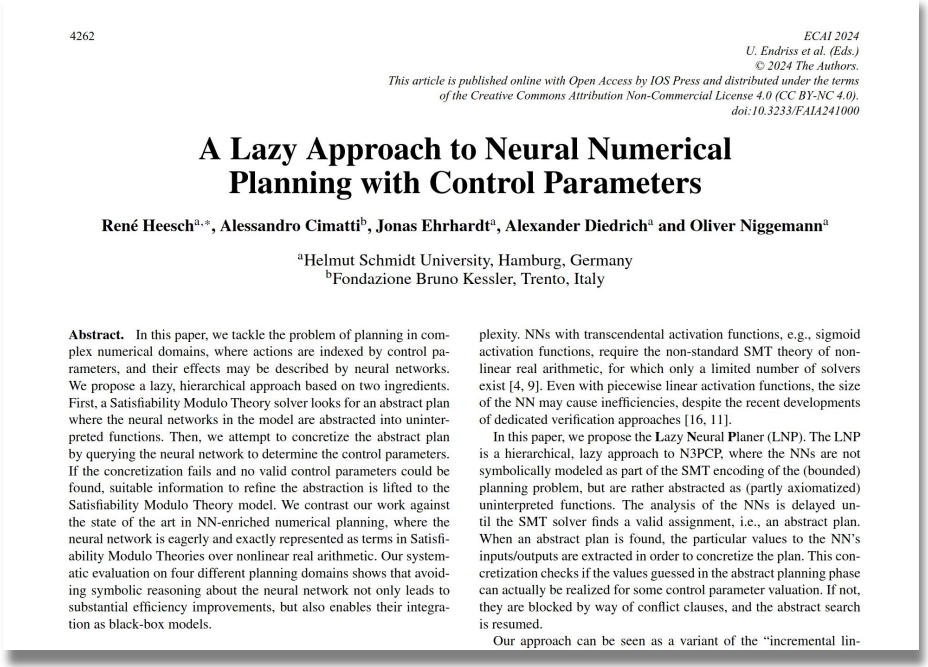
- Data: For both, the preconditions and the effects, we can only learn what is represented within the data.

- Control parameters influencing multiple actions

Outlook

- Extend to temporally N3PCP
- Improve the blocking, if a plan could not be concretized
- Integrate the reasoning about NN's directly in the SMT solver in a tighter way, following an online integration paradigm

Outlook



4262

ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA241000

A Lazy Approach to Neural Numerical Planning with Control Parameters

René Heesch[a,*], Alessandro Cimatti[b], Jonas Ehrhardt[a], Alexander Diedrich[a] and Oliver Niggemann[a]

[a]Helmut Schmidt University, Hamburg, Germany
[b]Fondazione Bruno Kessler, Trento, Italy

Abstract. In this paper, we tackle the problem of planning in complex numerical domains, where actions are indexed by control parameters, and their effects may be described by neural networks. We propose a lazy, hierarchical approach based on two ingredients. First, a Satisfiability Modulo Theory solver looks for an abstract plan where the neural networks in the model are abstracted into uninterpreted functions. Then, we attempt to concretize the abstract plan by querying the neural network to determine the control parameters. If the concretization fails and no valid control parameters could be found, suitable information to refine the abstraction is lifted to the Satisfiability Modulo Theory model. We contrast our work against the state of the art in NN-enriched numerical planning, where the neural network is eagerly and exactly represented as terms in Satisfiability Modulo Theories over nonlinear real arithmetic. Our systematic evaluation on four different planning domains shows that avoiding symbolic reasoning about the neural network not only leads to substantial efficiency improvements, but also enables their integration as black-box models.

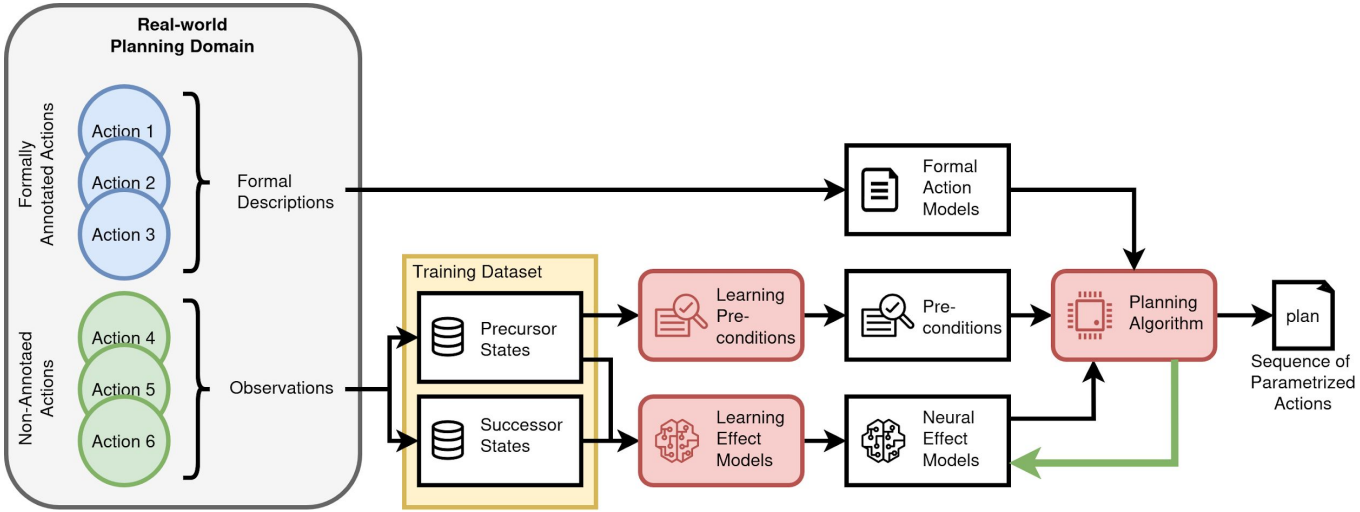
plexity. NNs with transcendental activation functions, e.g., sigmoid activation functions, require the non-standard SMT theory of nonlinear real arithmetic, for which only a limited number of solvers exist [4, 9]. Even with piecewise linear activation functions, the size of the NN may cause inefficiencies, despite the recent developments of dedicated verification approaches [16, 11].

In this paper, we propose the **L**azy **N**eural **P**laner (LNP). The LNP is a hierarchical, lazy approach to N3PCP, where the NNs are not symbolically modeled as part of the SMT encoding of the (bounded) planning problem, but are rather abstracted as (partly axiomatized) uninterpreted functions. The analysis of the NNs is delayed until the SMT solver finds a valid assignment, i.e., an abstract plan. When an abstract plan is found, the particular values to the NN's inputs/outputs are extracted in order to concretize the plan. This concretization checks if the values guessed in the abstract planning phase can actually be realized for some control parameter valuation. If not, they are blocked by way of conflict clauses, and the abstract search is resumed.

Our approach can be seen as a variant of the "incremental lin-

Heesch, R., Cimatti, A. Ehrhardt, J., Diedrich, A. & Niggemann, O. (2024). A Lazy Approach to Neural Numerical Planning with Control Parameters, 27TH European Conference on Artificial Intelligence (ECAI), Santiago de Compostela, Spain.

Questions?



René Heesch
Institute for Artificial Intelligence
Helmut Schmidt University
Holstenhofweg 85, 22043 Hamburg
tel.: +49 (0)40 / 6541-2875
email: rene.heesch@hsu-hh.de
github: https://github.com/rheesch

Jonas Ehrhardt
Institute for Artificial Intelligence
Helmut Schmidt University
Holstenhofweg 85, 22043 Hamburg
tel.: +49 (0)40 / 6541-3291
email: jonas.ehrhardt@hsu-hh.de
github: https://github.com/j-ehrhardt